U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

PhD

3rd CYCLE

FCUP
2026

U.PORTO

A Quantitative Study of Higher-Order Computations with Effects

Miguel Ramos

FC

U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

PhD

A Quantitative Study of Higher-Order Computations with Effects

Miguel Ramos
Doctoral Program in Computer Science
Department of Computer Science
Faculty of Sciences of the University of Porto

2026

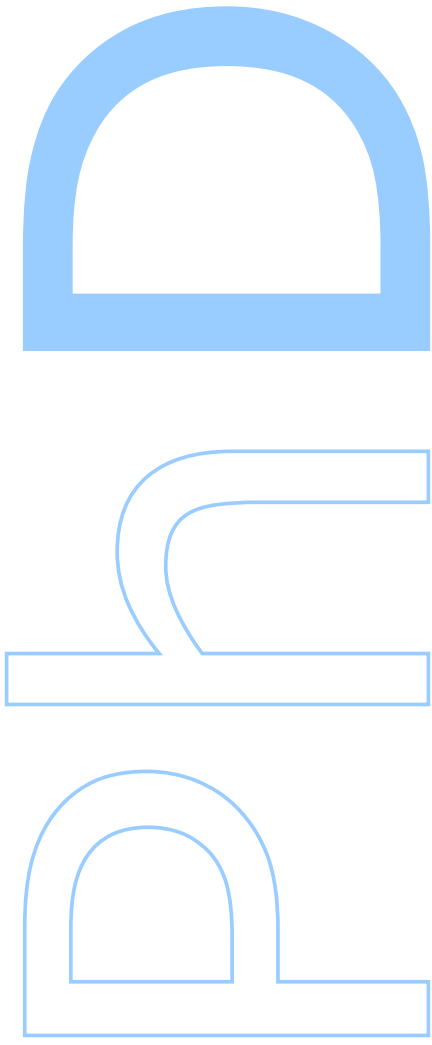
PhD

A Quantitative Study of Higher-Order Computations with Effects

Miguel Ramos
Thesis carried out as part of the
Doctoral Program in Computer Science
Department of Computer Science
2026

Supervisor
Professor Sandra Alves, Assistant Professor, Faculty of Sciences of the University of Porto

Co-supervisor
Professor Delia Kesner, Full Professor, Université Paris Cité



University of Porto

Doctoral Thesis

A Quantitative Study of Higher-Order Computations with Effects

Author:
Miguel Ramos

Supervisor:
Sandra Alves

Co-supervisor:
Delia Kesner

A thesis submitted in fulfilment of the requirements
for the degree of PhD in Computer Science

at the

Faculty of Sciences of the University of Porto
Department of Computer Science

June 25, 2026

“ The waves are not the ocean. ”

Pat Metheny

Acknowledgements

There are many people to whom I owe thanks, and it feels natural to acknowledge them in the order in which they entered my life.

I begin with my parents, Jorge and Margarida. Beyond the obvious biological reasons, I am deeply grateful to them for encouraging me and always trusting my judgement. Their constant support made it possible for me to follow a path that was not always straightforward, but that ultimately led me here.

More broadly, I would like to thank my entire family. Even without contributing directly to the writing of this thesis, their encouragement and presence throughout my life have played an essential role in everything that I will continue to achieve.

I would next like to thank Ana. Not only for her patience and support, but because she was the person who helped me discover my passion for mathematics and computer science. Without her, I would never have found the courage to change majors from biology to computer science. Thank you for always reminding me of who I am. (I am also grateful for something more important—but more on that later.)

I am also deeply thankful to Sandra. Thank you for introducing me to the λ-calculus and for being such a dedicated and generous advisor throughout these years—ten of them, I believe, at this point (sorry!).

I would also like to thank Delia, for being a wonderful advisor and for making sure that I always felt welcomed and safe during my visits to Paris.

During my PhD, I had the privilege of meeting many colleagues who made this journey richer, both intellectually and personally. I would like to thank João and Pedro for being there at the very beginning—helping me settle into our (now former) shared office, and for the many discussions that followed. I am also grateful to the friends I made in Paris: Daniele, Mariana, Victor, Adrienne, Riccardo, and Gabriele. Thank you all for the conversations, the companionship, and for making my time there truly enjoyable. I would also like to thank some of my more recent companions: Rodrigo, Guilherme, Pedro, Ana Jorge, and Ana Catarina.

Having thanked those who accompanied me during my PhD, I would also like to remember those who are no longer here, but whose lives shaped who I am today: Rosa, Arminda, and Fátima.

Now, I would like to thank someone who is not here *yet*. Thank you, Gabriel, for making this past year the best year of my life. You do not know it yet, but you were the one who helped me the most. Because of you, I was able to write this thesis with a smile on my face and a skip in my step—something that, I am told, is quite unusual.

For this reason, I dedicate this thesis to you, my little bean, Gabriel.

UNIVERSITY OF PORTO

Abstract

Faculty of Sciences of the University of Porto
Department of Computer Science

PhD in Computer Science

A Quantitative Study of Higher-Order Computations with Effects

by Miguel Ramos

Non-idempotent intersection types have emerged as a powerful logical framework for providing exact quantitative semantics of higher-order programs, allowing typing derivations to capture precise information about resource usage such as evaluation length. While these techniques are now well understood for the pure λ-calculus, extending them to richer language features involving control and computational effects remains a challenging problem.

This thesis develops non-idempotent intersection type systems for extensions of the λ-calculus featuring pattern matching mechanisms and computational effects, with the aim of providing exact, syntax-directed cost semantics grounded in operational semantics. The contributions are organized around two main axes.

The first part of the thesis studies pattern matching as a fundamental extension of the λ-calculus. Building on prior quantitative analyses of pattern-matching calculi, we develop refined and generalized non-idempotent intersection type systems that preserve exact correspondence between typing derivations and evaluation length. Particular attention is devoted to the treatment of stuck computations. Exploiting the classical encoding of exceptions via pattern matching, we extend this analysis to exceptional control flow, showing that exception raising and handling admit exact quantitative semantics despite constituting a non-algebraic computational effect.

The second part of the thesis addresses algebraic operations and effects. We focus on algebraic theories that admit non-trivial comodels, which provide a coalgebraic basis for defining fine-grained operational semantics via comodel-based transition systems. For calculi with global state, we define such operational semantics and develop non-idempotent intersection type systems that precisely account for both computational and effectful steps. We then generalize this approach by modeling state as a mapping from locations to infinite stacks, yielding a uniform quantitative framework capable of capturing a broad class of algebraic effects, including global state, interactive input/output, and finite nondeterminism.

Throughout the thesis, the $\lambda!$-calculus plays a unifying role, subsuming call-by-name and call-by-value evaluation strategies while retaining the operational granularity required for exact quantitative reasoning. Overall, this work significantly extends the scope of non-idempotent intersection types, demonstrating their applicability to languages with pattern matching, exceptions, and algebraic effects, and providing a unified operational and quantitative account of resource-sensitive computation.

UNIVERSIDADE DO PORTO

Resumo

Faculdade de Ciências da Universidade do Porto
Departamento de Ciência de Computadores

Doutoramento em Ciência de Computadores

Um Estudo Quantitativo de Computações de Ordem Superior com Efeitos

por Miguel Ramos

Os tipos de interseção não-idempotentes emergiram como um poderoso enquadramento lógico para fornecer semânticas quantitativas exatas de programas de ordem superior, permitindo que as derivações de tipagem captem informação precisa sobre o uso de recursos, como o comprimento da avaliação. Embora estas técnicas estejam atualmente bem compreendidas para o λ-cálculo puro, a sua extensão a funcionalidades linguísticas mais ricas que envolvem controlo e efeitos computacionais continua a ser um problema desafiante.

Esta tese desenvolve sistemas de tipos de interseção não-idempotentes para extensões do λ-cálculo que incorporam mecanismos de *pattern matching* e efeitos computacionais, com o objetivo de fornecer semânticas de custo exatas, dirigidas pela sintaxe e fundamentadas na semântica operacional. As contribuições organizam-se em torno de dois eixos principais.

A primeira parte da tese estuda o *pattern matching* como uma extensão fundamental do λ-cálculo. Com base em análises quantitativas prévias de cálculos com *pattern matching*, desenvolvemos sistemas de tipos de interseção não-idempotentes refinados e generalizados que preservam uma correspondência exata entre as derivações de tipagem e o comprimento da avaliação. É dada particular atenção ao tratamento de computações bloqueadas (*stuck computations*). Explorando a codificação clássica de exceções através de *pattern matching*, estendemos esta análise ao controlo excecional, mostrando que a propagação e o tratamento de exceções admitem semânticas quantitativas exatas, apesar de constituírem um efeito computacional não algébrico.

A segunda parte da tese aborda operações e efeitos algébricos. Concentramo-nos em teorias algébricas que admitem comodelos não triviais, os quais fornecem uma base coalgébrica para a definição de semânticas operacionais de granularidade fina através de sistemas de transição baseados em comodelos. Para cálculos com estado global, definimos tais semânticas operacionais e desenvolvemos sistemas de tipos de interseção não-idempotentes que contabilizam com precisão

tanto os passos computacionais como os passos associados a efeitos. Em seguida, generalizamos esta abordagem modelando o estado como um mapeamento de localizações para pilhas infinitas, obtendo um enquadramento quantitativo uniforme capaz de capturar uma vasta classe de efeitos algébricos, incluindo estado global, *input/output* interativo e não-determinismo finito.

Ao longo da tese, o $\lambda!$-cálculo desempenha um papel unificador, subsumindo as estratégias de avaliação *call-by-name* e *call-by-value*, ao mesmo tempo que preserva a granularidade operacional necessária para um raciocínio quantitativo exato. No seu conjunto, este trabalho alarga significativamente o âmbito dos tipos de interseção não-idempotentes, demonstrando a sua aplicabilidade a linguagens com *pattern matching*, exceções e efeitos algébricos, e fornecendo uma descrição operacional e quantitativa unificada da computação sensível a recursos.

Contents

Acknowledgements v

Abstract vii

Resumo ix

Contents xi

I Prelude 1

1 Introduction 3

II Prologue 13

2 The λ-Calculus 15
2.1 Syntax and Operational Semantics . . . 15
2.2 Call-By-Name and Call-By-Value . . . 18

3 The $\lambda!$-Calculus 21
3.1 Syntax and Operational Semantics . . . 22
3.2 Subsuming Call-By-Name and Call-By-Value . . . 24

4 Non-Idempotent Intersection Types 27
4.1 Call-By-Name and Call-By-Value . . . 29
4.2 Subsuming Call-By-Name and Call-By-Value . . . 36
4.3 Tight Typings . . . 40

5 Monads and Algebraic Operations 45
5.1 Monads . . . 45
5.2 Relevant Examples . . . 48
5.3 Algebraic Operations . . . 51
5.4 Comodel-Based Operational Semantics . . . 57

III Pattern-Matching Computations 63

6 A Quantitative Study of Pattern-Matching 65
6.1 The Pattern-Matching Calculus . . . 66

6.1.1 Syntax and Operational Semantics . . . 66
6.1.2 Encoding Exceptions . . . 73
6.1.3 Encoding the CBN and CBV λ-Calculi . . . 73
6.2 Pattern-Matching Non-Idempotent Intersection Types . . . 75
6.2.1 Quantitative Soundness . . . 80
6.2.2 Quantitative Completeness . . . 81

IV State-Passing Computations 83

7 A Quantitative Study of Global State 85
7.1 The λ-Calculus with Global State . . . 90
7.1.1 Syntax and Operational Semantics . . . 90
7.2 The $\lambda!$-Calculus with Global State . . . 95
7.2.1 Syntax and Operational Semantics . . . 95
7.2.2 Subsuming Call-By-Name and Call-By-Value . . . 98
7.2.3 Correctness of Full Simulations . . . 100
7.3 State-Passing Non-Idempotent Intersection Types . . . 103
7.3.1 Call-By-Name . . . 106
7.3.1.1 Quantitative Soundness . . . 113
7.3.1.2 Quantitative Completeness . . . 113
7.3.2 Call-By-Value . . . 114
7.3.2.1 Quantitative Soundness . . . 119
7.3.2.2 Quantitative Completeness . . . 120
7.3.3 Subsuming Call-By-Name and Call-By-Value . . . 120
7.3.3.1 Quantitative Soundness . . . 124
7.3.3.2 Quantitative Completeness . . . 125
7.3.3.3 Soundness and Completeness of Translations . . . 126

8 A Quantitative Study of Infinite Stacks 131
8.1 The Algebraic Theory of Infinite Stacks . . . 132
8.2 Handling Infinite Stacks using Global State . . . 133
8.2.1 The Comodel for Infinite Stacks . . . 136
8.3 The λ-Calculus with Infinite Stacks . . . 137
8.3.1 Syntax and Operational Semantics . . . 137
8.4 The $\lambda!$-Calculus with Infinite Stacks . . . 139
8.4.1 Syntax and Operational Semantics . . . 139
8.4.2 Subsuming Call-By-Name and Call-By-Value . . . 141
8.5 State-Passing Non-Idempotent Intersection Types . . . 143
8.5.1 Call-By-Name . . . 143
8.5.1.1 Quantitative Soundness . . . 146
8.5.1.2 Quantitative Completeness . . . 146
8.5.2 Call-By-Value . . . 147
8.5.2.1 Quantitative Soundness . . . 149
8.5.2.2 Quantitative Completeness . . . 150
8.5.3 Subsuming Call-By-Name and Call-By-Value . . . 150
8.5.3.1 Quantitative Soundness . . . 153
8.5.3.2 Quantitative Completeness . . . 153
8.5.3.3 Soundness and Completeness of Translations . . . 154

8.6 Handling State-Passing Computations using Infinite Stacks . . . 156
8.6.1 Computing the Handlers . . . 156
8.6.1.1 Global State . . . 156
8.6.1.2 Interactive Input . . . 158
8.6.1.3 Interactive Output . . . 159
8.6.1.4 Nondeterminism . . . 159
8.6.1.5 Interactive IO . . . 159
8.6.2 Examples of Using the Handlers . . . 160
8.6.2.1 Handling Global State Using Infinite Stacks . . . 161
8.6.2.2 Handling Interactive IO using Infinite Stacks . . . 166

V Epilogue **173**

9 Conclusion & Future Work **175**
9.1 Conclusion . . . 175
9.2 Future Work . . . 177
9.2.1 Completeness of Call-By-Value Translations for Stateful Extensions . . . 177
9.2.2 A General Quantitative Framework for Algebraic Effects . . . 178
9.2.3 Combining State-Passing Semantics with Relational Intersection Types . . . 178
9.2.4 Further Directions . . . 179

VI Full Proofs **181**

A The λ-Calculus **183**
A.1 Syntax and Operational Semantics . . . 183
A.2 Call-By-Name and Call-By-Value . . . 184

B The $\lambda!$-Calculus **187**
B.1 Syntax and Operational Semantics . . . 187
B.2 Subsuming Call-By-Name and Call-By-Value . . . 189

C Non-Idempotent Intersection Types **195**
C.1 Call-By-Name and Call-By-Value . . . 195
C.2 Subsuming Call-By-Name and Call-By-Value . . . 203

D A Quantitative Study of Pattern-Matching **215**
D.1 The Pattern-Matching Calculus . . . 215
D.1.1 Syntax and Operational Semantics . . . 215
D.1.2 Encoding the CBN and CBV λ-Calculi . . . 227
D.2 Pattern-Matching Non-Idempotent Intersection Types . . . 230
D.2.1 Quantitative Soundness . . . 235
D.2.2 Quantitative Completeness . . . 246

E A Quantitative Study of Global State **259**
E.1 The λ-Calculus with Global State . . . 259
E.1.1 Syntax and Operational Semantics . . . 259
E.2 The $\lambda!$-Calculus with Global State . . . 259
E.2.1 Syntax and Operational Semantics . . . 259

E.2.2 Subsuming Call-By-Name and Call-By-Value . . . 260
E.2.3 Correctness of Full Simulations . . . 260
E.3 State-Passing Non-Idempotent Intersection Types . . . 270
E.3.1 Call-By-Name . . . 270
E.3.1.1 Quantitative Soundness . . . 274
E.3.1.2 Quantitative Completeness . . . 279
E.3.2 Call-By-Value . . . 284
E.3.2.1 Quantitative Soundness . . . 286
E.3.2.2 Quantitative Completeness . . . 292
E.3.3 Subsuming Call-By-Name and Call-By-Value . . . 298
E.3.3.1 Quantitative Soundness . . . 301
E.3.3.2 Quantitative Completeness . . . 305
E.3.3.3 Soundness and Completeness of Translations . . . 310

F A Quantitative Study of Infinite Stacks **323**
F.1 State-Passing Non-Idempotent Intersection Types . . . 323
F.1.1 Call-By-Name . . . 323
F.1.1.1 Quantitative Soundness . . . 325
F.1.1.2 Quantitative Completeness . . . 328
F.1.2 Call-By-Value . . . 331
F.1.2.1 Quantitative Soundness . . . 332
F.1.2.2 Quantitative Completeness . . . 335
F.1.3 Subsuming Call-By-Name and Call-By-Value . . . 338
F.1.3.1 Quantitative Soundness . . . 339
F.1.3.2 Quantitative Completeness . . . 340
F.1.3.3 Soundness and Completeness of Translations . . . 341

Bibliography **343**

This thesis is dedicated to my little bean.

May you grow into a strong and beautiful tree.

Part I

Prelude

Chapter 1

Introduction

Understanding *how much* computation a program performs is as important as understanding *what* it computes. Beyond functional correctness, modern programming languages demand precise reasoning about resources such as time, space, and interactions with effects. Over the last two decades, *non-idempotent intersection types* have emerged as a particularly robust logical tool for providing *exact quantitative semantics* of higher-order programs.

Unlike traditional (idempotent) type systems, which primarily enforce qualitative properties such as termination or safety, non-idempotent intersection types retain fine-grained information about resource usage. By rejecting idempotency, these systems record how many times a term is used: duplicating a program component corresponds to duplicating its type. As a result, typing derivations themselves become quantitative objects, from which exact measures—such as the number of evaluation steps or the size of normal forms—can be extracted directly. This approach replaces semantic reducibility arguments with combinatorial reasoning on typing derivations, making quantitative analysis both precise and tractable.

A substantial body of work has established tight correspondences between non-idempotent intersection types and the operational semantics of the pure λ-calculus, for a variety of evaluation strategies and cost models [1–7]. These results show that typing derivations can provide not only upper bounds, but often *exact* measures of evaluation length.

Scope and contributions of the thesis. This thesis contributes to this line of research by developing non-idempotent intersection type systems for *extensions of the λ-calculus featuring pattern-matching mechanisms and computational effects*. Our overarching goal is to show that non-idempotent intersection types provide *exact, syntax-directed cost semantics* for a broad class of language features that go beyond the pure λ-calculus, while remaining firmly grounded in operational semantics.

The thesis is structured around two main axes:

- *Pattern-matching mechanisms*, viewed as a fundamental extension of the λ-calculus and a basis for reasoning about exceptional control flow;
- *Algebraic operations and effects*, restricted to those algebraic theories that admit non-trivial comodels, and hence support a fine-grained operational interpretation.

Together, these contributions extend the reach of quantitative typing beyond purely functional computation, encompassing both control and stateful effects.

The key notions informally introduced above form the conceptual backbone of the thesis and will be introduced formally and discussed in depth in the background chapters of this thesis (Chapters 2 to 5).

Quantitative Semantics via Non-Idempotent Intersection Types

Non-idempotent intersection types originated in the study of relevance and linearity [1, 2], and later re-emerged as a central tool for quantitative semantics [3, 4]. In contrast with idempotent typing systems, where $A \cap A = A$, non-idempotent systems distinguish multiple uses of the same type, allowing resource usage to be tracked explicitly.

Foundational work by de Carvalho [3, 4] and subsequent developments by Accattoli, Dal Lago, Kesner, Ventura, and others [5, 8–15] have shown that these systems yield exact characterizations of normalization, evaluation lengths, and sometimes even space usage, for several variants of the λ-calculus. These techniques have been extended to calculi with explicit substitutions [16], call-by-need evaluation [17], and refined cost models [7], as well as classical calculi [10].

This thesis builds on these foundations, extending quantitative typing techniques to richer language constructs where duplication, control flow, and effects interact in subtler ways.

Pattern-Matching Mechanisms and Exceptions

Pattern-matching is a ubiquitous programming construct, underlying features such as destructuring, case analysis, and exception handling. From an operational point of view, pattern matching introduces non-trivial interactions between substitution, control, and failure, and has been studied extensively both operationally and logically [18–27].

More recently, quantitative analyses of pattern-matching calculi have been developed using non-idempotent intersection types. In particular, Alves, Kesner, and Ventura [28] introduced a pattern-matching λ-calculus equipped with a fine-grained operational semantics and a non-idempotent type system providing exact bounds on evaluation length. Their work builds on earlier studies of observability and pattern-restricted calculi [29, 30].

The first contribution of this thesis is to refine and generalize this line of work. We develop non-idempotent intersection type systems for a broader class of pattern-matching constructs, while preserving exact quantitative correspondence between typing derivations and operational evaluation. Particular care is devoted to the treatment of *stuck* computations (*i.e.* computations that halt for some reason before reaching a result), allowing us to distinguish programs that successfully terminate from those that fail to match.

A second, closely related contribution concerns *exceptions*. It is well known that exception raising and handling can be encoded using pattern matching. Exploiting this observation, we lift our quantitative analysis to exceptional control flow, covering both exception propagation and exception capture. Importantly, *exception handling* constitutes a form of higher-order computational effect that is *not algebraic* (*cf.* Remark 5.30) in the sense of Plotkin and Power [31]. Nevertheless, by viewing exceptions through the lens of pattern matching, we show that non-idempotent intersection types still provide exact cost semantics for their execution.

This result demonstrates that the applicability of non-idempotent intersection types extends beyond algebraic effects, encompassing control mechanisms traditionally considered difficult to analyze quantitatively.

Call-by-Push-Value and the $\lambda!$-Calculus

A recurring theme throughout the thesis is the fine-grained decomposition of computation. Historically, call-by-name and call-by-value have been studied largely independently, each giving rise to its own calculi, operational semantics, and semantic models. While this separation has led to deep and well-developed theories, it has also made the transfer of results between evaluation strategies technically delicate and conceptually non-uniform. In particular, properties established in one setting often require substantial reworking to be adapted to the other.

Call-by-Push-Value (CBPV), introduced by Levy [32, 33], provides a foundational framework that unifies call-by-name and call-by-value by distinguishing *values* from *computations*. From a logical perspective, CBPV is closely related to linear logic and to calculi equipped with an explicit *bang* modality [34].

The $\lambda!$-calculus studied in this thesis can be seen as an untyped, operational counterpart of CBPV [35]. It subsumes both call-by-name and call-by-value λ-calculi within a single term language, making duplication and erasure explicit in direct correspondence with the structural rules of contraction and weakening in linear logic. In this setting, the *bang* modality governs precisely when resources may be duplicated or discarded, rather than allowing these operations

implicitly as in the traditional λ-calculus. This explicit treatment of resources makes the $\lambda!$-calculus particularly well suited for exact quantitative analysis using non-idempotent intersection types.

Throughout the thesis, the $\lambda!$-calculus plays a unifying role: in Chapter 3 it subsumes call-by-name and call-by-value evaluation of the pure λ-calculus; in Chapter 7 it supports quantitative analysis in the presence of global state; and in Chapter 8 it is extended to more general stateful effects.

Algebraic Operations and Comodel-Based Semantics

The second main contribution of the thesis concerns *algebraic operations and effects*. Algebraic effects, introduced by Plotkin and Power [36, 37], provide a modular and mathematically elegant framework for describing computational effects such as state, nondeterminism, or input/output. In this approach, effects are specified by a collection of operation symbols and equations, in the spirit of universal algebra, and are interpreted semantically as operations on suitable monads. This algebraic presentation allows effects to be combined and reasoned about independently of any particular programming language. However, not all algebraic theories admit an operational interpretation that is fine-grained enough to support exact quantitative analysis.

We therefore restrict attention to *algebraic theories that admit non-trivial comodels* in the sense of [38]. Comodels provide a coalgebraic perspective on effects, allowing operational semantics to be defined in terms of explicit *comodel-based transition systems* [39]. This yields step-by-step descriptions of effectful computation that integrate smoothly with non-idempotent typing techniques.

We first study an extension of the λ-calculus with algebraic operations modeling *global state*. Building on earlier work on intersection types for effectful calculi [40–44], we define a comodel-based operational semantics and develop a non-idempotent intersection type system that precisely accounts for both computational and effectful steps.

We then show that this analysis can be significantly generalized. By modeling state as a mapping from locations to *infinite stacks*, we capture a wide class of algebraic theories admitting non-trivial comodels. This construction provides a uniform operational framework for quantitative analysis of algebraic effects, highlighting the expressive power of comodel-based semantics when combined with non-idempotent intersection types.

Related Work

The present thesis builds on a substantial body of work on non-idempotent intersection types, also known as quantitative or multi type systems [1–7]. Since Gardner's early observation that

dropping idempotency turns typing derivations into resource-sensitive objects [1], this line of research has developed into a robust framework in which typability characterizes various forms of normalization, and the size of typing derivations yields tight—and often exact—bounds on evaluation length and normal form size [3–7, 11]. A key conceptual contribution is the replacement of semantic reducibility arguments by combinatorial reasoning on typing derivations, refining termination from a qualitative to a quantitative property [3, 5, 6].

Exact correspondences between non-idempotent intersection types and operational semantics have been established for a wide range of evaluation strategies, including call-by-name, call-by-value, call-by-need, and linear or head-driven strategies [3–7]. The development of tight typings has been particularly influential, showing that minimal derivations yield exact bounds on both evaluation length and normal-form size [5], establishing non-idempotent intersection types as a precise, syntax-directed tool for quantitative cost semantics in purely functional settings [3–5].

A related line of work connects non-idempotent intersection types with linear logic and relational semantics, where they can be seen as proof-relevant refinements of the relational model [3, 35]. This perspective provides a logical account of resource usage aligned with explicit control over duplication and erasure, and with CBPV-inspired calculi such as the $\lambda!$-calculus [35, 45], allowing uniform accounts of normalization [35, 45] and inhabitation [46] across multiple evaluation strategies.

Beyond the pure λ-calculus, pattern-matching calculi have long been studied using qualitative intersection types [29, 30]. More recently, non-idempotent intersection types have been applied to pattern-matching languages with pairs, constructors, and case analysis, yielding quantitative bounds—and sometimes exact measures—on evaluation length [28]. These systems track multiple resources and provide a fine-grained account of control flow and failure. The present thesis builds directly on this quantitative pattern-matching tradition, extending it to richer constructs and to the analysis of stuck computations and exceptions.

The treatment of computational effects introduces further challenges. Early adaptations of intersection types focused on qualitative properties, such as may/must convergence in nondeterministic or probabilistic calculi [47]. More recently, monadic intersection type systems for λ-calculi with algebraic operations à la Plotkin and Power provide modular accounts of a wide class of effects (output, cost, nondeterminism, probabilistic choice), characterizing observational behaviors in finitary and infinitary settings [48, 49]. However, these approaches largely remain qualitative, do not cover stateful computations, and extending them to exact quantitative analysis—especially in the presence of combined effects—remains an open problem [48, 49].

Complementary work develops quantitative types for the Functional Machine Calculus, an

effectful λ-calculus with global state, input/output, and probabilistic choice [50]. There, non-idempotent intersection types measure machine transitions exactly in a weak variant and provide upper bounds under a strong evaluation variant, showing that multi-types can be adapted to machine-based operational semantics for rich effectful languages. However, these results are not phrased in terms of algebraic effects at the source level.

On the semantic side, algebraic effects and handlers are well-understood via equational theories and free-model monads [51]. Comodels provide an environment-based semantics for effectful computations, yielding operational "machines" for algebraic theories [52], and Garner's costructure-cosemantics adjunction relates accessible monads to presheaf comonads encoding comodel behaviors [53, 54]. At the programming-language level, runners operationalize comodels: Ahman and Bauer's λ_{coop}-calculus treats runners as first-class resources with a denotational semantics enforcing linear use and finalization [55], connecting categorical comodel semantics with practical resource management [56].

Exceptions lie at the interface of pattern matching and effects. Their encodability via pattern matching allows non-idempotent intersection types to track exception-like control [28, 30], though no direct monadic treatments currently account simultaneously for computation and exception propagation. These studies confirm the feasibility of quantitative analyses for non-algebraic control constructs.

The present thesis contributes to this landscape by developing exact quantitative semantics for higher-order languages combining pattern matching and effects, grounded in operational semantics and supported by comodel-based interpretations of algebraic operations and runners for external resources.

Contributions and Structure of the Thesis

The research developed in this thesis consolidates and extends a series of publications produced during the PhD. Their correspondence with the chapters and sections of the thesis is summarized below.

- Journal Papers:
  - Sandra Alves, Delia Kesner, Miguel Ramos. A Quantitative Approach to Global State Composition. Mathematical Structures in Computer Science (MSCS), 2025.
    This paper corresponds to the results developed in Chapter 7, which presents a quantitative analysis of global state using non-idempotent intersection types. In particular, it underpins the operational semantics and the quantitative soundness and completeness results for the λ- and

$\lambda!$-calculi with global state. These results also serve as a conceptual and technical basis for the generalizations developed later in Chapter 8.

- Conference Papers:
  - Sandra Alves, Delia Kesner, Miguel Ramos. Extending the Quantitative Pattern--Matching Paradigm. Asian Symposium on Programming Languages and Systems (APLAS), 2024.
    This paper forms the core of Chapter 6, where the pattern-matching calculus, its non-idempotent intersection type system, and the associated quantitative soundness and completeness results are developed and extended.
  - Sandra Alves, Delia Kesner, Miguel Ramos. Quantitative Global Memory. Workshop on Logic, Language, Information and Computation (WoLLIC), 2023.
    This paper presents a first quantitative study of global state and motivates the development carried out in Chapter 7, where the analysis is substantially refined, extended, and completed.
- Workshop Papers:
  - Miguel Ramos. Non-Idempotent Intersection Types for Global State. Workshop on Intersection Types and Related Systems (ITRS), 2024.
    This paper presents an early quantitative analysis of global state restricted to the call-by-value setting. Its technical core is incorporated and generalized in Section 7.3, where a subsuming quantitative framework is developed.
  - Miguel Ramos, Riccardo Treglia, Delia Kesner. Quantitative Understanding of Exceptions. International Workshop on Trends in Linear Logic and Applications (TLLA), 2023.
    This paper is subsumed by Section 6.1.2, where exceptions are encoded via pattern matching and shown to admit an exact quantitative analysis, despite constituting a non-algebraic computational effect.
  - Sandra Alves, Delia Kesner, Miguel Ramos. Extending the Quantitative Pattern--Matching Paradigm. International Workshop on Logical and Semantic Frameworks, with Applications (LSFA), 2021.
    This paper contains early foundational results on quantitative pattern matching that are substantially extended and generalized throughout Chapter 6, and should be viewed as a precursor to the APLAS 2024 contribution and its full development in this thesis.

In summary, the main contributions of this thesis are:

- The extension of non-idempotent intersection type techniques to pattern-matching mechanisms, with exact quantitative semantics.
- An exact quantitative analysis of exceptions, obtained via their encoding as pattern-matching constructs.
- A comodel-based operational semantics for λ-calculi with algebraic operations admitting non-trivial comodels.
- A uniform quantitative framework for algebraic effects based on global state modeled by infinite stacks.

The remainder of the thesis is organized as follows.

- Part II (Chapters 2 to 5) develops the background and technical foundations. Chapter 2 recalls the syntax and operational semantics of the λ-calculus, with particular attention to call-by-name and call-by-value evaluation strategies. Chapter 3 introduces the $\lambda!$-calculus, a refinement inspired by linear logic that subsumes call-by-name and call-by-value within a single framework. Chapter 4 presents non-idempotent intersection types and their quantitative interpretation, culminating in the notion of tight typings, which yield exact cost semantics. Chapter 5 reviews the categorical background on monads and algebraic operations, and discusses relevant examples motivating the effectful languages studied in the remainder of the thesis.
- Part III (Chapter 6) is devoted to pattern-matching computations. Chapter 6 introduces a pattern-matching calculus extending the λ-calculus and presents its operational semantics. It shows how call-by-name and call-by-value evaluation, as well as exception raising and handling, can be encoded using pattern matching. A non-idempotent intersection type system is then developed, and quantitative soundness and completeness are established, providing exact bounds on evaluation lengths.
- Part IV (Chapters 7 and 8) addresses state-passing computations and algebraic effects. Chapter 7 studies extensions of the λ- and $\lambda!$-calculi with global state, defines comodel-based operational semantics, and develops non-idempotent intersection type systems for call-by-name, call-by-value, and their unifying presentation, together with full quantitative soundness and completeness results. Chapter 8 generalizes this approach by considering infinite stacks as a model of state, showing how a broad class of algebraic theories admitting non-trivial comodels can be captured. It presents the corresponding calculi and type systems, and

illustrates how infinite stacks can be used to handle effects such as global state, interactive input/output, and finite nondeterminism.

- Part VI contains full proofs of all lemmas, propositions, and theorems. For readability, all proofs are deferred to the appendices. The main body of the thesis focuses on the design of the calculi, their operational semantics, and the resulting quantitative type systems. While some proofs require technical care, the majority proceed by routine structural inductions on terms or typing derivations. Since these proofs are not conceptually central to the contributions of the thesis, presenting them inline would significantly interrupt the flow of the exposition.

Part II

Prologue

Chapter 2

The λ-Calculus

The λ-calculus was first introduced by Alonzo Church in the early 1930s as part of the foundational study of mathematics and logic [57]. In the 1960s, Peter Landin demonstrated that the λ-calculus could also be regarded as a programming language [58, 59]—a minimal yet expressive formalism particularly suitable for theoretical analysis. Since then, the λ-calculus has served as a core framework for exploring the semantics of programming languages.

Because the related literature is vast, this section focuses only on the definitions and results relevant to the quantitative perspective developed in this thesis. We begin by presenting the syntax of the λ-calculus, along with some fundamental properties, following the standard literature [60]. All proofs for this chapter are collected in Appendix A.

2.1 Syntax and Operational Semantics

Definition 2.1 (λ-Terms)**.** Let $\mathcal{X}$ be a countable *set of variables*. The set of *terms* of the λ-calculus is denoted by Λ and generated by the following grammar:

$$\text{Terms:} \quad t, u, r, e ::= x \in \mathcal{X} \mid \lambda x.t \mid t\, u$$

We call expressions of the form $\lambda x.t$ *abstraction* and those of the form $t\, u$ *application*. Moreover, in $t\, u$ we call t the *function* or *operator* of the application, and u the *argument* or *operand*. As customary, we treat application as left-associative, abstractions as right-associative, and group abstractions together. For example, multiple abstractions are right-associated: $\lambda xyz.x\, y\, z = (\lambda x.(\lambda y.(\lambda z.((x\, y)\, z))))$.

Definition 2.2 (Free and Bound Variables)**.** An occurrence of a variable $x \in \mathcal{X}$ is *free* in a λ-term t if it is not in the scope of a λx, and it occurs *bound* otherwise.

The set $\mathtt{fv}(t)$ of *free variables in t* can be defined inductively as follows:

$$\begin{aligned} \mathtt{fv}(x) &= \{x\} \\ \mathtt{fv}(\lambda x.t) &= \mathtt{fv}(t) \setminus \{x\} \\ \mathtt{fv}(t\,u) &= \mathtt{fv}(t) \cup \mathtt{fv}(u) \end{aligned}$$

The set $\mathtt{bv}(t)$ of *bound variables in t* can be defined inductively as follows:

$$\begin{aligned} \mathtt{bv}(x) &= \emptyset \\ \mathtt{bv}(\lambda x.t) &= \{x\} \cup \mathtt{bv}(t) \\ \mathtt{bv}(t\,u) &= \mathtt{bv}(t) \cup \mathtt{bv}(u) \end{aligned}$$

Definition 2.3 (Closed and Open λ-Terms)**.** A term $t \in \Lambda$ is said to be *closed* if and only if $\mathtt{fv}(t) = \emptyset$; otherwise t is *open*. The set of *closed terms* of the λ-calculus is denoted by $\Lambda_\circ$. For example, $\lambda x.x\,y$ is open, and $\lambda xy.x\,y$ is closed.

Definition 2.4 (Substitution)**.** The result of *substituting u for the free occurrences of x in t* is denoted by $t\{x \setminus u\}$ and defined as follows:

$$\begin{aligned} x\{x \setminus u\} &= u \\ y\{x \setminus u\} &= y && \text{if } x \neq y \\ (\lambda y.r)\{x \setminus u\} &= \lambda y.r\{x \setminus u\} && \text{if } x \neq y \text{ and } y \notin \mathtt{fv}(u) \\ (r\,e)\{x \setminus u\} &= r\{x \setminus u\}\,e\{x \setminus u\} \end{aligned}$$

If the side condition $y \notin \mathtt{fv}(u)$ is not satisfied, substitution is undefined.

Definition 2.5 (α-Conversion)**.** Two terms $t, u \in \Lambda$ are *α-equivalent*, denoted $t \sim_\alpha u$, if and only if $t = u$ or $t = \lambda x.r$ and $u = \lambda y.r\{x \setminus y\}$, where y does not occur free in u. Then, the *α-equivalence relation* $\equiv_\alpha$ is the smallest *congruence* generated by $\sim_\alpha$. Formally,

$$\frac{t \sim_\alpha u}{t \equiv_\alpha u} \qquad \frac{t \equiv_\alpha u}{\lambda x.t \equiv_\alpha \lambda x.u} \qquad \frac{t \equiv_\alpha r \qquad u \equiv_\alpha e}{t\,u \equiv_\alpha r\,e}$$

For example,

$$\lambda x.x\,y \sim_\alpha \lambda z.z\,y \not\sim_\alpha \lambda y.y\,y$$
$$\lambda x.x\,(\lambda x.x) \sim_\alpha \lambda y.y\,(\lambda x.x) \equiv_\alpha \lambda y.y\,(\lambda z.z).$$

As usual, we will identify α-congruent terms on a syntactic level by following the following two conventions:

1. Terms that are α-equivalent are identified.

2. If terms $t_1, \ldots, t_n$ occur in a certain mathematical context, their bound variables are distinct from all free variables appearing in the surrounding context.

These conventions capture the intuition that the particular choice of a bound variable, in an abstraction, does not (usually) matter. For instance, $\lambda x.x$ and $\lambda y.y$ are α-equivalent λ-terms. Moreover, it also allows one to avoid name collisions. In particular, notice that substitution is a partial function on terms, but a total function on α-equivalent classes of terms.

Given a (one-step) reduction relation $\to_R$, we denote its *reflexive* closure by $\to_R^=$, and its *reflexive* and *transitive closure* by $\twoheadrightarrow_R$. A term t is said to be in *R-normal form* or *R-irreducible* (written $t \not\to_R$) if and only if there is no u such that $t \to_R u$, and *R-normalizing* if and only if there is some R-normal form u such that $t \twoheadrightarrow_R u$. The reduction relation $\to_R$ is normalizing if and only if every term is R-normalizing. In general, reduction relations are non-deterministic, since several redexes may be present simultaneously. However, most programs are evaluated according to a particular strategy, *i.e.* a *deterministic* subset of a reduction relation specified by restricting the contexts in which reduction steps may occur.

Definition 2.6 (Contexts)**.** The set of *contexts* of the λ-calculus is generated by the following grammar:

$$\text{Contexts:} \quad \mathtt{C} ::= \Box \mid \lambda x.\mathtt{C} \mid \mathtt{C}\, t \mid t\, \mathtt{C}$$

The hole $\Box$ in a context acts as a placeholder for a term, and write $\mathtt{C}[t]$ for the λ-term obtained by *plugging* t into the unique hole of $\mathtt{C}$, possibly capturing free variables of t.

Example 2.7 (Plugging Holes)**.** *Let* $\mathtt{C} = \lambda x.\Box\, \lambda y.y$ *and* $t \in \Lambda$. *The plugging of* t *into* $\mathtt{C}$ *is* $\mathtt{C}[t] = \lambda x.t\, \lambda y.y$.

Let R be the closure by some evaluation context $\mathtt{C}$ of the union of some finite set of reduction rules $S = \{s_1, \ldots, s_n\}$, and t and u be two terms. If $t = \mathtt{C}[t_0] \to_R \mathtt{C}[u_0] = u$, such that $t_0 \mapsto_{s_i} u_0$, we may write $t \to_{R(s_i)} u$ to highlight the underlying reduction step, or simply $t \to_{s_i} u$ whenever R is clear from the context. If $t \twoheadrightarrow_R u$ in n_i s_i-steps, we may write $t \overset{+_{s_i \in S} n_i \cdot s_i}{\twoheadrightarrow_R} u$, or simply $t \overset{n}{\twoheadrightarrow_R} u$, whenever there is only one reduction rule.

Definition 2.8 (β-Reduction)**.** The β-reduction is denoted by $\to_\beta$ and defined as the closure by *all* contexts $\mathtt{C}$ of the following rewriting step:

$$(\lambda x.t)\, u \quad \mapsto_\beta \quad t\{x \setminus u\}$$

We call $(\lambda x.t)\, u$ a *β-redex* and $t\{x \setminus u\}$ a *β-reduct*, but may omit the $\beta-$ prefix whenever it is clear from context.

A keystone property of the λ-calculus is that its reduction relation (β-reduction) is *confluent*, *i.e.* if there are reduction sequences from any term t to two different terms u and r, then there

are reduction sequences from u and r to some common term e. As a consequence, if a term has a β-normal form, then it is unique.

Having established the core syntactic and reduction properties of the λ-calculus, we next refine its operational semantics to account for different evaluation orders.

Definition 2.9 (Syntactic β-Normal Forms)**.** A term $t \in \Lambda$ is said to be in β-normal form if there is no other $u \in \Lambda$, such that $t \to_\beta u$; in which case we write $t \not\to_\beta$. This notion can be characterized by means of the following grammar:

$$\begin{array}{lrcl} \text{Normal Forms:} & \mathtt{NO} & ::= & \lambda x.\mathtt{NO} \mid \mathtt{NE} \\ \text{Neutral Forms:} & \mathtt{NE} & ::= & x \in \mathcal{X} \mid \mathtt{NE}\,\mathtt{NO} \end{array}$$

Proposition 2.10 (Characterization of β-Normal Forms)**.** *Let $t \in \Lambda$. Then, $t \not\to_\beta$ if and only if $t \in \mathtt{NO}$.*

2.2 Call-By-Name and Call-By-Value

As presented thus far, the λ-calculus (together with β-reduction) can be understood as a linguistic abstraction for studying functions. Indeed, in 1937, Alan Turing showed that the class of functions expressible in the λ-calculus is exactly that of computable functions [61]. However, in order to make the connection between the λ-calculus and (functional) programming languages more precise, three further refinements are usually considered: picking a *reduction strategy*; restricting reduction to *closed terms*; and *restricting reduction to occur only outside abstractions*. The first refinement reflects the fact that, since a term may contain several redexes, reducing these redexes in different orders may yield different results∗. The second refinement means that *programs* are closed λ-terms *i.e.* λ-terms that do not contain any free variables. Finally, the third refinement is due to the fact that most functional programming languages do not evaluate functions until they are called. There are many such reduction strategies in the literature [60]. In this thesis, we are interested in the two particular ones introduced in [62]: *call-by-name* (CBN) and *call-by-value* (CBV). From now on, we work only with closed terms, unless stated otherwise.

Call-By-Name. Contexts are such that reduction steps can only occur outside of abstractions and arguments, and reduction steps are only allowed to occur at operators.

Definition 2.11 (CBN Contexts)**.** Let $t \in \Lambda_\circ$. The set of *CBN contexts* of the λ-calculus is generated by the following grammar:

$$\text{CBN Contexts:} \quad \mathtt{N} ::= \Box \mid \mathtt{N}\,t$$

∗Although this is not the case for the *pure* λ-calculus because β-reduction is *confluent*. However, once the syntax of the λ-calculus is extended with (certain) effectful operations, confluence will no longer hold.

Definition 2.12 (CBN Reduction)**.** The *CBN reduction strategy* is denoted by $\to_{\mathtt{CBN}}$ and defined over closed terms as the closure by CBN contexts of rewriting step $\mapsto_\beta$.

Example 2.13 (CBN Reduction)**.** *The following reduction sequence demonstrates that CBN reduction always requires operators to be evaluated first and arguments are never evaluated:*

$$((\lambda x.x)\,\lambda y.y)\,((\lambda x.x)\,\lambda y.y) \to_{\mathtt{CBN}} (\lambda y.y)\,((\lambda x.x)\,\lambda y.y) \to_{\mathtt{CBN}} (\lambda x.x)\,\lambda y.y \to_{\mathtt{CBN}} \lambda y.y.$$

Let $\Omega = (\lambda x.x\,x)\,\lambda x.x\,x$*. Clearly,* $\Omega \to_{\mathtt{CBN}} \Omega$ *diverges. However, since CBN reduction does not evaluate arguments, the following term has a finite reduction sequence:*

$$(\lambda x.\lambda y.y)\,\Omega \to_{\mathtt{CBN}} \lambda y.y.$$

Call-By-Value. Contexts are such that reduction can only be performed outside of abstractions, and rewriting step $\mapsto_\beta$ is restricted so that it can only be performed if arguments are *values*, *i.e.* variables or abstractions.

The set of values of the λ-calculus is denoted by $\mathrm{V} \subseteq \Lambda$ and generated by the following grammar:

$$\text{Values:}\quad v, w \quad ::= \quad x \in \mathcal{X} \mid \lambda x.t$$

Note that $\mathrm{V}_\circ = \Lambda_\circ \cap \mathrm{V}$ is the set of closed values.

Definition 2.14 (CBV Contexts)**.** Let $t \in \Lambda_\circ$ and $v \in \mathrm{V}_\circ$. The set of *CBV contexts* of the λ-calculus is generated by the following grammar:

$$\text{CBV Contexts}\quad \mathtt{V} \quad ::= \quad \Box \mid \mathtt{V}\,t \mid v\,\mathtt{V}$$

Definition 2.15 (CBV Reduction)**.** The *CBV reduction strategy* is denoted by $\to_{\mathtt{CBV}}$ and defined over closed terms (and values) as the closure by CBV contexts of the following rewriting step:

$$(\lambda x.t)\,v \mapsto_{\beta_{\mathrm{v}}} t\{x \setminus v\}$$

Example 2.16 (CBV Reduction)**.** *The following reduction sequence demonstrates that CBV reduction, besides requiring operators to be evaluated first (as in CBN), also requires arguments to be evaluated before being consumed:*

$$((\lambda x.x)\,\lambda y.y)\,((\lambda x.x)\,\lambda y.y) \to_{\mathtt{CBV}} (\lambda y.y)\,((\lambda x.x)\,\lambda y.y) \to_{\mathtt{CBV}} (\lambda y.y)\,\lambda y.y \to_{\mathtt{CBV}} \lambda y.y.$$

Now, consider $(\lambda x.\lambda y.y)\,\Omega$ *from Example 2.13. Since CBV reduction must evaluate* Ω *first, the whole term diverges:*

$$(\lambda x.\lambda y.y)\,\Omega \to_{\mathtt{CBV}} (\lambda x.\lambda y.y)\,\Omega \to_{\mathtt{CBV}} \cdot$$

The following proposition tells us that the sets of CBN- and CBV-normal forms coincide with the set of closed values.

Proposition 2.17 (Characterization of CBN- and CBV-Normal Forms)**.** *The sets of* CBN*-normal forms* $\mathtt{NO}_{\mathtt{CBN}}$ *and* CBV*-normal forms* $\mathtt{NO}_{\mathtt{CBV}}$ *are both the set of closed values* $\mathrm{V}_\circ$.

Notice that the sets of CBN- and CBV-normal forms coincide with that of closed values because we are working with closed terms. When working with open terms, these sets of CBN- and CBV-normal forms are different (see *e.g.* [44]).

The λ-calculus provides a minimal and elegant framework for studying computation. However, the operational strategies of interest to programming-language semantics—such as CBN and CBV—require additional structure: evaluation order and resource consumption should be made explicit; and it should be possible to distinguish between values and computations.

The next chapter introduces the $\lambda!$-calculus, a formalism that captures both evaluation strategies within a single unified framework.

Chapter 3

The $\lambda!$-Calculus

The λ-calculus can be understood as a simple syntactic and operational framework that can be easily adapted to formalize distinct features of functional programming languages. In particular, as we have seen in the previous section, the standard operational semantics of the λ-calculus can be adapted in order to match the reduction strategies adopted by most mainstream functional programming languages: CBN$*$ and CBV evaluation. As it turns out, these two strategies have quite different properties [62]. Indeed, while the *CBN λ-calculus* has a rich and refined syntactic and semantical theory [60], the *CBV λ-calculus* does not enjoy an equally rich syntactic and semantical theory [62]. However, it is well-known that both the CBN and the CBV λ-calculi can be faithfully translated into Linear Logic (LL) [34]. It was this fact that prompted the introduction of the $\lambda!$-calculus [35] as an intermediate formalism combining the conceptual simplicity of the λ-calculus and the operational expressiveness of LL proof nets. The $\lambda!$-calculus can be viewed as an untyped version of the purely functional implicative fragment of the Call-By-Push-Value (CBPV) calculus introduced by Paul Levy [33, 65]. In this thesis, we are interested in the $\lambda!$-calculus due to two main reasons: first, it is a simple syntactic and operational framework subsuming both the CBN and the CBV λ-calculus; second, due to its roots in LL proof nets, it can be endowed with a very simple denotational semantics based on LL that can be presented as a non-idempotent intersection type system (see Chapter 4). We now continue by presenting the syntax of the $\lambda!$-calculus, along with some fundamental properties, mostly following [35]. All proofs for this chapter are collected in Appendix B.

$*$To be precise, most functional programming languages implementing CBN actually implement an optimized version of CBN called Call-By-Need (CBNeed). Describing CBNeed is outside the scope of this thesis, so we recommend the interested reader to take a look at [63, 64] for details.

3.1 Syntax and Operational Semantics

Definition 3.1 ($\lambda!$-Terms)**.** Let $\mathcal{X}$ be a countable *set of variables*. The set of *values* and *terms* of the $\lambda!$-calculus are denoted by $!\mathrm{V}$ and $!\Lambda$, respectively, and generated by the following grammars:

$$\begin{array}{lrcl} \text{Values:} & v, w & ::= & x \in \mathcal{X} \mid !t \\ \text{Terms:} & t, u, r, e & ::= & v \mid \lambda x.t \mid t\, u \mid \mathtt{der}(t) \end{array}$$

We call expressions of the form $!t$ *bang-terms* and those of the form $\mathtt{der}(t)$ *derelictions*. In the $\lambda!$-calculus, bang and dereliction play the same role as *thunk* and *force* in the CBPV calculus [65], *i.e.* bangs freeze computations, and derelictions allow frozen computations to continue.

The notions of free and bound variables, α-congruence, substitution, and contexts, are the obvious extensions of those presented in Chapter 2 to bang-terms and derelictions.

Definition 3.2 (Closed $\lambda!$-Terms)**.** A term t is said to be *closed* if and only if $\mathtt{fv}(t) = \emptyset$. The set of *closed terms* is denoted by $!\Lambda_\circ$, and the set of *closed values*, denoted by $!\mathrm{V}_\circ$, is defined as $!\mathrm{V}_\circ := !\mathrm{V} \cap !\Lambda_\circ$.

Notice that the $\lambda!$-calculus can only encode CBN and CBV if reduction is not allowed to occur inside the bodies of abstractions or bang-terms.

Definition 3.3 (CBPV Contexts)**.** Let $t, \lambda x.u \in !\Lambda_\circ$. The set of *CBPV contexts* of the $\lambda!$-calculus is generated by the following grammar:

$$\text{CBPV Contexts:} \quad \mathtt{B} \quad ::= \quad \Box \mid \mathtt{B}\, t \mid (\lambda x.u)\, \mathtt{B} \mid \mathtt{der}(\mathtt{B})$$

Definition 3.4 (CBPV Reduction)**.** The *CBPV reduction strategy* is denoted by $\to_{\mathtt{CBPV}}$ and defined over closed terms (and values) as the closure by CBPV contexts of the following rewriting steps:

$$\begin{array}{rcl} (\lambda x.t)\, v & \mapsto_{\beta_{\mathrm{v}}} & t\{x \setminus v\} \\ \mathtt{der}(!t) & \mapsto_{\beta_!} & t \end{array}$$

Example 3.5 (CBPV Reduction)**.** *The following reduction sequence demonstrates that CBPV reduction not only requires operators to be evaluated first (as in both CBN and CBV), but also that arguments are evaluated before being consumed (as in CBV):*

$$\begin{array}{rcl} ((\lambda x.\mathtt{der}(x))\, !\lambda y.y)\, ((\lambda x.x)\, !\lambda y.y) & \to_{\mathtt{CBPV}} & \mathtt{der}(!\lambda y.y)\, ((\lambda x.x)\, !\lambda y.y) \\ & \to_{\mathtt{CBPV}} & (\lambda y.y)\, ((\lambda x.x)\, !\lambda y.y) \\ & \to_{\mathtt{CBPV}} & (\lambda y.y)\, !\lambda y.y \\ & \to_{\mathtt{CBPV}} & !\lambda y.y. \end{array}$$

Definition 3.6 (Syntactic CBPV-Normal Forms)**.** Let $t, \lambda x.u \in !\Lambda_\circ$. The set of CBPV-normal forms can be characterized by the following grammar:

$$\begin{array}{lrcl}
\text{Normal Forms:} & \mathtt{NO}_{\mathtt{CBPV}} & ::= & \mathtt{NA}_{\mathtt{CBPV}} \mid \mathtt{NB}_{\mathtt{CBPV}} \\
\text{Bang or Neutral Forms:} & \mathtt{NA}_{\mathtt{CBPV}} & ::= & !t \mid \mathtt{NE}_{\mathtt{CBPV}} \\
\text{Abstraction or Neutral Forms:} & \mathtt{NB}_{\mathtt{CBPV}} & ::= & \lambda x.u \mid \mathtt{NE}_{\mathtt{CBPV}} \\
\text{Neutral Forms:} & \mathtt{NE}_{\mathtt{CBPV}} & ::= & (\lambda x.u)\,\mathtt{NB}_{\mathtt{CBPV}} \mid \mathtt{NA}_{\mathtt{CBPV}}\,t \mid \mathtt{der}(\mathtt{NB}_{\mathtt{CBPV}})
\end{array}$$

The following proposition tells us that the set of CBPV-normal forms is indeed characterized by the grammar presented in Definition 3.6.

Proposition 3.7 (Characterization of CBPV-Normal Forms)**.** *Let $t \in !\Lambda_\circ$. Then, $t \not\to_{\mathtt{CBPV}}$ if and only if $t \in \mathtt{NO}_{\mathtt{CBPV}}$.*

As an untyped version of the CBPV calculus, it is natural for the λ!-calculus to also clearly distinguish between values and abstractions. Here, the distinction is based on the fact that *values* are the only terms that can be duplicated, discarded and accessed through dereliction, and *abstractions* are the only terms representing functions. These two sets of terms are disjoint and closed under CBPV-reduction. However, there are some terms that "violate" this distinction: they apply a value (which is not a function) to something else, or try to access an abstraction (which is not a value) through a dereliction, or apply a function to an abstraction (which is not a value). We call such terms *clashes*, and define them as follows.

Definition 3.8 (Clashes)**.** A *clash* is a term having one of the forms:

$$v\,t \qquad \mathtt{der}(\lambda x.t) \qquad t\,(\lambda x.u)$$

Clearly, clashes are operationally ill-formed terms, *i.e.* they are neither redexes nor represent the result of a correct computation. For this reason, we are going to focus only on terms that do not lead to a clash.

Definition 3.9 (Clash-Free Terms)**.** A term is *clash-free* if it does not reduce to a term containing a clash at *top level*, *i.e.* outside the scope of any lambda λ or bang ! constructor. In other words, a term $t \in !\Lambda$ is *not* clash-free if and only if there exist a CBPV context $\mathtt{B}$ and a clash u, such that $t \twoheadrightarrow_{\mathtt{CBPV}} \mathtt{B}[u]$. For example, $(\lambda x.!x\,x)\,!\lambda y.y$ is *not* clash-free, because $(!\lambda x.x)\,\lambda x.x$ is a clash, and $(\lambda x.!x\,x)\,!\lambda y.y \to_{\mathtt{CBPV}} (!\lambda x.x)\,\lambda x.x$.

Definition 3.10 (Clash-Free CBPV-Normal Forms)**.** The set of *clash-free* CBPV-normal forms of the λ!-calculus is $\mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}} := !\mathrm{V}_\circ \cup \{\lambda x.t \mid \lambda x.t \in !\Lambda_\circ\}$.

Proposition 3.11 (Characterization of Clash-Free CBPV-Normal Forms)**.** *Let $t \in !\Lambda_\circ$. Then, $t \not\to_{\mathtt{CBPV}}$ and t is clash-free if and only if $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$.*

Notice that the definition of clash is a purely syntactical one, but that of *clash-free* terms is not. However, as we will show later, the latter can be characterized via a non-idempotent intersection type system (see Section 4.2).

3.2 Subsuming Call-By-Name and Call-By-Value

As mentioned in the introduction of this section, one of the reasons we are interested in the $\lambda!$-calculus is due to its ability to subsume both the CBN and the CBV λ-calculus. The three most relevant CBN and CBV translations in the literature are those in [35, 45, 66]. Here, we will adopt those in [35], since they are the ones that most closely follow Levy's original encodings in [65].

Definition 3.12 (Translations of Terms)**.** Consider the following translations of terms of the λ-calculus to terms of the $\lambda!$-calculus:

$$\begin{array}{rcl} (\cdot)^{\mathtt{CBN}} : \Lambda & \to & !\Lambda \\ x & \mapsto & \mathtt{der}(x) \\ \lambda x.t & \mapsto & \lambda x.(t)^{\mathtt{CBN}} \\ t\,u & \mapsto & (t)^{\mathtt{CBN}}\,!(u)^{\mathtt{CBN}} \end{array} \qquad \begin{array}{rcl} (\cdot)^{\mathtt{CBV}} : \Lambda & \to & !\Lambda \\ x & \mapsto & x \\ \lambda x.t & \mapsto & !\lambda x.(t)^{\mathtt{CBV}} \\ t\,u & \mapsto & \mathtt{der}((t)^{\mathtt{CBV}})\,(u)^{\mathtt{CBV}} \end{array}$$

Remark 3.13. As it turns out, due to CBPV-reduction not occurring under abstractions or bangs, the CBN translation can only preserve reduction up to any $\beta_!$-redexes that result from the translation. For example,

$$(\lambda x.\lambda y.x)\,t \to_{\mathtt{CBN}(\beta)} \lambda y.t,$$

but

$$\begin{array}{rcl} ((\lambda x.\lambda y.x)\,t)^{\mathtt{CBN}} & = & \mathtt{der}(!\lambda x.!\lambda y.\mathtt{der}(x))\,!(t)^{\mathtt{CBN}} \\ & \to_{\mathtt{CBPV}(\beta_!)} & (\lambda x.!\lambda y.\mathtt{der}(x))\,!(t)^{\mathtt{CBN}} \\ & \to_{\mathtt{CBPV}(\beta_{\mathtt{v}})} & !\lambda y.\mathtt{der}(!(t)^{\mathtt{CBN}}) \\ & \neq & !\lambda y.(t)^{\mathtt{CBN}} \\ & = & (\lambda y.t)^{\mathtt{CBN}}. \end{array}$$

This well-known phenomenon that has been identified both in [32, 65] and [35]. In [32, 65], this is dealt with by considering that some of the technical results for the CBN translation concern not the *function* $(\cdot)^{\mathtt{CBN}}$ (which does not commute with substitution, in general), but a *relation* $\overset{\mathtt{CBN}}{\leq}$ from CBN to CBPV terms. Informally, $t \overset{\mathtt{CBN}}{\leq} u$ means that u is $(t)^{\mathtt{CBN}}$ with possibly some extra $\mathtt{der}(!(\cdot))$-redexes. In [35], this is dealt with by allowing *all* $\beta_!$-redexes to be removed, *i.e.* by

allowing $\beta_!$-steps to be fired under any context. Here, we will allow $\beta_!$-steps to be fired under any context by introducing the following administrative reduction relation.

Definition 3.14 (Administrative Contexts)**.** Let $t \in !\Lambda$. The set of *administrative contexts* of the $\lambda!$-calculus is generated by the following grammar:

$$\mathtt{A} \quad ::= \quad \lambda x.\Box \mid !\Box \mid \mathtt{A}\, t \mid t\, \mathtt{A} \mid \lambda x.\mathtt{A} \mid !\mathtt{A} \mid \mathtt{der}(\mathtt{A})$$

Notice that administrative contexts only allow holes to occur under abstractions and bang-terms, while CBPV contexts (see Definition 3.3) precisely do not, *i.e.* administrative and CBPV contexts do not overlap. As such, the administrative reduction relation in Definition 3.15 can be shown not to interfere with the CBPV reduction relation (see Lemma 3.16). Therefore, even when the administrative and CBPV reduction relations are put together, it is still possible to distinguish between $\beta_!$-redexes fired inside administrative contexts from those fired outside those contexts.

Definition 3.15 (Administrative Reduction Relation)**.** The *administrative reduction relation* is denoted by $\to_{\mathtt{ADMIN}}$ and defined as the closure of $\beta_!$-steps by *administrative contexts*.

Remark that the well-formedness of terms is preserved by the administrative reduction relation and that administrative steps never occur at top level.

At this point, what we would like to do is to extend CBPV-reduction with administrative steps, as discussed in the paragraph after Definition 3.12. This can be done by simply defining a new reduction relation $\to_{\mathtt{CBPV}\cup\mathtt{ADMIN}}$ that is the union of $\to_{\mathtt{CBPV}}$ and $\to_{\mathtt{ADMIN}}$. However, this means that administrative steps can be freely intertwined with CBPV-steps, thus we must ensure that administrative steps do not interfere with CBPV-reduction in any significant way, with respect to the CBN translation of terms. Indeed, given some term t, we must guarantee that, if no β_{v}-step can be performed in $(t)^{\mathtt{CBN}}$, then it is also the case after applying any number of administrative steps to $(t)^{\mathtt{CBN}}$, *i.e.* the CBN translation preserves normal forms, up-to the presence of $\beta_!$-redexes. Crucially, this then means that the number of β-steps needed to CBN-normalize a term is preserved by the CBN translation. We formalize this fact, *i.e.* that administrative steps do not create β_{v}-steps, as follows.

Lemma 3.16 (Non-Interference of Single Administrative Steps)**.** *Let $t \in \Lambda_\circ$. If $(t)^{\mathtt{CBN}} \not\to_{\mathtt{CBPV}(\beta_{\mathsf{v}})}$ and $(t)^{\mathtt{CBN}} \to_{\mathtt{ADMIN}} u$, then $u \not\to_{\mathtt{CBPV}(\beta_{\mathsf{v}})}$.*

Corollary 3.17 (Non-Interference of Administrative Steps)**.** *Let $t \in \Lambda_\circ$. If $(t)^{\mathtt{CBN}} \not\to_{\mathtt{CBPV}(\beta_{\mathsf{v}})}$ and $(t)^{\mathtt{CBN}} \to\!\!\!\!\to_{\mathtt{ADMIN}} u$, then $u \not\to_{\mathtt{CBPV}(\beta_{\mathsf{v}})}$.*

Thus, we can define the following reduction relation safely.

Definition 3.18 (Administrative CBPV Reduction Relations)**.** Consider the CBPV reduction relation $\to_{\mathtt{CBPV}}$ and the administrative reduction relation $\to_{\mathtt{ADMIN}}$. Then, $\to_{\mathtt{CBPV}\cup\mathtt{ADMIN}}$ is defined as the union $\to_{\mathtt{CBPV}} \cup \to_{\mathtt{ADMIN}}$, and $\twoheadrightarrow_{\mathtt{CBPV}\cup\mathtt{ADMIN}}$ is its reflexive and transitive closure.

Theorem 3.19 (Correctness of Full Simulations)**.** *Let $t, u \in \Lambda_\circ$.*

1. *If $t \to^n_{\mathtt{CBN}} u$, then $(t)^{\mathtt{CBN}} \twoheadrightarrow^{n\cdot\beta_{\mathtt{v}}+m\cdot\beta_!}_{\mathtt{CBPV}\cup\mathtt{ADMIN}} (u)^{\mathtt{CBN}}$, for some $m \geq 0$*
2. *If $t \to^n_{\mathtt{CBV}} u$, then $(t)^{\mathtt{CBV}} \twoheadrightarrow^{n\cdot\beta_{\mathtt{v}}+m\cdot\beta_!}_{\mathtt{CBPV}} (u)^{\mathtt{CBV}}$, for some $m \geq 0$.*

Going back to the example in Remark 3.13, we now have:

$$\begin{array}{rcl}
((\lambda x.\lambda y.x)\, t)^{\mathtt{CBN}} & = & \mathtt{der}(!\lambda x.!\lambda y.\mathtt{der}(x))\, !(t)^{\mathtt{CBN}} \\
& \to_{\mathtt{CBPV}(\beta_!)} & (\lambda x.!\lambda y.\mathtt{der}(x))\, !(t)^{\mathtt{CBN}} \\
& \to_{\mathtt{CBPV}(\beta_{\mathtt{v}})} & !\lambda y.\mathtt{der}(!(t)^{\mathtt{CBN}}) \\
& \to_{\mathtt{ADMIN}} & !\lambda y.(t)^{\mathtt{CBN}} \\
& = & (\lambda y.t)^{\mathtt{CBN}}.
\end{array}$$

Chapter 4

Non-Idempotent Intersection Types

Intersection Types. Intersection types were first introduced by Coppo and Dezani, in the late 1970s, to increase the typability power of simple type assignment systems [67, 68]. Intersection types extend simple types with a new type constructor $\cap$, which allows for *finite polymorphism*. Indeed, a term t is typable with $A \cap B$ if and only if it is typable with both A and B, independently. For example, if a variable x has type $(A \to A) \cap A$, then $x\,x$ has type A, and the β-strongly normalizing* term $\lambda x.x\,x$, which is untypable in the simple system, becomes typable as $(A \to A) \cap A \to A$. Among other things, intersection types can be used to characterize properties of λ-terms, such as *solvability* and *normalization*, as well as to describe models of the λ-calculus [69, 70].

Intersection types are, at the same time, *syntactic* and *semantic* objects: they provide information about the various roles played by a term at execution time. For comprehensive surveys, see [71–73]; here we focus on the non-idempotent variant relevant to quantitative semantics.

Non-Idempotent Intersection Types. Intersection types, as they were first introduced in the 1970s, enjoy the three basic algebraic properties of intersections: *associativity* $(A \cap B) \cap C = A \cap (B \cap C)$, *commutativity* $(A \cap B = B \cap A)$, and in particular idempotency $(A \cap A = A)$. However, as it turns out, there is a tight correspondence between the multiplicative connective of LL [34] and *non-idempotent* intersection types. Indeed, non-idempotent intersection types and *Multiplicative Linear Logic* (MLL) are very similar in how they track the use of resources: in the non-idempotent setting, a term t is typable with the type $A_1 \cap \cdots \cap A_n \to B$ if and only if t uses its argument n times. It was Gardner who first explored this connection in [1], by introducing the first type system based on non-idempotent types in order to uncover needed reduction. Almost contemporaneously, Kfoury [2] developed a non-idempotent intersection type discipline explicitly aimed at tracking resource consumption, making the quantitative interpretation of type multiplicities a central semantic feature. However, the connection between non-idempotent intersection

*A term t is β-strongly normalizing if there is no infinite β-reduction starting with t [60].

types and MLL can be made precise through the connections between intersection types and categorical semantics inspired by LL. The literature is vast (see [74, 75] for pointers), but a relatively simple yet informative categorical model for the λ-calculus is the *Relational Model* (MRel). This model is able to provide good *quantitative semantics* to the λ-calculus, in the sense of [76], and can be presented as a non-idempotent type system [35, 66]: the relational semantics of a term t turns out to be just the set of conclusions of the type derivations of t. Indeed, it is now well understood that the semantics induced by MRel corresponds to the non-idempotent type system of de Carvalho [3, 4], where such a type system was used to obtain exact bounds on the number of β-reduction steps and the size of β-normal forms for λ-terms. Remark that associative, commutative and idempotent intersections can be denoted by sets. Naturally, this means that *non-idempotent* intersections can be denoted by *multisets*. We follow this convention throughout this thesis.

We start by formally introducing non-idempotent intersection types for the CBN, CBV and CBPV variants of the λ-calculus presented in the previous chapters. All proofs for this chapter are collected in Appendix C.

Definition 4.1 (CBN, CBV, and CBPV Types)**.** Let $\mathtt{I}$ be a finite set. The sets of types for the CBN, CBV, and CBPV λ-calculi are denoted by $\mathcal{T}y_{\mathtt{CBN}}$, $\mathcal{T}y_{\mathtt{CBV}}$, and $\mathcal{T}y_{\mathtt{CBPV}}$, respectively, and generated by the following grammars:

	CBN	CBV	CBPV
Types:	$A ::= \star \mid M \to A$	$A ::= M \to N$	$A ::= \star \mid M \mid M \to A$
Multi-Types:	$M ::= \{\!\{A_i\}\!\}_{i\in\mathtt{I}}$	$M, N ::= \{\!\{A_i\}\!\}_{i\in\mathtt{I}}$	$M ::= \{\!\{A_i\}\!\}_{i\in\mathtt{I}}$,

where $\mathtt{I}$ denotes a countable set of indices.

We call $\star$ an *answer*, and $M \to A$ and $M \to N$ *arrow-types*.

The set of types for CBN and CBV are quite standard, and can be found in *e.g.* [35, 45]. The set of types for CBPV is exactly the same as in [77]. Notice that, while in CBV any value (and thus abstraction) can be typed with the empty multi-type $\{\!\{\,\}\!\}$, in CBN and CBPV this is not the case. Thus, it is necessary to introduce constant $\star$ as the type of any abstraction, in order to be able to type any normal form.

In order to introduce the non-idempotent type systems for the CBN, CBV, and CBPV λ-calculi, we must first introduce some notations and definitions that will be adapted and extended throughout this thesis.

Definition 4.2 (Typing Environments)**.**

1. *Typing environments* are denoted by Γ, Δ and *defined as functions from variables to multi-types* that assign the empty multi-type $\{\!\{\,\}\!\}$ to all but a finite set of variables.

2. The *domain of a typing environment* Γ denoted by $\mathtt{dom}(\Gamma)$ and defined as $\mathtt{dom}(\Gamma) := \{x \mid \Gamma(x) \neq \{\!\{\,\}\!\}\}$.

3. The *union of typing environments* Γ and Δ is denoted by $\Gamma + \Delta$ and defined as $(\Gamma + \Delta)(x) := \Gamma(x) \sqcup \Delta(x)$, where $\sqcup$ denotes multiset union. This notion is extended to a finite union of environments, written $+_{i \in \mathtt{I}} \Gamma_i$ (the empty environment is obtained when $\mathtt{I} = \emptyset$).

We write $\Gamma \setminus\!\setminus x$ for the environment $(\Gamma \setminus\!\setminus x)(x) = \{\!\{\,\}\!\}$ and $(\Gamma \setminus\!\setminus x)(y) = \Gamma(y)$ if $y \neq x$, and we write $\Gamma; x : M$ for $\Gamma + (x : M)$, whenever $x \notin \mathtt{dom}(\Gamma)$. In particular, notice that Γ and $\Gamma; x : \{\!\{\,\}\!\}$ both denote the same environment.

Definition 4.3 (Typing Judgments)**.** *Typing judgments* are denoted by $\Gamma \vdash t : A$ (resp. M) and, whenever Γ is empty, we write $\vdash t : A$ (resp. M) instead of $\emptyset \vdash t : A$ (resp. M).

Definition 4.4 (Typing Rules)**.** *Typing rules* are presented in the usual Gentzen-style layout: the typing judgments on the top of the rule are called *premises*, and the unique typing judgment on the bottom is called the *conclusion*.

Let $\mathtt{T}$ be the name of some type system. We write $\rhd_{\mathtt{T}} \Gamma \vdash t : A$ (resp. M) if there is a type derivation concluding with $\Gamma \vdash t : A$ (resp. M) using the typing rules of $\mathtt{T}$, in which case we say that t is *typable* in $\mathtt{T}$. We use letters $\Phi, \Psi, \ldots$ to name type derivations, by writing *e.g.* $\Phi \rhd_{\mathtt{T}} \Gamma \vdash t : A$ (resp. M).

Definition 4.5 (Size of Type Derivations)**.** Let Φ be a type derivation in some type system $\mathtt{T}$. Moreover, let $R = \{r_1, \ldots, r_n\}$ be a *subset* of the set of all typing rules of $\mathtt{T}$. We define $\#_R(\Phi)$ as a function that counts the number of occurrences of the typing rules R in Φ. We may write the result of $\#_R(\Phi)$ as a vector $+_{r_i \in R} n_i \cdot r_i$, where n_i is the number of occurrences of rule r_i in Φ, or simply n, whenever there is only one typing rule in R.

Example 4.6 (Size of Type Derivations)**.** *Consider the following abstract type derivation* Φ*:*

$$\dfrac{\dfrac{\dfrac{}{\ldots}\quad\dfrac{}{\ldots}}{\ldots}\,r_1 \qquad \dfrac{\dfrac{}{\ldots}\quad\dfrac{}{\ldots}}{\ldots}\,r_1}{\ldots}\,r_2$$

Then, $\#_{r_1}(\Phi) = 2 \cdot r_1$, $\#_{r_2}(\Phi) = 1 \cdot r_2$, *and* $\#_{r_1, r_2}(\Phi) = 2 \cdot r_1 + 1 \cdot r_2$.

4.1 Call-By-Name and Call-By-Value

In this section, we introduce the type system based on non-idempotent intersection types for the CBN and CBV λ-calculi.

Definition 4.7 (CBN Type System)**.** Recall the set of CBN types presented in Definition 4.1. The *CBN Type System* $\mathcal{N}$ assigns CBN (multi-)types to λ-terms and consists of the following typing rules:

$$\frac{}{x : \{\!\{A\}\!\} \vdash x : A}\ \mathtt{Ax} \qquad \frac{\Gamma \vdash t : A}{\Gamma \setminus\!\setminus x \vdash \lambda x.t : \Gamma(x) \to A}\ \mathtt{Abs} \qquad \frac{}{\varnothing \vdash \lambda x.t : \star}\ \mathtt{Ans}$$

$$\frac{\Gamma \vdash t : M \to A \qquad \Delta \vdash u : M}{\Gamma + \Delta \vdash t\, u : A}\ \mathtt{App} \qquad \frac{(\Gamma_i \vdash t : A_i)_{i \in \mathrm{I}}}{+_{i \in \mathrm{I}} \Gamma_i \vdash t : \{\!\{A_i\}\!\}_{i \in \mathrm{I}}}\ \mathtt{Many}$$

Definition 4.8 (CBV Type System)**.** Recall the set of CBV types presented in Definition 4.1. The *CBV Type System* $\mathcal{V}$ assigns CBV (multi-)types to λ-terms and consists of the following typing rules:

$$\frac{}{x : M \vdash x : M}\ \mathtt{Ax} \qquad \frac{(\Gamma_i \vdash t : M_i)_{i \in \mathrm{I}}}{+_{i \in \mathrm{I}} \Gamma_i \setminus\!\setminus x \vdash \lambda x.t : \{\!\{\Gamma_i(x) \to M_i\}\!\}_{i \in \mathrm{I}}}\ \mathtt{Abs}$$

$$\frac{\Gamma \vdash t : \{\!\{M \to N\}\!\} \qquad \Delta \vdash u : M}{\Gamma + \Delta \vdash t\, u : N}\ \mathtt{App}$$

The differences between the CBN and CBV type systems can be understood through Girard's translations [34] of CBN and CBV into LL: in CBN all λ-terms are duplicable, but in CBV only values are duplicable. This can also be seen in Definition 3.12: the $(\cdot)^{\mathtt{CBN}}$ translation puts a bang in front of any argument, but the $(\cdot)^{\mathtt{CBV}}$ translation only puts a bang in front of abstractions (*i.e.* values). Moreover, we say that a term $t \in \Lambda$ is *typable in* $\mathcal{N}$ whenever it is typable with a *type*, and *typable in* $\mathcal{V}$ whenever it is typable with a *multi-type*.

Just like their idempotent precursors, non-idempotent intersection types capture interesting computational properties of the λ-calculus. But they also grant a substantial improvement: they provide quantitative measures about these properties. For example, typability in a type system using non-idempotent intersections does not only characterize a qualitative property such as termination for the λ-calculus, but also provides quantitative information such as upper bounds for the number of evaluation steps needed to reach a normal form. This shift of perspective, from idempotent to non-idempotent intersection types, goes beyond lowering the logical complexity of the proof: the quantitative information provided by typing derivations in the non-idempotent setting unveils crucial quantitative relations between typing (statics) and reduction (dynamics) of programs.

The quantitative aspects of type systems $\mathcal{N}$ and $\mathcal{V}$ are materialized in weighted versions of the usual subject reduction and expansion properties. As usual, substitution and anti-substitution lemmas must be proved, but the following weighted versions of these lemmas are necessary.

Lemma 4.9 (Weighted Substitution for CBN and CBV)**.** *Let* $t, u \in \Lambda$, *and* $v \in \mathrm{V}$.

1. *If* $\Phi \rhd_{\mathcal{N}} \Gamma; x : M \vdash t : A$ *(resp. N) and* $\Psi \rhd_{\mathcal{N}} \varnothing \vdash u : M$, *then* $\Pi \rhd_{\mathcal{N}} \Gamma \vdash t\{x \setminus u\} : A$ *(resp. N), such that* $\#_{\mathtt{App}}(\Pi) = \#_{\mathtt{App}}(\Phi) + \#_{\mathtt{App}}(\Psi)$.

2. *If* $\Phi \rhd_{\mathcal{V}} \Gamma; x : M \vdash t : N$ *and* $\Psi \rhd_{\mathcal{V}} \varnothing \vdash v : M$*, then* $\Pi \rhd_{\mathcal{V}} \Gamma \vdash t\{x \setminus v\} : N$*, such that* $\#_{\mathtt{App}}(\Pi) = \#_{\mathtt{App}}(\Phi) + \#_{\mathtt{App}}(\Psi)$.

Example 4.10 (Weighted Substitution)**.** *Consider the open term* $z\,z$.

Example for CBN:

Suppose we want to substitute *the closed* term $(\lambda x.x)\,\lambda y.y$ *for* z*. Let* $A = \{\!\{\star\}\!\} \to \star$, Φ *be*

$$\dfrac{\dfrac{}{z : \{\!\{A\}\!\} \vdash z : A}\,\mathtt{Ax} \qquad \dfrac{\dfrac{}{z : \{\!\{\star\}\!\} \vdash z : \star}\,\mathtt{Ax}}{z : \{\!\{\star\}\!\} \vdash z : \{\!\{\star\}\!\}}\,\mathtt{Many}}{z : \{\!\{A, \star\}\!\} \vdash z\,z : \star}\,\mathtt{App}$$

where $\#_{\mathtt{App}}(\Phi) = 1$*, and* Ψ *be*

$$\dfrac{\dfrac{\dfrac{}{x : \{\!\{A\}\!\} \vdash x : A}\,\mathtt{Ax}}{\vdash \lambda x.x : \{\!\{A\}\!\} \to A}\,\mathtt{Abs} \qquad \dfrac{\dfrac{\dfrac{}{y : \{\!\{\star\}\!\} \vdash y : \star}\,\mathtt{Ax}}{\vdash \lambda y.y : A}\,\mathtt{Abs}}{\vdash \lambda y.y : \{\!\{A\}\!\}}\,\mathtt{Many}}{\vdash (\lambda x.x)\,\lambda y.y : A}\,\mathtt{App} \qquad \dfrac{\dfrac{\dfrac{}{x : \{\!\{\star\}\!\} \vdash x : \star}\,\mathtt{Ax}}{\vdash \lambda x.x : A}\,\mathtt{Abs} \qquad \dfrac{\dfrac{}{\vdash \lambda y.y : \star}\,\mathtt{Ans}}{\vdash \lambda y.y : \{\!\{\star\}\!\}}\,\mathtt{Many}}{\vdash (\lambda x.x)\,\lambda y.y : \star}\,\mathtt{App}}{\vdash (\lambda x.x)\,\lambda y.y : \{\!\{A, \star\}\!\}}\,\mathtt{Many}$$

where $\#_{\mathtt{App}}(\Psi) = 2$*. Notice that* $(z\,z)\{z \setminus (\lambda x.x)\,\lambda y.y\} = ((\lambda x.x)\,\lambda y.y)\,((\lambda x.x)\,\lambda y.y)$*. By Lemma 4.9, there is a derivation* $\Pi \rhd_{\mathcal{N}} \varnothing \vdash ((\lambda x.x)\,\lambda y.y)\,((\lambda x.x)\,\lambda y.y) : \star$*, such that* $\#_{\mathtt{App}}(\Pi) = \#_{\mathtt{App}}(\Phi) + \#_{\mathtt{App}}(\Psi)$*. Indeed, such a* Π *can be built from* Φ *and* Ψ *as follows:*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : \{\!\{A\}\!\} \vdash x : A}\,\mathtt{Ax}}{\vdash \lambda x.x : \{\!\{A\}\!\} \to A}\,\mathtt{Abs} \qquad \dfrac{\dfrac{\dfrac{}{y : \{\!\{\star\}\!\} \vdash y : \star}\,\mathtt{Ax}}{\vdash \lambda y.y : A}\,\mathtt{Abs}}{\vdash \lambda y.y : \{\!\{A\}\!\}}\,\mathtt{Many}}{\vdash (\lambda x.x)\,\lambda y.y : A}\,\mathtt{App} \qquad \dfrac{\dfrac{\dfrac{}{x : \{\!\{\star\}\!\} \vdash x : \star}\,\mathtt{Ax}}{\vdash \lambda x.x : A}\,\mathtt{Abs} \qquad \dfrac{\dfrac{}{\vdash \lambda y.y : \star}\,\mathtt{Ans}}{\vdash \lambda y.y : \{\!\{\star\}\!\}}\,\mathtt{Many}}{\vdash (\lambda x.x)\,\lambda y.y : \star}\,\mathtt{App}}{\vdash ((\lambda x.x)\,\lambda y.y)\,((\lambda x.x)\,\lambda y.y) : \star}\,\mathtt{App}$$

since $\#_{\mathtt{App}}(\Pi) = 3 = \#_{\mathtt{App}}(\Phi) + \#_{\mathtt{App}}(\Psi)$.

Example for CBV:

Suppose we want to substitute *the closed* value $\lambda x.x$ *for* z*. Let* Φ *be*

$$\dfrac{\dfrac{}{z : \{\!\{\,\}\!\} \to \{\!\{\,\}\!\}\} \vdash z : \{\!\{\,\}\!\} \to \{\!\{\,\}\!\}\}}\,\mathtt{Ax} \qquad \dfrac{}{z : \{\!\{\,\}\!\} \vdash z : \{\!\{\,\}\!\}}\,\mathtt{Ax}}{z : \{\!\{\,\}\!\} \to \{\!\{\,\}\!\}\} \vdash z\,z : \{\!\{\,\}\!\}}\,\mathtt{App}$$

where $\#_{\mathtt{App}}(\Phi) = 1$*, and* Ψ

$$\dfrac{\dfrac{}{x : \{\!\{\,\}\!\} \vdash x : \{\!\{\,\}\!\}}\,\mathtt{Ax}}{\vdash \lambda x.x : \{\!\{\,\}\!\} \to \{\!\{\,\}\!\}\}}\,\mathtt{Abs}$$

where $\#_{\mathtt{App}}(\Psi) = 0$. *Notice that* $(z\,z)\{z \setminus \lambda x.x\} = (\lambda x.x)\,\lambda x.x$. *By Lemma 4.9, there is a derivation* $\Pi \rhd_{\mathcal{V}} \emptyset \vdash (\lambda x.x)\,\lambda x.x : \{\!\{\,\}\!\}$, *such that* $\#_{\mathtt{App}}(\Pi) = \#_{\mathtt{App}}(\Phi) + \#_{\mathtt{App}}(\Psi)$. *Indeed, such a* Π *can be built from* Φ *and* Ψ *as follows:*

$$\dfrac{\dfrac{\dfrac{}{x : \{\!\{\,\}\!\} \vdash x : \{\!\{\,\}\!\}}\,\mathtt{Ax}}{\vdash \lambda x.x : \{\!\{\{\!\{\,\}\!\} \to \{\!\{\,\}\!\}\}\!\}}\,\mathtt{Abs} \qquad \dfrac{}{\vdash \lambda x.x : \{\!\{\,\}\!\}}\,\mathtt{Abs}}{\vdash (\lambda x.x)\,\lambda x.x : \{\!\{\,\}\!\}}\,\mathtt{App}$$

since $\#_{\mathtt{App}}(\Pi) = 1 = \#_{\mathtt{App}}(\Phi) + \#_{\mathtt{App}}(\Psi)$.

Lemma 4.11 (Weighted Anti-Substitution for CBN and CBV)**.** *Let* $t \in \Lambda$, $u \in \Lambda_{\circ}$, *and* $v \in \mathrm{V}_{\circ}$.

1. *If* $\Phi \rhd_{\mathcal{N}} \Gamma \vdash t\{x \setminus u\} : A$ *(resp. M), then* $\Psi \rhd_{\mathcal{N}} \Gamma; x : M \vdash t : A$ *(resp. M) and* $\Pi \rhd_{\mathcal{N}} \emptyset \vdash u : M$, *such that* $\#_{\mathtt{App}}(\Phi) = \#_{\mathtt{App}}(\Psi) + \#_{\mathtt{App}}(\Pi)$.
2. *If* $\Phi \rhd_{\mathcal{V}} \Gamma \vdash t\{x \setminus v\} : N$, *then* $\Psi \rhd_{\mathcal{V}} \Gamma; x : M \vdash t : N$ *and* $\Pi \rhd_{\mathcal{V}} \emptyset \vdash v : M$, *such that* $\#_{\mathtt{App}}(\Phi) = \#_{\mathtt{App}}(\Psi) + \#_{\mathtt{App}}(\Pi)$.

Example 4.12 (Weighted Anti-Substitution Lemma for CBN and CBV)**.**

Example for CBN:

Consider the closed term $(z\,z)\{z \setminus (\lambda x.x)\,\lambda y.y\} = ((\lambda x.x)\,\lambda y.y)\,((\lambda x.x)\,\lambda y.y)$ *from Example 4.10. Suppose we want to* extract $(\lambda x.x)\,\lambda y.y$ *from* $((\lambda x.x)\,\lambda y.y)\,((\lambda x.x)\,\lambda y.y)$. *By Lemma 4.11, there is a way of extracting type derivations for* $z\,z$ *and* $(\lambda x.x)\,\lambda y.y$ *from the type derivation for* $((\lambda x.x)\,\lambda y.y)\,((\lambda x.x)\,\lambda y.y)$. *Indeed, notice that derivations* Φ *and* Ψ *can also be extracted back from* Π *in Example 4.10.*

Example for CBV:

Consider the closed term $(z\,z)\{z \setminus \lambda x.x\} = (\lambda x.x)\,\lambda x.x$ *from Example 4.10. Suppose we want to* extract $\lambda x.x$ *from* $(\lambda x.x)\,\lambda x.x$. *By Lemma 4.11, there is a way of extracting type derivations for* $z\,z$ *and* $\lambda x.x$ *from the type derivation for* $(\lambda x.x)\,\lambda x.x$. *In particular, notice that derivations* Φ *and* Ψ *can also be extracted back from* Π *in Example 4.10.*

The weighted subject reduction and expansion properties follow from the above lemmas.

Lemma 4.13 (Weighted Subject Reduction for CBN and CBV)**.** *Let* $t, u \in \Lambda_{\circ}$.

1. *If* $t \to_{\mathtt{CBN}} u$ *and* $\Phi \rhd_{\mathcal{N}} \emptyset \vdash t : A$, *then there is* $\Psi \rhd_{\mathcal{N}} \emptyset \vdash u : A$, *such that* $\#_{\mathtt{App}}(\Phi) = \#_{\mathtt{App}}(\Psi) + 1$.
2. *If* $t \to_{\mathtt{CBV}} u$ *and* $\Phi \rhd_{\mathcal{V}} \emptyset \vdash t : M$, *then there is* $\Psi \rhd_{\mathcal{V}} \emptyset \vdash u : M$, *such that* $\#_{\mathtt{App}}(\Phi) = \#_{\mathtt{App}}(\Psi) + 1$.

Example 4.14 (Weighted Subject Reduction)**.**

Example for CBN:

Consider the term $(\lambda z.z\,z)\,((\lambda x.x)\,\lambda y.y)$*. According to the CBN reduction strategy:*

$$(\lambda z.z\,z)\,((\lambda x.x)\,\lambda y.y) \to_{\mathtt{CBN}(\beta)} ((\lambda x.x)\,\lambda y.y)\,((\lambda x.x)\,\lambda y.y).$$

Recall derivations Φ *and* Ψ *from Example 4.10. The following type derivation* Φ' *is a type derivation for* $(\lambda z.z\,z)\,((\lambda x.x)\,\lambda y.y)$*:*

$$\dfrac{\dfrac{\Phi \rhd_{\mathcal{N}} z : \{\!\{A, \star\}\!\} \vdash z\,z : \star}{\emptyset \vdash \lambda z.z\,z : \{\!\{A, \star\}\!\} \to \star}\,\mathtt{Abs} \qquad \Psi \rhd_{\mathcal{N}} \emptyset \vdash (\lambda x.x)\,\lambda y.y : \{\!\{A, \star\}\!\}}{\emptyset \vdash (\lambda z.z\,z)\,((\lambda x.x)\,\lambda y.y) : \star}\,\mathtt{App}$$

such that $\#_{\mathtt{App}}(\Phi') = \#_{\mathtt{App}}(\Phi) + \#_{\mathtt{App}}(\Psi) + 1$*. By Lemma 4.13, there should be a type derivation* $\Psi' \rhd_{\mathcal{N}} \emptyset \vdash ((\lambda x.x)\,\lambda y.y)\,(\lambda x.x)\,\lambda y.y : \star$*, such that* $\#_{\mathtt{App}}(\Phi') = \#_{\mathtt{App}}(\Psi') + 1$*. Indeed, notice that* $(\lambda z.z\,z)\,((\lambda x.x)\,\lambda y.y) = (z\,z)\{z \setminus (\lambda x.x)\,\lambda y.y\}$*. So, we can take* $\Psi' = \Pi$ *from Example 4.10, since* $\#_{\mathtt{App}}(\Phi') = \#_{\mathtt{App}}(\Phi) + \#_{\mathtt{App}}(\Psi) + 1 = \#_{\mathtt{App}}(\Pi)$*.*

Example for CBV:

Consider the term $(\lambda z.z\,z)\,\lambda x.x$*. According to the CBV reduction strategy:*

$$(\lambda z.z\,z)\,\lambda x.x \to_{\mathtt{CBV}(\beta_v)} (\lambda x.x)\,\lambda x.x.$$

Recall derivations Φ *and* Ψ *from Example 4.10. The following type derivation* Φ' *is a type derivation for* $(\lambda z.z\,z)\,\lambda x.x$*:*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{z : \{\!\{\,\}\!\} \to \{\!\{\,\}\!\} \to \{\!\{\,\}\!\} \vdash z : \{\!\{\,\}\!\} \to \{\!\{\,\}\!\} \to \{\!\{\,\}\!\}}\,\mathtt{Ax} \quad \dfrac{}{z : \{\!\{\,\}\!\} \vdash z : \{\!\{\,\}\!\}}\,\mathtt{Ax}}{z : \{\!\{\,\}\!\} \to \{\!\{\,\}\!\} \to \{\!\{\,\}\!\} \vdash z\,z : \{\!\{\,\}\!\} \to \{\!\{\,\}\!\}}\,\mathtt{App}}{\emptyset \vdash \lambda z.z\,z : (\{\!\{\,\}\!\} \to \{\!\{\,\}\!\} \to \{\!\{\,\}\!\}) \to \{\!\{\,\}\!\} \to \{\!\{\,\}\!\}}\,\mathtt{Abs} \qquad \dfrac{\dfrac{}{x : \{\!\{\,\}\!\} \vdash x : \{\!\{\,\}\!\}}\,\mathtt{Ax}}{\emptyset \vdash \lambda x.x : \{\!\{\,\}\!\} \to \{\!\{\,\}\!\} \to \{\!\{\,\}\!\}}\,\mathtt{Abs}}{\emptyset \vdash (\lambda z.z\,z)\,\lambda x.x : \{\!\{\,\}\!\} \to \{\!\{\,\}\!\}}\,\mathtt{App}$$

such that $\#_{\mathtt{App}}(\Phi') = \#_{\mathtt{App}}(\Phi) + \#_{\mathtt{App}}(\Psi) + 1$*. By Lemma 4.13, there is* $\Psi' \rhd_{\mathcal{V}} \emptyset \vdash (\lambda x.x)\,\lambda x.x : \{\!\{\,\}\!\} \to \{\!\{\,\}\!\}$*, such that* $\#_{\mathtt{App}}(\Phi') = \#_{\mathtt{App}}(\Psi') + 1$*. Indeed, notice that* $(\lambda z.z\,z)\,\lambda x.x = (z\,z)\{z \setminus \lambda x.x\}$*. So, we can take* $\Psi' = \Pi$ *from Example 4.10, since* $\#_{\mathtt{App}}(\Phi') = \#_{\mathtt{App}}(\Phi) + \#_{\mathtt{App}}(\Psi) + 1 = \#_{\mathtt{App}}(\Pi)$*.*

Lemma 4.15 (Weighted Subject Expansion for CBN and CBV)**.** *Let* $t, u \in \Lambda_\circ$*.*

1. *If* $t \to_{\mathtt{CBN}} u$ *and* $\Phi \rhd_{\mathcal{N}} \emptyset \vdash u : A$*, then there is* $\Psi \rhd_{\mathcal{N}} \emptyset \vdash t : A$*, such that* $\#_{\mathtt{App}}(\Phi) + 1 = \#_{\mathtt{App}}(\Psi)$*.*
2. *If* $t \to_{\mathtt{CBV}} u$ *and* $\Phi \rhd_{\mathcal{V}} \emptyset \vdash u : M$*, then there is* $\Psi \rhd_{\mathcal{V}} \emptyset \vdash t : M$*, such that* $\#_{\mathtt{App}}(\Phi) + 1 = \#_{\mathtt{App}}(\Psi)$*.*

Example 4.16 (Weighted Subject Expansion)**.**

Example for CBN:

Recall the CBN reduction step in Example 4.14:

$$(\lambda z.z\,z)\,((\lambda x.x)\,\lambda y.y) \to_{\mathtt{CBN}(\beta)} ((\lambda x.x)\,\lambda y.y)\,((\lambda x.x)\,\lambda y.y),$$

and type derivation Ψ' *for* $((\lambda x.x)\,\lambda y.y)\,((\lambda x.x)\,\lambda y.y)$*:*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : \{\!\{A\}\!\} \vdash x : A}\,\mathtt{Ax}}{\vdash \lambda x.x : \{\!\{A\}\!\} \to A}\,\mathtt{Abs} \qquad \dfrac{\dfrac{\dfrac{}{y : \{\!\{\star\}\!\} \vdash y : \star}\,\mathtt{Ax}}{\vdash \lambda y.y : A}\,\mathtt{Abs}}{\vdash \lambda y.y : \{\!\{A\}\!\}}\,\mathtt{Many}}{\vdash (\lambda x.x)\,\lambda y.y : A}\,\mathtt{App} \qquad \dfrac{\dfrac{\dfrac{}{x : \{\!\{\star\}\!\} \vdash x : \star}\,\mathtt{Ax}}{\vdash \lambda x.x : A}\,\mathtt{Abs} \qquad \dfrac{\dfrac{}{\vdash \lambda y.y : \star}\,\mathtt{Ans}}{\vdash \lambda y.y : \{\!\{\star\}\!\}}\,\mathtt{Many}}{\vdash (\lambda x.x)\,\lambda y.y : \star}\,\mathtt{App}}{\vdash ((\lambda x.x)\,\lambda y.y)\,((\lambda x.x)\,\lambda y.y) : \star}\,\mathtt{App}$$

where $\#_{\mathtt{App}}(\Psi') = 3$*. By Lemma 4.15, there is* $\Phi' \rhd_{\mathcal{N}} \emptyset \vdash (\lambda z.z\,z)\,((\lambda x.x)\,\lambda y.y) : \star$*, such that* $\#_{\mathtt{App}}(\Psi') + 1 = \#_{\mathtt{App}}(\Phi')$*. By Lemma 4.15, there is a way of using the information in* Ψ' *to build a type derivation* Φ' *for* $(\lambda z.z\,z)\,((\lambda x.x)\,\lambda y.y)$*. In this case, this is guaranteed by Lemma 4.11, and* Φ' *is precisely the one in Example 4.14, where* $\#_{\mathtt{App}}(\Phi') = 4$*.*

Example for CBV:

Recall the CBV reduction step in Example 4.14:

$$(\lambda z.z\,z)\,\lambda x.x \to_{\mathtt{CBV}(\beta_{\mathrm{v}})} (\lambda x.x)\,\lambda x.x,$$

and type derivation Ψ' *for* $(\lambda x.x)\,\lambda x.x$*:*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : \{\!\{\,\}\!\} \vdash x : \{\!\{\,\}\!\}}\,\mathtt{Ax}}{\vdash \lambda x.x : \{\!\{\{\!\{\,\}\!\} \to \{\!\{\,\}\!\}\}\!\}}\,\mathtt{Abs} \qquad \dfrac{}{\vdash \lambda x.x : \{\!\{\,\}\!\}}\,\mathtt{Abs}}{\vdash (\lambda x.x)\,\lambda x.x : \{\!\{\,\}\!\}}\,\mathtt{App}$$

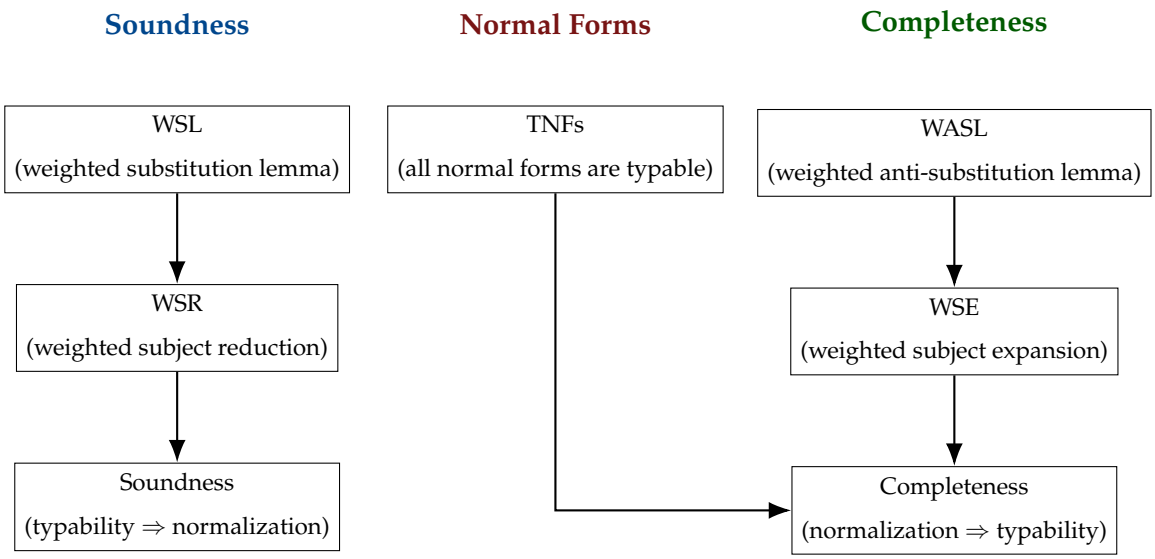
where $\#_{\mathtt{App}}(\Psi') = 1$*. By Lemma 4.15, there is* $\Phi' \rhd_{\mathcal{N}} \emptyset \vdash (\lambda z.z\,z)\,\lambda x.x : \{\!\{\,\}\!\}$*, such that* $\#_{\mathtt{App}}(\Psi') + 1 = \#_{\mathtt{App}}(\Phi')$*. By Lemma 4.15, there is a way of using the information in* Ψ' *to build a type derivation* Φ' *for* $(\lambda z.z\,z)\,((\lambda x.x)\,\lambda y.y)$*. In this case, this is also guaranteed by Lemma 4.11, and* Φ' *is precisely the one in Example 4.14, where* $\#_{\mathtt{App}}(\Phi') = 2$*.*

Finally, the only thing left to prove to show soundness and completeness for CBN and CBV is that all normal forms are typable. This is made precise by the following lemma.

Lemma 4.17 (Typability of CBN- and CBV-Normal Forms)**.** *If* $t \in \mathrm{V}_\circ$*, then there is a derivation* $\Phi \rhd_{\mathcal{N}\,(\text{resp. }\mathcal{V})} \emptyset \vdash t : A$ *(resp.* M*).*

Remark 4.18. The proof schema seen so far is quite standard and will serve as a template for showing similar properties for all the type systems and reduction strategies appearing in this thesis. Therefore, we are going to enumerate the steps here:

- Normal Forms:
  - Prove that all normal forms are typable (TNFs);
- Soundness:
  - Prove a weighted version of the *substitution lemma* (WSL);
  - Use WSL to prove a weighted version of the *subject reduction* property (WSR);
  - Use WSR to prove that typability is *sound* with respect to normalization.
- Completeness:
  - Prove a weighted version of the *anti-substitution lemma* (WASL);
  - Use WASL to prove a weighted version of the *subject expansion* property (WSE);
  - Use TNFs and WSE to prove that typability is *complete* with respect to normalization.



Now, we are able to prove that $\mathcal{N}$-(resp. $\mathcal{V}$-)typabilty is sound and complete with respect to CBN-(resp. CBV-)normalization.

Theorem 4.19 (Soundness and Completeness for CBN and CBV)**.** *Let $t \in \Lambda_\circ$. Then, t is:*

1. *$\mathcal{N}$-typable if and only if t CBN-normalizes to some term $u \in \mathtt{NO}_{\mathtt{CBN}}$. Moreover, if $\Phi \rhd_{\mathcal{N}} \emptyset \vdash t : A$, then $t \twoheadrightarrow^{n}_{\mathtt{CBN}} u$ and $\#_{\mathtt{App}}(\Phi) \geq n$.*

2. *$\mathcal{V}$-typable if and only if t $\mathtt{CBV}$-normalizes to some term $u \in \mathtt{NO}_{\mathtt{CBV}}$. Moreover, if $\Phi \rhd_{\mathcal{V}} \emptyset \vdash t : M$, then $t \twoheadrightarrow^{n}_{\mathtt{CBV}} u$ and $\#_{\mathtt{App}}(\Phi) \geq n$.*

Proof. The proofs for both statements follow the same overall structure. The $(\Rightarrow)$ direction holds by a weighted version of subject reduction (see Lemma 4.13), and the $(\Leftarrow)$ direction holds by the typability of normal forms (see Lemma 4.17) and a weighted version of subject expansion (see Lemma 4.15). The "moreover" statements easily hold by the weighted nature of the subject reduction and expansion statements. □

Example 4.20 (Soudness and Completeness)**.**

Example for CBN:

Recall term $(\lambda z.z\,z)\,((\lambda x.x)\,\lambda y.y)$ from Example 4.14 and its type derivation Φ' of size $\#_{\mathtt{App}}(\Phi') = 4$. Notice that $(\lambda z.z\,z)\,((\lambda x.x)\,\lambda y.y)$ $\mathtt{CBN}$-normalizes to $\lambda y.y \in \mathtt{NO}_{\mathtt{CBN}}$ in 4 β-steps:

$$\begin{array}{lll}(\lambda z.z\,z)\,((\lambda x.x)\,\lambda y.y) & \to_{\mathtt{CBN}(\beta)} & ((\lambda x.x)\,\lambda y.y)\,((\lambda x.x)\,\lambda y.y)\\ & \to_{\mathtt{CBN}(\beta)} & (\lambda y.y)\,((\lambda x.x)\,\lambda y.y)\\ & \to_{\mathtt{CBN}(\beta)} & (\lambda x.x)\,\lambda y.y\\ & \to_{\mathtt{CBN}(\beta)} & \lambda y.y.\end{array}$$

Example for CBV:

Recall term $(\lambda z.z\,z)\,\lambda x.x$ from Example 4.14 and its type derivation Φ' of size $\#_{\mathtt{App}}(\Phi') = 2$. Notice that $(\lambda z.z\,z)\,\lambda x.x$ $\mathtt{CBV}$-normalizes to $\lambda x.x \in \mathtt{NO}_{\mathtt{CBN}}$ in 2 $\beta_{\mathtt{v}}$-steps:

$$\begin{array}{lll}(\lambda z.z\,z)\,\lambda x.x & \to_{\mathtt{CBV}(\beta_{\mathtt{v}})} & (\lambda x.x)\,\lambda x.x\\ & \to_{\mathtt{CBV}(\beta_{\mathtt{v}})} & \lambda x.x.\end{array}$$

4.2 Subsuming Call-By-Name and Call-By-Value

In this section, we introduce a type system based on non-idempotent intersection types for the $\lambda!$-calculus.

Definition 4.21 (CBPV Type System)**.** Recall the set of CBPV types presented in Definition 4.1. The *CBPV Type System $\mathcal{P}$* assigns CBPV (multi-)types to $\lambda!$-terms and consists of the following typing rules:

$$\frac{}{x : M \vdash x : M}\ \mathtt{Ax} \qquad \frac{\Gamma \vdash t : A}{\Gamma \setminus\!\setminus x \vdash \lambda x.t : \Gamma(x) \to A}\ \mathtt{Abs} \qquad \frac{}{\emptyset \vdash \lambda x.t : \star}\ \mathtt{Ans}$$

$$\frac{\Gamma \vdash t : M \to A \qquad \Delta \vdash u : M}{\Gamma + \Delta \vdash t\,u : A}\ \mathtt{App} \qquad \frac{(\Gamma_i \vdash t : A_i)_{i \in \mathtt{I}}}{+_{i \in \mathtt{I}} \Gamma_i \vdash\, !t : \{\!\{A_i\}\!\}_{i \in \mathtt{I}}}\ \mathtt{Bg} \qquad \frac{\Gamma \vdash t : \{\!\{A\}\!\}}{\Gamma \vdash \mathtt{der}(t) : A}\ \mathtt{Der}$$

The exact quantitative soundness and completeness of type system $\mathcal{P}$ with respect to CBPV-normalization is shown by following the proof schema in Remark 4.18. However, since the introduction of ! leads to clashes, CBPV-normalization is restricted to clash-free normal forms. This means that the third point of the proof schema in Remark 4.18 must be restricted to clash-free normal forms, *i.e.* clash-free normal forms are the only typable normal forms.

Lemma 4.22 (Typability of Clash-Free CBPV-Normal Forms)**.** *If $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$, then there is a derivation $\Phi \rhd_{\mathcal{P}} \emptyset \vdash t : A$.*

Moreover, since we are only interested in clash-free CBPV-normalization (*i.e.* ending in a clash-free normal form), we must prove that that all typable terms are clash-free, *i.e.* normalize to a clash-free CBPV normal form.

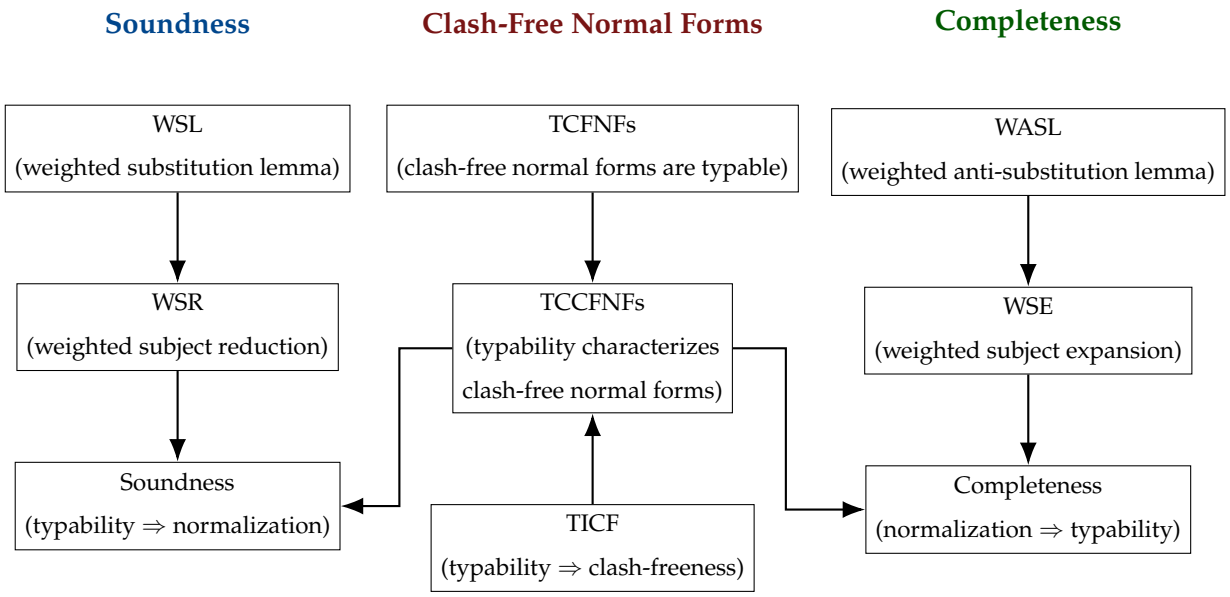
Lemma 4.23 (Typability Implies Clash-Freeness)**.** *Let $t \in !\Lambda_{\circ}$. If $\Phi \rhd_{\mathcal{P}} \emptyset \vdash t : A$, then t is clash-free.*

In fact, we can show that $\mathcal{P}$-typability *captures* the clash-freeness of CBPV-normal forms.

Lemma 4.24 (Typability Characterizes Clash-Free CBPV-Normal Forms)**.** *Let $t \in !\Lambda_{\circ}$. Then, $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$ if and only if $t \in \mathtt{NO}_{\mathtt{CBPV}}$ and t is $\mathcal{P}$-typable.*

Remark 4.25. In sum, the proof schema in Remark 4.18 for CBN and CBV can be adapted to CBPV as follows:

- Clash-Free Normal Forms
  - Prove that all *clash-free normal forms* are typable (TCFNFs)
  - Prove that *typability implies clash-freeness* (TICF);
  - Use TCFNFs and TICF to prove that *typability characterizes clash-free normal forms* (TCCFNFs);
- Soundness
  - Prove a weighted version of the *substitution lemma* (WSL);
  - Use WSL to prove a weighted version of the *subject reduction* property (WSR);
  - Use TCCFNFs and WSR to prove that typability is *sound* with respect to normalization.
- Completeness
  - Prove a weighted version of the *anti-substitution lemma* (WASL);
  - Use WASL to prove a weighted version of the *subject expansion* property (WSE);
  - Use TCCFNFs and WSE to prove that typability is *complete* with respect to normalization.



Now, we are able to prove that $\mathcal{P}$-typability is sound and complete with respect to clash-free CBPV-normalization. Again, we include the proof in order to illustrate the general reasoning used in the last point of Remark 4.25.

Theorem 4.26 (Soundness and Completeness for CBPV)**.** *Let $t \in !\Lambda_{\circ}$. Then, t is $\mathcal{P}$-typable if and only if t CBPV-normalizes to a term $u \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$. Moreover, if $\Phi \rhd_{\mathcal{P}} \emptyset \vdash t : A$ is a (not necessarily tight) type derivation, then $t \overset{n\cdot\beta_{\mathtt{v}}+m\cdot\beta_{!}}{\twoheadrightarrow}_{\mathtt{CBPV}} u$ and $\#_{\mathtt{App},\mathtt{Der}}(\Phi) \geq n \cdot \mathtt{App} + m \cdot \mathtt{Der}$.*

Proof. The soundness proof ($\Rightarrow$) is straightforward by Proposition 3.7 and Lemmas 4.24 and C.4. The completeness proof ($\Leftarrow$) is obtained by induction on the length of the CBPV-normalizing sequence:

- Case $n + m = 0$. Then, since $u \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$, we know that u is $\mathcal{P}$-typable by Lemma 4.24.
- Case $n + m = k + 1$. Then, $t \overset{n-1\cdot\beta_{\mathtt{v}}+m\cdot\beta_{!}}{\rightarrow}_{\mathtt{CBPV}} r \rightarrow_{\mathtt{CBPV}} u$ or $t \overset{n\cdot\beta_{\mathtt{v}}+m-1\cdot\beta_{!}}{\rightarrow}_{\mathtt{CBPV}} r \rightarrow_{\mathtt{CBPV}} u$. Notice that, by Lemma 4.24, u is $\mathcal{P}$-typable. By applying the *i.h.* to $k = n + m - 1$, we know r is $\mathcal{P}$-typable. Therefore, t is $\mathcal{P}$-typable by Lemma C.6.

The "moreover" statement easily holds by the "weightedness" of the subject reduction and expansion statements. □

We now show that the translations in Definition 3.12 are also sound and complete with respect to $\mathcal{P}$-typability.

At this point, it should be stressed that the weights are related to the number of steps needed to normalize terms. Indeed, in CBN and CBV, the weight of a type derivation is the number of

occurrences of rule App; in CBPV, it is the number of occurrences of rules App and Der. As per the soundness and completeness of the system, we know that these weights are upper-bounds on the length of normalization sequences. Since the translations on terms preserve the number of β- and β_v-steps, typability should therefore be not only sound and complete with respect to these translations, but should also reflect this preservation at the level of typing weights.

Theorem 4.27 (Soundness and Completeness of Translations)**.** *Let $t \in \Lambda$. Then*

1. $\Phi \triangleright_{\mathcal{N}} \Gamma \vdash t : A$ *if and only if* $\Psi \triangleright_{\mathcal{P}} \Gamma \vdash (t)^{\mathtt{CBN}} : A$. *Moreover,* $\#_{\mathtt{App}}(\Phi) = \#_{\mathtt{App}}(\Psi)$.
2. $\Phi \triangleright_{\mathcal{V}} \Gamma \vdash t : M$ *if and only if* $\Psi \triangleright_{\mathcal{P}} \Gamma \vdash (t)^{\mathtt{CBV}} : M$. *Moreover,* $\#_{\mathtt{App}}(\Phi) = \#_{\mathtt{App}}(\Psi)$.

As we have already mentioned, and can be seen in all subject reduction and expansion statements above, weights only give us upper-bounds on the length of normalization sequences. This is due to the fact that typability, as it is defined now for CBN, CBV and CBPV, does not guarantee that type derivation for (clash-free) normal forms have zero weight.

Example 4.28 (Type Derivations for Normal Forms)**.** *In CBN and CBV, the following are all admissible type derivations for $\lambda x.x\,x$:*

$$\dfrac{\dfrac{\dfrac{}{x : \{\!\{\{\!\{\,\}\!\} \to \star\}\!\} \vdash x : \{\!\{\,\}\!\} \to \star}\,\mathtt{Ax} \qquad \dfrac{}{\emptyset \vdash x : \{\!\{\,\}\!\}}\,\mathtt{Many}}{x : \{\!\{\{\!\{\,\}\!\} \to \star\}\!\} \vdash x\,x : \star}\,\mathtt{App}}{\triangleright_{\mathcal{N}} \emptyset \vdash \lambda x.x\,x : \{\!\{\{\!\{\,\}\!\} \to \star\}\!\} \to \star}\,\mathtt{Abs}$$

$$\dfrac{\dfrac{\dfrac{}{x : \{\!\{\{\!\{\,\}\!\} \to \{\!\{\,\}\!\}\}\!\} \vdash x : \{\!\{\{\!\{\,\}\!\} \to \{\!\{\,\}\!\}\}\!\}}\,\mathtt{Ax} \qquad \dfrac{}{\emptyset \vdash x : \{\!\{\,\}\!\}}\,\mathtt{Ax}}{x : \{\!\{\{\!\{\,\}\!\} \to \{\!\{\,\}\!\}\}\!\} \vdash x\,x : \{\!\{\,\}\!\}}\,\mathtt{App}}{\triangleright_{\mathcal{V}} \emptyset \vdash \lambda x.x\,x : \{\!\{\{\!\{\,\}\!\} \to \{\!\{\,\}\!\} \to \{\!\{\,\}\!\}\}\!\}}\,\mathtt{Abs}$$

In CBPV, the following is an admissible type derivation for $\lambda x.x\,!x$:

$$\dfrac{\dfrac{\dfrac{}{x : \{\!\{\{\!\{\,\}\!\} \to \star\}\!\} \vdash x : \{\!\{\,\}\!\} \to \star}\,\mathtt{Ax} \qquad \dfrac{}{\emptyset \vdash\, !x : \{\!\{\,\}\!\}}\,\mathtt{Bg}}{x : \{\!\{\{\!\{\,\}\!\} \to \star\}\!\} \vdash x\,!x : \star}\,\mathtt{App}}{\triangleright_{\mathcal{P}} \emptyset \vdash \lambda x.x\,!x : \{\!\{\{\!\{\,\}\!\} \to \star\}\!\} \to \star}\,\mathtt{Abs}$$

All of them have an occurrence of rule App*, even though $\lambda x.x\,x$ is irreducible in all strategies. However, $\lambda x.x\,x$ also has the following admissible type derivations in CBN and CBV:*

$$\dfrac{}{\triangleright_{\mathcal{N}} \emptyset \vdash \lambda x.x\,x : \star}\,\mathtt{Ans} \qquad \dfrac{}{\triangleright_{\mathcal{V}} \emptyset \vdash \lambda x.x\,x : \{\!\{\,\}\!\}}\,\mathtt{Abs}$$

And the following type derivations in CBPV:

$$\dfrac{}{\triangleright_{\mathcal{P}} \emptyset \vdash \lambda x.x\,!x : \star}\,\mathtt{Ans}$$

As it turns out, this can be refined via *tight typings* [5, 12].

4.3 Tight Typings

Following the pioneering work by Gardner and Kfoury [1, 2] and the previously mentioned work by de Carvalho [3, 4], non-idempotent intersection types have been extensively used to reason about programs from a quantitative point of view [6, 13, 16, 43, 45, 47, 78–81]. However, some care has to be taken in order to obtain good measures. More specifically, to measure the length of evaluation of a term, it is necessary to consider only *minimal typings*, *i.e.* the smallest typing derivations. In [5, 12], this is done by introducing the key notion of *tightness* for type derivations∗. Very succinctly, minimal type derivations are obtained by imposing *tightness conditions* on the typing of the final judgment of type derivations.

Definition 4.29 (Tight Typings for CBN, CBV, and CBPV)**.** Let $t \in \Lambda_\circ$ and $u \in !\Lambda_\circ$. A type derivation Φ is tight if and only if it is of the form:

- $\Phi \rhd_{\mathcal{N}} \emptyset \vdash t : \star$.
- $\Phi \rhd_{\mathcal{V}} \emptyset \vdash t : \{\!\{\,\}\!\}$.
- $\Phi \rhd_{\mathcal{P}} \emptyset \vdash u : \star$ or $\Phi \rhd_{\mathcal{P}} \emptyset \vdash u : \{\!\{\,\}\!\}$.

Intuitively, tight derivations are those whose final judgment forces the typing to reflect only computationally relevant structure, excluding administrative or persistent resources.

Now, by restricting type derivations to be *tight*, we can prove that having type derivations with zero weight is a characteristic of (clash-free) normal forms.

Lemma 4.30 (Zero Weight Tight Type Derivations Characterize (Clash-Free) Normal Forms)**.** *Let $t \in \Lambda_\circ$ and $u \in !\Lambda_\circ$.*

1. *If $\Phi \rhd_{\mathcal{N}} \emptyset \vdash t : A$ is tight, then $\#_{\mathtt{App}}(\Phi) = 0$ if and only if $t \in \mathtt{NO}_{\mathtt{CBN}}$.*
2. *If $\Phi \rhd_{\mathcal{V}} \emptyset \vdash t : M$ is tight, then $\#_{\mathtt{App}}(\Phi) = 0$ if and only if $t \in \mathtt{NO}_{\mathtt{CBV}}$.*
3. *If $\Phi \rhd_{\mathcal{P}} \emptyset \vdash u : A$ is tight, then $\#_{\mathtt{App},\mathtt{Der}}(\Phi) = 0 \cdot \mathtt{App} + 0 \cdot \mathtt{Der}$ if and only if $u \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$.*

Moreover, we can also show that all normal forms are typable with tight type derivations.

Example 4.31 (Zero Weight Tight Type Derivations for (Clash-Free) Normal Forms)**.** *Recall the following type derivation for CBN-, CBV-, and CBPV-normal forms, in Example 4.28:*

$$\frac{}{\rhd_{\mathcal{N}} \emptyset \vdash \lambda x.x\, x : \star}\ \mathtt{Ans} \qquad \frac{}{\rhd_{\mathcal{V}} \emptyset \vdash \lambda x.x\, x : \{\!\{\,\}\!\}}\ \mathtt{Abs} \qquad \frac{}{\rhd_{\mathcal{P}} \emptyset \vdash \lambda x.x\, !x : \star}\ \mathtt{Ans}$$

∗In this thesis we are going to focus only on *closed* terms, therefore the size explosion problem (see [82] for details) does not apply. However, it should be noted that, in [5, 12], the size explosion problem is solved by splitting the set of typing rules so that terms that disappear during evaluation (consumed) and terms that remain in the normal form (persistent) are typed differently.

Notice that all of these type derivations have no occurrences of rules `App` *(for CBN, CBV, and CBPV) or* `Der` *(for CBPV). Thus, all of these have size (weight) zero.*

Lemma 4.32 (Tight Typability of (Clash-Free) Normal Forms)**.** *Let $t \in \Lambda_\circ$ and $u \in !\Lambda_\circ$.*

1. *If $t \in \mathtt{NO}_{\mathtt{CBN}}$, then there is $\Phi \rhd_{\mathcal{N}} \emptyset \vdash t : A$ tight.*
2. *If $t \in \mathtt{NO}_{\mathtt{CBV}}$, then there is $\Phi \rhd_{\mathcal{V}} \emptyset \vdash t : M$ tight.*
3. *If $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$, then there is $\Phi \rhd_{\mathcal{P}} \emptyset \vdash u : A$ tight.*

By restricting to tight derivations, the quantitative measures become exact: the number of rule applications corresponds precisely to the number of reduction steps.

We can now state exact soundness and completeness results, where the size of a tight derivation exactly matches the length of the normalization sequence.

Theorem 4.33 (Exact Soundness and Completeness for CBN and CBV)**.** *Let $t \in \Lambda_\circ$. Then, t is:*

1. *$\mathcal{N}$-typable if and only if t* `CBN`*-normalizes to a term $u \in \mathtt{NO}_{\mathtt{CBN}}$. Moreover, if $\Phi \rhd_{\mathcal{N}} \emptyset \vdash t : A$ is tight, then $t \twoheadrightarrow^{n}_{\mathtt{CBN}} u$ and $\#_{\mathtt{App}}(\Phi) = n$.*
2. *$\mathcal{V}$-typable if and only if t* `CBV`*-normalizes to a term $u \in \mathtt{NO}_{\mathtt{CBV}}$. Moreover, if $\Phi \rhd_{\mathcal{V}} \emptyset \vdash t : M$ is tight, then $t \twoheadrightarrow^{n}_{\mathtt{CBV}} u$ and $\#_{\mathtt{App}}(\Phi) = n$.*

Example 4.34 (Exact Soundness and Completeness for CBN and CBV)**.** *Recall the terms in Example 4.20, as well as their respective type derivations:*

$$\Phi_1 \rhd_{\mathcal{N}} \emptyset \vdash (\lambda z.z\,z)\,((\lambda x.x)\,\lambda y.y) : \star \qquad \Phi_2 \rhd_{\mathcal{V}} \emptyset \vdash (\lambda z.z\,z)\,\lambda x.x : \{\!\{\,\}\!\}$$

Notice that both Φ_1 and Φ_2 are tight*. As it turns out, this is why the size of Φ_1 coincides with the length of the* `CBN`*-normalization sequence for $(\lambda z.z\,z)\,((\lambda x.x)\,\lambda y.y)$, and similarly with respect to the size of Φ_2 and the length of the* `CBV`*-normalization sequence for $(\lambda z.z\,z)\,\lambda x.x$.*

Theorem 4.35 (Exact Soundness and Completeness for CBPV)**.** *Let $t \in !\Lambda_\circ$. Then, t is $\mathcal{P}$-typable if and only if t* `CBPV`*-normalizes to a term $u \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$. Moreover, if $\Phi \rhd_{\mathcal{P}} \emptyset \vdash t : A$ is tight, then $t \overset{n\cdot\beta_{\mathtt{v}}+m\cdot\beta_{!}}{\twoheadrightarrow}_{\mathtt{CBPV}} u$ and $\#_{\mathtt{App},\mathtt{Der}}(\Phi) = n \cdot \mathtt{App} + m \cdot \mathtt{Der}$.*

Example 4.36 (Exact Soundness and Completeness for CBPV)**.** *Recall the CBPV reduction sequences in Example 3.5:*

$$((\lambda x.\mathtt{der}(x))\,!\lambda y.y)\,((\lambda x.x)\,!\lambda y.y) \overset{3\cdot\beta_{\mathtt{v}}+1\cdot\beta_{!}}{\to}_{\mathtt{CBPV}} !\lambda y.y,$$

where $!\lambda y.y \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$. *Let* $A = \{\!\{\ \}\!\} \to \{\!\{\ \}\!\}$, *and* Φ *be:*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : \{\!\{A\}\!\} \vdash x : \{\!\{A\}\!\}}\,\mathtt{Ax}}{x : \{\!\{A\}\!\} \vdash \mathtt{der}(x) : A}\,\mathtt{Der}}{\vdash \lambda x.\mathtt{der}(x) : \{\!\{A\}\!\} \to A}\,\mathtt{Abs} \quad \dfrac{\dfrac{\dfrac{}{y : \{\!\{\ \}\!\} \vdash y : \{\!\{\ \}\!\}}\,\mathtt{Ax}}{\vdash \lambda y.y : A}\,\mathtt{Abs}}{\vdash\ !\lambda y.y : \{\!\{A\}\!\}}\,\mathtt{Bg}}{\varnothing \vdash (\lambda x.\mathtt{der}(x))\,!\lambda y.y : A}\,\mathtt{App} \quad \dfrac{\dfrac{\dfrac{}{x : \{\!\{\ \}\!\} \vdash x : \{\!\{\ \}\!\}}\,\mathtt{Ax}}{\vdash \lambda x.x : A}\,\mathtt{Abs} \quad \dfrac{}{\vdash\ !\lambda y.y : \{\!\{\ \}\!\}}\,\mathtt{Bg}}{\varnothing \vdash (\lambda x.x)\,!\lambda y.y : \{\!\{\ \}\!\}}\,\mathtt{App}}{\varnothing \vdash ((\lambda x.\mathtt{der}(x))\,!\lambda y.y)\,((\lambda x.x)\,!\lambda y.y) : \{\!\{\ \}\!\}}\,\mathtt{App}$$

Notice that Φ *is* tight, *and* $\#_{\mathtt{App},\mathtt{Der}}(\Phi) = 3 \cdot \mathtt{App} + 1 \cdot \mathtt{Der}$. *Thus, this derivation respects Theorem 4.35.*

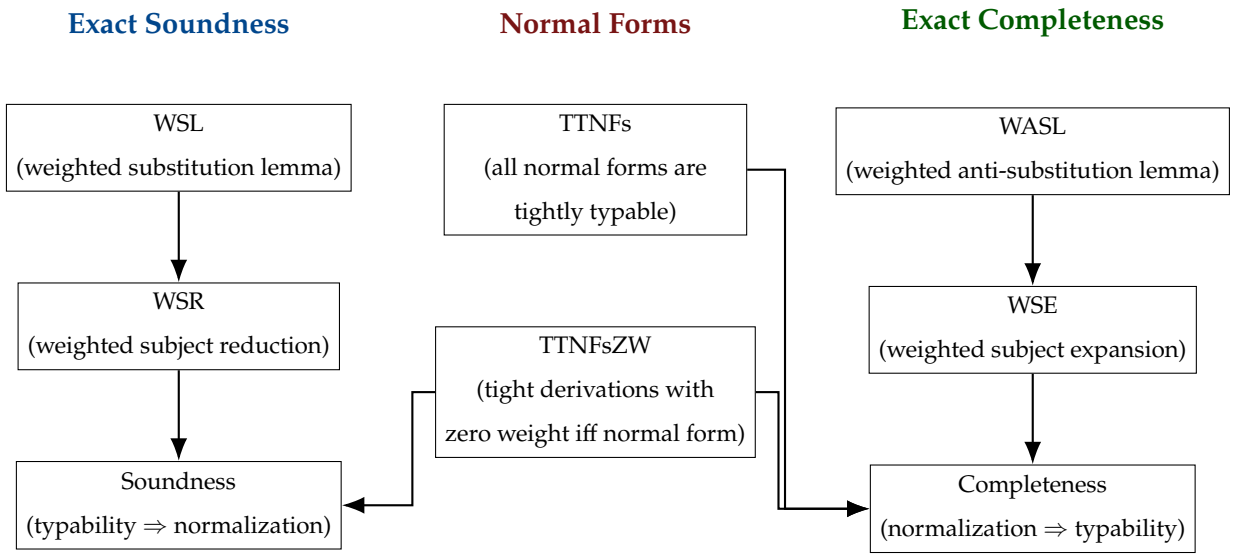
Finally, by Theorem 4.27, we can also conclude that the translations provide an exact correspondence between the length of normalizing sequences in CBN and CBV, and in CBPV.

In sum, *tight typability* and *exact soundness and completeness* are obtained by slightly tweaking the proof schemas in Remarks 4.18 and 4.25 as follows.

Remark 4.37 (Proof Schema for CBN and CBV). We refine the proof schema for CBN and CBV in Remark 4.18 as follows:

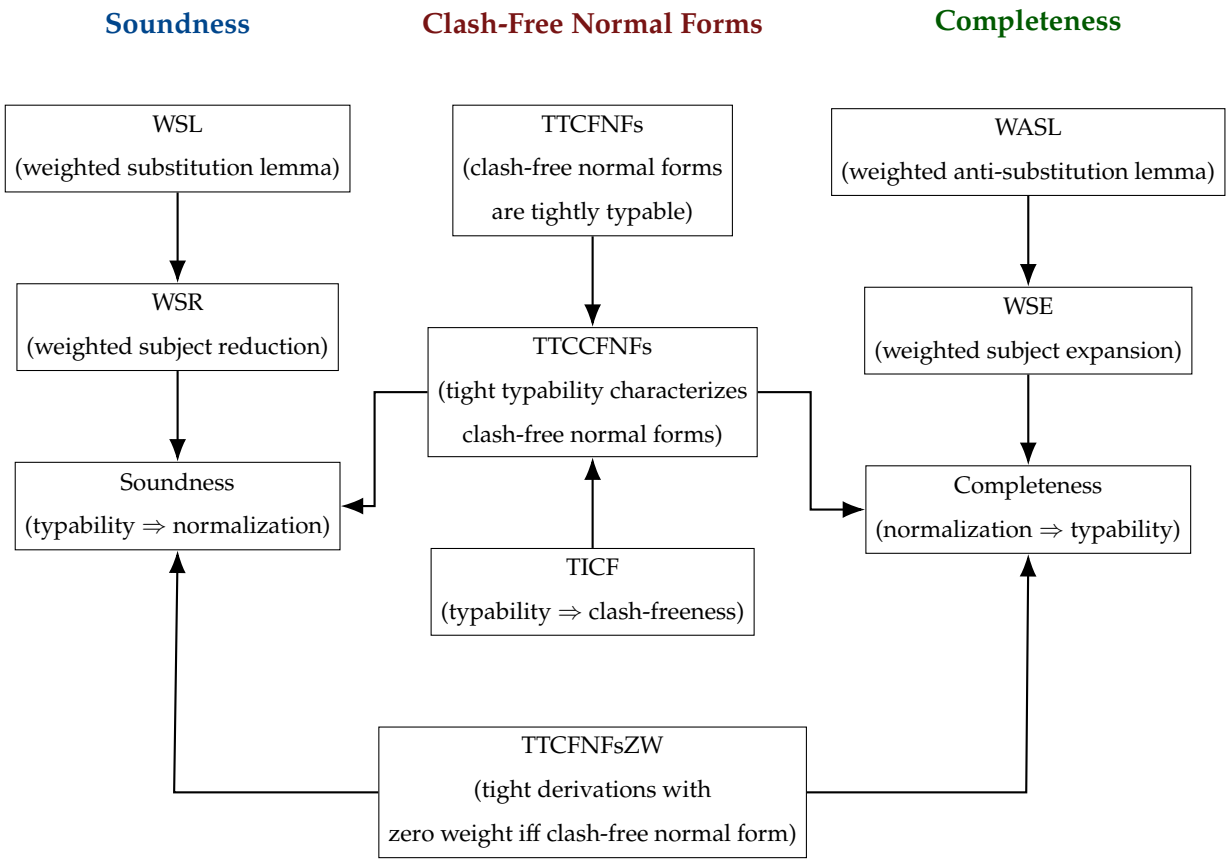
- Normal Forms:
  - Prove that all normal forms are *tightly* typable (TTNFs);
  - Prove that having *tight type derivations* with *zero weight* is a characteristic of *normal forms* (TTNFsZW)
- Exact Soundness:
  - Prove a weighted version of the *substitution lemma* (WSL);
  - Use WSL to prove a weighted version of the *subject reduction* property (WSR);
  - Use TTNFsZW and WSR to prove that typability is *sound* and *exact* with respect to normalization.
- Exact Completeness:
  - Prove a weighted version of the *anti-substitution lemma* (WASL);
  - Use WASL to prove a weighted version of the *subject expansion* property (WSE);
  - Use TTNFs and TTNFsZW, and WSE to prove that typability is *complete* and *exact* with respect to normalization.

We summarize the refined proof dependencies using the following diagrams:



Remark 4.38 (Proof Schema for CBPV). We refine the proof schema for CBPV in Remark 4.25 as follows:

- Clash-Free Normal Forms
  - Prove that all *clash-free normal forms* are *tightly* typable (TTCFNFs)
  - Prove that having *tight type derivations* with *zero weight* is a characteristic of *clash-free normal forms* (TTCFNFsZW);
  - Prove that *typability implies clash-freeness* (TICF);
  - Use TTCFNFs and TICF to prove that *tight typability characterizes clash-free normal forms* (TTCCFNFs);
- Exact Soundness
  - Prove a weighted version of the *substitution lemma* (WSL);
  - Use WSL to prove a weighted version of the *subject reduction* property (WSR);
  - Use TTCCFNFsZW and WSR to prove that typability is *sound* with respect to normalization.
- Exact Completeness
  - Prove a weighted version of the *anti-substitution lemma* (WASL);
  - Use WASL to prove a weighted version of the *subject expansion* property (WSE);
  - Use TTCCFNFs, TTCFNFsZW, and WSE to prove that typability is *complete* with respect to normalization.


Soundness
Clash-Free Normal Forms
Completeness
WSL
(weighted substitution lemma)
TTCFNFs
(clash-free normal forms
are tightly typable)
WASL
(weighted anti-substitution lemma)
WSR
(weighted subject reduction)
TTCCFNFs
(tight typability characterizes
clash-free normal forms)
WSE
(weighted subject expansion)
Soundness
(typability ⇒ normalization)
Completeness
(normalization ⇒ typability)
TICF
(typability ⇒ clash-freeness)
TTCFNFsZW
(tight derivations with
zero weight iff clash-free normal form)

Chapter 5

Monads and Algebraic Operations

In [31, 83], Moggi gave a unified account of computational effects as strong monads. However, monads only describe the structure of effectful computations abstractly, they do not by themselves specify the primitive operations that generate effects. To do so, Plotkin and Power suggested the representation of effects by equational theories [37, 84, 85]. Intuitively, an effect can be seen as a branching point in the execution of a program where each possible outcome corresponds to a different branch. The source of an effect is represented by an operation whose arity is the number of possible outcomes and whose arguments represent the branches. The properties of those sources are then described by equations between terms built using the corresponding operations. Almost all computational effects have such a representation, such as partiality, nondeterminism, input, output, and state. An example of a non-algebraic, yet monadic effect [86] is that of continuations [87], whereas (exception) handlers [88] are an example of non-algebraic, yet natural effects. The algebraic approach to computational effects has been quite successful: it gave an elegant denotational semantics of effectful programs in terms of Lawvere theories [84], showed how it induces the usual monadic understanding of effects [37], and provided simple ways of combining effects [89]. In this section, we introduce monads and algebraic operations both abstractly and through examples of monads modeling concrete computational effects.

This chapter proceeds as follows. We first recall monads and Kleisli categories, then introduce algebraic theories and their free models. We explain how algebraic theories induce monads and comonads, and finally describe comodel-based operational semantics.

5.1 Monads

In order to interpret a programming language in a category $\mathbb{C}$, Moggi distinguishes between the object X of *values* (of type X) from the object TX of *computations* (of type X), and takes as denotations of programs (of type X) the elements of TX. Moggi calls T a *notion of computation*, since it abstracts

away from the type of values computations may produce. It turns out that there are many choices for TX corresponding to different notions of computations. But rather than focusing on a specific T, Moggi focuses instead on finding the general properties common to all notions of computation. As it turns out, the only requirement that needs to be imposed is that programs should form a category.

Definition 5.1 (Category)**.** A *category* $\mathbb{C}$ consists of:

- a collection $\mathtt{obj}(\mathbb{C})$ of *objects*;
- for each pair of objects $x, y \in \mathtt{obj}(\mathbb{C})$, a collection $\mathbb{C}(x, y)$ of *morphisms* from x to y;
- for each pair of morphisms $f \in \mathbb{C}(x, y)$ and $g \in \mathbb{C}(y, z)$, a morphism $g \circ f \in \mathbb{C}(x, z)$, called their *composition*;
- for each object $x \in \mathtt{obj}(\mathbb{C})$, a morphism $\mathrm{id}_x \in \mathbb{C}(x, x)$, called the *identity morphism* on x;

such that the following properties hold:

- **Associativity:** for all $w, x, y, z \in \mathtt{obj}(\mathbb{C})$ and $h \in \mathbb{C}(w, x)$, $g \in \mathbb{C}(x, y)$, $f \in \mathbb{C}(y, z)$,

$$(f \circ g) \circ h = f \circ (g \circ h).$$

- **Unit laws:** for all $x, y \in \mathtt{obj}(\mathbb{C})$ and $f \in \mathbb{C}(x, y)$,

$$\mathrm{id}_y \circ f = f = f \circ \mathrm{id}_x.$$

For example, in the category called `Set` of sets and functions, objects are sets, morphisms are (total) functions. Composition is given by ordinary function composition, the identity morphism on a set is the identity function, and the associativity and unit laws follow directly from the corresponding properties of function composition.

Definition 5.2 (Kleisli Triple)**.** A *Kleisli triple* over a category $\mathbb{C}$ is a triple $\mathbb{T} = (T, \eta, \cdot^\dagger)$, consisting of:

- A *map* $T : \mathtt{obj}(\mathbb{C}) \to \mathtt{obj}(\mathbb{C})$ over objects of $\mathbb{C}$, called the *carrier* of $\mathbb{T}$;
- A $\mathbb{C}$-object-indexed family η of morphisms $\eta_X : X \to T(X)$, called the *unit* of $\mathbb{T}$ on X;
- An operation $\cdot^\dagger$ mapping a $\mathbb{C}$-morphism $f : X \to TY$ to $f^\dagger : TX \to TY$, called the *Kleisli extension* of f.

Let $f : X \to TY$ and $g : Y \to TZ$. Then, the above data must satisfy the following identities:

$$\begin{aligned} \eta_X{}^\dagger &= \mathtt{Id}_{TX} \\ f^\dagger \circ \eta_X &= f \\ g^\dagger \circ f^\dagger &= (g^\dagger \circ f)^\dagger \end{aligned}$$

Kleisli triples are in bijective correspondence with monads [90]. Throughout this thesis, we will therefore use the terms *Kleisli triple* and *monad* interchangeably.

Moreover, we work exclusively in the category `Set`. In particular, every monad is *strong*, and we can freely pass between the Kleisli and Eilenberg-Moore presentations without further comment (see [90] for details).

Given a Kleisli triple $\mathbb{T} = (T, \eta, \cdot^\dagger)$, the corresponding monad (T, η, μ) defined as follows:

$$\begin{aligned} T(f) &:= (\eta_X \circ f)^\dagger \\ \mu_X &:= (\mathtt{Id}_{TX})^\dagger. \end{aligned}$$

Conversely, given a monad (T, η, μ), the Kleisli extension of a morphism $f : X \to T(Y)$ is defined by:

$$f^\dagger := \mu_Y \circ Tf.$$

The Kleisli construction gives rise to a category whose morphisms represent effectful programs.

Definition 5.3 (Kleisli Category)**.** Given a Kleisli triple $\mathbb{T} = (T, \eta, \cdot^\dagger)$ over $\mathbb{C}$, the *Kleisli category* $\mathbb{C}_T$ is defined as follows:

- The objects of $\mathbb{C}_T$ are those of $\mathbb{C}$;
- The set $\mathbb{C}_T(X, Y)$ of morphisms from X to Y in $\mathbb{C}_T$ is $\mathbb{C}(X, TY)$;
- The identity on X in $\mathbb{C}_T$ is $\eta_X : X \to TX$;
- The composition of morphisms $f \in \mathbb{C}_T(X, Y)$ and $g \in \mathbb{C}_T(Y, Z)$ in $\mathbb{C}_T$ is $g^\dagger \circ f : X \to TZ$.

The amount of category theory used in this thesis is deliberately kept minimal. Nevertheless, categorical notions provide a convenient and precise vocabulary for describing the semantic structures underlying effectful computation.

When dealing with an arbitrary monad, we are going to adopt the above notation. However, whenever dealing with specific monads or with multiple monads at the same time, we are going to adopt the notational convention in [91], as follows. Given a monad $\mathbb{T} = (T, \eta, \cdot^\dagger)$, we always use a capital Latin letter to denote its carrier, whereas the name of the monad is given by the same letter written in "blackboard style". The unit of $\mathbb{T}$ will be denoted by T, *i.e.* with same letter used to denote its carrier, but written in small capital style, whereas we use the notation $\cdot^{\mathbb{T}}$ in place of $\cdot^\dagger$.

5.2 Relevant Examples

We now give examples of monads modeling interesting notions of computation.

Example 5.4 (Partiality). *Partial computations are modeled using the* partiality monad $\mathbb{M} = (M, \eta^{\mathbb{M}}, \cdot^{\mathbb{M}})$. *The carrier MX of $\mathbb{M}$ is defined as $X + \{\bot\}$, where $\bot$ is a special symbol (not appearing in X) used to denote diverging computations. The unit and Kleisli extension of $\mathbb{M}$ are defined as follows:*

$$\begin{aligned} \eta^{\mathbb{M}}_X(x) &:= \mathtt{in}_1(x) \\ f^{\mathbb{M}}(\chi) &:= \begin{cases} f(x) & \text{if } \chi = \mathtt{in}_1(x) \\ \mathtt{in}_2(\bot) & \text{if } \chi = \mathtt{in}_2(\bot), \end{cases} \end{aligned}$$

where $f : X \to MY$ is a function.

The following result by Hyland, Plotkin and Power [89], tells us that we can combine the maybe monad with any other monad $\mathbb{T}$ using an operation on Lawvere theories [92, 93] that was introduced to combine algebraic theories.

Proposition 5.5 (Proposition 3 of [91]). *Let $\mathbb{T} = (T, \tau, \cdot^{\mathbb{T}})$ be a monad, and also $\mathbb{TM} = (TM, \tau^{\mathbb{M}}, \cdot^{\mathbb{TM}})$ be the sum of $\mathbb{T}$ and $\mathbb{M}$, defined as follows:*

$$\begin{aligned} TMX &:= T(MX) \\ \tau^{\mathbb{M}}_X &:= \eta^{\mathbb{M}}_X ; \tau_{MX} \\ f^{\mathbb{TM}} &:= (f_M)^{\mathbb{T}} \end{aligned}$$

where, for a function $f : X \to TMY$, f_M is defined as:

$$f_M(\chi) := \begin{cases} f(x) & \text{if } \chi = x \\ \tau_{MX}(\bot) & \text{if } \chi = \bot. \end{cases}$$

Then, $\mathbb{TM}$ is a monad.

Example 5.6 (Exceptions). *Let $\mathcal{E}$ be a set of exception symbols. The* exception monad $\mathbb{E} = (E, \eta^{\mathbb{E}}, \cdot^{\mathbb{E}})$ *over $\mathcal{E}$ has carrier $E(X) = X + \mathcal{E}$ (assuming that $X \cap \mathcal{E} = \emptyset$). The unit and Kleisli extension of $\mathbb{E}$ are defined as follows:*

$$\begin{aligned} \eta^{\mathbb{E}}_X(x) &:= \mathtt{in}_1(x) \\ f^{\mathbb{E}}(\chi) &:= \begin{cases} f(x) & \text{if } \chi = \mathtt{in}_1(x) \\ \mathtt{in}_2(e) & \text{if } \chi = \mathtt{in}_2(e), \end{cases} \end{aligned}$$

where $f : X \to EY$ is a function. The exception monad models total computations that can raise exceptions. It has the same structure as $\mathbb{M}$, thus Proposition 5.5 also applies to $\mathbb{E}$. In particular, partial computations with exceptions can be modeled using the sum $\mathbb{ME}$.

These examples anticipate the coalgebraic/comodel treatment introduced later.

Example 5.7 (Interactive Input)**.** *Let $\mathcal{A}$ be a given alphabet. The* interactive input monad $\mathbb{I} = (I, \imath, \cdot^{\mathbb{I}})$ *has carrier $\mu Z.X + (\mathcal{A} \to Z)$, and the unit and Kleisli extension of $\mathbb{I}$ are defined as follows:*

$$\begin{array}{rcl} \imath_X(x) & := & x \\ f^{\mathbb{I}}(\chi) & := & \begin{cases} f(x) & \text{if } \chi = \mathtt{in}_1(x) \\ \mathtt{in}_2(f^{\mathbb{I}} \circ g) & \text{if } \chi = \mathtt{in}_2(g). \end{cases} \end{array}$$

*The interactive input monad models total computations that depend on a finite stream of input. Indeed, we can think of IX as the set of input-driven trees, with leaves labeled by values of type X, and internal nodes branching on elements of $\mathcal{A}$ (*i.e.*, waiting for input). Then, the unit maps a value x into the tree consisting of only one leaf labelled with x, and the Kleisli extension computes the tree obtained by replacing the leaves labelled by x of a given tree with the tree $f(x)$. More precisely,*

$$IX \cong X + (\mathcal{A} \to IX)$$

This is the canonical unfolding of the fixed point of the functor:

$$F(Z) = X + (\mathcal{A} \to Z)$$

So IX is the final coalgebra for functor F.

We can extend $\mathbb{I}$ to take into account divergent computations. To do so, we have to take into account infinite streams of input. This is not just "partiality + input". It is input-driven computation that may diverge only through infinite interaction.

Let $\mathcal{A}^\omega$ be the set of infinite strings over $\mathcal{A}$. The partial interactive input monad $\mathbb{I}^\omega = (I^\omega, \imath^\omega, \cdot^{\mathbb{I}^\omega})$ *has carrier $\nu Z.MX + (\mathcal{A} \to Z)$, and the unit and Kleisli extension of $\mathbb{I}^\omega$ are defined as follows:*

$$\begin{array}{rcl} \imath_X(x) & := & \mathtt{in}_1(x) \\ f^{\mathbb{I}^\infty}(\chi) & := & \begin{cases} f(x) & \text{if } \chi = \mathtt{in}_1(x) \\ \mathtt{in}_1(\bot) & \text{if } \chi = \mathtt{in}_1(\bot) \\ \mathtt{in}_2(f^{\mathbb{I}^\infty} \circ g) & \text{if } \chi = \mathtt{in}_2(g). \end{cases} \end{array}$$

The use of μ enforces finiteness of interaction, while ν allows infinite input streams and thus divergence.

Example 5.8 (Interactive Output)**.** *Let $\mathcal{A}$ be a given alphabet. The* interactive output monad $\mathbb{O} = (O, o, \cdot^{\mathbb{O}})$ *has carrier $\mu Z.X + (\mathcal{A} \times Z)$, which is isomorphic to $X \times \mathcal{A}^*$, where $\mathcal{A}^*$ is the set of finite strings over $\mathcal{A}$. Let ϵ denote the empty string, and uw the concatenation of $u, w \in A$. The unit and Kleisli extension*

of $\mathbb{O}$ are defined as follows:

$$\begin{aligned} o_X(x) &:= (x, \epsilon) \\ f^{\mathrm{O}}(x,u) &:= (y, uw) \end{aligned}$$

where $(w,y) = f(x)$. The interactive output monad models total computations with finite output. Indeed, we can think of OX as the set of output traces, with leaves labeled by values of type X, and internal nodes branching on elements of $\mathcal{A}$ (i.e., *waiting for input).*

We can also extend $\mathbb{O}$ to take into account divergent computations. To do so, we have to take into account that divergent computations have infinite output. This is not just "partiality + output". It is output-driven computation that may diverge only through infinite interaction.

Let $\mathcal{A}^{\leq\omega}$ be the set of finite and infinite strings over $\mathcal{A}$. Concatenation is extended to (possibly) infinite strings by defining uw to be u if u is infinite. The partial interactive output monad $\mathbb{O}^{\leq\omega} = (O^{\leq\omega}, o^{\leq\omega}, \cdot^{\mathbb{O}^{\leq\omega}})$ *has carrier $MX \times \mathcal{A}^{\leq\omega}$, and the unit and Kleisli extension of $\mathbb{O}^{\leq\omega}$ are defined as follows:*

$$\begin{aligned} o_X(x) &:= (x, \epsilon) \\ f^{\mathbb{O}^{\leq\omega}}(\chi, u) &:= \begin{cases} (\gamma, uw) & \text{if } \chi = x \text{ and } f(x) = (\gamma, w) \\ (\bot, u) & \text{if } \chi = \bot \end{cases} \end{aligned}$$

The use of μ enforces finiteness of interaction, while ν allows infinite output traces and thus divergence.

Example 5.9 (Nondeterminism)**.** *Nondeterministic computations are modelled using the* non-empty powerset monad $\mathbb{F} = (F, \mathrm{F}, \cdot^{\mathbb{F}})$. *The carrier FX of $\mathbb{F}$ is defined as $\{\chi \subseteq X \mid \chi \neq \emptyset\}$. The unit and Kleisli extension of $\mathbb{F}$ are defined as follows:*

$$\begin{aligned} \mathrm{F}_X(x) &:= \{x\} \\ f^{\mathbb{F}}(\chi) &:= \textstyle\bigcup_{x\in\chi} f(x). \end{aligned}$$

Intuitively, set $\chi \in FX$ denotes the set of possible (nondeterministic) outcomes of a computation. It is possible to model partial computations by considering the full powerset monad $\mathbb{P} = (P, \mathrm{P}, \cdot^{\mathbb{P}})$, where P and P are defined as for $\mathbb{F}$, or by using $\mathbb{FM}$, the sum $\mathbb{F}$ and $\mathbb{M}$. In the former case, the empty set models purely divergent computations, in the latter case, divergence is traced using the symbol $\bot$.

Example 5.10 (Global State)**.** *Let $\mathcal{L}$ be a set of location names and $\mathcal{V}$ be a countable set of values. A state is a function $s : \mathcal{L} \to \mathcal{V}$ and the set of states $\mathcal{S}$ is denoted by $\mathcal{V}^{\mathcal{L}}$. The* global state monad $\mathbb{G} = (G, \mathrm{G}, \cdot^{\mathbb{G}})$ *has carrier $\mathcal{S} \to (X \times \mathcal{S})$, and the unit and Kleisli extension of $\mathbb{G}$ are defined as follows:*

$$\begin{aligned} \mathrm{G}_X(x) &:= \lambda s.(x, s) \\ f^{\mathbb{G}}(g) &:= \lambda s.f(y)(q) \end{aligned}$$

where $g(s) = (y, q)$. Informally, an element of GX is a function g mapping a state s to a pair (configuration) (y, q). That is, g is a computation that, starting with the initial state s, finishes with the final state q, and returns the result y.

As for the sum of effects, the following result by Hyland, Plotkin and Power [89], tells us that we can combine the global state monad with any monad $\mathbb{T}$ using the so-called tensor of effects.

Proposition 5.11 (Proposition 4 of [91])**.** *Let $\mathbb{T} = (T, \tau, \cdot^\dagger)$ be a monad, and $\mathbb{T} \otimes \mathbb{G} = (T \otimes G, \tau \otimes \gamma, \cdot^{\mathbb{T}\otimes\mathbb{G}})$ be the tensor of $\mathbb{T}$ and $\mathbb{G}$, defined as follows:*

$$\begin{aligned}(T \otimes G)X &:= \mathcal{S} \to T(X \times \mathcal{S}) \\ (\tau \otimes \gamma)_X &:= \Lambda(\tau_{X\times\mathcal{S}}) \\ f^{\mathbb{T}\otimes\mathbb{G}}(g) &:= \lambda s.(\Lambda^{-1} f)^{\mathbb{T}}(g)(s)\end{aligned}$$

where, for $f : X \to Y^Z$ and $g : Z \times X \to Y$, we have $\Lambda^{-1} f : Z \times X \to Y$ and $\Lambda g : X \to Y^Z$. Then, $\mathbb{T} \otimes \mathbb{G}$ is a monad.

The restriction to algebraic operations will later play a crucial role in our quantitative analysis, as it ensures that effectful steps correspond to uniform, locally generated costs.

5.3 Algebraic Operations

As we have seen in the previous section, monads provide a unified framework for modeling a wide range of computational effects. However, as Plotkin and Power observed in their influential series of papers [36, 37, 84], monads capture the structure of effectful computation, but they do not provide an account of the primitives (operations) that produce such effects. To address this, Plotkin and Power proposed describing computational effects algebraically via equational theories, where operations denote effectful primitives and equations capture their meaning. As it turns out, many familiar effects (including all of those discussed so far) are determined by appropriate algebraic theories.

Definition 5.12 (Algebraic Signatures)**.** A *signature* Σ is given by a set of *operation symbols* $\mathtt{op}_i$, each associated with an *operation signature* consisting of a *parameter set* $\mathcal{P}_i$ and an *arity set* $\mathcal{A}_i$, and denoted by $\mathtt{op}_i : \mathcal{P}_i \rightsquigarrow \mathcal{A}_i$.

We are going to consider that parameter and arity sets are at most *countable*, and write $\mathtt{op}_i \in \Sigma$ instead of $\mathtt{op}_i : \mathcal{P}_i \rightsquigarrow \mathcal{A}_i \in \Sigma$ whenever the parameter and arity sets of $\mathtt{op}_i$ are clear from the context.

Definition 5.13 (Algebraic Terms)**.** Let X be a countable set of variables. The set $\mathtt{Tree}_\Sigma X$ of well-founded Σ-trees over X is defined inductively as follows:

- For every $x \in X$, $\mathtt{return}\, x \in \mathtt{Tree}_\Sigma X$, where tree leaves are labelled by `return` according to their role as pure effectful computations (see Definition 5.21);
- For every operation symbol $\mathtt{op}_i : \mathcal{P}_i \rightsquigarrow \mathcal{A}_i \in \Sigma$, parameter $p \in \mathcal{P}_i$, and continuation $\kappa : \mathcal{A}_i \to \mathtt{Tree}_\Sigma X$, $\mathtt{op}_i(p, \kappa) \in \mathtt{Tree}_\Sigma X$.

We will use the λ-notation to denote continuations, and $\mathtt{a}, \mathtt{b}, \ldots \in \mathcal{X}$ to denote free variables of algebraic theories.

An *algebraic theory* $\mathcal{T} = (\Sigma_\mathcal{T}, \mathrm{Eq}_\mathcal{T})$ is given by a signature $\Sigma_\mathcal{T}$ and a set of equations $\mathrm{Eq}_\mathcal{T}$ between $\Sigma_\mathcal{T}$-trees.

Let 0 be the empty set $\emptyset$, 1 be a singleton set $\{*\}$, and $\mathcal{B}$ be the set of Booleans.

Example 5.14 (Partiality)**.** *The algebraic theory $\mathcal{M}$ for partiality is defined as follows:*

$$\begin{aligned} \Sigma_\mathcal{M} &:= \{\mathtt{raise} : 1 \rightsquigarrow 0\} \\ Eq_\mathcal{M} &:= \emptyset. \end{aligned}$$

Example 5.15 (Exception)**.** *Let $\mathcal{E}$ be a set of exceptions. The algebraic theory $\mathcal{E}$ for exception is defined as follows:*

$$\begin{aligned} \Sigma_\mathcal{E} &:= \{\mathtt{raise} : \mathcal{E} \rightsquigarrow 0\} \\ Eq_\mathcal{E} &:= \emptyset. \end{aligned}$$

Example 5.16 (Interactive Input)**.** *Let $\mathcal{A}$ be a set of inputs. The algebraic theory $\mathcal{I}$ for interactive input is defined as follows:*

$$\begin{aligned} \Sigma_\mathcal{I} &:= \{\mathtt{input} : 1 \rightsquigarrow \mathcal{A}\} \\ Eq_\mathcal{I} &:= \emptyset. \end{aligned}$$

Example 5.17 (Interactive Output)**.** *Let $\mathcal{A}$ be a set of outputs. The algebraic theory $\mathcal{O}$ for interactive output is defined as follows:*

$$\begin{aligned} \Sigma_\mathcal{O} &:= \{\mathtt{output} : \mathcal{A} \rightsquigarrow 1\} \\ Eq_\mathcal{O} &:= \emptyset. \end{aligned}$$

Example 5.18 (Nondeterminism)**.** *Let $\mathcal{B}$ be a set of Booleans. The algebraic theory $\mathcal{F}$ for finite non-empty nondeterminism is defined as follows:*

$$\begin{aligned} \Sigma_\mathcal{F} &:= \{\mathtt{choose} : 1 \rightsquigarrow \mathcal{B}\} \\ Eq_\mathcal{F} &:= \left\{ \begin{array}{rcl} \mathtt{choose}(*, \lambda \mathtt{a}.\mathtt{t}) & \sim_\mathcal{F} & \mathtt{t}\ \textit{if}\ \mathtt{a} \notin \mathtt{fv}(\mathtt{t}) \\ \mathtt{choose}(*, \lambda \mathtt{a}.\mathtt{choose}(*, \lambda \mathtt{b}.\mathtt{t})) & \sim_\mathcal{F} & \mathtt{choose}(*, \lambda \mathtt{b}.\mathtt{choose}(*, \lambda \mathtt{a}.\mathtt{t})) \\ \mathtt{choose}(*, \lambda \mathtt{a}.\mathtt{t}) & \sim_\mathcal{F} & \mathtt{choose}(*, \lambda \mathtt{b}.\mathtt{t}\{\mathtt{a} \setminus \mathtt{not}\, \mathtt{b}\}) \end{array} \right\}. \end{aligned}$$

The algebraic theory presented above differs from the more standard join-semilattice presentation of nondeterminism, in that branching is encoded using Boolean structure, yielding idempotence, commutativity,

and involution via negation. Despite this difference in presentation, the resulting algebraic structure induces a monad that is equivalent to the non-empty powerset monad on `Set`*: Boolean combinations of outcomes correspond to finite, non-empty sets of possible results, modulo the usual identifications given by idempotence and commutativity. Thus, the expressive power is the same as in the standard semilattice-based approach, while offering a formulation better aligned with the presentation of the other algebraic theories.*

Example 5.19 (Global State)**.** *Let* $\mathtt{v}, \mathtt{w} \in \mathcal{V}$*, and* $l, l' \in \mathcal{L}$*, such that* $l \neq l'$*. The algebraic theory* $\mathcal{G}$ *for global state is defined as follows:*

$$\Sigma_{\mathcal{G}} \;:=\; \{\mathtt{get} : \mathcal{L} \rightsquigarrow \mathcal{V}, \mathtt{set} : \mathcal{L} \times \mathcal{V} \rightsquigarrow 1\}$$

$$Eq_{\mathcal{G}} \;:=\; \left\{ \begin{array}{rcl}
\mathtt{get}(l, \lambda \mathtt{a}.\mathtt{t}) & \sim_{\mathcal{G}} & \mathtt{t} \textit{ if } \mathtt{a} \notin \mathtt{fv}(\mathtt{t}) \\
\mathtt{get}(l, \lambda \mathtt{a}.\mathtt{set}((l, \mathtt{a}), \lambda \mathtt{b}.\mathtt{t})) & \sim_{\mathcal{G}} & \mathtt{t} \textit{ if } \mathtt{a} \notin \mathtt{fv}(\mathtt{t}) \\
\mathtt{get}(l, \lambda \mathtt{b}.\mathtt{get}(l, \lambda \mathtt{a}.\mathtt{t})) & \sim_{\mathcal{G}} & \mathtt{get}(l, \lambda \mathtt{a}.\mathtt{t}\{\mathtt{b} \setminus \mathtt{a}\}) \\
\mathtt{set}((l, \mathtt{v}), \lambda \mathtt{a}.\mathtt{get}(l, \lambda \mathtt{b}.\mathtt{t})) & \sim_{\mathcal{G}} & \mathtt{set}((l, \mathtt{v}), \lambda \mathtt{a}.\mathtt{t}\{\mathtt{b} \setminus \mathtt{v}\}) \\
\mathtt{set}((l, \mathtt{v}), \lambda \mathtt{a}.\mathtt{set}((l, \mathtt{w}), \lambda \mathtt{b}.\mathtt{t})) & \sim_{\mathcal{G}} & \mathtt{set}((l, \mathtt{w}), \lambda \mathtt{b}.\mathtt{t}) \\
\mathtt{get}(l, \lambda \mathtt{a}.\mathtt{get}(l', \lambda \mathtt{b}.\mathtt{t})) & \sim_{\mathcal{G}} & \mathtt{get}(l', \lambda \mathtt{b}.\mathtt{get}(l, \lambda \mathtt{a}.\mathtt{t})) \\
\mathtt{get}(l, \lambda \mathtt{a}.\mathtt{set}((l', \mathtt{v}), \lambda \mathtt{b}.\mathtt{t})) & \sim_{\mathcal{G}} & \mathtt{set}((l', \mathtt{v}), \lambda \mathtt{b}.\mathtt{get}(l, \lambda \mathtt{a}.\mathtt{t})) \\
\mathtt{set}((l, \mathtt{v}), \lambda \mathtt{a}.\mathtt{set}((l', \mathtt{w}), \lambda \mathtt{b}.\mathtt{t})) & \sim_{\mathcal{G}} & \mathtt{set}((l', \mathtt{w}), \lambda \mathtt{b}.\mathtt{set}((l, \mathtt{v}), \lambda \mathtt{a}.\mathtt{t}))
\end{array} \right\}.$$

Definition 5.20 (Models of Algebraic Theories)**.** Let $\mathcal{T} = (\Sigma_{\mathcal{T}}, \mathrm{Eq}_{\mathcal{T}})$ be some algebraic theory. A $\Sigma_{\mathcal{T}}$*-algebra* $\mathtt{I}$ is a pair $(|\mathtt{I}|, I)$ between:

- A *carrier* set $|\mathtt{I}|$;
- A set of *interpretation* functions $I = \{[\![\mathtt{op}_i]\!]_{\mathtt{I}} : \mathcal{P}_i \times (\mathcal{A}_i \to |\mathtt{I}|) \to |\mathtt{I}| \mid \mathtt{op}_i : \mathcal{P}_i \rightsquigarrow \mathcal{A}_i \in \Sigma_{\mathcal{T}}\}$.

$\Sigma_{\mathcal{T}}$-algebras can be extended to $\Sigma_{\mathcal{T}}$-trees. Let $\mathtt{t} \in \mathtt{Tree}_{\Sigma_{\mathcal{T}}} X$. Then, $[\![\mathtt{t}]\!]_{\mathtt{I}} : (X \to |\mathtt{I}|) \to |\mathtt{I}|$ is defined inductively over the structure of $\mathtt{t}$ as follows:

$$[\![\mathtt{return}\, x]\!]_{\mathtt{I}} := \lambda f.f\, x \qquad [\![\mathtt{op}_i(p, \kappa)]\!]_{\mathtt{I}} := \lambda f.[\![\mathtt{op}_i]\!]_{\mathtt{I}}(p, \lambda \mathtt{a}.[\![\kappa\, \mathtt{a}]\!]_{\mathtt{I}}\, f)$$

An equation $\mathtt{t} \sim_{\mathcal{T}} \mathtt{u} \in \mathrm{Eq}_{\mathcal{T}}$ is said to be *valid* for $\mathtt{I}$ whenever $[\![\mathtt{t}]\!]_{\mathtt{I}} = [\![\mathtt{u}]\!]_{\mathtt{I}}$. A $\mathcal{T}$*-algebra* (or $\mathcal{T}$*-model*) $\mathtt{M}$ is a $\Sigma_{\mathcal{T}}$-algebra that validates all the equations in $\mathrm{Eq}_{\mathcal{T}}$.

Definition 5.21 (Free Models of Algebraic Theories)**.** Let $\mathcal{T}$ be an algebraic theory and X a set. The *free* $\mathcal{T}$*-model*, also called the *free* $\mathcal{T}$*-algebra*, generated by X is a model $\mathtt{M}$ together with a function $\eta_X : X \to |\mathtt{M}|$, such that, for every $\mathcal{T}$-model $\mathtt{M}'$ and every function $f : X \to |\mathtt{M}'|$ there is a unique $\mathcal{T}$-homomorphism $\overline{f} : \mathtt{M} \to \mathtt{M}'$, such that $f = \overline{f} \circ \eta_X$.

As it turns out, every algebraic theory has a free $\mathcal{T}$-model $\mathtt{F}_{\mathcal{T}} X$ generated by an arbitrary set X. The carrier of $\mathtt{F}_{\mathcal{T}} X$ is $\mathtt{Tree}_{\Sigma_{\mathcal{T}}} X / \sim_{\mathcal{T}}$, *i.e.* $\mathtt{Tree}_{\Sigma_{\mathcal{T}}} X$ quotiented by the $\Sigma_{\mathcal{T}}$-congruence $\sim_{\mathcal{T}}$ generated

by $\mathrm{Eq}_{\mathcal{T}}$*. Let $[\cdot]_{\mathcal{T}}$ be the equivalence class induced by $\sim_{\mathcal{T}}$. Then, each $\mathtt{op}_i : \mathcal{P}_i \rightsquigarrow \mathcal{A}_i \in \Sigma_{\mathcal{T}}$ is interpreted in $\mathtt{F}_{\mathcal{T}}X$ as the following function:

$$\begin{array}{rcl} [\![\mathtt{op}_i]\!]_{\mathtt{F}_{\mathcal{T}}X} : \mathcal{P}_i \times (\mathcal{A}_i \to |\mathtt{F}_{\mathcal{T}}X|) & \to & |\mathtt{F}_{\mathcal{T}}X| \\ (p, \kappa) & \mapsto & [\mathtt{op}_i(p, \kappa)]_{\mathcal{T}} \end{array}$$

The function $\eta_X : X \to \mathtt{F}_{\mathcal{T}}X$ is defined as follows:

$$\eta_X(x) = [\mathtt{return}\, x].$$

The key insight of [37] is that many effects (monads) are determined by algebraic theories (notions of computation). Indeed, an algebraic theory $\mathcal{T}$ defines a unique monad $\mathbb{T}$ whose T-algebras are essentially the same as $\mathcal{T}$-models, and any monad corresponds to at most one algebraic theory in this fashion.

Example 5.22 (Partiality)**.** *It is usual to consider that the partiality monad* $\mathbb{M}$ *comes without any operations. This is due to divergence being an implicit feature of any Turing complete programming language. However, it is sometimes useful to consider an explicit divergence operation,* e.g. *to allow partial computations in a total programming language. As such, we will consider the functor* $\mathbb{M}$ *to be induced by the partiality monad* $\mathcal{M}$*, as defined in Example 5.14. The interpretation of the unique operation on* $\mathbb{M}$ *is as follows:*

$$\begin{array}{rcl} [\![\mathtt{raise}]\!]_{\mathbb{M}} : 1 \times (\emptyset \to X + \{\bot\}) & \to & X + \{\bot\} \\ (\star, \kappa) & \mapsto & \mathtt{in}_2(\bot), \end{array}$$

since $\emptyset \to X + \{\bot\} \cong \emptyset$*.*

Example 5.23 (Exception)**.** *The exception monad* $\mathbb{E}$ *is induced by* $\mathcal{E}$*, as defined in Example 5.15. Its unique operation is the same as the one for* $\mathcal{M}$*, but it is parameterized by the set of exception symbols* $\mathcal{E}$*. The interpretation on* $\mathbb{E}$ *is as follows:*

$$\begin{array}{rcl} [\![\mathtt{raise}]\!]_{\mathbb{E}} : \mathcal{E} \times (\emptyset \to X + \mathcal{E}) & \to & X + \mathcal{E} \\ (e, \kappa) & \mapsto & \mathtt{in}_2(e), \end{array}$$

since $\emptyset \to X + \mathcal{E} \cong \emptyset$*.*

As it turns out, it is possible to generalize Proposition 5.5 to $\Sigma_{\mathcal{T}}$-algebraic monads, *i.e.* a monad whose carrier functor is the free model functor of some algebraic theory.

*The sizes of the arity sets require some care here. Since $\Sigma_{\mathcal{T}}$ may contain operations of infinite arity (such as `input`, of arity A, or `get`, of arity $\mathcal{V}$), the inductive definition of $\mathtt{Tree}_{\Sigma_{\mathcal{T}}}X$ in Definition 5.13 does not close at stage ω: it closes at the first regular cardinal strictly above all arities occurring in $\Sigma_{\mathcal{T}}$ (ω_1 when all arities are countable). Such a bound always exists because Definition 5.12 takes $\Sigma_{\mathcal{T}}$ to be a *set* of operation symbols, each with a *set* as its arity; hence $\mathtt{Tree}_{\Sigma_{\mathcal{T}}}X$ is a set and the quotient below is well defined [94]. Some bound is essential: for signatures with operations of unbounded arity, free models may not exist. In particular, there is no free complete Boolean algebra on countably many generators [95, 96].

Proposition 5.24 (Proposition 5 of [91]). *Let $\mathbb{T} = (T, \tau, \cdot^{\mathbb{T}})$ be a $\Sigma_{\mathcal{T}}$-algebraic monad. Then, the monad $\mathbb{T}\mathbb{E}$ is $\Sigma_{\mathcal{T}+\mathcal{E}}$-algebraic.*

Example 5.25 (Interactive Input). *The total $\mathbb{I}$ and partial $\mathbb{I}^\infty$ interactive input monads are both induced by $\mathcal{I}$, as defined in Example 5.16. The interpretation on $\mathbb{I}$ and $\mathbb{I}^\infty$ are as follows:*

$$\begin{aligned} [\![\mathtt{input}]\!]_{\mathbb{I}} : 1 \times (\mathcal{A} \to \mu Z.X + (A \to Z)) &\to \mu Z.X + (A \to Z) \\ (\star, \kappa) &\mapsto \mathtt{in}_2(\kappa), \\ [\![\mathtt{input}]\!]_{\mathbb{I}^\infty} : 1 \times (\mathcal{A} \to \nu Z.MX + (A \to Z)) &\to \nu Z.MX + (A \to Z) \\ (\star, \kappa) &\mapsto \mathtt{in}_2(\kappa). \end{aligned}$$

Example 5.26 (Interactive Output). *The total $\mathbb{O}$ and partial $\mathbb{O}^\infty$ interactive output monads are both induced by $\mathcal{O}$, as defined in Example 5.17. The interpretation on $\mathbb{O}$ is as follows:*

$$\begin{aligned} [\![\mathtt{output}]\!]_{\mathbb{O}} : \mathcal{A} \times (1 \to X \times \mathcal{A}^*) &\to X \times \mathcal{A}^* \\ (a, \kappa) &\mapsto (x, aw), \end{aligned}$$

where $\kappa \star = (x, w)$. Notice that, for $f(x) = (y, u)$, $f^\dagger(x, aw) = (y, (aw)u) = (y, a(wu)) = [\![\mathtt{output}]\!]_{\mathbb{O}}(a, f^\dagger \circ \kappa)$. Thus, $\mathbb{O}$ is $\Sigma_{\mathcal{O}}$-algebraic.

The interpretation on $\mathbb{O}^\infty$ is as follows:

$$\begin{aligned} [\![\mathtt{output}]\!]_{\mathbb{O}^\infty} : \mathcal{A} \times (1 \to MX \times \mathcal{A}^\infty) &\to MX \times \mathcal{A}^\infty \\ (a, \kappa) &\mapsto (\chi, aw), \end{aligned}$$

where $\kappa \star = (\chi, w)$.

Example 5.27 (Nondeterminism). *Consider the algebraic theory $\mathcal{F}$ for finite non-empty non-determinism. The non-empty powerset monad $\mathbb{F}$ is induced by $\mathcal{F}$, as defined in Example 5.18. The interpretation on $\mathbb{F}$ is as follows:*

$$\begin{aligned} [\![\mathtt{choose}]\!]_{\mathbb{F}} : 1 \times (\mathcal{B} \to \{\chi \subseteq X \mid \chi \neq \emptyset\}) &\to \{\chi \subseteq X \mid \chi \neq \emptyset\} \\ (\star, \kappa) &\mapsto \textstyle\bigcup_{b \in \mathcal{B}} \kappa(b) \end{aligned}$$

Example 5.28 (Global State). *The global state monad $\mathbb{G}$ is induced by $\mathcal{G}$, as defined in Example 5.19. The interpretation on $\mathbb{G}$ is as follows:*

$$\begin{aligned} [\![\mathtt{get}]\!]_{\mathbb{G}} : \mathcal{L} \times (\mathcal{V} \to (\mathcal{S} \to (X \times \mathcal{S}))) &\to \mathcal{S} \to (X \times \mathcal{S}) \\ (l, \kappa) &\mapsto \lambda\sigma.\kappa(\sigma(l))(\sigma) \\ [\![\mathtt{set}]\!]_{\mathbb{G}} : (\mathcal{L} \times \mathcal{V}) \times (1 \to (\mathcal{S} \to (X \times \mathcal{S}))) &\to \mathcal{S} \to (X \times \mathcal{S}) \\ ((l, \mathtt{v}), \kappa) &\mapsto \lambda\sigma.\kappa(*)(\sigma\{l := \mathtt{v}\}) \end{aligned}$$

where $\sigma\,(l)$ is the value at location l of state σ, and $\sigma\{l := \mathrm{v}\}$ is the state that has value v at location l, and the same values as σ in all other locations.

Remark 5.29 (Algebraic Operations and Cost). In the setting of this thesis, algebraic operations are not only used to describe the qualitative structure of computational effects, but also play a quantitative role. Intuitively, each occurrence of an algebraic operation corresponds to a uniform computational event, which will later be assigned a cost.

This perspective will be made precise in subsequent chapters, where programs are interpreted in suitable comodels and cost semantics are obtained by counting transitions associated with algebraic operations. The present algebraic presentation thus serves as the qualitative backbone upon which the quantitative analysis developed later is built.

Remark 5.30 (Algebraicity). From an operational point of view, the *algebraicity* of an operation expresses a strong form of uniformity: the operational behavior of the operation does not depend on the particular arguments to which it is applied, nor on the surrounding evaluation context in which it is executed. In other words, invoking an algebraic operation produces an effect whose observable behavior is independent of the computational environment, apart from the continuation that receives its result. This restriction plays a crucial structuring role in effectful computation, as it enforces a clear separation between where effects are triggered and how their results are subsequently used.

Algebraicity makes it possible to organize effectful programs in a modular and compositional way, and to exercise fine-grained control over the generation and propagation of effects. From a semantic perspective, this uniformity is precisely what allows algebraic operations to be interpreted as operations on suitable monads, in the sense of Plotkin and Power. Most of the operation symbols introduced above-such as those modeling partiality, exception *raising*, input and output, nondeterministic choice, or global state—fall within this algebraic framework and admit canonical interpretations as algebraic operations on appropriate monads. However, not all computational effects satisfy this algebraicity condition. A classical example of a monadic but non-algebraic effect is provided by continuations [86, 87], whose behavior inherently depends on the surrounding evaluation context and cannot be described by context-independent operations. Similarly, exception *handling* constitute an example of operations that are non-algebraic yet natural: while *raising* exceptions themselves can be modeled algebraically, exception *handlers* explicitly manipulate control flow and therefore fall outside the scope of purely algebraic effects.

The recognition of such non-algebraic phenomena motivated the development of more expressive semantic frameworks in which algebraic operations are complemented by additional structure. In particular, the theory of effects and handlers extends algebraic effects by introducing handlers

as specific algebraic morphisms that transform and control the flow of computations. Handlers provide a disciplined way of interpreting, intercepting, and re-routing effects, thereby accommodating non-algebraic control constructs within a coherent semantic setting [51, 97–99]. While handlers are not treated as first-class syntactic or semantic objects in this thesis, they nonetheless provide useful semantic intuition for some of the constructions considered later on. In particular, in Section 8.6, where state-passing computations are handled via infinite stacks, the operational manipulation of the stack can be viewed as playing a role analogous to that of a handler, in that it governs how effectful operations are interpreted and propagated. This perspective helps motivate the encodings employed there, without requiring an explicit treatment of handlers within the quantitative framework developed in the thesis.

5.4 Comodel-Based Operational Semantics

In [36], Plotkin and Power focused on developing a unified theory of structural operational semantics for effects, but their approach did not account for global state, or any combination of effects with global state. However, in [100], Plotkin and Power start to rectify the situation by taking into consideration the role played by *coalgebra* in the dynamics of stateful computations.

Definition 5.31 (Comodels of Algebraic Theories)**.** Let $\mathcal{T} = (\Sigma_{\mathcal{T}}, \mathrm{Eq}_{\mathcal{T}})$ be some algebraic theory. A $\Sigma_{\mathcal{T}}$*-coalgebra* $\mathtt{I}$ is a pair $(|\mathtt{I}|, I)$ between:

- A *carrier* set $|\mathtt{I}|$;
- A set of *cointerpretation* functions

$$I = \{[\![\mathtt{op}_i]\!]^{\mathtt{I}} : \mathcal{P}_i \times |\mathtt{I}| \to \mathcal{A}_i \times |\mathtt{I}| \mid \mathtt{op}_i : \mathcal{P}_i \rightsquigarrow \mathcal{A}_i \in \Sigma_{\mathcal{T}}\}.$$

Here, the term "coalgebra" is used in the sense of Plotkin and Power, to denote a family of cooperations indexed by the operation symbols of the signature, rather than a coalgebra for a single endofunctor.

$\Sigma_{\mathcal{T}}$-coalgebras can be extended to $\Sigma_{\mathcal{T}}$-trees. Let $\mathtt{t} \in \mathtt{Tree}_{\Sigma_{\mathcal{T}}} X$. Then, $[\![\mathtt{t}]\!]^{\mathtt{I}} : |\mathtt{I}| \to X \times |\mathtt{I}|$ is defined inductively over the structure of $\mathtt{t}$ as follows:

$$[\![\mathtt{return}\, x]\!]^{\mathtt{I}} := \lambda w.(x, w) \qquad [\![\mathtt{op}_i(p, \kappa)]\!]^{\mathtt{I}} := \lambda w.[\![\kappa\,(x)]\!]^{\mathtt{I}}(w') \text{ where } [\![\mathtt{op}_i]\!]^{\mathtt{I}}(p, w) = (x, w')$$

An equation $\mathtt{t} \sim_{\mathcal{T}} \mathtt{u} \in \mathrm{Eq}_{\mathcal{T}}$ is said to be valid for $\mathtt{I}$ whenever $[\![\mathtt{t}]\!]^{\mathtt{I}} = [\![\mathtt{u}]\!]^{\mathtt{I}}$ as functions $|\mathtt{I}| \mapsto X \times |\mathtt{I}|$. A $\mathcal{T}$*-coalgebra* (or $\mathcal{T}$-comodel) $\mathtt{W}$ is a $\Sigma_{\mathcal{T}}$-coalgebra that validates all the equations in $\mathrm{Eq}_{\mathcal{T}}$.

A key insight of [100] is that algebraic theories not only determine monads, but also comonads. Indeed, an algebraic theory $\mathcal{T}$ defines a comonad $\mathbb{D}$ whose D-coalgebras are essentially the same as $\mathcal{T}$-comodels. However, while each monad corresponds to a unique algebraic theory, the same comonad may correspond to different algebraic theories.

In the sequel of this thesis, we are going to extend the syntax of the calculi in Chapter 2 and Chapter 3 with algebraic operations. The above result allows us to extend the operational semantics accordingly. In fact, the operational semantics that we are going to adopt are based on the Comodel-based Transitions Systems (CTS) of [39]. In a CTS, the state is described by a comodel, and transitions happen between program-state pairs (configurations), reflecting how programs interact with and transform the state.

The following adaptation of a result by Plotkin and Power [100] to the category of sets and functions tells us how to compute the tensor of an arbitrary $\mathcal{T}$-comodel $\mathtt{W}$ of an arbitrary algebraic theory $\mathcal{T}$ with any free model $\mathtt{F}_{\mathcal{T}}X$ on a set X.

Proposition 5.32 (Theorem 4.4 of [100])**.** *For any algebraic theory $\mathcal{T}$, $\mathcal{T}$-comodel $\mathtt{W}$, and set X, the tensor $\mathtt{F}_{\mathcal{T}}X \otimes \mathtt{W}$ of $\mathtt{W}$ with the free model $\mathtt{F}_{\mathcal{T}}X$ is given by the product $X \times |\mathtt{W}|$.*

The cooperations of the tensor are obtained by acting on the second component according to the cooperations of $\mathtt{W}$, while leaving the X-component unchanged.

Let us work out what are the carriers of comodels of the algebraic theories mentioned so far. In the following examples, we will replace some sets with their obvious isomorphic versions.

Example 5.33 (Partiality)**.** *Let $\mathcal{M}$ be the algebraic theory for partiality. A comodel $\mathtt{W}$ for $\mathcal{M}$ is a set $|\mathtt{W}|$ together with the following function:*

$$[\![\mathtt{raise}]\!]^{\mathtt{W}} : 1 \times |\mathtt{W}| \to 0 \times |\mathtt{W}| \cong 0.$$

The above function is uninhabited, unless $|\mathtt{W}| = \emptyset$, in which case it has exactly one element. Thus, there are no (non-trivial) comodels.

Example 5.34 (Exceptions)**.** *Let $\mathcal{E}$ be the algebraic theory for exceptions. A comodel $\mathtt{W}$ for $\mathcal{E}$ is a set $|\mathtt{W}|$ together with the following function:*

$$[\![\mathtt{raise}]\!]^{\mathtt{W}} : \mathcal{E} \times |\mathtt{W}| \to 0 \times |\mathtt{W}| \cong 0.$$

Just like for partiality, the above function is uninhabited. Thus, the carrier must be the empty set and there are no (non-trivial) comodels.

Example 5.35 (Interactive Input)**.** *Let $\mathcal{I}$ be the algebraic theory for interactive input. A comodel $\mathtt{W}$ for $\mathcal{I}$ is a set $|\mathtt{W}|$ together with the following function:*

$$[\![\mathtt{input}]\!]^{\mathtt{W}} : 1 \times |\mathtt{W}| \to \mathcal{A} \times |\mathtt{W}|$$

Then, $|\mathtt{W}|$ is the set of streams $\mathcal{A}^\infty$ of characters of $\mathcal{A}$, equipped with the usual head-tail decompositions, and

$$\begin{array}{rcl} [\![\mathtt{input}]\!]^{\mathtt{W}} : 1 \times \mathcal{A}^\infty & \to & \mathcal{A} \times \mathcal{A}^\infty \\ (*, aw) & \mapsto & (a, w). \end{array}$$

Example 5.36 (Interactive Output)**.** *Let $\mathcal{O}$ be the algebraic theory for interactive output. A comodel $\mathtt{W}$ for $\mathcal{O}$ is a set $|\mathtt{W}|$ together with the following function:*

$$[\![\mathtt{output}]\!]^{\mathtt{W}} : \mathcal{A} \times |\mathtt{W}| \to 1 \times |\mathtt{W}|$$

Then, $|\mathtt{W}|$ is the set of streams $\mathcal{A}^\infty$ of characters of $\mathcal{A}$, equipped with the canonical coalgebra structure induced by prefix extension, and

$$\begin{array}{rcl} [\![\mathtt{output}]\!]^{\mathtt{W}} : \mathcal{A} \times \mathcal{A}^\infty & \to & 1 \times \mathcal{A}^\infty \\ (a, w) & \mapsto & (*, aw). \end{array}$$

Example 5.37 (Nondeterminism)**.** *Let $\mathcal{F}$ be the algebraic theory for finite non-empty non-determinism. A comodel $\mathtt{W}$ for $\mathcal{F}$ is a set $|\mathtt{W}|$ together with the following function:*

$$[\![\mathtt{choose}]\!]^{\mathtt{W}} : 1 \times |\mathtt{W}| \to \mathcal{B} \times |\mathtt{W}|$$

subject to some of the following equalities:

$$\begin{array}{rcl} [\![\mathtt{choose}(*, \lambda \mathtt{a}.\mathtt{t})]\!]^{\mathtt{W}} & = & [\![\mathtt{t}]\!]^{\mathtt{W}} \textit{ if } \mathtt{a} \notin \mathtt{fv}(\mathtt{t}) \\ [\![\mathtt{choose}(*, \lambda \mathtt{a}.\mathtt{choose}(*, \lambda \mathtt{b}.\mathtt{t}))]\!]^{\mathtt{W}} & = & [\![\mathtt{choose}(*, \lambda \mathtt{b}.\mathtt{choose}(*, \lambda \mathtt{a}.\mathtt{t}))]\!]^{\mathtt{W}} \\ [\![\mathtt{choose}(*, \lambda \mathtt{a}.\mathtt{t})]\!]^{\mathtt{W}} & = & [\![\mathtt{choose}(*, \lambda \mathtt{b}.\mathtt{t}\{\mathtt{a} \setminus \mathtt{not}\,\mathtt{b}\})]\!]^{\mathtt{W}} \end{array}$$

As it turns out, there is only a (non-trivial) comodel for the empty equational theory *over the nondeterministic signature [38]. In such a case, $|\mathtt{W}|$ is the set of boolean streams $\mathcal{B}^\infty$.*

$$\begin{array}{rcl} [\![\mathtt{choose}]\!]^{\mathtt{I}} : 1 \times \mathcal{B}^\infty & \to & \mathcal{B} \times \mathcal{B}^\infty \\ (*, bw) & \mapsto & (b, w). \end{array}$$

Example 5.38 (Global State)**.** *Let $\mathcal{G}$ be the algebraic theory for global state. A comodel $\mathtt{W}$ for $\mathcal{G}$ is a set $|\mathtt{W}|$ together with the following functions:*

$$[\![\mathtt{get}]\!]^{\mathtt{W}} : \mathcal{L} \times |\mathtt{W}| \to \mathcal{V} \times |\mathtt{W}|$$
$$[\![\mathtt{set}]\!]^{\mathtt{W}} : (\mathcal{L} \times \mathcal{V}) \times |\mathtt{W}| \to 1 \times |\mathtt{W}|$$

subject to the following equalities:

$$\begin{aligned}
[\![\mathtt{get}(l, \lambda \mathtt{a}.\mathtt{t})]\!]^{\mathtt{W}} &= [\![\mathtt{t}]\!]^{\mathtt{W}} \text{ if } \mathtt{a} \notin \mathtt{fv}(\mathtt{t}) \\
[\![\mathtt{get}(l, \lambda \mathtt{a}.\mathtt{set}((l, \mathtt{a}), \lambda \mathtt{b}.\mathtt{t}))]\!]^{\mathtt{W}} &= [\![\mathtt{t}]\!]^{\mathtt{W}} \text{ if } \mathtt{a} \notin \mathtt{fv}(\mathtt{t}) \\
[\![\mathtt{get}(l, \lambda \mathtt{b}.\mathtt{get}(l, \lambda \mathtt{a}.\mathtt{t}))]\!]^{\mathtt{W}} &= [\![\mathtt{get}(l, \lambda \mathtt{a}.\mathtt{t}\{\mathtt{b} \setminus \mathtt{a}\})]\!]^{\mathtt{W}} \\
[\![\mathtt{set}((l, \mathtt{v}), \lambda \mathtt{a}.\mathtt{get}(l, \lambda \mathtt{b}.\mathtt{t}))]\!]^{\mathtt{W}} &= [\![\mathtt{set}((l, \mathtt{v}), \lambda \mathtt{a}.\mathtt{t}\{\mathtt{b} \setminus \mathtt{v}\})]\!]^{\mathtt{W}} \\
[\![\mathtt{set}((l, \mathtt{v}), \lambda \mathtt{a}.\mathtt{set}((l, \mathtt{w}), \lambda \mathtt{b}.\mathtt{t}))]\!]^{\mathtt{W}} &= [\![\mathtt{set}((l, \mathtt{w}), \lambda \mathtt{b}.\mathtt{t})]\!]^{\mathtt{W}} \\
[\![\mathtt{get}(l, \lambda \mathtt{a}.\mathtt{get}(l', \lambda \mathtt{b}.\mathtt{t}))]\!]^{\mathtt{W}} &= [\![\mathtt{get}(l', \lambda \mathtt{b}.\mathtt{get}(l, \lambda \mathtt{a}.\mathtt{t}))]\!]^{\mathtt{W}} \\
[\![\mathtt{get}(l, \lambda \mathtt{a}.\mathtt{set}((l', \mathtt{v}), \lambda \mathtt{b}.\mathtt{t}))]\!]^{\mathtt{W}} &= [\![\mathtt{set}((l', \mathtt{v}), \lambda \mathtt{b}.\mathtt{get}(l, \lambda \mathtt{a}.\mathtt{t}))]\!]^{\mathtt{W}} \\
[\![\mathtt{set}((l, \mathtt{v}), \lambda \mathtt{a}.\mathtt{set}((l', \mathtt{w}), \lambda \mathtt{b}.\mathtt{t}))]\!]^{\mathtt{W}} &= [\![\mathtt{set}((l', \mathtt{w}), \lambda \mathtt{b}.\mathtt{set}((l, \mathtt{v}), \lambda \mathtt{a}.\mathtt{t}))]\!]^{\mathtt{W}}
\end{aligned}$$

Then, $|\mathtt{W}|$ *is a set of states* $\mathcal{S} = \mathcal{V}^{\mathcal{L}}$*, and*

$$\begin{aligned}
[\![\mathtt{get}]\!]^{\mathtt{W}} : \mathcal{L} \times \mathcal{S} &\to \mathcal{V} \times \mathcal{S} \\
(l, s) &\mapsto (s(l), s), \\
[\![\mathtt{set}]\!]^{\mathtt{W}} : (\mathcal{L} \times \mathcal{V}) \times \mathcal{S} &\to 1 \times \mathcal{S} \\
((l, \mathtt{v}), s) &\mapsto (*, s\{l := \mathtt{v}\}).
\end{aligned}$$

Remark 5.39 (Comodel-Based Transition Systems). If one understands the elements of the free $\mathcal{T}$-model $\mathtt{F}_{\mathcal{T}}X$ over a set X as effectful computations and the elements of a $\mathcal{T}$-comodel $\mathtt{W}$ as runtime environments, the tensor $\mathtt{F}_{\mathcal{T}}\mathrm{X} \otimes \mathtt{W}$ represents the interaction between $\mathtt{F}_{\mathcal{T}}X$ and $\mathtt{W}$. Indeed, $\mathtt{F}_{\mathcal{T}}\mathrm{X} \otimes \mathtt{W}$ induces the following equivalence relation over the product set $|\mathtt{F}_{\mathcal{T}}X| \times |\mathtt{W}|$:

$$([\![\mathtt{op}_i]\!]_{\mathtt{F}_{\mathcal{T}}X}(p, \kappa), w) \sim (\kappa\, a, w') \text{ where } [\![\mathtt{op}_i]\!]^{\mathtt{W}}(p, \kappa) = (a, w').$$

The above equivalence has an operational reading when read left-to-right: to perform the operation $[\![\mathtt{op}_i]\!]_{\mathtt{F}_{\mathcal{T}}X}(p, \kappa)$ in the runtime environment w, first run the corresponding cooperation $[\![\mathtt{op}_i]\!]^{\mathtt{W}}(p, w)$ to obtain $a \in \mathcal{A}_i$ and the next runtime environment w', then continue by executing $\kappa\, a$ in w'.

Moreover, this perspective extends to combinations of effects, where the runtime environment must simultaneously support several theories, *e.g.* both input and output. Recall from [89] that two algebraic theories $\mathcal{T}$ and $\mathcal{T}'$ admit two canonical combinations, both with signature the disjoint union $\Sigma_{\mathcal{T}} \cup \Sigma_{\mathcal{T}'}$. The *sum* $\mathcal{T} + \mathcal{T}'$ has equations $\mathrm{Eq}_{\mathcal{T}} \cup \mathrm{Eq}_{\mathcal{T}'}$, imposing no interaction between the two theories. The *tensor* $\mathcal{T} \otimes \mathcal{T}'$ additionally requires every operation of $\mathcal{T}$ to commute with every operation of $\mathcal{T}'$, *i.e.* its equations are those of the sum together with the schema

$$\mathtt{op}(p, \lambda \mathtt{a}.\mathtt{op}'(p', \lambda \mathtt{b}.\mathtt{t})) \sim_{\mathcal{T} \otimes \mathcal{T}'} \mathtt{op}'(p', \lambda \mathtt{b}.\mathtt{op}(p, \lambda \mathtt{a}.\mathtt{t}))$$

for all $\mathtt{op} : \mathcal{P} \rightsquigarrow \mathcal{A} \in \Sigma_{\mathcal{T}}$, $\mathtt{op}' : \mathcal{P}' \rightsquigarrow \mathcal{A}' \in \Sigma_{\mathcal{T}'}$, $p \in \mathcal{P}$, and $p' \in \mathcal{P}'$ (the cross-location equations

of Example 5.19 have exactly this shape). The following adaptation of results by Plotkin and Power [100] to the category of sets and functions tells us how to combine comodels along either combination; in both cases, the computation of the tensor with the free model is just Proposition 5.32 instantiated at the combined theory.

Proposition 5.40 (*cf.* Theorem 4.4 and Corollary 6.1 of [100]). *Let $\mathcal{T}$ and $\mathcal{T}'$ be algebraic theories with $\mathcal{T}$-comodel $\mathtt{W}$ and $\mathcal{T}'$-comodel $\mathtt{W}'$, and let $\mathcal{T}''$ be either the sum $\mathcal{T}+\mathcal{T}'$ or the tensor $\mathcal{T}\otimes\mathcal{T}'$. The* composition $\mathtt{W}\circ\mathtt{W}'$ *of the two comodels,* i.e. *the set $|\mathtt{W}'|\times|\mathtt{W}|$ whose $\Sigma_{\mathcal{T}}$-cooperations act on the second component and whose $\Sigma_{\mathcal{T}'}$-cooperations act on the first, is a $\mathcal{T}''$-comodel. Consequently, if $\mathtt{F}_{\mathcal{T}''}X$ is the free $\mathcal{T}''$-model on a set X, the tensor $\mathtt{F}_{\mathcal{T}''}X\otimes(\mathtt{W}\circ\mathtt{W}')$ is given by the triple $X\times|\mathtt{W}'|\times|\mathtt{W}|$.*

For the sum, the composition is merely the canonical choice: any set equipped with both a $\mathcal{T}$-comodel and a $\mathcal{T}'$-comodel structure is a $\mathcal{T}+\mathcal{T}'$-comodel, since no interaction between the two families of cooperations is required.

We are going to define our CTS transition system as a function $X\times|\mathtt{W}|\to X\times|\mathtt{W}|$, between pairs of programs of type X and the carrier of some $\mathcal{T}$-comodel $\mathtt{W}$, whose algebraic theory $\mathcal{T}$ corresponds to some monad $\mathbb{T}$. Moreover, by currying $X\times|\mathtt{W}|\to X\times|\mathtt{W}|$ to get $X\to|\mathtt{W}|\to(X\times|\mathtt{W}|)$, we can immediately see that a CTS transition system can also be understood as a $\mathbb{G}$-coalgebra $X\to G(X)$. In this sense, CTSs provide a coalgebraic presentation of effectful operational semantics. This also applies to combinations of comodels, as per the above proposition.

In sum, after extending the syntax of the λ- and $\lambda!$-calculi with algebraic operations, we extend their operational semantics as follows. For each new algebraic operation $\mathsf{op}_i(p,\kappa)$, we add a reduction step based on its cointerpretation:

$$(\mathsf{op}_i(p,\kappa),w)\mapsto_{\mathsf{op}_i}(\kappa\,x,w')\text{ where }[\![\mathsf{op}_i]\!]^{\mathtt{W}}(p,w)=(x,w'),$$

where x is the result produced by the environment.

Having established the theoretical framework underlying this thesis—from the λ-calculus and its evaluation strategies to non-idempotent intersection types and the categorical background for algebraic effects—we will apply this machinery to concrete effectful systems in subsequent chapters.

Part III

Pattern-Matching Computations

Chapter 6

A Quantitative Study of Pattern-Matching

This chapter extends the pattern calculi in [28–30] with tagged products and case expressions, which capture Abstract Data Types (ADTs). The calculus that is obtained, called the λ_{PM}-calculus, models functional programming languages more closely. The main contributions of this chapter are the following. A detailed operational treatment of the λ_{PM}-calculus in Section 6.1; a purely syntactical characterization of *clash-free* (*i.e.* error-free) normal forms; and that, not only are we able to encode exceptions using the λ_{PM}-calculus, but also that it can act as a subsuming framework for the CBN and CBV λ-calculi [62]. The fact that the λ_{PM}-calculus can encode exceptions might seem like a simple detail, but it is an important one. As was discussed in Chapter 5, *raising exceptions* is *algebraic*, but *catching exceptions* is *not*. Therefore, while the calculi that we are going to deal with in Chapters 7 and 8 can (and will be) used to study *state-passing* algebraic effects (*i.e.* algebraic effects with non-trivial comodels), the λ_{PM}-calculus can then be used to study exceptions. Ultimately, we could have used the λ_{PM}-calculus as the base calculus for the study of state-passing algebraic effects in order to study both kinds of effects via a single unified framework. However, we have decided instead to base Chapters 7 and 8 on the $\lambda!$-calculus, even though it does not seem to be able to encode exceptions, since it also encodes the CBN and CBV λ-calculi, but has a much simpler syntax. However, it is indeed possible to study state-passing algebraic effects and exceptions by replacing the $\lambda!$-calculus with the λ_{PM}-calculus in Chapters 7 and 8. Finally, we define a non-idempotent intersection type system for pattern matching in Section 6.2 that not only extends the type system first presented in [28] to more general pattern-matching constructs, but also refines the type system in [101] by characterizing the length of clash-free normalization sequences (*i.e.* normalization sequences that do not end in clashes) in a *precise* (*i.e. exact*) manner: the soundness and completeness of our type system with respect to clash-free normalization not

only establishes an equivalence between typability and clash-free normalization, but also provides the exact length of the clash-free normalization sequence. All proofs for this chapter are collected in Appendix D.

6.1 The Pattern-Matching Calculus

Before introducing the syntax of the pattern-matching λ_{PM}-calculus, we clarify some basic notation related to *tagged products*.

Tagged Products. Let $\mathtt{c}$ be a *tag* taken out of an enumerable set of tags $\mathtt{C}$. A *constructor* $\mathtt{c}^n$ is built by assigning a unique *arity* $n \in \mathbb{N}$ to a each *tag* $\mathtt{c}$. Since each tag has a *unique* arity, constructors are identified uniquely by their tags. Constructors cannot be applied partially: a constructor $\mathtt{c}^n$ *must* be applied to n arguments. Accordingly, the application $\mathtt{c}^n\ a_1 \cdots a_n$ will be written $\mathtt{c}^n(a_1, \ldots, a_n)$ and called a *tagged product*, where the sequence of arguments $a_1 \cdots a_n$ is represented as a *tuple* (or *product* or *vector*) of arguments $(a_1, \ldots, a_n)$ of length n. It is possible to take advantage of the tuple notation and simply write $(a_1, \ldots, a_n)$ as $(a_i)_n$ (meaning that $1 \leq i \leq n$) in order to keep notation light. Tagged products represent *data* and will thus be referred to as such throughout the rest of this work. Moreover, the arity of constructors will always be left implicit, and all data is assumed to be well-formed (constructors are always applied to tuples of arguments of the appropriate length). Now, we can proceed with the syntax and operational semantics for the λ_{PM}-calculus.

6.1.1 Syntax and Operational Semantics

The syntax of the λ_{PM}-calculus is based on the λ-calculus, but it extends it with patterns and case expression.

Definition 6.1 (λ-Terms)**.** Let $\mathcal{X}$ be a countable set of *variables*. The set of *term* of the λ_{PM}-calculus is denoted by Λ^{PM} and generated by the following grammars:

$$\begin{array}{lrcl} \text{Patterns:} & p, q & ::= & x \mid \mathtt{c}(p_i)_n \\ \text{Branches:} & b & ::= & \mathtt{c}(p_i)_n.t \\ \text{Terms:} & t, u, r, e & ::= & x \mid \lambda p.t \mid t\ u \mid t[p \backslash u] \mid \mathtt{c}(t_i)_n \mid \mathtt{case}\ t\ \mathtt{of}\ (b_1, \ldots, b_n) \end{array}$$

Let Λ^{PM} denote the set of terms of the λ_{PM}-calculus. In order to keep notation light, symbols P, B, and T are going to be used to denote *tuples of patterns*, *tuples of branches*, and *tuples of terms*, respectively. The pattern x is called a *variable pattern*, and $\mathtt{c}P$ is a *data pattern*. The term $\lambda p.t$ is a *generalized abstraction*, $t[p \backslash u]$ is a *matching closure* (where $[p \backslash u]$ is a called an *explicit matching operation*), $\mathtt{c}T$ is a *data term*, and $\mathtt{case}\ t\ \mathtt{of}\ B$ is a *case expression*. Remark in particular that the explicit

matching operator is a new constructor in the language, and it is not on the meta-level. We write $\mathtt{I}$ as a special term representing the identity function $\lambda x.x$.

Terms can be surrounded by list contexts, which are sequences of explicit matching operations.

Definition 6.2 (List Contexts)**.** The set of *list contexts* for the λ_{PM}-calculus is generated by the following grammar:

$$\text{List Contexts:} \quad \mathtt{L} \quad ::= \quad \Box \mid \mathtt{L}[p\backslash t].$$

Indeed, given a list context $\mathtt{L}$ and a term t, $\mathtt{L}(\!|t|\!)$ denotes the term obtained by *plugging the term t inside context* $\mathtt{L}$, *i.e.* by replacing the unique occurrence of $\Box$ in $\mathtt{L}$ with t (possibly allowing the capture of free variables of t). Thus, *e.g.* if $\mathtt{L} = \Box[x_1\backslash u_1][x_2\backslash u_2]$, then $\mathtt{L}(\!|x_1|\!)$ is the term $x_1[x_1\backslash u_1][x_2\backslash u_2]$. List contexts are important as they allow matching operations to be postponed, thus exposing redexes that might otherwise stay hidden due to premature normal forms. We use three special predicates to distinguish terms possibly affected by a list of explicit matching operators: $\mathtt{isabs}(t)$ if and only if $t = \mathtt{L}(\!|\lambda p.u|\!)$; $\mathtt{iscase}(t)$ if and only if $t = \mathtt{L}(\!|\mathtt{case}\ u\ \mathtt{of}\ B|\!)$; and $\mathtt{isdata}_{\mathtt{c}}(t)$ if and only if $t = \mathtt{L}(\!|\mathtt{c}T|\!)$, in which case we may also write $\mathtt{isdata}(t)$ whenever $\mathtt{c}$ is irrelevant.

Definition 6.3 (Free and Bound Variables)**.** The *set of variables* of a pattern p is denoted by $\mathtt{vars}(p)$ and defined as expected. It will be useful to write $\hat{p}$ to denote *data patterns*, so that $\mathtt{case}\ t\ \mathtt{of}\ (\mathtt{c}_i P_i.u_i)_n$ can be written as $\mathtt{case}\ t\ \mathtt{of}\ (\hat{p}_i.u_i)_n$, where $\hat{p}_i = \mathtt{c}_i P_i$. The *sets of free and bound variables* of branches, terms and list contexts are also defined as expected. In particular:

$$\begin{array}{rcl}
\mathtt{fv}(\lambda p.t) & := & \mathtt{fv}(t) \setminus \mathtt{vars}(p) \\
\mathtt{fv}(t[p\backslash u]) & := & (\mathtt{fv}(t) \setminus \mathtt{vars}(p)) \cup \mathtt{fv}(u) \\
\mathtt{fv}(\mathtt{case}\ t\ \mathtt{of}\ (\hat{p}_i.u_i)_n) & := & \mathtt{fv}(t) \bigcup_{i\in\{1,\dots,n\}} (\mathtt{fv}(u_i) \setminus \mathtt{vars}(\hat{p}_i)) \\
\mathtt{bv}(\mathtt{L}[p\backslash t]) & := & \mathtt{bv}(\mathtt{L}) \cup \mathtt{vars}(p)
\end{array}$$

Linear Patterns. Patterns are assumed to be *linear*, *i.e.* each variable occurs at most once, and so are tuples of patterns, *i.e.* no variable is shared between different patterns in the same tuple. Most modern programming languages with pattern matching implement linear patterns since this does not affect the expressivity of the language, while avoiding to check for equality of terms.

Case Expressions. As mentioned in the introduction, case expressions capture the pattern-matching mechanism over tagged unions. Note that each branch of a case expression is built out of a tagged product and a term. Therefore, we require all tagged products to have a different top-level tag, thus branches are disjoint and matching is thus unambiguous. As such, case expressions can indeed be understood as *deconstructors* for *tagged unions*. As an example, the following

case expression is a function whose behavior depends on whether its argument is a *pair* or a *triple*, like in $\lambda x.\mathtt{case}\ x\ \mathtt{of}\ (\mathtt{pair}(x,y).y, \mathtt{triple}(x,y,z).x)$.

We are going to base the operational semantics of the λ_{PM}-calculus on a *reduction relation* with the following selective lazy behavior: generalized abstractions built from *variable patterns*, applications are going to behave like in CBN, *i.e.* arguments are never evaluated before they are plugged into a term; generalized abstractions built from *data patterns* and case expressions need to be (partially) evaluated to some data in order to satisfy the matching conditions.

Definition 6.4 (Weak Head Contexts)**.** The set of *weak head contexts* of the λ_{PM}-calculus is generated by the following grammar:

$$\mathtt{WH} \quad ::= \quad \Box \mid \mathtt{WH}\, t \mid \mathtt{WH}[p \backslash u] \mid t[\hat{p} \backslash \mathtt{WH}] \mid \mathtt{case}\ \mathtt{WH}\ \mathtt{of}\ B$$

Definition 6.5 (Weak Head Reduction)**.** The *weak head reduction relation* is denoted by $\to_{\mathtt{WH}}$ and defined over open terms as the closure by *weak head contexts* of the following rewriting steps:

$$\begin{array}{rcll}
\mathtt{L}(\!|\lambda p.t|\!)u & \mapsto_{\mathtt{dB}} & \mathtt{L}(\!|t[p\backslash u]|\!) & \text{if } \mathtt{bv}(\mathtt{L}) \cap \mathtt{fv}(u) = \emptyset \\
\mathtt{case}\ \mathtt{L}(\!|\mathtt{c}(u_i)_n|\!)\ \mathtt{of}\ (\ldots, \mathtt{c}(p_i)_n.t, \ldots) & \mapsto_{\mathtt{branch}} & \mathtt{L}(\!|t[p_1\backslash u_1]\cdots[p_n\backslash u_n]|\!) & \text{if } \mathtt{bv}(\mathtt{L}) \cap \mathtt{fv}(t) = \emptyset \\
t[\mathtt{c}(p_i)_n \backslash \mathtt{L}(\!|\mathtt{c}(u_i)_n|\!)] & \mapsto_{\mathtt{match}} & \mathtt{L}(\!|t[p_1\backslash u_1]\cdots[p_n\backslash u_n]|\!) & \text{if } \mathtt{bv}(\mathtt{L}) \cap \mathtt{fv}(t) = \emptyset \\
t[x\backslash u] & \mapsto_{\mathtt{sub}} & t\{x \setminus u\} &
\end{array}$$

Notice that the dB-step is based on the usual β-step of λ-calculi, but substitutions are not performed immediately. Instead, they are carried explicitly in the syntax and the substitution is performed via sub-steps, which produce a *capture avoiding* substitution as a result of solving a matching between a variable pattern and a term. Also, since matchings and substitutions can be delayed, all terms are formally plugged into a (possibly empty) *list context*. Then, branch- and match-steps solve (top-level) matches between data patterns and data terms, respectively. In particular, this means that abstractions can be separated from their arguments by an arbitrary amount of explicit substitutions. Thus, we say that dB-steps are performed *at a distance*. Finally, it is worth noticing that the side conditions in these three rules are necessary in order to avoid the capture of free variables. We implicitly work modulo α-equivalence, and we assume bound variables in list contexts and terms are always renamed apart so that the side conditions of reduction rules are satisfied whenever possible.

Example 6.6 (Weak Head Reduction). *Let* $\mathtt{c}_0, \mathtt{c}_1, \mathtt{c}_2 \in \mathtt{C}$. *Then*

$$
\begin{array}{ll}
 & t = (\lambda x.\mathtt{case}\ x\ \mathtt{of}\ (\mathtt{pair}(x,y).y, \mathtt{triple}(x,y,z).x))\ \mathtt{triple}(\mathtt{c}_0,\mathtt{c}_1,\mathtt{c}_2) \\
\to_{\mathtt{WH}(\mathtt{dB})} & \mathtt{case}\ x\ \mathtt{of}\ (\mathtt{pair}(x,y).y, \mathtt{triple}(x,y,z).x)[x\backslash \mathtt{triple}(\mathtt{c}_0,\mathtt{c}_1,\mathtt{c}_2)] \\
\to_{\mathtt{WH}(\mathtt{sub})} & \mathtt{case}\ \mathtt{triple}(\mathtt{c}_0,\mathtt{c}_1,\mathtt{c}_2)\ \mathtt{of}\ (\mathtt{pair}(x,y).y, \mathtt{triple}(x,y,z).x) \\
\to_{\mathtt{WH}(\mathtt{branch})} & x[x\backslash \mathtt{c}_0][y\backslash \mathtt{c}_1][z\backslash \mathtt{c}_2] \\
\to_{\mathtt{WH}(\mathtt{sub})} & x[x\backslash \mathtt{c}_0][y\backslash \mathtt{c}_1] \\
\to_{\mathtt{WH}(\mathtt{sub})} & x[x\backslash \mathtt{c}_0] \\
\to_{\mathtt{WH}(\mathtt{sub})} & \mathtt{c}_0.
\end{array}
$$

Notice that relation $\to_{\mathtt{WH}}$ is not deterministic. Indeed, the λ-term in Example 6.6 also has the following reduction sequence, which differs from the one in Example 6.6 after the branch-step, by applying sub-steps right-to-left, they are applied left-to-right:

$$t \overset{1\cdot \mathtt{dB}+1\cdot \mathtt{branch}+1\cdot \mathtt{sub}}{\to\!\!\!\!\to_{\mathtt{WH}}} x[x\backslash \mathtt{c}_0][y\backslash \mathtt{c}_1][z\backslash \mathtt{c}_2] \to_{\mathtt{WH}(\mathtt{sub})} \mathtt{c}_0[y\backslash \mathtt{c}_1][z\backslash \mathtt{c}_2] \to_{\mathtt{WH}(\mathtt{sub})} \mathtt{c}_0[z\backslash \mathtt{c}_2] \to_{\mathtt{WH}(\mathtt{sub})} \mathtt{c}_0.$$

Therefore, we are going to define the operational semantics of the λ_{PM}-calculus by fixing a particular evaluation strategy for $\to_{\mathtt{WH}}$ (see Proposition 6.11). Despite nondeterminism $\to_{\mathtt{WH}}$ is is confluent (see Lemma 6.9).

Definition 6.7 (One-Step Diamond Property)**.** A reduction relation $\to_{\mathtt{WH}}$ is said to enjoy the *one-step diamond property* if for every term $t \in \Lambda^{PM}$, if $u\ {}_{\mathtt{WH}}\!\leftarrow t \to_{\mathtt{WH}} r$ with $u \neq r$, there exists a term $e \in \Lambda^{PM}$, such that $u \to_{\mathtt{WH}} e\ {}_{\mathtt{WH}}\!\leftarrow r$.

Lemma 6.8 (One-Step Diamond)**.** *The reduction relation* $\to_{\mathtt{WH}}$ *enjoys the one-step diamond property.*

As a corollary we obtain:

Corollary 6.9 (Confluence)**.** *The reduction relation* $\to_{\mathtt{WH}}$ *is confluent.*

Weak Head Evaluation. As we have discussed in Section 2.1, one of the the difference between reduction *relations* and evaluation *strategies* is that, while the former are (usually) *not* deterministic, the latter *are* (usually) deterministic, *i.e.* given some evaluation strategy, if some term t is not a normal form, only one reduction can apply to t. Therefore, we define the operational semantics for λ_{PM}-calculus in Definition 6.10 as a particular choice of strategy for the weak head reduction relation in Definition 6.5.

Definition 6.10 (Deterministic Weak Head Relation)**.** The *deterministic* weak head reduction relation is denoted by $\to_{\mathtt{PM}}$ and defined by the following set of reduction rules:

$$\frac{\mathtt{bv}(\mathtt{L}) \cap \mathtt{fv}(u) = \emptyset}{\mathtt{L}(\!|\lambda p.t|\!)u \to_{\mathtt{PM}} \mathtt{L}(\!|t[p\backslash u]|\!)}\ (\mathtt{dB}) \qquad \frac{\neg\mathtt{isabs}(t) \quad t \to_{\mathtt{PM}} t'}{tu \to_{\mathtt{PM}} t'u}\ (\mathtt{appL})$$

$$\frac{\mathtt{bv}(\mathtt{L}) \cap \mathtt{fv}(t) = \emptyset}{t[\mathtt{c}(p_i)_n\backslash\mathtt{L}(\!|\mathtt{c}(u_i)_n|\!)] \to_{\mathtt{PM}} \mathtt{L}(\!|t[p_1\backslash u_1]\cdots[p_n\backslash u_n]|\!)}\ (\mathtt{match}) \qquad \frac{}{t[x\backslash u] \to_{\mathtt{PM}} t\{x \setminus u\}}\ (\mathtt{sub})$$

$$\frac{\neg\mathtt{isdata}_{\mathtt{c}}(u) \quad t \to_{\mathtt{PM}} t'}{t[\mathtt{c}P\backslash u] \to_{\mathtt{PM}} t'[\mathtt{c}P\backslash u]}\ (\mathtt{esL}) \qquad \frac{\neg\mathtt{isdata}_{\mathtt{c}}(u) \quad t \not\to_{\mathtt{PM}} \quad u \to_{\mathtt{PM}} u'}{t[\mathtt{c}P\backslash u] \to_{\mathtt{PM}} t[\mathtt{c}P\backslash u']}\ (\mathtt{esR})$$

$$\frac{\mathtt{bv}(\mathtt{L}) \cap \mathtt{fv}(t) = \emptyset}{\mathtt{case}\ \mathtt{L}(\!|\mathtt{c}(u_i)_n|\!)\ \mathtt{of}\ (\ldots,\mathtt{c}(p_i)_n.t,\ldots) \to_{\mathtt{PM}} \mathtt{L}(\!|t[p_1\backslash u_1]\cdots[p_n\backslash u_n]|\!)}\ (\mathtt{branch})$$

$$\frac{t \to_{\mathtt{PM}} t' \quad \neg\mathtt{isdata}_{\mathtt{c}_i}(t) \text{ for all } i \in \{1,\ldots,n\}}{\mathtt{case}\ t\ \mathtt{of}\ (\mathtt{c}_iP_i.u_i)_n \to_{\mathtt{PM}} \mathtt{case}\ t'\ \mathtt{of}\ (\mathtt{c}_iP_i.u_i)_n}\ (\mathtt{caseArg})$$

Proposition 6.11 (Determinism)**.** *The reduction relation $\to_{\mathtt{PM}}$ is deterministic.*

Note that $\to_{\mathtt{PM}}$ can also be defined as the closure, by a particular subset of the weak head contexts in Definition 6.4, of the reduction steps in Definition 6.5. However, given that some of the rules have very specific conditions, we prefer the presentation above instead. Moreover, it is not difficult to show that $\to_{\mathtt{PM}}$ has the same set of normal forms as $\to_{\mathtt{WH}}$.

In order to syntactically describe the set of irreducible forms for $\to_{\mathtt{PM}}$ (and thus for $\to_{\mathtt{WH}}$), it is necessary to place special care in the treatment of *stuck matchings*. The latter can arise in two ways: (1) from rule (dB) whenever the abstraction is built from a data pattern; and (2) from rule (branch). Consider the following two examples:

$$(\lambda\mathtt{pair}(x,y).y)(\mathtt{duo}(t,u)) \to_{\mathtt{dB}} y[\mathtt{pair}(x,y)\backslash\mathtt{duo}(t,u)] \not\to_{\mathtt{match}} \qquad (1)$$

$$\mathtt{case}\ \mathtt{duo}(t,u)\ \mathtt{of}\ (\mathtt{one}(x).x, \mathtt{pair}(x,y).y) \not\to_{\mathtt{branch}} \qquad (2)$$

In both cases, evaluation is *stuck* due to a stuck matching. Clearly, these irreducible terms are not the desired results of computations. Moreover, the same is true if $\mathtt{duo}(t,u)$ is replaced with an abstraction. These kinds of situations will be dealt with later by introducing the notion of *clash* and the set of *clash-free* normal forms (see Lemma 6.19). Still, to provide a syntactical description of irreducible terms, these situations need to be considered. Note that stuck matchings that result from matching a data pattern with a different constructor (in the examples above, data $\mathtt{duo}(t,u)$ is matched against patterns $\mathtt{pair}(x,y)$ and $\mathtt{one}(x)$) can be described by: (1) inspecting the matching operation $[\hat{p}\backslash t]$ in a matching closure $u[\hat{p}\backslash t]$, and checking that whenever t is data, it has the same constructor as $\hat{p}$; (2) inspecting t in a case expression $\mathtt{case}\ t\ \mathtt{of}\ (b_1,\ldots,b_n)$, and checking that whenever t is data, it has the same constructor as the data pattern of one of the branches $b_1,\ldots,b_n$. Naturally, different data patterns in closure matchings and sets of data patterns in

vectors of branches generate different sets of stuck matching. Thus, the set of irreducible terms of the λ_{PM}-calculus is the union over the sets of normal forms with respect to a particular constructor $\mathtt{c} \in \mathtt{C}$. To capture that, we define three categories of terms: *neutral*, *neutral data*, and *normal*. The set of *neutral forms* is written $\mathtt{NE}$ and does not depend on any constructor.

Irreducibility in the presence of pattern matching depends on the constructor against which matching is attempted; for this reason, normal forms are indexed by constructors.

Definition 6.12 (Indexed Normal Forms)**.** Let $\mathtt{C}' \subseteq \mathtt{C}$. Moreover, let us define $\mathtt{NO}_{\neg \mathtt{C}'} := \{t \in \mathtt{NO}_{\mathtt{c}} \mid \mathtt{c} \in \mathtt{C} \setminus \mathtt{C}'\}$ and $\mathtt{WH}(\!|\mathtt{NO}_{\neg \mathtt{C}'}|\!) := \{\mathtt{WH}(\!|t|\!) \mid t \in \mathtt{NO}_{\neg \mathtt{C}'}\}$. Then, the set of $\mathtt{PM}$-normal forms $\mathtt{NO}_{\mathtt{c}}$ is generated by the following grammars:

$$
\begin{array}{llcl}
\textbf{(Normal)} & \mathtt{NO}_{\mathtt{c}} & ::= & \mathtt{NA}_{\mathtt{c}} \mid \lambda p.t \mid \mathtt{NO}_{\mathtt{c}}[\mathtt{c}'P \backslash \mathtt{NO}_{\neg \mathtt{c}'}] \\
\textbf{(Neutral or Constants)} & \mathtt{NA}_{\mathtt{c}} & ::= & \mathtt{NE} \mid \mathtt{c}T \mid \mathtt{NA}_{\mathtt{c}}[\mathtt{c}'P \backslash \mathtt{NO}_{\neg \mathtt{c}'}] \\
\textbf{(Neutral)} & \mathtt{NE} & ::= & x \mid \mathtt{NA}\ t \mid \mathtt{NE}[\mathtt{c}'P \backslash \mathtt{NO}_{\neg \mathtt{c}'}] \mid \mathtt{case}\ \mathtt{NO}_{\neg \{\mathtt{c}_1, \ldots, \mathtt{c}_n\}}\ \mathtt{of}\ (\mathtt{c}_i P_i.u_i)_n
\end{array}
$$

and $\mathtt{NO}_{\neg \mathtt{c}'}$ is written for $\mathtt{NO}_{\neg \{\mathtt{c}'\}}$.

Intuitively, $t \in \mathtt{NO}_{\mathtt{c}}$ if and only if t is an irreducible term, and either $\neg\mathtt{isdata}(t)$ or $\mathtt{isdata}_{\mathtt{c}}(t)$. The set of neutral forms $\mathtt{NE}$ corresponds to irreducible terms that do not produce any redexes whenever plugged into any context $\mathtt{WH}$, *i.e.* the set of terms $t \in \mathtt{NO}_{\mathtt{c}}$, such that $\neg\mathtt{isabs}(t)$ and $\neg\mathtt{isdata}(t)$ hold. The set $\mathtt{NA}_{\mathtt{c}}$ is the set of irreducible terms (with respect to $\mathtt{c}$) that do not produce any redexes whenever plugged into a context of the form $\mathtt{WH}\ t$, *i.e.* the set of terms $t \in \mathtt{NO}_{\mathtt{c}}$, such that $\neg\mathtt{isabs}(t)$ holds.

Definition 6.13 (Normal Forms)**.** The *set of normal (resp. neutral data) forms* is written $\mathtt{NO}$ (resp. $\mathtt{NA}$) and defined as $\mathtt{NO} := \bigcup_{\mathtt{c} \in \mathtt{C}} \mathtt{NO}_{\mathtt{c}}$ (resp. $\mathtt{NA} := \bigcup_{\mathtt{c} \in \mathtt{C}} \mathtt{NA}_{\mathtt{c}}$), where $\mathtt{NO}_{\mathtt{c}}$ (resp. $\mathtt{NA}_{\mathtt{c}}$) is the *set of normal (resp. neutral data) forms with respect to* $\mathtt{c} \in \mathtt{C}$, as defined in Definition 6.12.

Example 6.14 (Syntactic Normal Forms)**.** *The following terms are* $\mathtt{PM}$*-normal forms:*

$$
\begin{array}{cc}
\mathtt{case}\ \mathtt{duo}(\mathtt{I},\mathtt{I})\ \mathtt{of}\ (\mathtt{pair}(x,y).y) \in \mathtt{NE} & \mathtt{case}\ \mathtt{I}\ \mathtt{of}\ (\mathtt{pair}(x,y).y) \in \mathtt{NE} \\
\mathtt{pair}(\mathtt{I}\,\mathtt{I},\mathtt{I}) \in \mathtt{NO}_{\mathtt{pair}} \subseteq \mathtt{NO} & \mathtt{pair}(\mathtt{II},\mathtt{I})\ \mathtt{I} \in \mathtt{NO}
\end{array}
$$

In particular, note that $\mathtt{duo}(\mathtt{I},\mathtt{I}) \in \mathtt{NO}_{\mathtt{duo}} \subseteq \mathtt{NO}_{\neg\mathtt{pair}}$ *and* $\mathtt{I} \in \mathtt{NO}_{\neg\mathtt{pair}}$.

Set $\mathtt{NO}$ precisely captures the notion of irreducible term as follows.

Lemma 6.15 (Characterization of Normal Forms)**.** *Let* $t \in \Lambda^{PM}$*. Then,* $t \in \mathtt{NO}$ *if and only if* $t \not\to_{\mathtt{PM}}$.

Note that the set of normal forms was defined for *open* terms. This is a technical detail that allows induction to go through smoothly in the proofs. The set of normal forms for *closed* terms (programs) is obtained by removing x from the definition of normal forms and assuming every term to be closed.

Similarly to what happened for the $\lambda!$-calculus (see Chapter 3), some $\mathtt{PM}$-normal forms are operationally ill-formed.

Definition 6.16 (Clashes)**.** The set of *clashes* for the λ_{PM}-calculus is generated by the following grammars:

$$\begin{array}{rrcl} \text{Clashes:} & \mathtt{CL} & ::= & \mathtt{CL}^0 \mid \mathtt{CL}\ t \mid \mathtt{CL}[p\backslash u] \mid t[p\backslash \mathtt{CL}] \mid \mathtt{case}\ \mathtt{CL}\ \mathtt{of}\ B \\ \text{Base Clashes:} & \mathtt{CL}^0 & ::= & \mathtt{L}(\!|\mathtt{c}T|\!)\ u \mid t[\mathtt{c}P\backslash \mathtt{L}(\!|\lambda q.u|\!)] \mid t[\mathtt{c}P\backslash \mathtt{L}(\!|\mathtt{c}'T|\!)] \text{ where } \mathtt{c} \neq \mathtt{c}' \\ & & \mid & \mathtt{case}\ \mathtt{L}(\!|\lambda p.t|\!)\ \mathtt{of}\ B \\ & & \mid & \mathtt{case}\ \mathtt{L}(\!|\mathtt{c}T|\!)\ \mathtt{of}\ (\mathtt{c}_i P_i.u_i)_n \text{ where } \mathtt{c} \notin \{\mathtt{c}_1, \ldots, \mathtt{c}_n\}. \end{array}$$

By construction, the set of clashes is closed under term formation and list contexts, so that any term containing a clash is itself a clash.

Example 6.17 (Clashes)**.** *Going back to Example 6.14, term* $\mathtt{pair}(\mathtt{I}\,\mathtt{I}, \mathtt{I}) \notin \mathtt{CL}$ *is* not *a clash. But the three following terms are clashes:*

$$\mathtt{case}\ \mathtt{duo}(\mathtt{I}, \mathtt{I})\ \mathtt{of}\ (\mathtt{pair}(x, y).y) \in \mathtt{CL}$$
$$\mathtt{case}\ \mathtt{I}\ \mathtt{of}\ (\mathtt{pair}(x, y).y) \in \mathtt{CL} \qquad \mathtt{pair}(\mathtt{I}\,\mathtt{I}, \mathtt{I})\ \mathtt{I} \in \mathtt{CL}$$

Definition 6.18 (Clash-Free Syntactical Normal Forms)**.** Clearly, it is easy to refine the set of normal forms by excluding clashes. The resulting set is called the *set of clash-free normal forms* and is generated by the following grammars:

$$\begin{array}{rrcl} \text{Clash-Free Normal:} & \mathtt{NO}^{\mathtt{CF}} & ::= & \mathtt{NE}^{\mathtt{CF}} \mid \lambda p.t \mid \mathtt{c}T \mid c[\mathtt{c}P\backslash \mathtt{NE}^{\mathtt{CF}}] \\ \text{Clash-Free Neutral:} & \mathtt{NE}^{\mathtt{CF}} & ::= & x \mid \mathtt{NE}^{\mathtt{CF}}\ t \mid \mathtt{NE}^{\mathtt{CF}}[\mathtt{c}P\backslash \mathtt{NE}^{\mathtt{CF}}] \mid \mathtt{case}\ \mathtt{NE}^{\mathtt{CF}}\ \mathtt{of}\ B \end{array}$$

Set $\mathtt{NO}^{\mathtt{CF}}$ precisely captures the notion of clash-free normal form as follows.

Lemma 6.19 (Clash-Free Normal Forms)**.** *Let* $t \in \Lambda^{PM}$*. Then,* $t \in \mathtt{NO}^{\mathtt{CF}}$ *if and only if* $t \not\to_{\mathtt{PM}}$ *and* $t \notin \mathtt{CL}$

Again, note that the set of clash-free normal forms was defined for *open* terms. Unsurprisingly, by restricting this set to *closed* terms, the set of clash-free normal forms collapses as follows.

Lemma 6.20 (Closed Clash-Free Normal Forms)**.** *Let* $t \in \Lambda^{PM}$ *be a closed term. Then,* $t \not\to_{\mathtt{PM}}$ *and* $t \notin \mathtt{CL}$ *if and only if* $t = \lambda p.u$ *or* $t = \mathtt{c}T$ *for* $\mathtt{c} \in \mathtt{C}$*.*

Programs that are neither clashes nor in normal form can still *evaluate* to clashes. As an example, term $((\lambda x.\mathtt{pair}(\mathtt{I}, \mathtt{I}))\mathtt{I})\mathtt{I} \notin \mathtt{CL}$, but evaluates to the *clash* $\mathtt{pair}(\mathtt{I}, \mathtt{I})\ \mathtt{I} \in \mathtt{CL}$ from Example 6.17.

Formally, a program t is a *clash-free program* if and only if there is no $u \in \mathtt{CL}$ such that $t \twoheadrightarrow_{\mathtt{PM}} u$. It is clearly not possible to provide a syntactical characterization of clash-freeness. However, as we shall see later, *clash-free* terms can be characterized via a non-idempotent intersection type system for pattern matching (see Section 6.2).

6.1.2 Encoding Exceptions

In this subsection we are going to show how the λ_{PM}-calculus can be used to encode exceptions à la Moggi [31, 83].

Consider the following fragment of the λ_{PM}-calculus:

$$\begin{array}{llll} \text{Value:} & v, w & ::= & x \mid \lambda x.t \\ \text{Terms:} & t, u & ::= & \mathtt{v}(v) \mid \mathtt{e}(t) \mid vw \mid t[x \backslash u] \mid \mathtt{case}\ t\ \mathtt{of}\ (\mathtt{v}(x).u, \mathtt{e}(y).y) \\ & & \mid & \mathtt{case}\ t\ \mathtt{of}\ (\mathtt{v}(x).u, \mathtt{e}(y).\mathtt{e}(y)) \end{array}$$

where $\mathtt{v}$ and $\mathtt{e}$ are unary constructors used to tag values and exceptions, respectively. Terms $\mathtt{v}(v)$ and $\mathtt{e}(t)$ distinguish between values and exceptions.

Case expressions of the form

$$\mathtt{case}\ t\ \mathtt{of}\ (\mathtt{v}(x).u, \mathtt{e}(y).y)$$

compose terms sequentially by forcing the evaluation of t before proceeding to the continuation. This is due to terms either reducing to values or exceptions, which contain a continuation that must be returned. Moreover, an application of the form tu can be encoded as

$$\mathtt{case}\ t\ \mathtt{of}\ (\mathtt{v}(x).\mathtt{case}\ u\ \mathtt{of}\ (\mathtt{v}(y).xy, \mathtt{e}(z).\mathtt{e}(z)), \mathtt{e}(z).\mathtt{e}(z)).$$

The following examples illustrate three terms encoding the expected behavior of a term of the form $(\lambda x.u)t$ in the presence of exceptions, depending on whether t evaluates (successfully) to $\mathtt{v}(v)$ or (exceptionally) to $\mathtt{e}(r)$:

$$\begin{array}{rcl} t_1 & = & \mathtt{case}\ \mathtt{v}(v)\ \mathtt{of}\ (\mathtt{v}(x).u, \mathtt{e}(y).y) \twoheadrightarrow_{\mathtt{PM}} u\{x \setminus v\} \\ t_2 & = & \mathtt{case}\ \mathtt{e}(r)\ \mathtt{of}\ (\mathtt{v}(x).u, \mathtt{e}(y).y) \twoheadrightarrow_{\mathtt{PM}} r \\ t_3 & = & \mathtt{case}\ \mathtt{e}(r)\ \mathtt{of}\ (\mathtt{v}(x).u, \mathtt{e}(y).\mathtt{e}(y)) \twoheadrightarrow_{\mathtt{PM}} \mathtt{e}(r). \end{array}$$

The first term t_1 encodes a successful application $(\lambda x.u)(\mathtt{v}(v)) \twoheadrightarrow_{\mathtt{PM}} u\{x \setminus v\}$; t_2 encodes a short-circuited application $(\lambda x.u)(\mathtt{e}(r)) \rightsquigarrow r$ that handles the exception $\mathtt{e}$; and t_3 encodes a short-circuited application $(\lambda x.u)(\mathtt{e}(r)) \rightsquigarrow \mathtt{e}(r)$ that simply propagates exception $\mathtt{e}$ outwards.

6.1.3 Encoding the CBN and CBV λ-Calculi

In this subsection, we are going to show how the λ_{PM}-calculus can be used to encode the CBN and CBV λ-calculi.

The CBN λ-Calculus. Recall the set of λ-terms, CBN contexts, and CBN reduction, for the CBN λ-calculus (see Chapter 2).

λ-Terms. The set of *terms* of the CBN λ-calculus is denoted by Λ and generated by the following grammar:

$$\text{Terms:} \quad t, u, r, e ::= x \in \mathcal{X} \mid \lambda x.t \mid t\, u$$

CBN Contexts. The set of *CBN contexts* of the λ-calculus is generated by the following grammar:

$$\text{CBN Contexts:} \quad \mathtt{N} \quad ::= \quad \Box \mid \mathtt{N}\, t$$

CBN Reduction. *CBN reduction* is denoted by $\to_{\mathtt{CBN}}$ and defined over closed terms as the closure by CBN contexts of the following rewriting step:

$$(\lambda x.t)\, u \mapsto_\beta t\{x \setminus u\}.$$

The CBN λ-calculus can be fully embedded into the λ_{PM}-calculus through the identity translation. Let $\Lambda^{PM}_{\mathtt{CBN}}$ be the set of terms in the image of the trivial translation, together with matching closures restricted to variable patterns and terms of the form $t[x \setminus u]$.

The operational semantics of the CBN λ-calculus is captured as follows.

Lemma 6.21 (CBN Simulation)**.** *Let $t, u \in \Lambda$. If $t \twoheadrightarrow_{\mathtt{CBN}} u$, then $t \twoheadrightarrow_{\mathtt{WH}} u$.*

This translation is not very interesting, but notice that we have to consider the non-deterministic reduction relation $\to_{\mathtt{WH}}$ instead of the deterministic one $\to_{\mathtt{PM}}$.

The CBV λ-Calculus. More interestingly, the CBV λ-calculus can also be encoded into the λ_{PM}-calculus, but this simple fact does not seem to have been noticed in the literature until recently (see [101]).

Recall the set of λ-terms, CBV contexts, and CBV reduction, for the CBV λ-calculus (see Chapter 2).

λ-Terms. The set of *values* and *terms* of the CBV λ-calculus are denoted by V and Λ and generated respectively by the following grammars:

$$\begin{array}{lrcl} \text{Values:} & v, w & ::= & x \in \mathcal{X} \mid \lambda x.t \\ \text{Terms:} & t, u, r, e & ::= & x \in \mathcal{X} \mid \lambda x.t \mid t\, u. \end{array}$$

CBV Contexts. The set of *CBV contexts* of the λ-calculus is generated by the following grammar:

$$\text{CBV Contexts} \quad \mathtt{V} \quad ::= \quad \Box \mid \mathtt{V}\, t \mid v\, \mathtt{V}$$

CBV Reduction. *CBV reduction* is denoted by $\to_{\mathtt{CBV}}$ and defined over closed terms as the closure by CBV contexts of the following rewriting step:

$$(\lambda x.t)\, v \mapsto_{\beta_{\mathtt{v}}} t\{x \setminus v\}.$$

The CBV λ-calculus can be fully embedded into the λ_{PM}-calculus through translation $(\cdot)^\bullet$. Let $\Lambda^{PM}_{\mathtt{CBV}}$ and V denote the sets of terms and values of the CBV λ-calculus, respectively. Translation $(\cdot)^\bullet$ is defined as follows:

$$\begin{array}{rclcrcl}
(\cdot)^\bullet & : & \Lambda \mapsto \Lambda^{PM}_{\mathtt{CBV}} & & (\cdot)^\circ & : & \mathrm{V} \mapsto \Lambda^{PM}_{\mathtt{CBV}} \\
(v)^\bullet & = & \mathtt{v}((v)^\circ) & & (x)^\circ & = & x \\
(t\,u)^\bullet & = & (x\,y)[\mathtt{v}(y)\backslash(u)^\bullet][\mathtt{v}(x)\backslash(t)^\bullet] & & (\lambda x.t)^\circ & = & \lambda x.(t)^\bullet
\end{array}$$

where $\mathtt{v}$ is a unary constructor used to tag values.

This allows us to capture the operational semantics of the CBV λ-calculus. First we need to show that the CBV translation preserves substitutions.

Lemma 6.22 (The CBV Translation Preserves Substitutions)**.** *Let* $t, v \in \Lambda$. *Then,* $(t\{x \setminus v\})^\bullet = (t)^\bullet\{x \setminus (v)^\circ\}$.

Then, we can show the main simulation result.

Lemma 6.23 (CBV Simulation)**.** *Let* $t, u \in \Lambda$. *If* $t \twoheadrightarrow_{\mathtt{CBV}} u$, *then* $(t)^\bullet \twoheadrightarrow_{\mathtt{WH}} (u)^\bullet$.

Note that the use of the explicit matching operation (*i.e.* explicit substitutions) in the translation of an application is crucial in order to obtain the simulation result above. Since in Plotkin's CBV, an application of the form $t\,u$ can (in general) be evaluated by first evaluating either t or u, thus translation $(\cdot)^\bullet$ must allow this as well. Let $t \to_{\mathtt{CBV}} t'$. Consider an alternative translation $(t\,u)^\bullet = (\lambda\mathtt{v}(x).\lambda\mathtt{v}(y).x\,y)(t)^\bullet(u)^\bullet$. Then, it is not difficult to see that Lemma 6.23 does not hold anymore. Now, consider another alternative translation $(t\,u)^\bullet = \mathtt{case}\ u\ \mathtt{of}\ (\mathtt{v}(y).\mathtt{case}\ t\ \mathtt{of}\ (\mathtt{v}(x).x\,y))$. Then, $\to_{\mathtt{WH}}$ reduction forces u to be evaluated first. Alternatively, we could have considered another alternative translation $(t\,u)^\bullet = \mathtt{case}\ u\ \mathtt{of}\ (\mathtt{v}(y).\mathtt{case}\ t\ \mathtt{of}\ (\mathtt{v}(x).x\,y))$. However, the same problem arises if one simply realizes that the translations must also take into account the case where $u \to_{\mathtt{CBV}} u'$ happens first.

6.2 Pattern-Matching Non-Idempotent Intersection Types

In this section, we introduce a non-idempotent intersection type system for the λ_{PM}-calculus based on the non-idempotent intersection type system in [101], that establishes an exact equivalence between typability and PM-normalization, and is able to discriminate the exact length of

PM-normalization sequences according to the different reductions steps that are performed.

Definition 6.24 (Pattern-Matching Types)**.** The set of types for the λ_{PM}-calculus is generated by the following grammars:

$$\begin{array}{rrcl} \text{Data Types:} & D & ::= & \mathtt{c}(M_i)_n \\ \text{Multi-Types:} & M & ::= & \{\!\{A_i\}\!\}_{i\in \mathtt{I}} \text{ where } I \text{ is a finite set} \\ \text{Types:} & A & ::= & \star \mid D \mid M \to A \end{array}$$

Operations over multi-types, such as multiset difference and multiset union are defined as usual. Data types are tagged products of multi-types. Type $\star$ is a special constant type that is used to type abstractions whose bodies are not evaluated due to weak head evaluation.

Definition 6.25 (Pattern-Matching Typing Environments)**.** *Pattern-Matching typing environments* are denoted by Γ, Δ, and defined as functions from variables to multi-types, that assign the empty multi-type $\{\!\{\,\}\!\}$ to all but a finite set of variables.

The *domain of a typing environment*, the *union of typing environments*, and any other properties of typing environments, are defined as in Definition 4.2.

The *restriction of a typing environment* Γ *to a set of variables* $X \subseteq \mathcal{X}$ is denoted by $\Gamma|_X$ and defined as follows:

$$(\Gamma|_X)(x) = \begin{cases} \Gamma(x) & \text{if } x \in X \\ \{\!\{\,\}\!\} & \text{otherwise} \end{cases}$$

The *removal of a set of variables* $X \subseteq \mathcal{X}$ *from a typing environment* Γ is denoted by $\Gamma \setminus\!\setminus X$ and defined as follows:

$$(\Gamma \setminus\!\setminus X)(x) = \begin{cases} \{\!\{\,\}\!\} & \text{if } x \in X \\ \Gamma(x) & \text{otherwise} \end{cases}$$

Moreover, given some typing environment Γ and pattern p, we may sometimes write $\Gamma|_p$ and $\Gamma \setminus\!\setminus p$ to mean $\Gamma|_{\mathtt{vars}(p)}$ and $\Gamma \setminus\!\setminus \mathtt{vars}(p)$, respectively.

We can now introduce the set of typing rules for the λ_{PM}-calculus.

Definition 6.26 (Pattern-Matching Type System)**.** The *Pattern-Matching Type System* $\mathcal{PM}$ assigns matching (multi-)types to λ-terms and consists of the following typing rules:

Rules For Patterns

$$\frac{}{x : M \Vdash x : M}\ \texttt{VPat} \qquad \frac{(\Gamma_i \Vdash p_i : M_i)_{i\in\{1,\dots,n\}}}{+_{i\in\{1,\dots,n\}}\Gamma_i \Vdash \mathtt{c}(p_1,\dots,p_n) : \{\!\{\mathtt{c}(M_1,\dots,M_n)\}\!\}}\ \texttt{DPat}$$

Rules For Terms

$$\frac{}{x : \{\!\{A\}\!\} \vdash x : A}\ \texttt{Ax} \qquad \frac{\Gamma \vdash t : A \quad \Gamma|_{\mathtt{vars}(p)} \Vdash p : M}{\Gamma \setminus\!\setminus \mathtt{vars}(p) \vdash \lambda p.t : M \to A}\ \texttt{Abs} \qquad \frac{}{\emptyset \vdash \lambda p.t : \star}\ \texttt{Ans}$$

$$\frac{\Gamma \vdash t : M \to A \quad \Delta \vdash u : M}{\Gamma + \Delta \vdash t\,u : A}\ \texttt{App} \qquad \frac{(\Gamma_i \vdash t : A_i)_{i\in\mathtt{I}}}{+_{i\in\mathtt{I}}\Gamma_i \vdash t : \{\!\{A_i\}\!\}_{i\in\mathtt{I}}}\ \texttt{Many}$$

$$\frac{(\Gamma_i \vdash t_i : M_i)_{i\in\{1,\dots,n\}}}{+_{i\in\{1,\dots,n\}}\Gamma_i \vdash \mathtt{c}(t_1,\dots,t_n) : \mathtt{c}(M_1,\dots,M_n)}\ \texttt{Data}$$

$$\frac{\Gamma \vdash t : A \quad \Gamma|_{\mathtt{vars}(p)} \Vdash p : M \quad \Delta \vdash u : M}{(\Gamma \setminus\!\setminus \mathtt{vars}(p)) + \Delta \vdash t[p\backslash u] : A}\ \texttt{Match}$$

$$\frac{\Delta \vdash t : M \quad \Gamma|_{\mathtt{vars}(\hat{p}_k)} \Vdash \hat{p}_k : M \quad \Gamma \vdash u_k : A \quad \exists k \in \{1,\dots,n\}}{(\Gamma \setminus\!\setminus \mathtt{vars}(\hat{p}_k)) + \Delta \vdash \mathtt{case}\ t\ \mathtt{of}\ (\hat{p}_i.u_i)_n : A}\ \texttt{Case}$$

Moreover, we say that a term $t \in \Lambda^{PM}$ is typable in $\mathcal{PM}$ whenever it is typable with a *type*.

The Typing Rules. Rules `VPat` and `DPat` are used to type variable patterns and data patterns, respectively. The remaining rules are used to type terms. Rule `Ans` is necessary due to evaluation being weak, *i.e.* not occurring in the body of λ-abstractions. Notice that rule `Ans` allows us to trivially type any abstraction with type $\star$. However, such typings are impossible unless the abstractions typed by rule `Ans` appear on their own. Rule `Case` is very similar to `Match`, but the former appears to have a non-deterministic flavor, but it is not. As it turns out, all typable programs terminate in either a generalized λ-abstraction or a data. More specifically, programs typed with arrow types terminate in generalized λ-abstractions, and programs typed with data pattern types terminate in data with the same top-level tag. Consider a typable case expression $t = \mathtt{case}\ u\ \mathtt{of}\ (\hat{p}_i.s_i)_n$. Then, t must be typed using rule `Case`, and u must be typable as well. The type assigned to u must match the type assigned to one of the data patterns in the tuple of branches $(\hat{p}_i.s_i)_n$. Recall that all $\hat{p}_i$ are distinct by construction. Therefore, their associated data types are pairwise distinct, since constructor tags uniquely determine data types. Moreover, type of t can only match the type of the data pattern of *exactly* one of the branches. Thus, rule `Case`, *is* deterministic.

Example 6.27 (Typing Derivation)**.** *We can type the term from Example 6.6 with the following type derivation* Φ*. Since* Φ *is quite big, we are going to split it into the following pieces.*

Let Φ_1 *be the following type derivation for variable* x*:*

$$\dfrac{\dfrac{}{x : \{\!\{\mathtt{triple}(\{\!\{\mathtt{c}_0\}\!\},\{\!\{\,\}\!\},\{\!\{\,\}\!\})\}\!\} \vdash x : \mathtt{triple}(\{\!\{\mathtt{c}_0\}\!\},\{\!\{\,\}\!\},\{\!\{\,\}\!\})}\,\mathtt{Ax}}{x : \{\!\{\mathtt{triple}(\{\!\{\mathtt{c}_0\}\!\},\{\!\{\,\}\!\},\{\!\{\,\}\!\})\}\!\} \vdash x : \{\!\{\mathtt{triple}(\{\!\{\mathtt{c}_0\}\!\},\{\!\{\,\}\!\},\{\!\{\,\}\!\})\}\!\}}\,\mathtt{Many}$$

Let Φ_2 *be the following type derivation for the data pattern* $\mathtt{triple}(x,y,z)$*:*

$$\dfrac{\dfrac{}{x : \{\!\{\mathtt{c}_0\}\!\} \Vdash x : \{\!\{\mathtt{c}_0\}\!\}}\,\mathtt{VPat} \quad \dfrac{}{\emptyset \Vdash y : \{\!\{\,\}\!\}}\,\mathtt{VPat} \quad \dfrac{}{\emptyset \Vdash z : \{\!\{\,\}\!\}}\,\mathtt{VPat}}{x : \{\!\{\mathtt{c}_0\}\!\} \Vdash \mathtt{triple}(x,y,z) : \{\!\{\mathtt{triple}(\{\!\{\mathtt{c}_0\}\!\},\{\!\{\,\}\!\},\{\!\{\,\}\!\})\}\!\}}\,\mathtt{DPat}$$

Let Φ_3 *be the following type derivation for the variable pattern* x*:*

$$\dfrac{}{x : \{\!\{\mathtt{triple}(\{\!\{\mathtt{c}_0\}\!\},\{\!\{\,\}\!\},\{\!\{\,\}\!\})\}\!\} \Vdash x : \{\!\{\mathtt{triple}(\{\!\{\mathtt{c}_0\}\!\},\{\!\{\,\}\!\},\{\!\{\,\}\!\})\}\!\}}\,\mathtt{VPat}$$

Then, Ψ_1 *is the following type derivation for the abstraction:*

$$\dfrac{\dfrac{\Phi_1 \quad \Phi_2 \quad \dfrac{}{x : \{\!\{\mathtt{c}_0\}\!\} \vdash x : \mathtt{c}_0}\,\mathtt{Ax}}{x : \{\!\{\mathtt{triple}(\{\!\{\mathtt{c}_0\}\!\},\{\!\{\,\}\!\},\{\!\{\,\}\!\})\}\!\} \vdash \mathtt{case}\ x\ \mathtt{of}\ (\mathtt{pair}(x,y).y, \mathtt{triple}(x,y,z).x) : \mathtt{c}_0}\,\mathtt{Case} \quad \Phi_3}{\vdash \lambda x.\mathtt{case}\ x\ \mathtt{of}\ (\mathtt{pair}(x,y).y, \mathtt{triple}(x,y,z).x) : \{\!\{\mathtt{triple}(\{\!\{\mathtt{c}_0\}\!\},\{\!\{\,\}\!\},\{\!\{\,\}\!\})\}\!\} \to \mathtt{c}_0}\,\mathtt{Abs}$$

Now, let Ψ_2 *be the following type derivation for the data* $\mathtt{triple}(\mathtt{c}_0,\mathtt{c}_1,\mathtt{c}_2)$*:*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\emptyset \vdash \mathtt{c}_0 : \mathtt{c}_0}\,\mathtt{Data}}{\emptyset \vdash \mathtt{c}_0 : \{\!\{\mathtt{c}_0\}\!\}}\,\mathtt{Many} \quad \dfrac{}{\emptyset \vdash \mathtt{c}_1 : \{\!\{\,\}\!\}}\,\mathtt{Many} \quad \dfrac{}{\emptyset \vdash \mathtt{c}_2 : \{\!\{\,\}\!\}}\,\mathtt{Many}}{\emptyset \vdash \mathtt{triple}(\mathtt{c}_0,\mathtt{c}_1,\mathtt{c}_2) : \mathtt{triple}(\{\!\{\mathtt{c}_0\}\!\},\{\!\{\,\}\!\},\{\!\{\,\}\!\})}\,\mathtt{Data}}{\vdash \mathtt{triple}(\mathtt{c}_0,\mathtt{c}_1,\mathtt{c}_2) : \{\!\{\mathtt{triple}(\{\!\{\mathtt{c}_0\}\!\},\{\!\{\,\}\!\},\{\!\{\,\}\!\})\}\!\}}\,\mathtt{Many}$$

Then, type derivation Φ *can be built as follows:*

$$\dfrac{\Psi_1 \quad \Psi_2}{\emptyset \vdash (\lambda x.\mathtt{case}\ x\ \mathtt{of}\ (\mathtt{pair}(x,y).y, \mathtt{triple}(x,y,z).x))\ \mathtt{triple}(\mathtt{c}_0,\mathtt{c}_1,\mathtt{c}_2) : \mathtt{c}_0}\,\mathtt{App}$$

Note that $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi) = 1\cdot\mathtt{App} + 4\cdot\mathtt{VPat} + 1\cdot\mathtt{DPat}$ *and recall that* $t \overset{1\cdot\mathtt{dB}+4\cdot\mathtt{sub}+1\cdot\mathtt{branch}}{\underset{\mathtt{WH}}{\longrightarrow}} \mathtt{c}_0$ *(see Example 6.6).*

The above example seems to suggest that it is sufficient to count the number of occurrences of rules App, VPat, and DPat in order to draw quantitative information related to the number of reduction steps necessary to PM-normalize a term. Indeed, each App rule corresponds to a dB-step, each VPat corresponds to a sub-step, and each DPat rule corresponds to either a branch- or a match-step. The distinction between branch and match is operational and does not affect the type system, which records only the occurrence of a constructor-pattern interaction. Moreover, since the total number of reduction steps taken in Example 6.6 seems to coincide with the total number

of occurrences of rules App, VPat, and DPat in Example 6.27, it also seems that Φ might be a *tight* derivation. In [101] we did not consider *tight* type derivations, but we are going to do so here.

Definition 6.28 (Tight Typings for Pattern-Matching)**.** Let $t \in \Lambda^{PM}$. A type derivation Φ is *tight* if and only if it is of the form $\Phi \rhd_{\mathcal{PM}} \emptyset \vdash t : A$, such that A is $\star$ or $\mathtt{c}(\{\!\{\ \}\!\})_n$, for some $n \in \mathtt{N}$.

We start by showing that multiset-typings can be split and merged, as expected.

Lemma 6.29 (Split and Merge for Multi-Types)**.** *Let $t \in \Lambda^{PM}$ and $M = \sqcup_{i\in \mathtt{I}} M_i$. Then, there exists $\Phi \rhd \Gamma \vdash t : M$ if and only if there exist $(\Phi_i \rhd \Gamma_i \vdash t : M_i)_{i\in\mathtt{I}}$, such that $\Gamma = +_{i\in\mathtt{I}}\Gamma_i$. Moreover, $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi) = +_{i\in\mathtt{I}}\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_i)$.*

The equality on weights relies on the non-idempotent nature of multi-types, where multiplicities are preserved.

And that typing contexts are necessarily empty whenever terms are closed.

Lemma 6.30 (Contexts and Free Variables for Pattern-Matching)**.** *Let $t \in \Lambda^{PM}$ and $\Phi \rhd \Gamma \vdash t : A$. Then, $\mathtt{dom}(\Gamma) \subseteq \mathtt{fv}(t)$.*

The fact that having tight type derivations with zero weight is a characteristic of clash-free normal forms is formalized as follows.

Lemma 6.31 (Zero Weight Tight Type Derivations Characterize (Clash-Free) Normal Forms)**.** *Let $t \in \Lambda^{PM}_{\circ}$. If $\Phi \rhd_{\mathcal{PM}} \emptyset \vdash t : A$ is tight, then $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi) = 0 \cdot \mathtt{App} + 0 \cdot \mathtt{VPat} + 0 \cdot \mathtt{DPat}$ if and only if $t \in \mathtt{NO}^{\mathtt{CF}}$.*

Now, we show that $\mathcal{PM}$-typable terms are necessarily clash-free, *i.e.* normalize to a clash-free PM-normal form.

Lemma 6.32 (Typability Implies Clash-Freeness)**.** *Let $t \in \Lambda^{PM}$. If $\Gamma \vdash t : A$, then t is clash-free.*

The following two lemmas guarantee that all closed clash-free normal forms are tightly typable.

Lemma 6.33 (Patterns are Always Typable)**.** *Given any typing context Γ and pattern p, there exists $\Pi \rhd \Gamma|_p \Vdash p : M$. Moreover, if $p = \mathtt{c}(q_i)_n$, then there exist $\Pi_i \rhd_{\mathcal{PM}} \Gamma|_{q_i} \Vdash q_i : M_i$, such that $M = \{\!\{\mathtt{c}(M_i)_n\}\!\}$.*

Lemma 6.34 (Tight Typability of Closed Clash-Free PM-Normal Forms)**.** *Let $t \in \Lambda^{PM}_{\circ}$. If $t \in \mathtt{NO}^{\mathtt{CF}}$, then there is a derivation $\Phi \rhd_{\mathcal{PM}} \emptyset \vdash t : A$ that is tight.*

Proof. If $t \in \mathtt{NO}^{\mathtt{CF}}$, then $t = \lambda x.u$ or $t = \mathtt{c}T$:

- Let $t = \lambda x.u$. Then, we can build Φ using rule $\mathtt{Ans}$. Since $\mathtt{Ans}$ has no premises and assigns type $\star$ to $\lambda x.u$, the resulting type derivation is tight.
- Let $t = \mathtt{c}T$. Let $T = (u_1, \ldots, u_n)$, for some $n \in \mathtt{N}$. Then, we can build Φ as follows:

$$\dfrac{\left(\dfrac{}{\varnothing \vdash u_i : \{\!\{\,\}\!\}}\ \mathtt{Many}\right)_{i \in \{1,\ldots,n\}}}{\varnothing \vdash \mathtt{c}(u_1, \ldots, u_n) : \mathtt{c}(\{\!\{\,\}\!\})_n}\ \mathtt{Data}$$

  We can conclude since the above type derivation is tight.

□

In fact, we can show that $\mathcal{PM}$-typability *captures* the clash-freeness of $\mathtt{PM}$-normal forms.

Lemma 6.35 (Typability Characterizes Closed Clash-Free Normal Forms)**.** *Let $t \in \Lambda_{\circ}^{PM}$. Then, $t \in \mathtt{NO}^{\mathsf{CF}}$ if and only if $t \in \mathtt{NO}$ and t is $\mathcal{PM}$-typable.*

We now establish quantitative soundness and completeness of $\mathcal{PM}$-typability.

6.2.1 Quantitative Soundness

In this subsection we are going to show that $\mathcal{PM}$-typability is exactly quantitatively sound with respect to $\mathtt{PM}$-normalization, and that the total number of rules $\mathtt{App}$, $\mathtt{VPat}$, and $\mathtt{DPat}$ appearing in tight type derivations is the total length of $\mathtt{PM}$-normalization sequences. The structure of the proof follows the left-hand-side column of the proof schema for CBPV in Remark 4.38.

We proceed by proving a weighted version of the usual substitution lemma.

Lemma 6.36 (Weighted Substitution)**.** *Let $t, u \in \Lambda^{PM}$ and assume $\Phi_t \rhd_{\mathcal{PM}} \Gamma; x : M \vdash t : A$ and $\Phi_u \rhd_{\mathcal{PM}} \Delta \vdash u : M$. Then, there exists $\Phi_{t\{x \setminus u\}} \rhd \Gamma + \Delta \vdash t\{x \setminus u\} : A$, such that $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{t\{x\setminus u\}}) = \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_t) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u)$.*

The exact version of the subject reduction property follows from the above lemma.

Lemma 6.37 (Exact Subject Reduction)**.** *Let $t \in \Lambda^{PM}$, and $S \in \{\mathtt{dB}, \mathtt{branch}, \mathtt{match}, \mathtt{sub}\}$. If $t \to_{\mathtt{PM}(S)} t'$ and $\Phi_t \rhd_{\mathcal{PM}} \Gamma \vdash t : A$, then there is $\Phi_{t'} \rhd_{\mathcal{PM}} \Gamma \vdash t' : A$, such that $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_t) = \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{t'}) + 1 \cdot R$, where*

- *$R = \mathtt{App}$, if $S = \mathtt{dB}$;*
- *$R = \mathtt{VPat}$, if $S = \mathtt{sub}$;*
- *$R = \mathtt{DPat}$ if $S \in \{\mathtt{branch}, \mathtt{match}\}$.*

Finally, we can show that $\mathcal{PM}$-typability is quantitatively sound with respect to clash-free $\mathtt{PM}$-normalization.

Theorem 6.38 (Soundness). *Let $t \in \Lambda_{\circ}^{PM}$. If there is $\Phi \rhd_{\mathcal{PM}} \emptyset \vdash t : A$ tight, such that $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi) = n \cdot \mathtt{App} + m \cdot \mathtt{VPat} + l \cdot \mathtt{DPat}$, then $t \overset{n\cdot\mathtt{dB}+m\cdot\mathtt{sub}+l_1\cdot\mathtt{branch}+l_2\cdot\mathtt{match}}{\twoheadrightarrow_{\mathtt{PM}}} t'$, such that $l_1 + l_2 = l$ and $t' \in \mathtt{NO}^{\mathtt{CF}}$.*

Notice that type system $\mathcal{PM}$ does not distinguish between `branch` and `match`, since both correspond to the same constructor-pattern interaction, but this is not a problem since the quantitative results hold nonetheless.

6.2.2 Quantitative Completeness

In this subsection we are going to show that $\mathcal{PM}$-typability is complete with respect to `PM`-normalization, and that the total length of normalization sequences is the total number of rules `App`, `VPat`, and `DPat` appearing in tight type derivations. The structure of the proof follows the right-hand-side column of the proof schema for CBPV in Remark 4.38.

We start by proving the weighted version of the usual anti-substitution lemma.

Lemma 6.39 (Weighted Anti-substitution). *Let $t \in \Lambda^{PM}$ and $u \in \Lambda_{\circ}^{PM}$ and assume $\Phi_{t\{x\setminus u\}} \rhd_{\mathcal{PM}} \Gamma \vdash t\{x \setminus u\} : A$. Then, there exist $\Phi_t \rhd \Sigma; x : M \vdash t : A$ and $\Phi_u \rhd \Delta \vdash u : M$, such that $\Gamma = \Sigma + \Delta$ and $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{t\{x\setminus u\}}) = \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_t) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u)$.*

The quantitative version of the usual subject expansion property follows from the above lemma.

Lemma 6.40 (Exact Subject Expansion). *Let $t \in \Lambda^{PM}$, and $S \in \{\mathtt{dB}, \mathtt{branch}, \mathtt{match}, \mathtt{sub}\}$. If $t \to_{\mathtt{PM}(S)} t'$ and $\Phi_{t'} \rhd_{\mathcal{PM}} \Gamma \vdash t' : A$, then there is $\Phi_t \rhd_{\mathcal{PM}} \Gamma \vdash t : A$, such that $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{t'}) + 1 \cdot R = \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_t)$, where*

- $R = \mathtt{App}$, *if* $S = \mathtt{dB}$;
- $R = \mathtt{VPat}$, *if* $S = \mathtt{sub}$;
- $R = \mathtt{DPat}$ *if* $S \in \{\mathtt{branch}, \mathtt{match}\}$.

Finally, we can show that $\mathcal{PM}$-typability is quantitatively complete with respect to clash-free `PM`-normalization.

Theorem 6.41 (Completeness). *Let $t \in \Lambda_{\circ}^{PM}$. If $t \overset{n\cdot\mathtt{dB}+m\cdot\mathtt{sub}+l_1\cdot\mathtt{branch}+l_2\cdot\mathtt{match}}{\twoheadrightarrow_{\mathtt{PM}}} t'$, such that $t' \in \mathtt{NO}^{\mathtt{CF}}$, then there is $\Phi \rhd_{\mathcal{PM}} \emptyset \vdash t : A$ tight, such that $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi) = n \cdot \mathtt{App} + m \cdot \mathtt{VPat} + l \cdot \mathtt{DPat}$, where $l_1 + l_2 = l$.*

Notice that completeness holds only for normalization sequences that do not end in clashes, which reinforces the fact that clashes should be untypable.

This concludes the quantitative study of pattern matching. In the next chapters, we are going to present a quantitative study of higher-order state-passing computations.

Part IV

State-Passing Computations

Chapter 7

A Quantitative Study of Global State

This section is based on [43, 44], and develops a quantitative study of the extension of the λ-calculus with algebraic operations for reading and writing on a global memory, that we will call the $\lambda^{\mathcal{G}}$-calculus.

In [43], a quantitatively sound and complete (tight) type system was introduced for an extension of the CBV λ-calculus. In [44], this system was refined and extended to the CBN λ-calculus. Both works consider reduction over *open* λ-terms. In contrast, here we restrict reduction to *closed* terms (programs), and further extend the approach in [43] to the $\lambda!$-calculus (see Chapter 3).

The main technical deviation here from [44] is the way we treat the algebraic theory of global state. As it turns out, our approach here is identical to that in [43]. However, the point of view from which it is understood is quite different. To explain this, we start by explaining the difference between *linear* and *non-linear* λ-terms (see *e.g.* [102] and [103] for a more complete exposition of the topic).

Remark 7.1 (Linear vs. Non-Linear λ-Terms.).

Operationally...

Consider the following λ-term t:

$$t = (\lambda x.x\, x)\, (\lambda y.y).$$

Notice that t is *non-linear* due to the subterm $\lambda x.x\, x$. In particular, since x occurs twice in the body of $\lambda x.x\, x$, then $\lambda y.y$ is duplicated by the β-step :

$$(\lambda x.x\, x)\, (\lambda y.y) \mapsto_\beta (\lambda y.y)\, \lambda y.y.$$

Now, consider the following λ-term t':

$$t' = (\lambda x_1.(\lambda x_2.x_1\, x_2)\, \lambda y.y)\, \lambda y.y.$$

Notice that t' β-reduces (in two β-steps) to the same λ-term as t:

$$(\lambda x_1.(\lambda x_2.x_1\, x_2)\, \lambda y.y)\, \lambda y.y \mapsto_\beta (\lambda x_2.(\lambda y.y)\, x_2)\, \lambda y.y \mapsto_\beta (\lambda y.y)\, \lambda y.y.$$

This is no coincidence: t' is the *linearized* version of t in the sense of [102], obtained by making all variable occurrences linear. As such, t and t' have the same β-normal form. However, notice that t β-normalizes in fewer β-steps than t', even though no duplication occurs in the β-normalization of t'.

Denotationally...

Now, let us consider what happens when we type t and t' in (*e.g.*) type system $\mathcal{N}$. Let $A = \{\!\{\star\}\!\} \to \star$, Φ be the following *tight* type derivation for t:

$$\dfrac{\dfrac{\dfrac{}{x : \{\!\{A\}\!\} \vdash x : A}\,\mathtt{Ax} \qquad \dfrac{\dfrac{}{x : \{\!\{\star\}\!\} \vdash x : \star}\,\mathtt{Ax}}{x : \{\!\{\star\}\!\} \vdash x : \{\!\{\star\}\!\}}\,\mathtt{Many}}{x : \{\!\{A, \star\}\!\} \vdash x\, x : \star}\,\mathtt{App}}{\emptyset \vdash \lambda x.x\, x : \{\!\{A, \star\}\!\} \to \star}\,\mathtt{Abs} \qquad \dfrac{\dfrac{\dfrac{}{y : \{\!\{\star\}\!\} \vdash y : \star}\,\mathtt{Ax}}{\emptyset \vdash \lambda y.y : A}\,\mathtt{Abs} \qquad \dfrac{}{\emptyset \vdash \lambda y.y : \star}\,\mathtt{Ans}}{\emptyset \vdash \lambda y.y : \{\!\{A, \star\}\!\}}\,\mathtt{Many}}{\emptyset \vdash (\lambda x.x\, x)\, (\lambda y.y) : \star}\,\mathtt{App}$$

and Φ' be the following *tight* type derivation for t':

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x_1 : \{\!\{A\}\!\} \vdash x_1 : A}\,\mathtt{Ax} \qquad \dfrac{\dfrac{}{x_2 : \{\!\{\star\}\!\} \vdash x_2 : \star}\,\mathtt{Ax}}{x_2 : \{\!\{\star\}\!\} \vdash x_2 : \{\!\{\star\}\!\}}\,\mathtt{Many}}{x_1 : \{\!\{A\}\!\}, x_2 : \{\!\{\star\}\!\} \vdash x_1\, x_2 : \star}\,\mathtt{App}}{x_1 : \{\!\{A\}\!\} \vdash \lambda x_2.x_1\, x_2 : A}\,\mathtt{Abs} \qquad \dfrac{\dfrac{}{\emptyset \vdash \lambda y.y : \star}\,\mathtt{Ans}}{\emptyset \vdash \lambda y.y : \{\!\{\star\}\!\}}\,\mathtt{Many}}{x_1 : \{\!\{A\}\!\} \vdash (\lambda x_2.x_1\, x_2)\, \lambda y.y : \star}\,\mathtt{App}}{\emptyset \vdash \lambda x_1.(\lambda x_2.x_1\, x_2)\, \lambda y.y : \{\!\{A\}\!\} \to \star}\,\mathtt{Abs} \qquad \dfrac{\dfrac{\dfrac{}{y : \{\!\{\star\}\!\} \vdash y : \star}\,\mathtt{Ax}}{\emptyset \vdash \lambda y.y : A}\,\mathtt{Abs}}{\emptyset \vdash \lambda y.y : \{\!\{A\}\!\}}\,\mathtt{Many}$$
$$\emptyset \vdash (\lambda x_1.(\lambda x_2.x_1\, x_2)\, \lambda y.y)\, \lambda y.y : \star\,\mathtt{App}$$

Notice that each occurrence of $\lambda y.y$ in Φ' is typed with a singleton multiset (resp. $\{\!\{A\}\!\}$ or $\{\!\{\star\}\!\}$), and the single occurrence of $\lambda y.y$ in Φ is typed with multiset $\{\!\{A, \star\}\!\}$. Thus, even though t' is the linearization of t, they clearly have different–although related–type derivations: in Φ, $\lambda y.y$ appears once and is typed twice; in Φ', $\lambda y.y$ appears twice and each occurrence is typed once. Accordingly, the size of Φ and Φ' reflect the difference in the length of the β-normalization sequences for t and t': $\#_{\mathtt{App}}(\Phi) = 2$; and $\#_{\mathtt{App}}(\Phi') = 3$.

As we shall see immediately, an analogous situation arises in the algebraic theory of global state.

Remark 7.2 (Linear vs Non-Linear Global State Equations).

Operationally...

Consider the following equality from Example 5.19:

$$\texttt{get}(l, \lambda \texttt{b}.\texttt{get}(l, \lambda \texttt{a}.\texttt{t})) \sim_{\mathcal{G}} \texttt{get}(l, \lambda \texttt{a}.\texttt{t}\{\texttt{b} \setminus \texttt{a}\}).$$

According to Example 5.38, the above equation is cointerpreted as the following operational equality. Let $s \in S$. Then, both variables are substituted with the same value returned by the state:

$$((\texttt{t}\{\texttt{b} \setminus s(l)\})\{\texttt{a} \setminus s(l)\}, s) = ((\texttt{t}\{\texttt{b} \setminus \texttt{a}\})\{\texttt{a} \setminus s(l)\}, s).$$

If we now take the set of λ-terms Λ as the carrier of the coalgebra for global state, λ-abstractions denote continuations, and let the state be a mapping from locations (taken from $\mathcal{L}$) to λ-terms (taken from Λ), then we can reinterpret the above equation as follows:

$$(((x_1\, x_2)\{x_1 \setminus s(l)\})\{x_2 \setminus s(l)\}, s) = (((x_1\, x_2)\{x_2 \setminus x_1\})\{x_1 \setminus s(l)\}, s) = ((x_1\, x_1)\{x_1 \setminus s(l)\}, s).$$

That is,

$$(((x_1\, x_2)\{x_1 \setminus s(l)\})\{x_2 \setminus s(l)\}, s) = ((x_1\, x_1)\{x_1 \setminus s(l)\}, s).$$

Here continuations are interpreted in a call-by-name fashion, so substitutions are not evaluated eagerly.

Notice that this is very similar to what we saw in Remark 7.1, if we take $s(l) = \lambda y.y$. Indeed, recall the reduction sequences for $(\lambda x.x\, x)\, (\lambda y.y)$ (α-renamed below for clarity) and $(\lambda x_1.(\lambda x_2.x_1\, x_2)\, \lambda y.y)\, \lambda y.y$, but let us leave the substitutions unperformed. Then

$$(\lambda x_1.x_1\, x_1)\, (\lambda y.y) \mapsto_\beta (x_1\, x_1)\{x_1 \setminus \lambda y.y\},$$

and

$$\begin{aligned}(\lambda x_1.(\lambda x_2.x_1\, x_2)\, \lambda y.y)\, \lambda y.y &\mapsto_\beta ((\lambda x_2.x_1\, x_2)\, \lambda y.y)\{x_1 \setminus \lambda y.y\} \\ &\mapsto_\beta ((x_1\, x_2)\{x_1 \setminus \lambda y.y\})\{x_2 \setminus \lambda y.y\}.\end{aligned}$$

The similarity is striking: $(\lambda x_1.(\lambda x_2.x_1\, x_2)\, \lambda y.y)\, \lambda y.y$ is the linearization of $(\lambda x.x\, x)\, (\lambda y.y)$ with respect to $\lambda y.y$; and $(\texttt{get}(l, \lambda x_2.\texttt{get}(l, \lambda x_1.x_1\, x_2)), s)$ seems to be the linearization of the term $(\texttt{get}(l, \lambda x_1.x_1\, x_2\{x_2 \setminus x_1\}), s) = (\texttt{get}(l, \lambda x_1.x_1\, x_1), s)$ with respect to $s(l)$. Moreover, since x_1 occurs twice in the continuation of $(\texttt{get}(l, \lambda x_1.x_1\, x_1), s)$, $s(l)$ is duplicated once the get-operation is performed; since each x_1 and x_2 occur exactly once in the continuation, $s(l)$ is not duplicated, but two get-operations are performed.

Denotationally...

Now, let us understand how to type configurations such as $(\texttt{get}(l, \lambda x.t), s)$. If states are mappings from locations to λ-terms, it seems reasonable to type states with mappings from locations to types, *i.e.* the type of s should be a mapping S, such that for each location $l \in \mathcal{L}$, $S(l)$ is the type of $s(l)$. This corresponds to the following typing rule:

$$\frac{(\Gamma_l \vdash s(l) : S(l))_{l \in \mathcal{L}}}{+_{l \in \mathcal{L}} \Gamma_l \vdash s : S} \ \texttt{State}$$

where we assume that only finitely many locations are assigned non-empty types.

Now, we must understand how to type the operation `get`. Given how the type of state is a mapping of locations to types, it seems reasonable to take advantage of the coalgebraic structure of the global state to do so. At this point, it is best to leave the details to Section 7.1. Here, it is enough to notice that, if S is the type of s, the type of x in $\texttt{get}(l, \lambda x.t)$ should be $S(l)$, *i.e.* the type that results from running operation `get` on state S. For example, say we want to type $(\texttt{get}(l, \lambda x_1.x_1\, x_1), s)$. Then, taking into consideration the similarity to $(\lambda x_1.x_1\, x_1)\, (\lambda y.y)$, the type of x_1 should be $\{\!\{A, \star\}\!\}$ (see Remark 7.1), and thus that of $s(l)$, $\{\!\{A, \star\}\!\}$. Now, say we want to type $(\texttt{get}(l, \lambda x_1.x_1\, x_2\{x_2 \setminus x_1\}), s)$. Then, taking into consideration the similarity to $(\lambda x_1.(\lambda x_2.x_1\, x_2)\, \lambda y.y)\, \lambda y.y$, the type of x_1 and x_2 should be $\{\!\{A\}\!\}$ and $\{\!\{\star\}\!\}$, respectively, and thus that of $s(l)$ both $\{\!\{A\}\!\}$ and $\{\!\{\star\}\!\}$. Indeed, contrary to $(\lambda x_1.(\lambda x_2.x_1\, x_2)\, \lambda y.y)\, \lambda y.y$, which has two copies of $\lambda y.y$, $s(l)$ only contains one copy of $\lambda y.y$. Say we also type $s(l)$ with $\{\!\{A, \star\}\!\}$ for this case. Then, running operation `get` on state S twice (one for each `get` in $(\lambda x_1.(\lambda x_2.x_1\, x_2)\, \lambda y.y)\, \lambda y.y$) gives $\{\!\{A, \star\}\!\}$ in both cases. But, we want $\{\!\{A\}\!\}$ to be the type of x_1 and $\{\!\{\star\}\!\}$ to be the type of x_2.

One possible approach would be to define a very simple subtyping relation over multiset-types based on the obvious multi-subset ordering $\sqsubseteq$. However, this approach is not cost sensitive. Indeed, if two different `get` operations require the same type $\{\!\{A\}\!\}$ twice, there is no way to distinguish *e.g.* between $\{\!\{A\}\!\}$ and $\{\!\{A, A\}\!\}$, since $\{\!\{A\}\!\} \sqsubseteq \{\!\{A\}\!\}$ and $\{\!\{A\}\!\} \sqsubseteq \{\!\{A, A\}\!\}$. Nevertheless, it should be noted that this approach does seem to work for *idempotent* intersection types: subtyping collapses quantitative distinctions.

Another possible approach is to consider a different notion of state for the type of states. Instead, of the type of states being a mapping from locations to multi-sets, it becomes a mapping from locations to lists of multi-sets. For example, if we need $S(l)$ to be both $\{\!\{A\}\!\}$ and $\{\!\{\star\}\!\}$, we simply assign it both as a list *e.g.* $\{\!\{A\}\!\} \cdot \{\!\{\star\}\!\}$. Then, each time a `get` operation is run on S, the head of the list is *consumed*. This way, $\{\!\{A\}\!\}$ and $\{\!\{A\}\!\} \cdot \{\!\{A\}\!\}$ are indeed different. This was the approach taken in [44]. At the time, it seemed more different from [43] than we now recognize.

Indeed, the crucial point is to be able to update the type of the state as each `get` operation is run on it. The representation of $S(l)$ as a list $\{\!\{A\}\!\} \cdot \{\!\{A\}\!\}$ or a multiset-type $\{\!\{A, A\}\!\}$ is inconsequential. In the former case, each `get` operation consumes the head of the list (cf. [44]). In the latter case, each `get` operation consumes some element of the multiset-type (cf. [43]). In fact, given some list of multiset types L, we can define $M := \sqcup_{m \in L} m$ as a denotationally equivalent multiset type∗. However, notice that typing `get` operations is no longer as simple as running the `get` operation on S. Indeed, to type a `get` operation it is necessary to first run a `get` operation on S in order to get the type $S(l)$; and then to run a `set` operation (see Section 7.3) on S, in order to update $S(l)$. For example, consider again configuration $(\texttt{get}(l, \lambda x_1.x_1\, x_2\{x_2 \setminus x_1\}), s)$. If $S(l) = \{\!\{A\}\!\} \cdot \{\!\{\star\}\!\}$, the type of x_1 is obtained by running a `get` operation on S to obtain $\{\!\{A\}\!\}$, and then running a `set` operation on S to update $S(l)$ to $\{\!\{\star\}\!\}$. Then, the type of x_2 is obtained by running a `get` operation on S to obtain $\{\!\{\star\}\!\}$, and then running a `set` operation on S to update $S(l)$ to the empty list.

The key point of the above discussion is the following: even though `get` operations behave *non-linearly* with respect to state s, they need to behave *linearly* with respect to the type S of s. In particular, s is a mapping of locations to λ-terms and S is mapping of locations to lists of multiset-types. Then, every time a `get` operation is run on s, a `get` operation followed by a `set` operation is run on S. As it turns out, this is crucial in order to extend non-idempotent intersection type systems while preserving their quantitative precision. However, in the next chapter (see Chapter 8), we are going to repair this mismatch by introducing an algebraic theory that directly captures the notion of state required by the type systems in this chapter: the algebraic theory for infinite stacks.

In sum, the above discussion on the difference between the operational and the denotational behavior of operations can be summarized in Table 7.1.

	Operational Behavior (State $s \in \mathcal{V}^{\mathcal{L}}$)	**Denotational Behavior** (State $\mathcal{S} \in (M^*)^{\mathcal{L}}$)
$\texttt{get}(l, \lambda x.t)$	$[\![\texttt{get}]\!]^{\mathcal{V}^{\mathcal{L}}}(l, s) = (s\,(l), s)$ Variable x is replaced with $s\,(l)$. The updated state is s.	$[\![\texttt{get}]\!]^{(M^*)^{\mathcal{L}}}(l, \mathcal{S}) = (\mathcal{S}\,(l), \mathcal{S})$ $[\![\texttt{set}]\!]^{(M^*)^{\mathcal{L}}}((l, \texttt{tail}(\mathcal{S}\,(l))), \mathcal{S}) = (\star, \mathcal{S}\{l := \texttt{tail}(\mathcal{S}\,(l))\})$ Variable x has type $\texttt{head}(\mathcal{S}\,(l))$. The updated state has type $\mathcal{S}\{l := \texttt{tail}(\mathcal{S}\,(l))\}$.
$\texttt{set}((l, \texttt{v} \in \mathcal{V}), t)$	$[\![\texttt{set}]\!]^{\mathcal{V}^{\mathcal{L}}}((l, \texttt{v} \in \mathcal{V}), s) = (\star, s\{l := \texttt{v}\})$ The updated state is $s\{l := \texttt{v}\}$	$[\![\texttt{set}]\!]^{(M^*)^{\mathcal{L}}}((l, M^*), \mathcal{S}) = (\star, \mathcal{S}\{l := M^*\})$ The updated state has type $\mathcal{S}\{l := M^*\}$

Table 7.1: Comparison between the operational and denotational behavior of operations interacting with a global state.

∗This is a simple observation that will become important in Section 8.6.2.

We are now ready to extend the λ- and $\lambda!$-calculi with global state. All the proofs for this chapter are collected in Appendix E.

7.1 The λ-Calculus with Global State

The effect of global state allows computations to read and update a shared state whose content persists throughout evaluation. This section extends the λ-calculus with primitive operations for manipulating such a state, providing the simplest setting in which to study quantitative properties of stateful computation. We will call this new calculus the $\lambda^{\mathcal{G}}$-calculus.

7.1.1 Syntax and Operational Semantics

The syntax of the $\lambda^{\mathcal{G}}$-calculus is that of the λ-calculus extended with a notion of state corresponding to mapping from locations to arguments$*$, algebraic operations for reading and updating the state (according to Example 5.19), and configurations (*i.e.* term-state pairs).

Definition 7.3 (States and $\lambda^{\mathcal{G}}$-Terms)**.** Let $\mathcal{L}$ be a finite set of location names, $\mathcal{V}$ be a countable set of arguments, and $\mathcal{X}$ be a countable *set of variables*. The sets of terms $\Lambda^{\mathcal{G}}$, states $\mathcal{S}$, and configurations $\Lambda^{\mathcal{G}} \times \mathcal{S}$ of the $\lambda^{\mathcal{G}}$-calculus are generated by the three following grammars, respectively:

$$\begin{array}{lrcl} \text{Terms:} & t,u,r,e & ::= & x \in \mathcal{X} \mid \lambda x.t \mid t\,u \mid \mathtt{get}(l, \lambda x.t) \mid \mathtt{set}((l, t \in \mathrm{V}), t) \\ \text{States:} & s,q & ::= & \{l \mapsto t_l \in \mathcal{V}\}_{l \in \mathcal{L}} \\ \text{Configurations:} & c,c' & ::= & (t,s). \end{array}$$

Notice that the set of arguments $\mathcal{V}$ is left unspecified, since it will depend on the reduction strategy that is being considered. Moreover, states are total mappings from locations to arguments.

Notice also that states are defined according to Example 5.38. Regardless, we formally define the operations over states as follows.

Definition 7.4 (Operations On States)**.** Let $s := \{l \mapsto t_l \in \mathcal{V}\}_{l \in \mathcal{L}} \in \mathcal{S}$, and $l' \in \mathcal{L}$. Then

- we write $s\,(l')$ for the argument at location l' of s, *i.e.* $s\,(l') := t_{l'} \in \mathcal{V}$;
- and $s\{l' := t \in \mathcal{V}\}$ for the state obtained by replacing the argument at location l' of s with t, *i.e.* such that for all $l_0 \in \mathcal{L}$, $s\{l' := t \in \mathcal{V}\}\,(l_0) := s\,(l_0)$, if $l_0 \neq l'$, and $s\{l' := t \in \mathcal{V}\}\,(l_0) := t \in \mathcal{V}$, if $l_0 = l'$.

$*$We are going to call the set of values appearing in the algebraic theory for global state (see Example 5.19) the set of *arguments* to avoid confusion with the set of $\lambda^{\mathcal{G}}$-values of the CBV $\lambda^{\mathcal{G}}$-calculus. Indeed, we are going to consider this set to be parametric with respect to the calling mechanism that is being considered: for the CBN $\lambda^{\mathcal{G}}$-calculus, the set of arguments is going to be the set of $\lambda^{\mathcal{G}}$-terms; for the CBV $\lambda^{\mathcal{G}}$-calculus, the set of arguments is going to be the set of $\lambda^{\mathcal{G}}$-values.

Definition 7.5 (Free and Bound Variables)**.** Let $t, u \in \Lambda^{\mathcal{G}}$ and $s \in \mathcal{S}$.

The sets $\mathtt{fv}(t)$ and $\mathtt{fv}(s)$ of *free variables in t and s*, respectively, are defined by induction as follows:

$$\begin{aligned}
\mathtt{fv}(x) &= \{x\} \\
\mathtt{fv}(\lambda x.t) &= \mathtt{fv}(t) \setminus \{x\} \\
\mathtt{fv}(t\,u) &= \mathtt{fv}(t) \cup \mathtt{fv}(u) \\
\mathtt{fv}(\mathtt{get}(l, \lambda x.t)) &= \mathtt{fv}(t) \setminus \{x\} \\
\mathtt{fv}(\mathtt{set}((l, u \in \mathcal{V}), t)) &= \mathtt{fv}(u) \cup \mathtt{fv}(t) \\
\mathtt{fv}(\{l \mapsto t_l \in \mathcal{V}\}_{l \in \mathcal{L}}) &= \bigcup_{l \in \mathcal{L}} \mathtt{fv}(t_l)
\end{aligned}$$

The sets $\mathtt{bv}(t)$ and $\mathtt{bv}(s)$ of *bound variables in t and s*, respectively, are defined by induction as follows:

$$\begin{aligned}
\mathtt{bv}(x) &= \emptyset \\
\mathtt{bv}(\lambda x.t) &= \{x\} \cup \mathtt{bv}(t) \\
\mathtt{bv}(t\,u) &= \mathtt{bv}(t) \cup \mathtt{bv}(u) \\
\mathtt{bv}(\mathtt{get}(l, \lambda x.t)) &= \{x\} \cup \mathtt{bv}(t) \\
\mathtt{bv}(\mathtt{set}((l, u \in \mathcal{V}), t)) &= \mathtt{bv}(u) \cup \mathtt{bv}(t) \\
\mathtt{bv}(\{l \mapsto t_l \in \mathcal{V}\}_{l \in \mathcal{L}}) &= \bigcup_{l \in \mathcal{L}} \mathtt{bv}(t_l)
\end{aligned}$$

Example 7.6 (Free and Bound Variables)**.** *Let $\mathcal{L} = \{l_1, l_2\}$. Moreover, consider the following configuration $(\mathtt{get}(l, \lambda x.y\,x), \{l_1 \mapsto \lambda z.z, l_2 \mapsto \lambda z.z\})$. Then*

$$\begin{aligned}
\mathtt{fv}((\mathtt{get}(l, \lambda x.y\,x), \{l_1 \mapsto \lambda z.z, l_2 \mapsto \lambda z.z\})) &= \mathtt{fv}(\mathtt{get}(l, \lambda x.y\,x)) \cup \mathtt{fv}(\{l_1 \mapsto \lambda z.z, l_2 \mapsto z\}) \\
&= ((\mathtt{fv}(y\,x)) \setminus \{x\}) \cup (\mathtt{fv}(\lambda z.z) \cup \mathtt{fv}(z)) \\
&= (\{y, x\} \setminus \{x\}) \cup (\emptyset \cup \{z\}) \\
&= \{y\} \cup (\emptyset \cup \{z\}) \\
&= \{y, z\}
\end{aligned}$$

and

$$\begin{aligned}
\mathtt{bv}((\mathtt{get}(l, \lambda x.y\,x), \{l_1 \mapsto \lambda z.z, l_2 \mapsto \lambda z.z\})) &= \mathtt{bv}(\mathtt{get}(l, \lambda x.y\,x)) \cup \mathtt{bv}(\{l_1 \mapsto \lambda z.z, l_2 \mapsto z\}) \\
&= (\{x\} \cup (\mathtt{bv}(y\,x))) \cup (\mathtt{bv}(\lambda z.z) \cup \mathtt{bv}(z)) \\
&= (\{x\} \cup \emptyset) \cup (\{z\} \cup \mathtt{bv}(z)) \\
&= \{x\} \cup \{z\} \\
&= \{x, z\}.
\end{aligned}$$

The conventions regarding α-congruence are extended naturally to the above syntax.

Definition 7.7 (Closed and Open $\lambda^{\mathcal{G}}$-Terms and States)**.** A state $s \in \mathcal{S}$ and a term $t \in \Lambda^{\mathcal{G}}$ are said to be *closed* if and only if $\mathtt{fv}(s) = \emptyset$ and $\mathtt{fv}(t) = \emptyset$, respectively, and *open* otherwise. The sets of

closed states and terms of the $\lambda^{\mathcal{G}}$-calculus are denoted by $\mathcal{S}_\circ$ and $\Lambda^{\mathcal{G}}_\circ$, respectively. A configuration is said to be *closed* if and only if both its components are closed, and *open* otherwise.

Example 7.8 (Closed and Open $\lambda^{\mathcal{G}}$-Terms and States)**.** *Let $x \in \mathcal{X}$ and $l \in \mathcal{L}$. Then, x is an* open *$\lambda^{\mathcal{G}}$-term, and $\{l \mapsto x\}$ is an* open *state. However, $\lambda x.x$ is a* closed *$\lambda^{\mathcal{G}}$-term, and $\{l \mapsto \lambda x.x\}$ is a* closed *state.*

Definition 7.9 (Substitution)**.** Let $x \in \mathcal{X}$, and $t, u, r \in \Lambda^{\mathcal{G}}$. The result of *substituting u for the free occurrences of x in t* is denoted by $t\{x \setminus u\}$ and defined by induction as follows (recall that we are working modulo α-conversion):

$$\begin{array}{rcll} x\{x \setminus u\} & = & u & \\ y\{x \setminus u\} & = & y & \text{if } x \neq y \\ (\lambda y.r)\{x \setminus u\} & = & \lambda y.r\{x \setminus u\} & \\ (r\,e)\{x \setminus u\} & = & r\{x \setminus u\}\, e\{x \setminus u\} & \\ (\mathtt{get}(l, \lambda y.t))\{x \setminus u\} & = & \mathtt{get}(l, \lambda y.t\{x \setminus u\}) & \\ (\mathtt{set}((l, r \in \mathcal{V}), t))\{x \setminus u\} & = & \mathtt{set}((l, r\{x \setminus u\}), t\{x \setminus u\}) & \end{array}$$

Example 7.10 (Substitution)**.** *Let $\mathtt{get}(l, \lambda x.x\,y), \mathtt{set}((l, x), y) \in \Lambda^{\mathcal{G}}$. Then, for $u \in \Lambda^{\mathcal{G}}$, we have*

$$\begin{aligned} (\mathtt{get}(l, \lambda x.x\,y))\{x \setminus t\} &= \mathtt{get}(l, \lambda x.x\,y) \\ (\mathtt{get}(l, \lambda x.x\,y))\{y \setminus t\} &= \mathtt{get}(l, \lambda x.x\,t) \\ (\mathtt{set}((l, x), y))\{x \setminus t\} &= \mathtt{set}((l, t), y) \\ (\mathtt{set}((l, x), y))\{y \setminus t\} &= \mathtt{set}((l, x), y). \end{aligned}$$

Recall the following equation from Example 5.19:

$$\mathtt{get}(l, \lambda \mathtt{a}.\mathtt{set}((l', \mathtt{v}), \lambda \mathtt{b}.\mathtt{t})) \quad \sim_{\mathcal{G}} \quad \mathtt{set}((l', \mathtt{v}), \lambda \mathtt{b}.\mathtt{get}(l, \lambda \mathtt{a}.\mathtt{t})).$$

As it turns out, in order for it to hold, $\mathtt{a}$ must not appear free in v, *i.e.* $\mathtt{a} \notin \mathtt{fv}(v)$. Otherwise, $\mathtt{a}$ is bound in the term on the left, but free in the one on the right. Thus, we must impose the following condition over terms.

Definition 7.11 (Continuation Variables and Well-Formed $\lambda^{\mathcal{G}}$-Terms)**.** Let $t, u \in \Lambda^{\mathcal{G}}$ and $s \in \mathcal{S}$.

The sets $\mathtt{cv}(t)$ and $\mathtt{cv}(s)$ of *continuation variables* in t and s, respectively, are defined by induction as follows:

$$\begin{aligned}
\mathtt{cv}(x) &= \emptyset \\
\mathtt{cv}(\lambda x.t) &= \mathtt{cv}(t) \\
\mathtt{cv}(t\,u) &= \mathtt{cv}(t) \cup \mathtt{cv}(u) \\
\mathtt{cv}(\mathtt{get}(l, \lambda x.t)) &= \{x\} \cup \mathtt{cv}(t) \\
\mathtt{cv}(\mathtt{set}((l, u \in \mathcal{V}), t)) &= \mathtt{cv}(u) \cup \mathtt{cv}(t) \\
\mathtt{cv}(\{l \mapsto t_l \in \mathcal{V}\}_{l \in \mathcal{L}}) &= \textstyle\bigcup_{l \in \mathcal{L}} \mathtt{cv}(t_l).
\end{aligned}$$

The sets $\mathtt{cv}_l(t)$ and $\mathtt{cv}_l(s)$ of *continuation variables* in t and s *with respect to location* l, respectively, are defined by induction as follows:

$$\begin{aligned}
\mathtt{cv}_l(x) &= \emptyset \\
\mathtt{cv}_l(\lambda x.t) &= \mathtt{cv}_l(t) \\
\mathtt{cv}_l(t\,u) &= \mathtt{cv}_l(t) \cup \mathtt{cv}_l(u) \\
\mathtt{cv}_l(\mathtt{get}(l', \lambda x.t)) &= \begin{cases} \{x\} \cup \mathtt{cv}_l(t) & \text{if } l = l' \\ \mathtt{cv}_l(t) & \text{otherwise} \end{cases} \\
\mathtt{cv}_l(\mathtt{set}((l, u \in \mathcal{V}), t)) &= \mathtt{cv}_l(u) \cup \mathtt{cv}_l(t) \\
\mathtt{cv}_l(\{l' \mapsto t_{l'} \in \mathcal{V}\}_{l' \in \mathcal{L}}) &= \textstyle\bigcup_{l' \in \mathcal{L}} \mathtt{cv}_l(t_{l'}).
\end{aligned}$$

We say that a $\lambda^{\mathcal{G}}$*-term* $t \in \Lambda^{\mathcal{G}}$ *is well-formed* if and only if for every subterm $\mathtt{set}((l, r \in \mathcal{V}), u)$, $(\mathtt{fv}(r) \cap \mathtt{cv}(t)) \subseteq \mathtt{cv}_l(t)$. That is, the set of free continuation variables appearing in r is a subset of the set of continuation variables with respect to location l.

Intuitively, continuation variables record which variables may be substituted with state arguments originating from a given location.

Example 7.12 (Well-Formed $\lambda^{\mathcal{G}}$-Terms)**.** *The following is an example of a* well-formed $\lambda^{\mathcal{G}}$*-term:*

$$\mathtt{get}(l, \lambda x.\mathtt{get}(l, \lambda y.\mathtt{set}((l, x\,y), \lambda z.z))).$$

The following $\lambda^{\mathcal{G}}$*-term is* not*, due to continuation variable* y *now referring to location* l'*:*

$$\mathtt{get}(l, \lambda x.\mathtt{get}(l', \lambda y.\mathtt{set}((l, x\,y), \lambda z.z))).$$

Notice that the set of continuation variables is simply a subset of the set of bound variables. As such, any notion that applies to bound variables (such as α-conversion) also applies to continuation variables. From now on, we will assume that all $\lambda^{\mathcal{G}}$-terms are well-formed.

Call-By-Name. The set of arguments $\mathcal{V}$ in this setting is the set of $\lambda^{\mathcal{G}}$-terms $\Lambda^{\mathcal{G}}$. CBN contexts (in Definition 2.11) are naturally lifted to closed configurations. Let $s \in \mathcal{S}_\circ$ and $t \in \Lambda^{\mathcal{G}}_\circ$. We define

$\mathtt{N}[(t,s)] := (\mathtt{N}[t],s)$ to be the configuration obtained by replacing the hole $\Box$ in $\mathtt{N}$ with the term t. CBN reduction is defined for closed configurations as follows.

Definition 7.13 (CBN Reduction)**.** The *CBN reduction strategy* is denoted by $\Rightarrow_{\mathtt{CBN}}$ and defined over closed configurations as the closure by CBN contexts of the following three rewriting steps:

$$\begin{array}{rcl}((\lambda x.t)\,u,s) & \mapsto_{\beta} & (t\{x \setminus u\},s)\\ (\mathtt{get}(l,\lambda x.t),s) & \mapsto_{\mathtt{get}} & (t\{x \setminus s\,(l)\},s)\\ (\mathtt{set}((l,u),t),s) & \mapsto_{\mathtt{set}} & (t,s\{l := u\}).\end{array}$$

Observe that the rewriting steps for operations are defined according to Example 5.38 and Remark 5.39.

Example 7.14 (CBN Reduction)**.** *Notice how CBN reduction does not evaluate the term at the parameter position of* `set` *operations, but that any* $\lambda^{\mathcal{G}}$*-term is allowed to appear there:*

$$\begin{array}{rcl}(\mathtt{set}((l,(\lambda x.x)\,\lambda y.y),\mathtt{get}(l,\lambda z.z\,z)),s) & \Rightarrow_{\mathtt{CBN}(\mathtt{set})} & (\mathtt{get}(l,\lambda z.z\,z),s\{l := (\lambda x.x)\,\lambda y.y\})\\ & \Rightarrow_{\mathtt{CBN}(\mathtt{get})} & (((\lambda x.x)\,\lambda y.y)\,((\lambda x.x)\,\lambda y.y),s\{l := (\lambda x.x)\,\lambda y.y\})\\ & \Rightarrow_{\mathtt{CBN}(\beta)} & (\lambda y.y\,((\lambda x.x)\,\lambda y.y),s\{l := (\lambda x.x)\,\lambda y.y\})\\ & \Rightarrow_{\mathtt{CBN}(\beta)} & ((\lambda x.x)\,\lambda y.y,s\{l := (\lambda x.x)\,\lambda y.y\})\\ & \Rightarrow_{\mathtt{CBN}(\beta)} & (\lambda y.y,s\{l := (\lambda x.x)\,\lambda y.y\}).\end{array}$$

Call-By-Value. The set of ($\lambda^{\mathcal{G}}$-)values of the $\lambda^{\mathcal{G}}$-calculus denoted by $\mathrm{V}^{\mathcal{G}} \subseteq \Lambda^{\mathcal{G}}$ and generated by the following grammar:

$$\text{Values:}\quad v,w \quad ::= \quad x \in \mathcal{X} \mid \lambda x.t.$$

Notice that $\mathrm{V}^{\mathcal{G}}_{\circ} = \Lambda^{\mathcal{G}}_{\circ} \cap \mathrm{V}^{\mathcal{G}}$ is the set of closed values.

The set of arguments $\mathcal{V}$ is the set of values $\mathrm{V}^{\mathcal{G}}$. CBV contexts (in Definition 2.14) are naturally lifted to closed configurations. Let $s \in \mathcal{S}_{\circ}$ and $t \in \Lambda^{\mathcal{G}}_{\circ}$. We define $\mathtt{V}[(t,s)] := (\mathtt{V}[t],s)$ to be the configuration obtained by replacing the hole $\Box$ in $\mathtt{V}$ with the term t. CBV reduction is defined for closed configurations as follows.

Definition 7.15 (CBV Reduction)**.** The *CBV reduction strategy* is denoted by $\Rightarrow_{\mathtt{CBV}}$ and defined over closed configurations as the closure by CBV contexts of the following three rewriting steps:

$$\begin{array}{rcl}((\lambda x.t)\,v,s) & \mapsto_{\beta_{\mathrm{v}}} & (t\{x \setminus v\},s)\\ (\mathtt{get}(l,\lambda x.t),s) & \mapsto_{\mathtt{get}} & (t\{x \setminus s\,(l)\},s)\\ (\mathtt{set}((l,v),t),s) & \mapsto_{\mathtt{set}} & (t,s\{l := v\}).\end{array}$$

Notice that the only differences between the above rules and those for CBN reduction (see Definition 7.13) are related to the fact that only values can be substituted. Thus, only rules β and

set change. Rule get has the same shape as that for CBN, but now $s\,(l)$ is necessarily a value, as enforced by the syntactic definition of the state (cf. Definition 7.3).

Example 7.16 (CBV Reduction)**.** *Notice how CBV reduction also does not evaluate the term at the parameter position of* set *operations, but (contrary to CBN) only values are allowed to appear there:*

$$\begin{aligned}(\mathtt{set}((l,\lambda x.x),\mathtt{get}(l,\lambda y.y\,((\lambda z.z)\,y))),s) &\Rightarrow_{\mathtt{CBV}(\mathtt{set})} (\mathtt{get}(l,\lambda y.y\,((\lambda z.z)\,y)),s\{l:=\lambda x.x\})\\ &\Rightarrow_{\mathtt{CBV}(\mathtt{get})} ((\lambda x.x)\,((\lambda z.z)\,\lambda x.x),s\{l:=\lambda x.x\})\\ &\Rightarrow_{\mathtt{CBV}(\beta_{\mathrm{v}})} ((\lambda x.x)\,\lambda x.x,s\{l:=\lambda x.x\})\\ &\Rightarrow_{\mathtt{CBV}(\beta_{\mathrm{v}})} (\lambda x.x,s\{l:=\lambda x.x\}).\end{aligned}$$

Definition 7.17 (Syntactic CBN-Normal Forms and CBV-Normal Forms)**.** The sets of (closed) syntactic CBN-normal forms $\mathtt{NO}_{\mathtt{CBN}}$ and CBV-normal forms $\mathtt{NO}_{\mathtt{CBV}}$ of the $\lambda^{\mathcal{G}}$-calculus are both $\{(t,s) \mid t \in \mathrm{V}^{\mathcal{G}}_{\circ}, s \in \mathcal{S}_{\circ}\}$.

Lemma 7.18 (Characterization of CBN-Normal Forms and CBV-Normal Forms)**.** *Let* $(t,s) \in \Lambda^{\mathcal{G}} \times \mathcal{S}_{\circ}$. *Then,* $(t,s) \not\Rightarrow_{\mathtt{CBN}\,(\textit{resp. }\mathtt{CBV})}$ *if and only if* $t \in \mathtt{NO}_{\mathtt{CBN}}$ *(resp.* $\mathtt{NO}_{\mathtt{CBV}}$*).*

As was also the case for the CBN and CBV λ-calculi *without* global state, the sets of normal forms are the same for both CBN and CBV. This does not mean that CBN and CBV are a distinction without a difference. As an example, we have seen in Examples 2.13 and 2.16 that some terms have a CBN-normal forms but not a CBV-normal form, and that some terms CBV-normalize in fewer steps than they CBN-normalize.

7.2 The $\lambda!$-Calculus with Global State

We now extend the $\lambda!$-calculus introduced in Chapter 3 with primitive operations for interacting with the global state. This yields a system capable of representing both CBN and CBV interactions with state, while preserving the quantitative structure introduced earlier. We will call this new calculus the $\lambda!^{\mathcal{G}}$-calculus.

7.2.1 Syntax and Operational Semantics

The syntax of the $\lambda!^{\mathcal{G}}$-calculus is the extension of the syntax of the $\lambda!$-calculus according to Example 5.19.

Definition 7.19 (States and $\lambda!^{\mathcal{G}}$-Terms)**.** Let $\mathcal{L}$ be a finite set of location names and $\mathcal{X}$ be a countable *set of variables*. The sets of values $!\mathrm{V}^{\mathcal{G}}$, terms $!\Lambda^{\mathcal{G}}$, of states $\mathcal{S}$, and configurations $!\Lambda^{\mathcal{G}} \times \mathcal{S}$ of the

$\lambda!^{\mathcal{G}}$-calculus are generated by the following four grammars, respectively:

$$\begin{array}{lrcl}
\text{Values:} & v, w & ::= & x \in \mathcal{X} \mid !t \\
\text{Terms:} & t, u, r, e & ::= & v \mid \lambda x.t \mid t\,u \mid \mathtt{der}(t) \mid \mathtt{get}(l, \lambda x.t) \mid \mathtt{set}((l, v), t) \\
\text{State:} & s, q & ::= & \{l \mapsto v_l\}_{l \in \mathcal{L}} \\
\text{Configurations:} & c, c' & ::= & (t, s).
\end{array}$$

Here again, $\mathcal{V}$ is the set of values $!\mathrm{V}^{\mathcal{G}}$. States and terms can be put together as term-state pairs called *configurations*.

The notions of *free and bound variables* for the $\lambda!^{\mathcal{G}}$-calculus, as well as those of *α-congruence*, *substitution*, and *context*, are the obvious adaptations of those presented in Section 7.1 for the $\lambda^{\mathcal{G}}$-calculus.

Definition 7.20 (Closed and Open $\lambda!^{\mathcal{G}}$-Terms and States)**.** A state $s \in \mathcal{S}$ and a term $t \in !\Lambda^{\mathcal{G}}$ are said to be *closed* if and only if $\mathtt{fv}(s) = \emptyset$ and $\mathtt{fv}(t) = \emptyset$, respectively, and *open* otherwise. The sets of *closed state and terms* of the $\lambda!^{\mathcal{G}}$-calculus are denoted by $\mathcal{S}_\circ$ and $!\Lambda^{\mathcal{G}}_\circ$, respectively. A configuration is said to be *closed* if and only if both its components are closed.

Example 7.21 (Closed and Open $\lambda!^{\mathcal{G}}$-Terms and States)**.** *Let $x \in \mathcal{X}$ and $l, l' \in \mathcal{L}$. Then, x and $!x$ are* open *$\lambda!^{\mathcal{G}}$-terms, and $\{l \mapsto x, l' \mapsto \lambda x.x\}$ is an* open *state, due to x being an open $\lambda!^{\mathcal{G}}$-term. However, $\lambda x.x$ and $!\lambda x.x$ are both* closed *$\lambda!^{\mathcal{G}}$-terms, and $\{l \mapsto \lambda x.x, l' \mapsto \lambda x.x\}$ is a* closed *$\lambda!^{\mathcal{G}}$-term.*

CBPV contexts (in Definition 3.3) are naturally lifted to closed configurations. Let $s \in \mathcal{S}_\circ$ and $t \in !\Lambda^{\mathcal{G}}_\circ$. We define $\mathtt{B}[(t, s)] := (\mathtt{B}[t], s)$ to be the configuration obtained by replacing the hole $\Box$ in $\mathtt{B}$ with the term t. CBPV reduction is defined for closed configurations as follows.

Definition 7.22 (CBPV Reduction)**.** The *CBPV reduction strategy* is denoted by $\Rightarrow_{\mathtt{CBPV}}$ and defined over closed configurations as the closure by CBPV contexts of the following three rewriting steps:

$$\begin{array}{rcl}
((\lambda x.t)\, v, s) & \mapsto_{\beta_{\mathsf{v}}} & (t\{x \setminus v\}, s) \\
(\mathtt{der}(!t), s) & \mapsto_{\beta_!} & (t, s) \\
(\mathtt{get}(l, \lambda x.t), s) & \mapsto_{\mathtt{get}} & (t\{x \setminus s\,(l)\}, s) \\
(\mathtt{set}((l, v), t), s) & \mapsto_{\mathtt{set}} & (t, s\{l := v\}).
\end{array}$$

Observe that the rewriting steps for operations are defined according to Example 5.38 and Remark 5.39.

Example 7.23 (CBPV Reduction). *We are going to present two examples of CBPV reduction. The following corresponds to the CBN translation (see Definition 7.32) of Example 7.14. Let* $q = (s)^{\mathtt{CBN}}$*:*

$$
\begin{aligned}
& (\mathtt{set}((l, !((\lambda x.\mathtt{der}(x))\, !\lambda y.\mathtt{der}(y))), \mathtt{get}(l, \lambda z.\mathtt{der}(z)\, !\mathtt{der}(z))), q) \\
\Rightarrow_{\mathtt{set}} \quad & (\mathtt{get}(l, \lambda z.\mathtt{der}(z) !\mathtt{der}(z)\ ), q\{l := !((\lambda x.\mathtt{der}(x))\, !\lambda y.\mathtt{der}(y))\}) \\
\Rightarrow_{\mathtt{get}} \quad & (\mathtt{der}(!((\lambda x.\mathtt{der}(x))\, !\lambda y.\mathtt{der}(y)))\, !\mathtt{der}(!((\lambda x.\mathtt{der}(x))\, !\lambda y.\mathtt{der}(y))), \\
& q\{l := !(\lambda x.\mathtt{der}(x))\, !\lambda y.\mathtt{der}(y)\}) \\
\Rightarrow_{\beta_!} \quad & (((\lambda x.\mathtt{der}(x))\, !\lambda y.\mathtt{der}(y))\, !\mathtt{der}(!((\lambda x.\mathtt{der}(x))\, !\lambda y.\mathtt{der}(y))), \\
& q\{l := !(\lambda x.\mathtt{der}(x))\, !\lambda y.\mathtt{der}(y)\}) \\
\Rightarrow_{\beta_{\mathsf{v}}} \quad & (\mathtt{der}(!\lambda y.\mathtt{der}(y))\, !\mathtt{der}(!((\lambda x.\mathtt{der}(x))\, !\lambda y.\mathtt{der}(y))), q\{l := !(\lambda x.\mathtt{der}(x))\, !\lambda y.\mathtt{der}(y)\}) \\
\Rightarrow_{\beta_!} \quad & ((\lambda y.\mathtt{der}(y))\, !\mathtt{der}(!((\lambda x.\mathtt{der}(x))\, !\lambda y.\mathtt{der}(y))), q\{l := !(\lambda x.\mathtt{der}(x))\, !\lambda y.\mathtt{der}(y)\}) \\
\Rightarrow_{\beta_{\mathsf{v}}} \quad & (\mathtt{der}(!\mathtt{der}(!((\lambda x.\mathtt{der}(x))\, !\lambda y.\mathtt{der}(y)))), q\{l := !(\lambda x.\mathtt{der}(x))\, !\lambda y.\mathtt{der}(y)\}) \\
\Rightarrow_{\beta_!} \quad & (\mathtt{der}(!((\lambda x.\mathtt{der}(x))\, !\lambda y.\mathtt{der}(y))), q\{l := !(\lambda x.\mathtt{der}(x))\, !\lambda y.\mathtt{der}(y)\}) \\
\Rightarrow_{\beta_!} \quad & ((\lambda x.\mathtt{der}(x))\, !\lambda y.\mathtt{der}(y), q\{l := !(\lambda x.\mathtt{der}(x))\, !\lambda y.\mathtt{der}(y)\}) \\
\Rightarrow_{\beta_{\mathsf{v}}} \quad & (\mathtt{der}(!\lambda y.\mathtt{der}(y)), q\{l := !(\lambda x.\mathtt{der}(x))\, !\lambda y.\mathtt{der}(y)\}) \\
\Rightarrow_{\beta_!} \quad & (\lambda y.\mathtt{der}(y), q\{l := !(\lambda x.\mathtt{der}(x))\, !\lambda y.\mathtt{der}(y)\});
\end{aligned}
$$

The following corresponds to the CBV translation (see Definition 7.32) of Example 7.16. Let $q = (s)^{\mathtt{CBN}}$*. Let* $(s)^{\mathtt{CBV}} = q$*.*

$$
\begin{aligned}
& (\mathtt{set}((l, !\lambda x.x), \mathtt{get}(l, \lambda y.\mathtt{der}(y)\ (\mathtt{der}(!(\lambda z.z))\ y))), q) \\
\Rightarrow_{\mathtt{set}} \quad & (\mathtt{get}(l, \lambda y.\mathtt{der}(y)\ (\mathtt{der}(!(\lambda z.z))\ y)), q\{l := !\lambda x.x\}) \\
\Rightarrow_{\mathtt{get}} \quad & (\mathtt{der}(!\lambda x.x)\ (\mathtt{der}(!(\lambda z.z))\, !\lambda x.x), q\{l := !\lambda x.x\}) \\
\Rightarrow_{\beta_!} \quad & (\lambda x.x\ (\mathtt{der}(!(\lambda z.z))\, !\lambda x.x), q\{l := !\lambda x.x\}) \\
\Rightarrow_{\beta_!} \quad & (\lambda x.x\ ((\lambda z.z)\, !\lambda x.x), q\{l := !\lambda x.x\}) \\
\Rightarrow_{\beta_{\mathsf{v}}} \quad & ((\lambda x.x)\, !\lambda x.x, q\{l := !\lambda x.x\}) \\
\Rightarrow_{\beta_{\mathsf{v}}} \quad & (!\lambda x.x, q\{l := !\lambda x.x\}).
\end{aligned}
$$

Definition 7.24 (Syntactic CBPV-Normal Forms)**.** The set of (closed) syntactic CBPV-normal forms of the $\lambda!^{\mathcal{G}}$-calculus is $\{(t,s) \mid t \in \mathtt{NO}_{\mathtt{CBPV}}, s \in \mathcal{S}_\circ\}$, where $\mathtt{NO}_{\mathtt{CBPV}}$ is the set of (closed) syntactic CBPV-normal forms of the $\lambda!^{\mathcal{G}}$-calculus, generated by the following grammar:

$$
\begin{array}{lrcl}
\text{Normal Forms:} & \mathtt{NO}_{\mathtt{CBPV}} & ::= & \mathtt{NA}_{\mathtt{CBPV}} \mid \mathtt{NB}_{\mathtt{CBPV}} \\
\text{Bang or Neutral Forms:} & \mathtt{NA}_{\mathtt{CBPV}} & ::= & !t \mid \mathtt{NE}_{\mathtt{CBPV}} \\
\text{Abstraction or Neutral Forms:} & \mathtt{NB}_{\mathtt{CBPV}} & ::= & \lambda x.u \mid \mathtt{NE}_{\mathtt{CBPV}} \\
\text{Neutral Forms:} & \mathtt{NE}_{\mathtt{CBPV}} & ::= & (\lambda x.u)\, \mathtt{NB}_{\mathtt{CBPV}} \mid \mathtt{NA}_{\mathtt{CBPV}}\, t \mid \mathtt{der}(\mathtt{NB}_{\mathtt{CBPV}}).
\end{array}
$$

Proposition 7.25 (Characterization of CBPV-Normal Forms). *Let* $(t,s) \in !\Lambda^{\mathcal{G}} \times \mathcal{S}_\circ$. *Then,* $(t,s) \not\to_{\mathtt{CBPV}}$ *if and only if* $t \in \mathtt{NO}_{\mathtt{CBPV}}$.

The set of clashes is naturally lifted to configurations: if t is a clash, then so is (t,s), for any $s \in \mathcal{S}_\circ$. The set of clash-free terms is also naturally lifted to configurations.

Definition 7.26 (Clash-Free Configurations). A configuration is *clash-free* if it does not reduce to a configuration containing a term with a clash (see Definition 3.8) outside the scope of any lambda λ or bang ! constructor. In other words, let $s \in \mathcal{S}_\circ$ and $t \in !\Lambda^{\mathcal{G}}$. Then, (t,s) is *not* clash-free if and only if there exist a CBPV context $\mathtt{B}$, a clash u, and a state $q \in \mathcal{S}_\circ$, such that $(t,s) \twoheadrightarrow_{\mathtt{CBPV}} \mathtt{B}[(u,q)]$.

Example 7.27 (Clash-Free Configurations). *The following configuration c is* not *a clash-free configuration:*

$$c = (\mathtt{get}(l, \lambda x.x\, x), s\{l := !\lambda y.y\}) \Rightarrow_{\mathtt{CBPV}} ((!\lambda y.y)\, !\lambda y.y, s\{l := !\lambda y.y\}).$$

But the following configuration c' is clash-free:

$$\begin{aligned} c' = (\mathtt{get}(l, \lambda x.\mathtt{der}(x)\, x), s\{l := !\lambda y.y\}) &\Rightarrow_{\mathtt{CBPV}} (\mathtt{der}((!\lambda y.y))\, !\lambda y.y, s\{l := !\lambda y.y\}) \\ &\Rightarrow_{\mathtt{CBPV}} ((\lambda y.y)\, !\lambda y.y, s\{l := !\lambda y.y\}) \\ &\Rightarrow_{\mathtt{CBPV}} (!\lambda y.y, s\{l := !\lambda y.y\}). \end{aligned}$$

Definition 7.28 (Syntactic Clash-Free CBPV-Normal Forms). The set of *clash-free* CBPV-normal forms of the $\lambda!^{\mathcal{G}}$-calculus is $\{(t,s) \mid t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}, s \in \mathcal{S}_\circ\}$, where $\mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}} := !\mathrm{V}^{\mathcal{G}}_\circ \cup \{\lambda x.t \mid \lambda x.t \in !\Lambda^{\mathcal{G}}_\circ\}$.

Proposition 7.29 (Characterization of Clash-Free CBPV-Normal Forms). *Let* $(t,s) \in !\Lambda^{\mathcal{G}}$. *Then,* $(t,s) \not\to_{\mathtt{CBPV}}$ *and t is clash-free if and only if* $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$.

Proof. By following the same structure as the proof of Proposition 3.11. □

As we will see later, *clash-free* terms can be characterized via a lifting of the non-idempotent intersection type system presented in Section 4.2 (see Section 7.3.3).

7.2.2 Subsuming Call-By-Name and Call-By-Value

The unifying framework of the $\lambda!^{\mathcal{G}}$-calculus allows us once again to capture both CBN and CBV interactions with state. This is essential for transferring quantitative results between evaluation strategies, since effects such as reading or updating a state interact differently with duplication and evaluation order.

We now show how the encodings of Section 3.2 can be extended to the $\lambda!^{\mathcal{G}}$-calculus. Naturally, this require us to extend the translation functions for CBN and CBV to include operations, and to additionally define how to encode states.

Definition 7.30 (Translations of Terms)**.** Consider the following inductively defined translation of terms of the $\lambda^{\mathcal{G}}$-calculus to terms of the $\lambda!^{\mathcal{G}}$-calculus:

$$\begin{array}{rcl}(\cdot)^{\mathtt{CBN}} : \Lambda^{\mathcal{G}} & \to & !\Lambda^{\mathcal{G}} \\ x & \mapsto & \mathtt{der}(x) \\ \lambda x.t & \mapsto & \lambda x.(t)^{\mathtt{CBN}} \\ t\,u & \mapsto & (t)^{\mathtt{CBN}}\,!(u)^{\mathtt{CBN}} \\ \mathtt{get}(l,\lambda x.t) & \mapsto & \mathtt{get}(l,\lambda x.(t)^{\mathtt{CBN}}) \\ \mathtt{set}((l,u),t) & \mapsto & \mathtt{set}((l,!(u)^{\mathtt{CBN}}),(t)^{\mathtt{CBN}})\end{array} \qquad \begin{array}{rcl}(\cdot)^{\mathtt{CBV}} : \Lambda^{\mathcal{G}} & \to & !\Lambda^{\mathcal{G}} \\ x & \mapsto & x \\ \lambda x.t & \mapsto & !\lambda x.(t)^{\mathtt{CBV}} \\ t\,u & \mapsto & \mathtt{der}((t)^{\mathtt{CBV}})\,(u)^{\mathtt{CBV}} \\ \mathtt{get}(l,\lambda x.t) & \mapsto & \mathtt{get}(l,\lambda x.(t)^{\mathtt{CBV}}) \\ \mathtt{set}((l,v),t) & \mapsto & \mathtt{set}((l,(v)^{\mathtt{CBV}}),(t)^{\mathtt{CBV}}).\end{array}$$

The above definition is the natural extensions of Definition 3.12 to operations, *i.e.* all terms (with the exception of operations) are translated in much the same way. However, notice that terms occurring at the parameter position of operations must belong to the set of values $!\mathrm{V}^{\mathcal{G}}$ of the $\lambda!^{\mathcal{G}}$-calculus. Accordingly, the image of the translation must be a subset of $!\mathrm{V}^{\mathcal{G}}$. The image of $(\cdot)^{\mathtt{CBV}}$ is indeed a subset of $!\mathrm{V}^{\mathcal{G}}$ (see Lemma 7.31), but the image of $(\cdot)^{\mathtt{CBN}}$ is not. However, this can be solved by decorating the term in the image of $(\cdot)^{\mathtt{CBN}}$ with a bang.

Lemma 7.31 (Preservation of Values)**.** *Let* $v \in \mathrm{V}^{\mathcal{G}}$*, then* $(v)^{\mathtt{CBV}} \in !\mathrm{V}^{\mathcal{G}}$.

The above definition only tells us how to translate terms (and parameters), but we want to be able to translate *configurations*.

Definition 7.32 (Translations of Configurations)**.** Consider the following inductively defined translation of states of the $\lambda^{\mathcal{G}}$-calculus to states of the $\lambda!^{\mathcal{G}}$-calculus:

$$\begin{array}{rcl}(\cdot)^{\mathtt{CBN}} : \mathcal{S} & \to & \mathcal{S} \\ \{l \mapsto t_l\}_{l\in\mathcal{L}} & \mapsto & \{l \mapsto !(t_l)^{\mathtt{CBN}}\}_{l\in\mathcal{L}}\end{array} \qquad \begin{array}{rcl}(\cdot)^{\mathtt{CBV}} : \mathcal{S} & \to & \mathcal{S} \\ \{l \mapsto v_l\}_{l\in\mathcal{L}} & \mapsto & \{l \mapsto (v_l)^{\mathtt{CBV}}\}_{l\in\mathcal{L}}.\end{array}$$

The translation of configurations of the $\lambda^{\mathcal{G}}$-calculus to configurations of the $\lambda!^{\mathcal{G}}$-calculus are then defined as follows:

$$\begin{array}{rcl}(\cdot)^{\mathtt{CBN}} : \Lambda^{\mathcal{G}} \times \mathcal{S} & \to & !\Lambda^{\mathcal{G}} \times \mathcal{S} \\ (t,s) & \mapsto & ((t)^{\mathtt{CBN}},(s)^{\mathtt{CBN}})\end{array} \qquad \begin{array}{rcl}(\cdot)^{\mathtt{CBV}} : \Lambda^{\mathcal{G}} \times \mathcal{S} & \to & !\Lambda^{\mathcal{G}} \times \mathcal{S} \\ (t,s) & \mapsto & ((t)^{\mathtt{CBV}},(s)^{\mathtt{CBV}}).\end{array}$$

Example 7.33 (Translations of Configurations)**.** *In Example 7.23, we have the following translations of the configurations in Example 7.14 and Example 7.16:*

$$\begin{gathered}((\mathtt{set}((l,(\lambda x.x)\,\lambda y.y),\mathtt{get}(l,\lambda z.z\,z)),s))^{\mathtt{CBN}}\\=\\(\mathtt{set}((l,!((\lambda x.\mathtt{der}(x))\,!\lambda y.\mathtt{der}(y))),\mathtt{get}(l,\lambda z.z\,!\mathtt{der}(z))),(s)^{\mathtt{CBN}})\end{gathered}$$

$$\begin{gathered}((\mathtt{set}((l,\lambda x.x),\mathtt{get}(l,\lambda y.y\,((\lambda z.z)\,y))),s))^{\mathtt{CBV}}\\=\\(\mathtt{set}((l,!\lambda x.x),\mathtt{get}(l,\lambda y.\mathtt{der}(y)\,(\mathtt{der}(!(\lambda z.z))\,y))),(s)^{\mathtt{CBV}}).\end{gathered}$$

7.2.3 Correctness of Full Simulations

In this section we show that the CBN (resp. CBV) translation presented in the previous section preserves CBN- (resp. CBV)-normal forms, as well as the number of β- (resp. $\beta_{\mathtt{v}}$-), get-, and set-steps, respectively.

Just like we did in Section 3.2, we are going to introduce administrative contexts to allow $\beta_!$-steps to be fired under any Contexts.

Definition 7.34 (Administrative Contexts)**.** Let $t \in \Lambda^{\mathcal{G}}$. The set of *administrative contexts* of the $\lambda!^{\mathcal{G}}$-calculus are generated by the following grammar:

$$\begin{array}{rcl}\mathtt{A} & ::= & \lambda x.\Box \mid !\Box \mid \mathtt{A}\,t \mid t\,\mathtt{A} \mid \lambda x.\mathtt{A} \mid !\mathtt{A} \mid \mathtt{der}(\mathtt{A})\\ & \mid & \mathtt{get}(l,\lambda x.\mathtt{A}) \mid \mathtt{set}((l,\mathtt{A}),t) \mid \mathtt{set}((l,v),\mathtt{A}).\end{array}$$

Definition 7.35 (Administrative Reduction Relation)**.** The *administrative reduction relation* is denoted by $\Rightarrow_{\mathtt{ADMIN}}$ and defined as the closure of $\beta_!$-steps by *administrative contexts*.

Notice that administrative steps never occur at top level.

Example 7.36 (Administrative Reduction Relation)**.** *Consider the following configuration:*

$$(\mathtt{set}((l,!\mathtt{der}(!\lambda x.x)),\mathtt{der}(!\lambda x.x)),s) \in \Lambda^{\mathcal{G}} \times \mathcal{S}_{\mathcal{G}}.$$

Then, by the administrative reduction relation, the $\beta_!$-redex occurring at the parameter position of the set *operation can be fired:*

$$(\mathtt{set}((l,!\mathtt{der}(!\lambda x.x)),\mathtt{der}(!\lambda x.x)),s) \Rightarrow_{\mathtt{ADMIN}} (\mathtt{set}((l,!\lambda x.x),\mathtt{der}(!\lambda x.x)),s).$$

However, the only other $\beta_!$-redex occurring in the above configuration cannot be fired by an administrative step due to not occurring under a bang.

Notice that Definition 7.35 is the obvious extension of Chapter 3 to the `get` and `set` operations, and thus also preserves the well-formedness of terms. Moreover, notice also that the administrative reduction relation never changes the state. Nevertheless, it is important to stress how important it is to allow administrative steps to occur at the parameter position of operations. Consider the following configuration and its CBN translation:

$$(\mathtt{set}((l, r\{x \setminus u\}), t), s)$$
$$(\mathtt{set}((l, r\{x \setminus u\}), t), s)^{\mathtt{CBN}} = (\mathtt{set}((l, !(r\{x \setminus u\})^{\mathtt{CBN}}), (t)^{\mathtt{CBN}}), (s)^{\mathtt{CBN}}).$$

Notice that the translation of $r\{x \setminus u\}$ is $!(r\{x \setminus u\})^{\mathtt{CBN}}$. Now, consider what happens when we apply a `set`-step to both:

$$(\mathtt{set}((l, r\{x \setminus u\}), t), s) \mapsto_{\mathtt{set}} (t, s\{l := r\{x \setminus u\}\})$$
$$(\mathtt{set}((l, !(r\{x \setminus u\})^{\mathtt{CBN}}), (t)^{\mathtt{CBN}}), (s)^{\mathtt{CBN}}) \mapsto_{\mathtt{set}} ((t)^{\mathtt{CBN}}, (s)^{\mathtt{CBN}}\{l := !(r\{x \setminus u\})^{\mathtt{CBN}}\}).$$

Notice that $r\{x \setminus u\}$ and $!(r\{x \setminus u\})^{\mathtt{CBN}}$ are placed at location l of s and $(s)^{\mathtt{CBN}}$, respectively. Now, consider the following configuration instead:

$$(\mathtt{set}((l, !(r)^{\mathtt{CBN}})\{x \setminus !(u)^{\mathtt{CBN}}\}, (t)^{\mathtt{CBN}}), (s)^{\mathtt{CBN}}).$$

Notice that, as long as $(r)^{\mathtt{CBN}}\{x \setminus (u)^{\mathtt{CBN}}\} = (r\{x \setminus u\})^{\mathtt{CBN}}$, then it is the same as the one above. But, as we shall see, this property is not the case in CBN. Indeed, it is only the case that $((r)^{\mathtt{CBN}}\{x \setminus (u)^{\mathtt{CBN}}\}, q) \twoheadrightarrow_{\mathtt{ADMIN}} ((r\{x \setminus u\})^{\mathtt{CBN}}, q)$, for any state $q \in \mathcal{S}$. Moreover, if we consider what happens when we apply a `set`-step to it:

$$(\mathtt{set}((l, !(r)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}), (t)^{\mathtt{CBN}}), (s)^{\mathtt{CBN}}) \mapsto_{\mathtt{set}} ((t)^{\mathtt{CBN}}, (s)^{\mathtt{CBN}}\{l := !(r)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}\}).$$

Notice that $(r)^{\mathtt{CBN}}\{x \setminus (u)^{\mathtt{CBN}}\}$ is placed at location l of $(s)^{\mathtt{CBN}}$. Therefore, we know that $s\{l := (r\{x \setminus u\})^{\mathtt{CBN}}\} \neq s\{l := (r)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}\}$, and thus $((t)^{\mathtt{CBN}}, s\{l := (r\{x \setminus u\})^{\mathtt{CBN}}\}) \neq ((t)^{\mathtt{CBN}}, s\{l := (r)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}\})$. However, if we allow administrative steps to occur at the parameter position of operations, then we can evaluate $(r\{x \setminus u\})^{\mathtt{CBN}}$ to $(r)^{\mathtt{CBN}}\{x \setminus (u)^{\mathtt{CBN}}\}$ before applying any `set`-steps, obtaining:

$$(\mathtt{set}((l, !(r)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}), (t)^{\mathtt{CBN}}), (s)^{\mathtt{CBN}}) \Rightarrow_{\mathtt{ADMIN}} (\mathtt{set}((l, !(r\{x \setminus u\})^{\mathtt{CBN}}), (t)^{\mathtt{CBN}}), (s)^{\mathtt{CBN}}).$$

The following lemmas tell us that administrative reduction steps allow us to show that the CBN translation preserves normal forms, up-to the presence of $\beta_!$-redexes. Crucially, this then means that the number of β-steps needed to `CBN`-normalize a term is preserved by the CBN translation.

Lemma 7.37 (Stability of Administrative Steps)**.** *Let $(t, s) \in \Lambda^{\mathcal{G}}_{\circ} \times \mathcal{S}_{\circ}$ and $(t, s) \Rightarrow_{\mathtt{ADMIN}} (u, s)$. Then, u has the same top level constructor (application, abstraction, dereliction, bang-term, or operation) than t.*

Lemma 7.38 (Non-Interference of Single Administrative Steps). *Let* $S \in \{\beta_{\mathsf{v}}, \mathtt{get}, \mathtt{set}\}$, *and* $(t,s) \in \Lambda^{\mathcal{G}}_{\circ} \times \mathcal{S}_{\circ}$. *If* $((t,s))^{\mathtt{CBN}} \not\Rightarrow_{\mathtt{CBPV}(S)}$ *and* $((t,s))^{\mathtt{CBN}} \Rightarrow_{\mathtt{ADMIN}} (u,(s)^{\mathtt{CBN}})$, *then* $(u,(s)^{\mathtt{CBN}}) \not\Rightarrow_{\mathtt{CBPV}(S)}$.

Corollary 7.39 (Non-Interference of Administrative Steps). *Let* $S \in \{\beta_{\mathsf{v}}, \mathtt{get}, \mathtt{set}\}$, *and* $(t,s) \in \Lambda^{\mathcal{G}}_{\circ} \times \mathcal{S}_{\circ}$. *If* $((t,s))^{\mathtt{CBN}} \not\Rightarrow_{\mathtt{CBPV}(S)}$ *and* $((t,s))^{\mathtt{CBN}} \Rightarrow\!\!\!\!\Rightarrow_{\mathtt{ADMIN}} (u,(s)^{\mathtt{CBN}})$, *then* $(u,(s)^{\mathtt{CBN}}) \not\Rightarrow_{\mathtt{CBPV}(S)}$.

Lemma 7.40 (Preservation under Substitution). *Let* $s \in \mathcal{S}$, $t \in \Lambda^{\mathcal{G}}$, $u \in \Lambda^{\mathcal{G}}_{\circ}$, *and* $v \in \mathrm{V}^{\mathcal{G}}_{\circ}$:

1. $((t)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}, s) \Rightarrow^{=}_{\beta_!} \Rightarrow\!\!\!\!\Rightarrow_{\mathtt{ADMIN}} ((t\{x \setminus u\})^{\mathtt{CBN}}, s)$.
2. $((t)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\}, s) = ((t\{x \setminus v\})^{\mathtt{CBV}}, s)$.

Now, we show that one-step simulations are correct. In particular, we show that reduction steps are preserved by the CBN and CBV translations, but only up-to some additional $\beta_!$-steps, possibly occurring at administrative contexts in the case of the CBN translation.

Lemma 7.41 (Correctness of One-Step Simulations). *Let* $s \in \mathcal{S}_{\circ}$ *and* $t \in \Lambda^{\mathcal{G}}_{\circ}$.

1. *Let* $S \in \{\beta, \mathtt{get}, \mathtt{set}\}$. *If* $(t,s) \Rightarrow_{\mathtt{CBN}(S)} (u,q)$, *then* $((t,s))^{\mathtt{CBN}} \Rightarrow\!\!\!\!\Rightarrow^{1\cdot S' + m\cdot\beta_!}_{\mathtt{CBPV}\cup\mathtt{ADMIN}} ((u,q))^{\mathtt{CBN}}$, *for some* $m \geq 0$, *such that* $S' = \beta_{\mathsf{v}}$ *if* $S = \beta$, *and* $S' = S$, *otherwise.*
2. *Let* $S \in \{\beta_{\mathsf{v}}, \mathtt{get}, \mathtt{set}\}$. *If* $(t,s) \Rightarrow_{\mathtt{CBV}(S)} (u,q)$, *then* $((t,s))^{\mathtt{CBV}} \Rightarrow\!\!\!\!\Rightarrow^{1\cdot S + m\cdot\beta_!}_{\mathtt{CBPV}} ((u,q))^{\mathtt{CBV}}$, *for some* $m \geq 0$.

Before presenting the main correctness result, we recall that a simulation is full when every reduction step in the source calculus is faithfully reproduced in the target, possibly with additional administrative steps that do not affect quantitative measures.

Theorem 7.42 (Correctness of Full Simulations). *Let* $s \in \mathcal{S}_{\circ}$, *and* $t \in \Lambda^{\mathcal{G}}_{\circ}$.

1. *If* $(t,s) \overset{n\cdot\beta + g\cdot\mathtt{get} + s\cdot\mathtt{set}}{\Rightarrow\!\!\!\!\Rightarrow}_{\mathtt{CBN}} (u,q)$, *then* $((t,s))^{\mathtt{CBN}} \overset{n\cdot\beta_{\mathsf{v}} + m\cdot\beta_! + g\cdot\mathtt{get} + s\cdot\mathtt{set}}{\Rightarrow\!\!\!\!\Rightarrow}_{\mathtt{CBPV}\cup\mathtt{ADMIN}} ((u,q))^{\mathtt{CBN}}$, *for some* $m \geq 0$.
2. *If* $(t,s) \overset{n\cdot\beta_{\mathsf{v}} + g\cdot\mathtt{get} + s\cdot\mathtt{set}}{\Rightarrow\!\!\!\!\Rightarrow}_{\mathtt{CBV}} (u,q)$, *then* $((t,s))^{\mathtt{CBV}} \overset{n\cdot\beta_{\mathsf{v}} + m\cdot\beta_! + g\cdot\mathtt{get} + s\cdot\mathtt{set}}{\Rightarrow\!\!\!\!\Rightarrow}_{\mathtt{CBPV}} ((u,q))^{\mathtt{CBV}}$, *for some* $m \geq 0$.

This establishes that the $\lambda!^{\mathcal{G}}$-calculus accurately reproduces the behavior of both the CBN and CBV $\lambda^{\mathcal{G}}$-calculi, forming the foundation for a unified quantitative analysis.

Example 7.43 (Full Simulations). *Recall the CBN reduction sequence for the configuration in Example 7.14:*

$$(\mathtt{set}((l, (\lambda x.x)\,\lambda y.y), \mathtt{get}(l, \lambda z.z\,z)), s) \overset{3\cdot\beta + 1\cdot\mathtt{get} + 1\cdot\mathtt{set}}{\Rightarrow\!\!\!\!\Rightarrow}_{\mathtt{CBN}} (\lambda y.y, s\{l := (\lambda x.x)\,\lambda y.y\});$$

and the CBV reduction sequence for the configuration in Example 7.16:

$$(\mathtt{set}((l, \lambda x.x), \mathtt{get}(l, \lambda y.y\,((\lambda z.z)\,y))), s) \overset{2\cdot\beta_{\mathsf{v}} + 1\cdot\mathtt{get} + 1\cdot\mathtt{set}}{\Rightarrow\!\!\!\!\Rightarrow}_{\mathtt{CBV}} (\lambda x.x, s\{l := \lambda x.x\}).$$

In Example 7.23, we can see how Theorem 7.42 holds for the translation of the above configurations.

The following is the CBPV reduction sequence corresponding to the CBN translation of the first configuration:

$$(\mathtt{set}((l, !((\lambda x.\mathtt{der}(x))\, !\lambda y.\mathtt{der}(y)))), \mathtt{get}(l, \lambda z.\mathtt{der}(z)\, !\mathtt{der}(z))), q)$$

$$\overset{3\cdot\beta_{\mathsf{v}}+5\cdot\beta_{!}+1\cdot\mathtt{get}+1\cdot\mathtt{set}}{\Rrightarrow_{\mathsf{CBPV}}}$$

$$(\lambda y.\mathtt{der}(y), q\{l := !(\lambda x.\mathtt{der}(x))\, !\lambda y.\mathtt{der}(y)\})$$

The following is the CBPV reduction sequence corresponding to the CBV translation of the second configuration:

$$(\mathtt{set}((l, !\lambda x.x), \mathtt{get}(l, \lambda y.\mathtt{der}(y)\, ((\lambda z.z)\, y))), q) \overset{2\cdot\beta_{\mathsf{v}}+2\cdot\beta_{!}+1\cdot\mathtt{get}+1\cdot\mathtt{set}}{\Rrightarrow_{\mathsf{CBPV}}} (!\lambda x.x, q\{l := !\lambda x.x\})$$

In this subsection we have shown how the operational semantics fo the $\lambda!^{\mathcal{G}}$-calculus subsumes the operational semantics of the CBN and CBV $\lambda^{\mathcal{G}}$-calculi. In the next subsection, we will present a non-idempotent intersection type system for the $\lambda!^{\mathcal{G}}$-calculus and show that it subsumes the type systems for the CBN and CBV $\lambda^{\mathcal{G}}$-calculi.

7.3 State-Passing Non-Idempotent Intersection Types

Recall the sets of types for CBN and CBV, as defined in Definition 4.1. The set of types denoted by A is composed of types of different shapes, each reflecting the behavior of the term to which it is assigned. In the CBN, CBV, and CBPV calculi, terms either behave like functions or arguments, and that is roughly it. However, by extending these calculi with operations, we have extended their behaviors with effects. In general, the behavior of an operation depends on the type of effect that is determined by its algebraic theory. As we have seen in Section 5.1, notions of computation (effects) described by algebraic theories determine monads, and those monads reflect the behavior of the operations. This means that it is possible to reflect the behavior of the operations added to the syntax of the CBN and CBV calculi in Chapter 2, by lifting the set of types in Definition 4.1 according to the appropriate monad (see Section 5.1). But the difference between the two calling mechanisms must also be taken into account [31]:

> "... the semantics of functional types should reflect the choice of calling mechanism:
>
> - in call-by-value a procedure of type $A \to B$ expects a value of type A and computes a result of type B, so the interpretation of $A \to B$ is $A \to TB$;
> - in call-by-name a procedure of type $A \to B$ expects a computation of type A, which is evaluated only when needed, and computes a result of type B, so the interpretation of $A \to B$ is $TA \to TB$."

This observation has a direct impact on how function types must be lifted in the presence of effects: depending on the calling strategy, the set of types is lifted according to Moggi's following monadic translations.

Let T be the functor of some monad $\mathbb{T}$ as defined in Section 5.1, and $A \to B$ be an arrow type, for some types A and B. Then

$$(A \to B)^{\mathtt{M}_{\mathtt{CBN}}} := T(A)^{\mathtt{M}_{\mathtt{CBN}}} \to T(B)^{\mathtt{M}_{\mathtt{CBN}}} \qquad (A \to B)^{\mathtt{M}_{\mathtt{CBV}}} := (A)^{\mathtt{M}_{\mathtt{CBV}}} \to T(B)^{\mathtt{M}_{\mathtt{CBV}}}.$$

If we take $\mathbb{T}$ to be the monad for global state $\mathbb{G}$ and the set of states $\mathcal{S}$ to be the carrier $|\mathtt{W}|$ defined in Example 5.10, the monadic type TA becomes:

$$GA = |\mathtt{W}| \to (A \times |\mathtt{W}|).$$

This matches exactly the interpretation of $\Sigma_{\mathcal{T}}$-trees in Definition 5.31 and is justified via the equivalence between comodels and stateful runners [38].

Notice that $|\mathtt{W}|$ denotes the set of elements of the carrier of comodel $\mathtt{W}$. This means that $|\mathtt{W}| \to (A \times |\mathtt{W}|)$ denotes a λ-term that expects an element of type $|\mathtt{W}|$ as the input state, and returns an element of type $|\mathtt{W}|$ as the output state, once it finishes computing the result of the computation (of type A). In our comodel-based operational semantics, every computation requires an initial state, so the type of the output state is completely determined by the type of the initial state. Thus, instead of assigning to λ-terms monadic types of the form GA, we are going to assign to them monadic types of the form:

$$\mathtt{w} \to (A \times \mathtt{w}'),$$

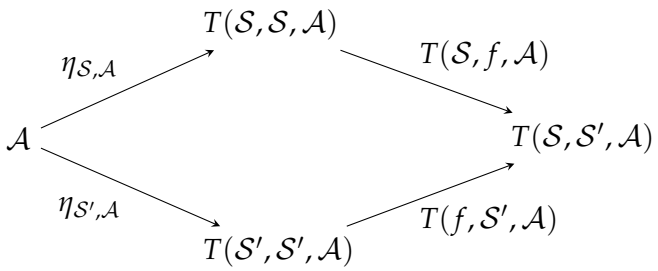
where $\mathtt{w}, \mathtt{w}' \in |\mathtt{W}|$. Clearly, this is not an element of the global state monad over the set of types, so we will instead consider a notion of the global state monad parameterised by $|\mathtt{W}|$. To do so, we briefly introduce Atkey's parameterised notions of computation [104].

Definition 7.44 (Parameterised Monads [104])**.** Given a category $\mathbb{C}$ with finite products and a category $\mathbb{S}$, an $\mathbb{S}$-*parameterised monad* (T, η, μ) on $\mathbb{C}$ consists of the following∗:

- A functor $T : \mathbb{S}^{\mathrm{op}} \times \mathbb{S} \times \mathbb{C} \to \mathbb{C}$;

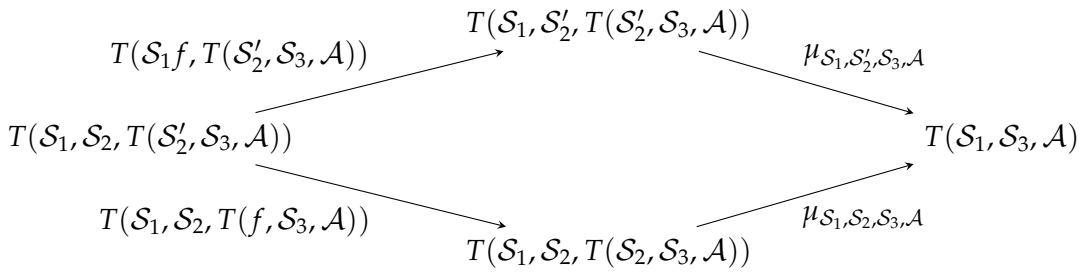
∗We omit the mention of strength since we are working in `Set`, where it comes for free.

- A unit $\eta_A^{\mathcal{S}} : \mathcal{A} \to T(\mathcal{S}, \mathcal{S}, \mathcal{A})$, natural in $\mathcal{A}$ and dinatural in $\mathcal{S}$. The *dinaturality* requirement amounts to the commutativity of the diagram



  for every arrow $f : \mathcal{S} \to \mathcal{S}'$ in $\mathbb{S}$;

- A multiplication $\mu_{\mathcal{A}}^{\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3} : T(\mathcal{S}_1, \mathcal{S}_2, T(\mathcal{S}_2, \mathcal{S}_3, \mathcal{A})) \to T(\mathcal{S}_1, \mathcal{S}_3, \mathcal{A})$, natural in $\mathcal{S}_1, \mathcal{S}_3$ and $\mathcal{A}$, and dinatural in $\mathcal{S}_2$. The *dinaturality* requirement amounts to the commutativity of the diagram



  for all $f : \mathcal{S}_2 \to \mathcal{S}_2'$.

We thus take $\mathbb{G} = (G, \eta, \mu)$ to be the $|\mathbb{W}|$-parameterized monad whose functor G is as follows

$$G(\mathtt{w}, \mathtt{w}', X) := \mathtt{w} \to (X \times \mathtt{w}');$$

and unit $\eta_X^{\mathtt{w}} : X \to G(\mathtt{w}, \mathtt{w}, X)$, multiplication $\mu_X^{\mathtt{w},\mathtt{w}',\mathtt{w}''} : G(\mathtt{w}, \mathtt{w}', G(\mathtt{w}', \mathtt{w}'', X)) \to G(\mathtt{w}, \mathtt{w}'', X)$, and Kleisli extension $\cdot^{\dagger}_{\mathtt{w},\mathtt{w}',\mathtt{w}''} : X \to G(\mathtt{w}', \mathtt{w}'', X) \to G(\mathtt{w}, \mathtt{w}', X) \to G(\mathtt{w}, \mathtt{w}'', X)$ are defined just as for the global state monad.

This allows us to keep a precise track of how the initial state (of type $\mathtt{w} \in |\mathbb{W}|$) reaches the final state (of type $\mathtt{w}' \in |\mathbb{W}|$), but only if we choose the right notion of state. The obvious choice would be to pick the same notion of state as the one for the operational semantics, *i.e.* if for the operational semantics $\mathcal{S} = \mathcal{V}^{\mathcal{L}}$, then for the type system we should pick $\mathcal{S} = M^{\mathcal{L}}$, where M is the set of types of values. However, there is a crucial mismatch due to the behavior of the `get` operations over the same location discussed in Remark 7.2! Indeed, while sequences of `get` operations over the same location always return the same value, each of these occurrences can be used differently, and thus have a different type. Therefore, we must consider the set of states to be a mapping from

locations to *lists* of types, each corresponding to a different use of the same value by a different `get` operation, *i.e.* $\mathcal{S} = (M^*)^{\mathcal{L}}$. This representation makes explicit how each use of the state contributes separately to the quantitative behavior of a program, allowing distinct state accesses to be counted independently.

We are now ready to introduce the non-idempotent type systems for the CBN and CBV $\lambda^{\mathcal{G}}$-calculi.

7.3.1 Call-By-Name

We now adapt the language of types for the CBN λ-calculus (see Definition 4.1) to account for the extended syntax and operational semantics.

Definition 7.45 (CBN Types)**.** The set of types $\mathcal{T}y^{\mathcal{G}}_{\mathtt{CBN}}$ for the CBN $\lambda^{\mathcal{G}}$-calculus is generated by the following extension of the grammar in Definition 4.1 for the CBN λ-calculus:

$$
\begin{array}{lrcl}
\text{List Types:} & L, L' & ::= & [\,] \mid M \cdot L' \\
\text{State Types:} & S, Q & ::= & \{l \mapsto L_l\}_{l \in \mathcal{L}} \\
\text{Monadic Types:} & \mathbb{A} & ::= & S \Rightarrow P \\
\text{Product Types:} & P & ::= & A \times S \\
\text{Multi-Monadic Types:} & M, N & ::= & \{\!\{\, \mathbb{A}_i \,\}\!\}_{i \in \mathtt{I}} \\
\text{Types:} & A & ::= & \star \mid M \to \mathbb{A}
\end{array}
$$

We use to highlight the parts of the grammar that are similar to those in Definition 4.1, and note that $\mathbb{A}$ expands to $S \Rightarrow (A \times Q)$, and P to $A \times Q$, for some type A and some $S, Q \in \mathcal{S}$. Moreover, given a multi-monadic type M and a list type L', we write $L := M \cdot L'$ for the list type resulting from adding M to the head of the list L. Moreover, we define $\mathtt{head}(L) := M$ and $\mathtt{tail}(L) := L'$. Notice that the carrier $|\mathtt{W}|$ is the set $\mathcal{S}$ of state types.

Definition 7.46 (Operations On State Types)**.** Let $S := \{l \mapsto L_l\}_{l \in \mathcal{L}}$ be a state type, and $l' \in \mathcal{L}$. Then

- we write $S\,(l')$ for the list type at location l' of S, *i.e.* $S\,(l') := L_{l'}$;
- and $S\{l' := L\}$ for the state obtained by replacing the list type at location l' of S with L, *i.e.* such that for all $l_0 \in \mathcal{L}$, $S\{l' := L\}\,(l_0) := S\,(l_0)$, if $l_0 \neq l'$, and $S\{l' := L\}\,(l_0) := L$, otherwise.

Monadic types are obtained by decorating types with information about the input and output states, *i.e.* in a monadic type $S \Rightarrow (A \times Q)$, S is the type of the input state, and Q is the type of the output state. Also, notice that, instead of the usual arrow $\to$ (used for functions), we opted to use a different arrow $\Rightarrow$ for monadic types. This is merely an aesthetical choice. Indeed, $S \Rightarrow (A \times Q)$

is used to denote the fact that $\lambda^{\mathcal{G}}$-terms behave as a function that takes a starting state and returns a value together with a final state, *i.e.* a configuration of product type $A \times Q$. Finally, since every $\lambda^{\mathcal{G}}$-term now behaves as a state carrying function, abstractions must be typed with an arrow type that expects a multi-set of monadic types, *i.e.* a multi-monadic type. List types are used to type the term at each location as many times as necessary, once for each time it is consumed by a `get` operations. Then, since CBN states are mappings from locations to $\lambda^{\mathcal{G}}$-terms, CBN state types are mappings from locations to list types.

Definition 7.47 (CBN Typing Environments)**.** *CBN typing environments* are denoted by Γ, Δ and *defined as functions from variables to multi-monadic types* that assign the empty multi-monadic type $\{\!\{\ \}\!\}$ to all but a finite set of variables.

The *domain of a typing environment*, the *union of typing environments*, and any other properties of typing environments, are defined as in Definition 4.2 for the $\lambda^{\mathcal{G}}$-calculus.

Definition 7.48 (CBN Typing Judgments)**.** There are two kinds of typing judgments:

- *CBN typing judgments for terms* are denoted by $\Gamma \vdash t : \mathbb{A}$ (resp. M) and, whenever Γ is empty, we may write $\vdash t : \mathbb{A}$ (resp. M) instead of $\emptyset \vdash t : \mathbb{A}$ (resp. M).
- *CBN typing judgments for configurations* are denoted by $\Gamma \vdash (t, s) : P$ and, whenever Γ is empty, we may write $\vdash (t, s) : P$ instead of $\emptyset \vdash (t, s) : P$.

CBN typing rules are presented just as in Definition 4.4 for the λ-calculus, along with any additional notation and definitions.

Definition 7.49 (CBN Type System)**.** The *CBN Type System* $\mathcal{N}_{\mathcal{G}}$ assigns CBN (multi-monadic) types to $\lambda^{\mathcal{G}}$-terms, and consists of the following translations of the typing rules in Definition 4.7 for the CBN λ-calculus:

$$\frac{}{x : \{\!\{ \mathbb{A} \}\!\} \vdash x : \mathbb{A}}\ \mathtt{Ax} \qquad \frac{\Gamma \vdash t : \mathbb{A}}{\Gamma \setminus\!\setminus x \vdash \lambda x.t : S \Rightarrow ((\Gamma(x) \to \mathbb{A}) \times S)}\ \mathtt{Abs}$$

$$\frac{}{\emptyset \vdash \lambda x.t : S \Rightarrow (\star \times S)}\ \mathtt{Ans}$$

$$\frac{\Gamma \vdash t : S \Rightarrow ((M \to (Q \Rightarrow P)) \times Q) \qquad \Delta \vdash u : M}{\Gamma + \Delta \vdash t\, u : S \Rightarrow P}\ \mathtt{App} \qquad \frac{(\Gamma_i \vdash t : \mathbb{A}_i)_{i \in \mathtt{I}}}{+_{i \in \mathtt{I}} \Gamma_i \vdash t : \{\!\{ \mathbb{A}_i \}\!\}_{i \in \mathtt{I}}}\ \mathtt{Many}$$

Along with the following six additional rules (recall Definition 7.4):

$$\frac{\Gamma; x : \mathtt{head}(S\,(l)) \vdash t : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P}{\Gamma \vdash \mathtt{get}(l, \lambda x.t) : S \Rightarrow P}\ \mathtt{Get} \qquad \frac{\Gamma \vdash u : L \quad \Delta \vdash t : S\{l := L\} \Rightarrow P}{\Gamma + \Delta \vdash \mathtt{set}((l,u),t) : S \Rightarrow P}\ \mathtt{Set}$$

$$\frac{\Gamma \vdash t : M \quad \Delta \vdash t : L}{\Gamma + \Delta \vdash t : M \cdot L}\ \mathtt{List} \qquad \frac{}{\varnothing \vdash t : [\,]}\ \mathtt{EList}$$

$$\frac{(\Gamma_l \vdash t_l : L_l)_{l \in \mathcal{L}}}{+_{l \in \mathcal{L}} \Gamma_l \vdash \{l \mapsto t_l\}_{l \in \mathcal{L}} : \{l \mapsto L_l\}_{l \in \mathcal{L}}}\ \mathtt{State} \qquad \frac{\Gamma \vdash t : S \Rightarrow P \quad \Delta \vdash s : S}{\Gamma \vdash (t,s) : P}\ \mathtt{Conf}$$

We use to highlight the parts of the first set of rules that are similar to those in Definition 4.7, and note that $\mathbb{A}$ expands to $Q \Rightarrow (A \times R)$, and P to $A \times R$, for some type A and some $Q, R \in \mathcal{S}$.

Typing Rules. Rules `Ax` and `Many` are simple adaptations of the corresponding rules in Definition 4.7. The same could be said of rules `Abs` and `Ans`, but they are obtained by (implicitly) applying the η^S_A transformation to abstractions. Rule `App` is the one that looks more complicated. We start by noticing that, if the monadic decorations are omitted, then we are left with a rule looking like the corresponding one in Definition 4.7. This suggests that, to explain the rule, it is enough to explain the decorations. The right premise of the rule has no top-level decorations because in CBN arguments are never evaluated. The left premise of the rule has a top level decoration that denotes the fact that t is evaluated to an abstraction of type $M \to (Q \Rightarrow P)$, where $P = A \times R$, and that the (current) state (of type S) is updated to a state of type Q. Then, since the argument is consumed without being evaluated, the state is not changed and thus the decoration on the type of the body of the abstraction denotes the fact that the body is evaluated to a term of type A and that the (current) state (of type Q) is updated to a state of type R. The decoration on the type of the conclusion of the rule summarizes this behavior and is obtained by (implicitly) applying the $\mu^{S,Q,R}_A$ transformation to the application. The five additional rules `Get`, `Set`, `List`, `State`, and `Conf` are used to type operations, states, and configurations. We start by explaining rules `List`, `State`, and `Conf`, as they are the simpler ones. Rule `List` simply allows us to type a term multiple times (similarly to rule `Many`) and to collect this information into a list, and rule `State` allows us to type states (mappings of locations to arguments) with state types (mappings of locations to lists of multi-monadic types, *i.e.* the type of arguments in CBN). Rule `Conf` allows us to type a term-state pair by typing the term with a type, and the state with the state type corresponding to the type of the initial state. Finally, we explain rules `Get` and `Set`. The meaning of $\mathtt{get}(l, \lambda x.t)$ is given by the cointerpretation of operation `get`: the occurrences of x in t are replaced with the term at location l of the state. Let us call this term u. Notice that u is not consumed by the operation. It is simply copied, and thus remains accessible to any further `get` operations. Accordingly, this means that u must be typed multiple times in two distinct ways: (i) once per each time it is obtained by a `get` operation; and

(ii) once per each time it is duplicated *after* it is obtained. The first number is reflected by the fact that u must be assigned a list type, and the second number is reflected by the fact that list types are lists of multi-monadic types. In sum, the length of the list type assigned to u reflects the number of times it is copied by different `get` operations, and the size of each multi-monadic type reflects the number of times u is copied after being copied by each `get` operation. This behavior is reflected in rule `Get` as follows. Since the term at location l (say u) is copied by the `get` operation, x must have the type corresponding to the type at the top of the list of types at location l in S (the type of the initial state). Then, t can contain other occurrences of `get` operations, but these no longer have access to the multi-monad type at the top of the list of types at location l, otherwise one would not be able to count how many times u is used by *different* `get` operations. Instead, any further `get` operations must use further multi-monadic types in the list of types assigned to u, and thus S is updated accordingly. The meaning of $\mathtt{set}((l,u),t)$ is given by the cointerpretation operation `set`, but is quite simpler. Operation `set` simply replaces the term at location l of the state with u, and thus the list type at location l in S is accordingly replaced by the list type assigned to u.

Remark 7.50 (On rules `Abs` and `Ans`...). In the CBN system, rules `Abs` and `Ans` are parametric in the input and output state types. Consequently, any type derivation for an abstraction ending in `Abs` or `Ans` may be instantiated with arbitrary (but equal) input and output state types.

Remark 7.51 (Minimal Typings for Global State). To consider only minimal typings it is necessary to understand what are the minimal (or *tight*) state types, *i.e.* what is the minimal information that can be extracted from a state. Notice that each location is mapped to a value, each value can be typed multiple times, but whenever a value cannot be copied by any further `get` operations, it should be assigned a list type of zero length. Moreover, recall rule `Set` from Definition 7.49:

$$\frac{\Gamma \vdash u : L \qquad \Delta \vdash t : S\{l := L\} \Rightarrow P}{\Gamma + \Delta \vdash \mathtt{set}((l,u),t) : S \Rightarrow P}\ \mathtt{Set}$$

Notice that condition $S\,(l) = [\,]$ is necessary in order to guarantee that `get`-steps can only reduce the size of the type derivation by a single `Get` rule. Indeed, if the condition is omitted, then $S\,(l)$ might be typed by an arbitrarily big type derivation, and thus the size of the derivation might decrease by an arbitrary amount once $S\,(l)$ is replaced by some other list type. In sum, requiring that $S\,(l) = [\,]$ in rule `Set` is necessary in order to obtain *tight* measures, but not otherwise. As an example, consider the following configuration:

$$c = (\mathtt{set}((l,t),u), \{l \mapsto r\}).$$

Let us assume that c is typable by the following type derivation Φ:

$$\dfrac{\dfrac{\Phi_1 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash t : L \quad \Phi_2 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash u : \{l \mapsto L\} \Rightarrow P}{\emptyset \vdash \mathtt{set}((l,t),u) : \{l \mapsto L'\} \Rightarrow P}\,\mathtt{Set} \quad \dfrac{\Phi_3 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash r : L'}{\emptyset \vdash \{l \mapsto r\} : \{l \mapsto L'\}}\,\mathtt{State}}{\emptyset \vdash (\mathtt{set}((l,t),u), \{l \mapsto r\}) : P}\,\mathtt{Conf}$$

Notice that the size of Φ not only depends on the size of the type derivation for $\mathtt{set}((l,t),u)$, but also on that for the state. Indeed,

$$\#_{\mathtt{Abs},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{Abs},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{Abs},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{Set} + \#_{\mathtt{Abs},\mathtt{Get},\mathtt{Set}}(\Phi_3).$$

Now, consider what happens after a single $\mathtt{set}$-steps is applied to s:

$$c = (\mathtt{set}((l,t),u), \{l \mapsto r\}) \Rightarrow_{\mathtt{set}} (u, \{l \mapsto t\}) = c'$$

In particular, notice that r is completely overwritten by t after the $\mathtt{set}$-step is performed. Now, consider the following possible type derivation Φ' for c' (built from Φ according to Lemma 7.62):

$$\dfrac{\Phi_2 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash u : \{l \mapsto L\} \Rightarrow P \quad \dfrac{\Phi_1 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash t : L}{\emptyset \vdash \{l \mapsto t\} : \{l \mapsto L\}}\,\mathtt{State}}{\emptyset \vdash (u, \{l \mapsto t\}) : P}\,\mathtt{Conf}$$

As expected, since r does not appear in c', Φ_3 does not appear in Φ'. Moreover, this means that the size of Φ' now only depends on the size of the derivation for u and t. That is,

$$\#_{\mathtt{Abs},\mathtt{Get},\mathtt{Set}}(\Phi') = \#_{\mathtt{Abs},\mathtt{Get},\mathtt{Set}}(\Phi_2) + \#_{\mathtt{Abs},\mathtt{Get},\mathtt{Set}}(\Phi_1).$$

Therefore,

$$\#_{\mathtt{Abs},\mathtt{Get},\mathtt{Set}}(\Phi') = \#_{\mathtt{Abs},\mathtt{Get},\mathtt{Set}}(\Phi) - 1 \cdot \mathtt{Set} - \#_{\mathtt{Abs},\mathtt{Get},\mathtt{Set}}(\Phi_3),$$

and thus the size of Φ is only guaranteed to decrease by a single occurrence of rule $\mathtt{Set}$ after the $\mathtt{set}$-step whenever Φ_3 has size zero, which is precisely what happens whenever L' is required to be the empty list type $[\,]$.

In sum, this means that minimal typings for global state require a genuinely *global* restriction on type derivations: each occurrence of rule $\mathtt{Set}$ must require $S(l) = [\,]$. Note, however, that this is only necessary because rule $\mathtt{Set}$ overwrites the state type. For instance, in [44], rule $\mathtt{Set}$ does *not* overwrite the state type (see *e.g.* Figure 13 of [44]), and thus local restrictions on the shape of the conclusion of the final rule of type derivations were sufficient.

Definition 7.52 (Tight Typings for CBN)**.** Let $t \in \Lambda^{\mathcal{G}}_{\circ}$ and $s \in \mathcal{S}_{\circ}$. A type derivation Φ is tight if and only if it is of the form $\Phi \rhd_{\mathcal{N}} \emptyset \vdash (t,s) : \star \times \{l \mapsto [\,]\}_{l \in \mathcal{L}}$, and each occurrence of rule $\mathtt{Set}$ requires $S(l) = [\,]$. Moreover, t must be assigned $\star$, just as in Definition 4.29.

Example 7.53 (Tight Typings for CBN). *Recall type derivation* Φ *in Remark 7.51 for configuration* $(\mathtt{set}((l,t),u),\{l \mapsto r\})$*:*

$$\dfrac{\dfrac{\Phi_1 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash t : L \quad \Phi_2 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash u : \{l \mapsto L\} \Rightarrow P}{\emptyset \vdash \mathtt{set}((l,t),u) : \{l \mapsto L'\} \Rightarrow P}\,\mathtt{Set} \quad \dfrac{\Phi_3 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash r : L'}{\emptyset \vdash \{l \mapsto r\} : \{l \mapsto L'\}}\,\mathtt{State}}{\emptyset \vdash (\mathtt{set}((l,t),u),\{l \mapsto r\}) : P}\,\mathtt{Conf}$$

If we assume that Φ *is a* tight *type derivation, then* L' *must be the empty list type* $[\,]$*, and* P *must be of the form* $\star \times \{l \mapsto [\,]\}$*. That is, if* Φ *is tight, then it must be of the following form:*

$$\dfrac{\dfrac{\Phi_1 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash t : L \quad \Phi_2 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash u : \{l \mapsto L\} \Rightarrow \star \times \{l \mapsto [\,]\}}{\emptyset \vdash \mathtt{set}((l,t),u) : \{l \mapsto [\,]\} \Rightarrow \star \times \{l \mapsto [\,]\}}\,\mathtt{Set} \quad \dfrac{\Phi_3 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash r : [\,]}{\emptyset \vdash \{l \mapsto r\} : \{l \mapsto [\,]\}}\,\mathtt{State}}{\emptyset \vdash (\mathtt{set}((l,t),u),\{l \mapsto r\}) : \star \times \{l \mapsto [\,]\}}\,\mathtt{Conf}$$

We now move towards exact quantitative soundness and completeness by stating that multiset-typings can be split and merged, as expected.

Lemma 7.54 (CBN Multiset-Typings Split and Merge). *Let* $t \in \Lambda^{\mathcal{G}}$*. Then, there is* $\Phi \rhd_{\mathcal{N}_\mathcal{G}} \Gamma \vdash t : M$*, such that* $M = \sqcup_{i\in \mathtt{I}} M_i$ *if and only if there are* $\Phi_i \rhd_{\mathcal{N}_\mathcal{G}} \Gamma_i \vdash t : M_i$*, such that* $\Gamma = +_{i\in \mathtt{I}}\Gamma_i$*, and* $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = +_{i\in \mathtt{I}}\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_i)$*.*

And that typing contexts are necessarily empty whenever terms are closed.

Lemma 7.55 (Contexts and Free Variables for CBN). *If* $\Phi \rhd_{\mathcal{N}_\mathcal{G}} \Gamma \vdash t : \mathbb{A}$ *(resp. M), then* $\mathtt{dom}(\Gamma) \subseteq \mathtt{fv}(t)$*.*

The fact that having a tight type derivation with zero weight is a characteristic of normal forms (recall Definition 7.17) is formalized as follows.

Lemma 7.56 (Zero Weight Tight Type Derivations Characterize CBN-Normal Forms). *Let* $t \in \Lambda^{\mathcal{G}}_{\circ}$ *and* $s \in \mathcal{S}_\circ$*. If* $\Phi \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash (t,s) : P$ *is tight, then* $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = 0 \cdot \mathtt{App} + 0 \cdot \mathtt{Get} + 0 \cdot \mathtt{Set}$ *if and only if* $t \in \mathtt{NO}_{\mathtt{CBN}}$*.*

The following lemma ensures that all normal forms are tightly typable.

Lemma 7.57 (Tight Typability of CBN-Normal Forms). *Let* $s \in \mathcal{S}_\circ$*. If* $t \in \mathtt{NO}_{\mathtt{CBN}}$*, then there is a derivation* $\Phi \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash (t,s) : P$ *that is tight.*

Example 7.58 (Tight Type Derivation). *Let* $\mathbb{A}_0 = \{l \mapsto [\,]\} \Rightarrow ((\{\!\{\mathbb{A}_1\}\!\} \to \mathbb{A}_1) \times \{l \mapsto [\,]\})$*,* $\mathbb{A}_1 = \{l \mapsto [\,]\} \Rightarrow ((\{\!\{\mathbb{A}_2\}\!\} \to \mathbb{A}_2) \times \{l \mapsto [\,]\})$*, and* $\mathbb{A}_2 = \{l \mapsto [\,]\} \Rightarrow (\star \times \{l \mapsto [\,]\})$*. Then, we can type the configuration from Example 7.14 with the following* tight *type derivation* Φ*. Since* Φ *is quite big, we are going to split it into the following pieces.*

Let Φ_s be the following type derivation for the state:

$$\dfrac{\dfrac{}{\varnothing \vdash s\,(l) : [\,]}\,\texttt{EList}}{\varnothing \vdash s : \{l \mapsto [\,]\}}\,\texttt{State}$$

The parameter of the `set` *operation is going to be typed twice. Let the following type derivation Φ_{p1} be one of those type derivations:*

$$\dfrac{\dfrac{\dfrac{}{x : \{\!\{\mathbb{A}_1\}\!\} \vdash x : \mathbb{A}_1}\,\texttt{Ax}}{\varnothing \vdash \lambda x.x : \mathbb{A}_0}\,\texttt{Abs} \quad \dfrac{\dfrac{\dfrac{}{y : \{\!\{\mathbb{A}_2\}\!\} \vdash y : \mathbb{A}_2}\,\texttt{Ax}}{\varnothing \vdash \lambda y.y : \mathbb{A}_1}\,\texttt{Abs}}{\varnothing \vdash \lambda y.y : \{\!\{\mathbb{A}_1\}\!\}}\,\texttt{Many}}{\varnothing \vdash (\lambda x.x)\,\lambda y.y : \mathbb{A}_1}\,\texttt{App}$$

And the following type derivation Φ_{p2} be the other one:

$$\dfrac{\dfrac{\dfrac{}{x : \{\!\{\mathbb{A}_2\}\!\} \vdash x : \mathbb{A}_2}\,\texttt{Ax}}{\varnothing \vdash \lambda x.x : \mathbb{A}_1}\,\texttt{Abs} \quad \dfrac{\dfrac{}{\varnothing \vdash \lambda y.y : \mathbb{A}_2}\,\texttt{Ans}}{\varnothing \vdash \lambda y.y : \{\!\{\mathbb{A}_2\}\!\}}\,\texttt{Many}}{\varnothing \vdash (\lambda x.x)\,\lambda y.y : \mathbb{A}_2}\,\texttt{App}$$

Finally, let Φ_c be the type derivation for the continuation of the `set` *operation:*

$$\dfrac{\dfrac{\dfrac{}{z : \{\!\{\mathbb{A}_1\}\!\} \vdash z : \mathbb{A}_1}\,\texttt{Ax} \quad \dfrac{\dfrac{}{z : \{\!\{\mathbb{A}_2\}\!\} \vdash z : \mathbb{A}_2}\,\texttt{Ax}}{z : \{\!\{\mathbb{A}_2\}\!\} \vdash z : \{\!\{\mathbb{A}_2\}\!\}}\,\texttt{Many}}{z : \{\!\{\mathbb{A}_1, \mathbb{A}_2\}\!\} \vdash z\,z : \mathbb{A}_2}\,\texttt{App}}{\varnothing \vdash \texttt{get}(l, \lambda z.z\,z) : \{l \mapsto \{\!\{\mathbb{A}_1, \mathbb{A}_2\}\!\} \cdot [\,]\} \Rightarrow (\star \times \{l \mapsto [\,]\})}\,\texttt{Get}$$

Then, Φ is as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Phi_{p1} \quad \Phi_{p2}}{\varnothing \vdash (\lambda x.x)\,\lambda y.y : \{\!\{\mathbb{A}_1, \mathbb{A}_2\}\!\}}\,\texttt{Many} \quad \dfrac{}{\varnothing \vdash (\lambda x.x)\,\lambda y.y : [\,]}\,\texttt{EList}}{\varnothing \vdash (\lambda x.x)\,\lambda y.y : \{\!\{\mathbb{A}_1, \mathbb{A}_2\}\!\} \cdot [\,]}\,\texttt{List} \quad \Phi_c}{\varnothing \vdash \texttt{set}((l, (\lambda x.x)\,\lambda y.y), \texttt{get}(l, \lambda z.z\,z)) : \{l \mapsto [\,]\} \Rightarrow (\star \times \{l \mapsto [\,]\})}\,\texttt{Set} \quad \Phi_s}{\varnothing \vdash (\texttt{set}((l, (\lambda x.x)\,\lambda y.y), \texttt{get}(l, \lambda z.z\,z)), s) : \star \times \{l \mapsto [\,]\}}\,\texttt{Conf}$$

Notice that $\#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Phi) = 3 \cdot \texttt{App} + 1 \cdot \texttt{Get} + 1 \cdot \texttt{Set}$. This means that the number of occurrences of rule `App` *(resp.* `Get` *or* `Set`*) matches the number of β-steps (resp.* `get`*- or* `set`*-steps) needed to normalize the above configuration. In the following, we show that this coincidence is a property of type system $\mathcal{N}_{\mathcal{G}}$.*

The following lemma tells us how to pop multiset types from the list type assigned to some location.

Lemma 7.59. *Let $l \in \mathcal{L}$ and $s \in \mathcal{S}$. There is $\Phi \rhd_{\mathcal{N}_{\mathcal{G}}} \varnothing \vdash s : S$, such that $S\,(l) \neq [\,]$, if and only if there are $\Psi \rhd_{\mathcal{N}_{\mathcal{G}}} \varnothing \vdash s\,(l) : \texttt{head}(S\,(l))$ and $\Pi \rhd_{\mathcal{N}_{\mathcal{G}}} \varnothing \vdash s : S\{l := \texttt{tail}(S\,(l))\}$. Moreover, $\#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Phi) = \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Psi) + \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Pi)$.*

The following lemma tells us how to update the list type assigned to a location, as long as the initial list type is empty.

Lemma 7.60. *Let $l \in \mathcal{L}$ and $s \in \mathcal{S}$. There are $\Phi \triangleright_{\mathcal{N}_\mathcal{G}} \emptyset \vdash s : S$, such that $S(l) = [\,]$, and $\Psi \triangleright_{\mathcal{N}_\mathcal{G}} \emptyset \vdash t : L$ if and only if there is $\Pi \triangleright_{\mathcal{N}_\mathcal{G}} \emptyset \vdash s\{l := t\} : S\{l := L\}$. Moreover, $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi)$.*

We now show that this refined typing discipline yields exact quantitative information about stateful normalization. In particular, that typing derivations provide exact measures for the number of β-, `get`-, and `set`-steps in normalization sequences.

7.3.1.1 Quantitative Soundness

We proceed by proving a weighted version of the usual substitution lemma.

Lemma 7.61 (Weighted Substitution for CBN)**.** *Let $t, u \in \Lambda^{\mathcal{G}}$. If we have $\Phi \triangleright_{\mathcal{N}_\mathcal{G}} \Gamma; x : M \vdash t : \mathbb{A}$ (resp. N or L) and $\Psi \triangleright_{\mathcal{N}_\mathcal{G}} \emptyset \vdash u : M$, then $\Pi \triangleright_{\mathcal{N}_\mathcal{G}} \Gamma \vdash t\{x \setminus u\} : \mathbb{A}$ (resp. N or L), such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi)$.*

The exact version of the subject reduction property follows from the above lemma, but only if $S(l) = [\,]$ is added as a requirement for rule `Set` (recall Definition 7.52). This is to be assumed from now on, but no other tightness criteria is to be assumed.

Lemma 7.62 (Exact Subject Reduction for CBN)**.** *Let $t \in \Lambda^{\mathcal{G}}_\circ$, $s \in \mathcal{S}_\circ$, and $S \in \{\beta, \mathtt{get}, \mathtt{set}\}$. If $(t,s) \Rightarrow_{\mathtt{CBN}(S)} (u,q)$ and $\Phi \triangleright_{\mathcal{N}_\mathcal{G}} \emptyset \vdash (t,s) : P$, then there is $\Psi \triangleright_{\mathcal{N}_\mathcal{G}} \emptyset \vdash (u,q) : P$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) + 1 \cdot R$, where*

- *$R = \mathtt{App}$, if $S = \beta$;*
- *$R = \mathtt{Get}$, if $S = \mathtt{get}$;*
- *and $R = \mathtt{Set}$, if $S = \mathtt{set}$.*

Finally, we can show that $\mathcal{N}_\mathcal{G}$-typability is exactly quantitatively sound with respect to `CBN`-normalization.

Theorem 7.63 (Exact Soundness for CBN)**.** *Let $t \in \Lambda^{\mathcal{G}}_\circ$ and $s \in \mathcal{S}_\circ$. If there is $\Phi \triangleright_{\mathcal{N}_\mathcal{G}} \emptyset \vdash (t,s) : P$ tight, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = n \cdot \mathtt{App} + g \cdot \mathtt{Get} + s \cdot \mathtt{Set}$, then $(t,s) \overset{n\cdot\beta + g\cdot\mathtt{get} + s\cdot\mathtt{set}}{\Rightarrow\!\!\!\!\!\!\Rightarrow}_{\mathtt{CBN}} (u,q)$, such that $u \in \mathtt{NO}_{\mathtt{CBN}}$.*

7.3.1.2 Quantitative Completeness

We proceed by proving a weighted version of the usual anti-substitution lemma.

Lemma 7.64 (Weighted Anti-Substitution for CBN). *Let $t \in \Lambda^{\mathcal{G}}$ and $u \in \Lambda^{\mathcal{G}}_{\circ}$. If $\Phi \rhd_{\mathcal{N}_{\mathcal{G}}} \Gamma \vdash t\{x \setminus u\} : \mathbb{A}$ (resp. N or L), then there exist $\Psi \rhd_{\mathcal{N}_{\mathcal{G}}} \Gamma; x : M \vdash t : \mathbb{A}$ (resp. N or L) and $\Pi \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash u : M$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi)$.*

The exact version of the subject expansion property follows from the above lemma, but only if $S(l) = [\,]$ is added as a requirement for rule `Set` (recall Definition 7.52). This is to be assumed from now on, but no other tightness criteria.

Lemma 7.65 (Exact Subject Expansion for CBN). *Let $t \in \Lambda^{\mathcal{G}}_{\circ}$, $s \in \mathcal{S}_{\circ}$, and $S \in \{\beta, \mathtt{get}, \mathtt{set}\}$. If $(t,s) \Rightarrow_{\mathtt{CBN}(S)} (u,q)$ and $\Phi \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash (u,q) : P$, then there is $\Psi \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash (t,s) : P$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) + 1 \cdot R = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi)$, where*

- $R = \mathtt{App}$, *if* $S = \beta$;
- $R = \mathtt{Get}$, *if* $S = \mathtt{get}$;
- *and* $R = \mathtt{Set}$, *if* $S = \mathtt{set}$.

Finally, we can conclude that $\mathcal{N}_{\mathcal{G}}$-typability is quantitatively complete with respect to `CBN`-normalization.

Theorem 7.66 (Exact Completeness for CBN). *Let $t \in \Lambda^{\mathcal{G}}_{\circ}$ and $s \in \mathcal{S}_{\circ}$. If $(t,s) \overset{n\cdot\beta + g\cdot\mathtt{get} + s\cdot\mathtt{set}}{\Rightarrow\!\!\!\!\Rightarrow}_{\mathtt{CBN}} (u,q)$, such that $u \in \mathtt{NO}_{\mathtt{CBN}}$, then there is $\Phi \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash (t,s) : P$ tight, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = n \cdot \mathtt{App} + g \cdot \mathtt{Get} + s \cdot \mathtt{Set}$.*

7.3.2 Call-By-Value

Definition 7.67 (CBV Types). The sets of types $\mathcal{T}y^{\mathcal{G}}_{\mathtt{CBV}}$ for the CBV $\lambda^{\mathcal{G}}$-calculus is generated by the following extension of the grammar in Definition 4.1 for the CBV λ-calculus:

$$
\begin{array}{lrcl}
\text{List Types:} & L, L' & ::= & [\,] \mid M \cdot L' \\
\text{State Types:} & S, Q & ::= & \{l \mapsto L_l\}_{l \in \mathcal{L}} \\
\text{Monadic Multi-Types:} & \mathbb{M} & ::= & S \Rightarrow P \\
\text{Product Types:} & P & ::= & M \times Q \\
\text{Multi-Types} & M, N & ::= & \{\!\{ A_i \}\!\}_{i \in \mathtt{I}} \\
\text{Types:} & A & ::= & M \to \mathbb{M}
\end{array}
$$

We use to highlight the parts of the grammar that are similar to those in Definition 4.1, and note that $\mathbb{M}$ expands to $S \Rightarrow (M \times Q)$, for some monadic multi-type M and some $S, Q \in \mathcal{S}$. Moreover, operations over list types are defined as for the CBN $\lambda^{\mathcal{G}}$-calculus (see Definition 7.45). The carrier $|\mathtt{W}|$ is the set $\mathcal{S}$ of state types.

Monadic types are obtained by decorating multi-types with information about the input and output states, *i.e.* in a monadic type $S \Rightarrow (M \times Q)$, S is the type of the input state, and Q is the type of the output state. Finally, since every $\lambda^{\mathcal{G}}$-term now behaves as a state carrying function, abstractions must be typed with an arrow type that expects a multi-set of types, *i.e.* a monadic multi-type. List types are used to type the term at each location as many times as necessary, once for each time it is consumed by a `get` operations. Then, since CBV states are mappings from locations to value $\lambda^{\mathcal{G}}$-terms, CBV state types are mappings from locations to list types.

Definition 7.68 (CBV Typing Environments)**.** *CBV typing environments* are denoted by Γ, Δ and *defined as functions from variables to multi-monadic types*, that assign the empty multi-type $\{\!\{\ \}\!\}$ to all but a finite set of variables.

The *domain of a typing environment*, the *union of typing environments*, and any other properties of typing environments, are defined as in Definition 4.2 for the $\lambda^{\mathcal{G}}$-calculus.

Definition 7.69 (CBV Typing Judgments)**.** There are two kinds of typing judgments:

- *CBV typing judgments for terms* are denoted by $\Gamma \vdash t : \mathbb{M}$ (resp. M) and, whenever Γ is empty, we write $\vdash t : \mathbb{M}$ (resp. M) instead of $\emptyset \vdash t : \mathbb{M}$ (resp. M).
- *CBV typing judgments for configurations* are denoted by $\Gamma \vdash (t, s) : P$ and, whenever Γ is empty, we write $\vdash (t, s) : P$ instead of $\emptyset \vdash (t, s) : P$.

CBV typing rules are presented just as in Definition 4.4 for the λ-calculus, along with any additional notation and definitions.

Definition 7.70 (CBV Type System)**.** The *CBV Type System* $\mathcal{V}_{\mathcal{G}}$ assigns CBV (multi-monadic) types to $\lambda^{\mathcal{G}}$-terms, and consists of the following translations of the typing rules in Definition 4.8 for the CBV λ-calculus:

$$\frac{}{x : M \vdash x : M}\ \texttt{Ax} \qquad \frac{(\Gamma_i \vdash t : \mathbb{M}_i)_{i \in \mathrm{I}}}{+_{i \in \mathrm{I}} \Gamma_i \setminus\!\setminus x \vdash \lambda x.t : \{\!\{ \Gamma_i(x) \to \mathbb{M}_i \}\!\}_{i \in \mathrm{I}}}\ \texttt{Abs}$$

$$\frac{\Gamma \vdash t : S \Rightarrow (\{\!\{ M \to (R \Rightarrow P) \}\!\} \times Q) \qquad \Delta \vdash u : Q \Rightarrow (M \times R)}{\Gamma + \Delta \vdash t\, u : S \Rightarrow P}\ \texttt{App}$$

Along with the following seven additional rules:

$$\frac{\Gamma \vdash v : M}{\Gamma \vdash v : S \Rightarrow (M \times S)}\ \texttt{Lift}$$

$$\frac{\Gamma; x : \texttt{head}(S\,(l)) \vdash t : S\{l := \texttt{tail}(S\,(l))\} \Rightarrow P}{\Gamma \vdash \texttt{get}(l, \lambda x.t) : S \Rightarrow P}\ \texttt{Get} \qquad \frac{\Gamma \vdash v : L \quad \Delta \vdash t : S\{l := L\} \Rightarrow P}{\Gamma + \Delta \vdash \texttt{set}((l, v), t) : S \Rightarrow P}\ \texttt{Set}$$

$$\frac{\Gamma \vdash v : M \quad \Delta \vdash v : L}{\Gamma + \Delta \vdash v : M \cdot L}\ \texttt{List} \qquad \frac{}{\emptyset \vdash v : [\,]}\ \texttt{EList}$$

$$\frac{(\Gamma_l \vdash v_l : L_l)_{l \in \mathcal{L}}}{+_{l \in \mathcal{L}} \Gamma_l \vdash \{l \mapsto v_l\}_{l \in \mathcal{L}} : \{l \mapsto L_l\}_{l \in \mathcal{L}}}\ \texttt{State} \qquad \frac{\Gamma \vdash t : S \Rightarrow P \quad \Delta \vdash s : S}{\Gamma \vdash (t, s) : P}\ \texttt{Conf}$$

We use to highlight the parts of the first set of rules that are similar to those in Definition 4.7, and note that $\mathbb{M}$ expands to $Q \Rightarrow (M \times R)$, and P to $N \times T$, for some monadic multi-type M, type A and some $Q, R, T \in \mathcal{S}$.

Typing Rules. Rules `Ax` and `Abs` are simple adaptations of the corresponding rules in Definition 4.8, but the types of their conclusions can then be lifted to monadic types using rule `Lift`, which corresponds to (implicitly) applying the η^S_M transformation to variables and abstractions. Again, rule `App` is the one that looks more complicated. Like we did in CBN, we start by noticing that, if the monadic decorations are omitted, then we are left with a rule looking like the corresponding one in Definition 4.8. Thus, we explain the rule by explaining the decorations. The main difference between rule `App` in CBV and that same rule in CBN is that, in CBV, its right premise also has top level decorations. This is due to arguments *not* being evaluated in CBN, but being required to be evaluated (to values) before being substituted. The left premise of the rule is very similar to that of CBN. It has a top level decoration that denotes the fact that t is evaluated to an abstraction of type $M \to (R \Rightarrow (N \times T))$ (omitting the irrelevant multiset notation for singletons) and that the (current) state (of type S) is updated to a state of type Q. Then, since now arguments are evaluated before being consumed, u is evaluated to a value of type M and the (current) state (of type Q) is updated to a state of type R. The decoration on the type of the body of the abstraction denotes the fact that the body is evaluated to a value of type N and that the (current) state (of type R) is updated to a state of type T. At first glance, this intricate composition of states might seem ad-hoc. It is *not*. Composing $M \to (R \Rightarrow (N \times T))$ with $Q \Rightarrow (M \times R)$ is possible by (implicitly) applying the Kleisli extension transformation $\cdot^\dagger_{Q,R,T}$ to abstractions. The decoration on the type of the conclusion of the rule is obtained by (implicitly) applying the $\mu^{S,Q,T}_M$ transformation to the application. The six (excluding `Lift`) new additional rules `Get`, `Set`, `List`, `EList`, `State`, and `Conf` are very similar to the ones in CBN (see Definition 7.49).

Remark 7.71 (On rule `Lift`...). In the CBV system, rule `Lift`is parametric in the input and output state types. Consequently, any type derivation for a value ending in `Lift` may be instantiated with arbitrary (but equal) input and output state types. This is key in Lemmas 7.82 and 7.85.

Just like in CBN (see Remark 7.51), to consider only minimal typings, it is necessary to add condition $S(l) = [\,]$ to the premises of rule `Set`. Moreover, in contrast with CBN, tightness in CBV reflects the fact that effects may only arise during the evaluation of computations, not values.

Definition 7.72 (Tight Typings for CBV)**.** Let $t \in \Lambda_{\circ}^{\mathcal{G}}$ and $s \in \mathcal{S}_{\circ}$. A type derivation Φ is tight if and only if it is of the form $\Phi \rhd_{\mathcal{V}} \emptyset \vdash (t, s) : \{\!\{\,\}\!\} \times \{l \mapsto [\,]\}_{l \in \mathcal{L}}$, and each occurrence of rule `Set` requires $S(l) = [\,]$. Moreover, t must be assigned $\{\!\{\,\}\!\}$, just as in Definition 4.29.

We now move towards exact quantitative soundness and completeness by stating that multiset-typings can be split and merged, as expected.

Lemma 7.73 (CBV Multiset-Typings Split and Merge)**.** *Let $v \in \mathrm{V}^{\mathcal{G}}$. Then, there is $\Phi \rhd_{\mathcal{V}_{\mathcal{G}}} \Gamma \vdash v : M$, such that $M = \sqcup_{i \in \mathrm{I}} M_i$, if and only if there are $\Phi_i \rhd_{\mathcal{V}_{\mathcal{G}}} \Gamma_i \vdash v : M_i$, such that $\Gamma = +_{i \in \mathrm{I}} \Gamma_i$, and $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = +_{i \in \mathrm{I}} \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_i)$.*

That typing contexts are necessarily empty whenever terms are closed.

Lemma 7.74 (Contexts and Free Variables for CBV)**.** *If $\Phi \rhd_{\mathcal{V}_{\mathcal{G}}} \Gamma \vdash t : \mathbb{M}$ (resp. M), then $\mathtt{dom}(\Gamma) \subseteq \mathtt{fv}(t)$.*

And that the type system reflects the fact that values are not effectful.

Lemma 7.75 (CBV Values Are Not Effectful)**.** *Let $t \in \Lambda^{\mathcal{G}}$ and $\Phi \rhd_{\mathcal{V}_{\mathcal{G}}} \Gamma \vdash t : S \Rightarrow (M \times Q)$. If $t \in \mathrm{V}^{\mathcal{G}}$, then $Q = S$ and Φ ends with rule `Lift`.*

The fact that having tight type derivations with zero weight is a characteristic of normal forms that is formalized as follows.

Lemma 7.76 (Zero Weight Tight Type Derivations Characterize `CBV`-Normal Forms)**.** *Let $t \in \Lambda_{\circ}^{\mathcal{G}}$ and $s \in \mathcal{S}_{\circ}$. If $\Phi \rhd_{\mathcal{V}_{\mathcal{G}}} \emptyset \vdash (t, s) : P$ is tight, then $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = 0 \cdot \mathtt{App} + 0 \cdot \mathtt{Get} + 0 \cdot \mathtt{Set}$ if and only if $t \in \mathtt{NO}_{\mathtt{CBV}}$.*

The following lemma ensures that all normal forms are tightly typable.

Lemma 7.77 (Tight Typability of `CBV`-Normal Forms)**.** *Let $s \in \mathcal{S}_{\circ}$. If $t \in \mathtt{NO}_{\mathtt{CBV}}$, then there is a derivation $\Phi \rhd_{\mathcal{V}_{\mathcal{G}}} \emptyset \vdash (t, s) : P$ that is tight.*

Example 7.78 (Tight Type Derivation). *Let* $A_1 = \{\!\{\,\}\!\} \to (\{l \mapsto []\} \Rightarrow (\{\!\{\,\}\!\} \times \{l \mapsto []\}))$. *Then, we can type the configuration from Example 7.16 with the following* tight *type derivation* Φ. *Since* Φ *is quite big, we are going to split it into the following pieces.*

Let Φ_s *be the following type derivation for the state:*

$$\dfrac{\dfrac{}{\varnothing \vdash s\,(l) : []}\ \texttt{EList}}{\varnothing \vdash s : \{l \mapsto []\}}\ \texttt{State}$$

The parameter of the `set` *operation is only typed once. Let* Φ_p *be such a type derivation:*

$$\dfrac{\dfrac{\dfrac{}{\varnothing \vdash x : \{\!\{\,\}\!\}}\ \texttt{Ax}}{\varnothing \vdash x : \{l \mapsto []\} \Rightarrow (\{\!\{\,\}\!\} \times \{l \mapsto []\})}\ \texttt{Lift}}{\varnothing \vdash \lambda x.x : \{\!\{A_1\}\!\}}\ \texttt{Abs}$$

Let Φ_l *be the following type derivation for the operator appearing in the* `get` *operation:*

$$\dfrac{\dfrac{}{y : \{\!\{A_1\}\!\} \vdash y : \{\!\{A_1\}\!\}}\ \texttt{Ax}}{y : \{\!\{A_1\}\!\} \vdash y : \{l \mapsto []\} \Rightarrow (\{\!\{A_1\}\!\} \times \{l \mapsto []\})}\ \texttt{Lift}$$

Let Φ_r *be the following type derivation for the term at the right of the application corresponding to the continuation of the* `get` *operation:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\varnothing \vdash z : \{\!\{\,\}\!\}}\ \texttt{Ax}}{\varnothing \vdash z : \{l \mapsto []\} \Rightarrow (\{\!\{\,\}\!\} \times \{l \mapsto []\})}\ \texttt{Lift}}{\varnothing \vdash \lambda z.z : \{\!\{A_1\}\!\}}\ \texttt{Abs}}{\varnothing \vdash \lambda z.z : \{l \mapsto []\} \Rightarrow (\{\!\{A_1\}\!\} \times \{l \mapsto []\})}\ \texttt{Lift} \qquad \dfrac{\dfrac{}{\varnothing \vdash y : \{\!\{\,\}\!\}}\ \texttt{Ax}}{\varnothing \vdash y : \{l \mapsto []\} \Rightarrow (\{\!\{\,\}\!\} \times \{l \mapsto []\})}\ \texttt{Lift}}{\varnothing \vdash (\lambda z.z)\,y : \{l \mapsto []\} \Rightarrow (\{\!\{\,\}\!\} \times \{l \mapsto []\})}\ \texttt{App}$$

Finally, let Φ_c *be the following type derivation for the continuation of the* `get` *operation:*

$$\dfrac{\Phi_l \quad \Phi_r}{y : \{\!\{A_1\}\!\} \vdash y\,((\lambda z.z)\,y) : \{l \mapsto []\} \Rightarrow (\{\!\{\,\}\!\} \times \{l \mapsto \{\!\{\,\}\!\}\})}\ \texttt{App}$$

Then, Φ *is as follows:*

$$\dfrac{\dfrac{\Phi_p \quad \dfrac{}{\varnothing \vdash \lambda x.x : []}\ \texttt{EList}}{\varnothing \vdash \lambda x.x : \{\!\{A_1\}\!\} \cdot []}\ \texttt{List} \qquad \dfrac{\Phi_c}{\varnothing \vdash \texttt{get}(l, \lambda y.y\,((\lambda z.z)\,y)) : \{l \mapsto \{\!\{A_1\}\!\} \cdot []\} \Rightarrow (\{\!\{\,\}\!\} \times \{l \mapsto \{\!\{\,\}\!\}\})}\ \texttt{Get}}{\varnothing \vdash \texttt{set}((l, \lambda x.x), \texttt{get}(l, \lambda y.y\,((\lambda z.z)\,y))) : \{l \mapsto []\} \Rightarrow (\{\!\{\,\}\!\} \times \{l \mapsto []\})}\ \texttt{Set} \qquad \Phi_s}{\varnothing \vdash (\texttt{set}((l, \lambda x.x), \texttt{get}(l, \lambda y.y\,((\lambda z.z)\,y))), s) : \{\!\{\,\}\!\} \times \{l \mapsto []\}}\ \texttt{Conf}$$

Notice that $\#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Phi) = 2 \cdot \texttt{App} + 1 \cdot \texttt{Get} + 1 \cdot \texttt{Set}$. *This means that the number of occurrences of rule* `App` *(resp.* `Get` *or* `Set`*) matches the number of* β_{v}*-steps (resp.* `get`*- or* `set`*-steps) needed to normalize the above configuration. In the following, we show that this coincidence is a property of type system* $\mathcal{V}_{\mathcal{G}}$.

The following lemma tells us how to pop multiset types from the list type assigned to some location.

Lemma 7.79. *Let $l \in \mathcal{L}$ and $s \in \mathcal{S}$. There is $\Phi \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash s : S$, such that $S(l) \neq [\,]$, if and only if there are $\Psi \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash s\,(l) : \mathtt{head}(S\,(l))$ and $\Pi \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash s : S\{l := \mathtt{tail}(S\,(l))\}$. Moreover, $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi)$.*

The following lemma tells us how to update the list type assigned to a location, as long as the initial list type is empty.

Lemma 7.80. *Let $l \in \mathcal{L}$ and $s \in \mathcal{S}$. There are $\Phi \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash s : S$, such that $S(l) = [\,]$, and $\Psi \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash v : L$ if and only if there is $\Pi \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash s\{l := v\} : S\{l := L\}$. Moreover, $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi)$.*

We now establish quantitative soundness and completeness for the CBV variant. These results are analogous to those for the CBN variant, but are now concerned with the number of $\beta_\mathtt{v}$, `get`, and `set`-steps.

7.3.2.1 Quantitative Soundness

We proceed by proving a weighted version of the usual substitution lemma.

Lemma 7.81 (Weighted Substitution for CBV)**.** *Let $t \in \Lambda^{\mathcal{G}}_{\circ}$, and $v \in \mathrm{V}^{\mathcal{G}}$. If $\Phi \rhd_{\mathcal{V}_\mathcal{G}} \Gamma; x : M \vdash t : \mathbb{M}$ (resp. N or L) and $\Psi \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash v : M$, then $\Pi \rhd_{\mathcal{V}_\mathcal{G}} \Gamma \vdash t\{x \setminus v\} : \mathbb{M}$ (resp. N or L), such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi)$.*

The exact version of the subject reduction property follows from the above lemma, but only if $S(l) = [\,]$ is added as a requirement for rule `Set` (recall Definition 7.72). This is to be assumed from now on, but no other tightness criteria are to be assumed.

Lemma 7.82 (Exact Subject Reduction for CBV)**.** *Let $t \in \Lambda^{\mathcal{G}}_{\circ}$, $s \in \mathcal{S}_{\circ}$, and $S \in \{\beta_\mathtt{v}, \mathtt{get}, \mathtt{set}\}$. If $(t,s) \Rightarrow_{\mathtt{CBV}(S)} (u,q)$ and $\Phi \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash (t,s) : P$, then there is $\Psi \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash (u,q) : P$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) + 1 \cdot R$, where*

- *$R = \mathtt{App}$, if $S = \beta_\mathtt{v}$;*
- *$R = \mathtt{Get}$, if $S = \mathtt{get}$;*
- *and $R = \mathtt{Set}$, if $S = \mathtt{set}$.*

Finally, we can show that $\mathcal{V}_\mathcal{G}$-typability is exactly quantitatively sound with respect to `CBV`-normalization.

Theorem 7.83 (Exact Soundness for CBV)**.** *Let $t \in \Lambda^{\mathcal{G}}_{\circ}$. If there is $\Phi \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash (t,s) : P$ tight, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = n \cdot \mathtt{App} + g \cdot \mathtt{Get} + s \cdot \mathtt{Set}$, then $(t,s) \overset{n\cdot\beta_\mathtt{v} + g\cdot\mathtt{get} + s\cdot\mathtt{set}}{\Rightarrow_{\mathtt{CBV}}} (u,q)$, such that $u \in \mathtt{NO}_{\mathtt{CBV}}$.*

7.3.2.2 Quantitative Completeness

We proceed by proving a weighted version of the usual anti-substitution lemma.

Lemma 7.84 (Weighted Anti-Substitution for CBV). *Let $t \in \Lambda^{\mathcal{G}}$ and $v \in \mathrm{V}^{\mathcal{G}}_{\circ}$. If $\Phi \rhd_{\mathcal{V}_{\mathcal{G}}} \Gamma \vdash t\{x \setminus v\} : \mathbb{M}$ (resp. N or L), then $\Psi \rhd_{\mathcal{V}_{\mathcal{G}}} \Gamma; x : M \vdash t : \mathbb{M}$ (resp. N or L) and $\Pi \rhd_{\mathcal{V}_{\mathcal{G}}} \varnothing \vdash v : M$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi)$.*

The exact version of the subject expansion properties follows from the above lemma, but only if $S(l) = [\,]$ is added as a requirement for rule $\mathtt{Set}$. This is to be assumed from now on, but no other tightness criteria are to be assumed.

Lemma 7.85 (Exact Subject Expansion for CBV). *Let $t, u \in \Lambda^{\mathcal{G}}_{\circ}$, $s \in \mathcal{S}_{\circ}$, and $S \in \{\beta_{\mathrm{v}}, \mathtt{get}, \mathtt{set}\}$. If $(t,s) \Rightarrow_{\mathtt{CBV}(S)} (u,q)$ and $\Phi \rhd_{\mathcal{V}_{\mathcal{G}}} \varnothing \vdash (u,q) : P$, then there is $\Psi \rhd_{\mathcal{V}_{\mathcal{G}}} \varnothing \vdash (t,s) : P$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) + 1 \cdot R = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi)$, where*

- *$R = \mathtt{App}$, if $S = \beta_{\mathrm{v}}$;*
- *$R = \mathtt{Get}$, if $S = \mathtt{get}$;*
- *and $R = \mathtt{Set}$, if $S = \mathtt{set}$.*

Finally, we can conclude that $\mathcal{V}_{\mathcal{G}}$-typability is quantitatively complete with respect to $\mathtt{CBV}$-normalization.

Theorem 7.86 (Exact Completeness for CBV). *Let $t \in \Lambda^{\mathcal{G}}_{\circ}$. If $(t,s) \overset{n\cdot\beta_{\mathrm{v}}+g\cdot\mathtt{get}+s\cdot\mathtt{set}}{\Rightarrow\!\!\!\!\Rightarrow}_{\mathtt{CBV}} (u,q)$, such that $u \in \mathtt{NO}_{\mathtt{CBV}}$, then there is $\Phi \rhd_{\mathcal{V}_{\mathcal{G}}} \varnothing \vdash (t,s) : P$ tight, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = n \cdot \mathtt{App} + g \cdot \mathtt{Get} + s \cdot \mathtt{Set}$.*

7.3.3 Subsuming Call-By-Name and Call-By-Value

Having established quantitative soundness and completeness for each strategy in isolation, we now show that the $\lambda!^{\mathcal{G}}$-calculus subsumes both strategies, while preserving the quantitative correspondence between typing and reduction lengths.

Recall the set of types for the CBN and CBV $\lambda^{\mathcal{G}}$-calculus in Definition 7.45 and Definition 7.67. Notice that the set of CBN types differs from the set of CBV types in one essential way: in CBN, multi-types contain monadic types; in CBV, multi-types contain types. This is due to the difference between β- and β_{v}-steps. Indeed, while β-steps substitute variables with terms (denoted by monadic types), β_{v}-steps substitute variables with values (denoted by types). Now, recall the way in which CBPV subsumes CBN and CBV according to Theorem 7.42. Crucially, CBPV must be able to simulate both β- and β_{v}-steps. Since, in CBPV, β_{v}-steps substitute variables with values (*i.e.* variables or bang-terms), we need to encode applications according to Definition 3.12: to encode

CBN into CBPV, every argument must be encoded as a bang-term; to encode CBV into CBPV, every value must be encoded as a bang-term. Thus, in CBPV, multi-types must contain monadic types. Moreover, since the set of CBPV-normal forms contains both (closed) abstraction and values (*i.e.* bang-terms), both types and multi-types must be monadic.

Definition 7.87 (CBPV Types)**.** The sets of types $\mathcal{T}y^{\mathcal{G}}_{\mathtt{CBPV}}$ for the $\lambda!^{\mathcal{G}}$-calculus is generated by the following extension of the grammar in Definition 4.1 for the $\lambda!$-calculus:

$$\begin{array}{lrcl}
\text{List Types:} & L, L' & ::= & [\,] \mid M \cdot L' \\
\text{State Types:} & S, Q & ::= & \{l \mapsto L_l\}_{l \in \mathcal{L}} \\
\text{Monadic Types:} & \mathbb{A} & ::= & S \Rightarrow P \\
\text{Product Types:} & P & ::= & A \times S \\
\text{Multi-Monadic Types} & M, N & ::= & \{\!\{\, \mathbb{A}_i \,\}\!\}_{i \in \mathtt{I}} \\
\text{Types:} & A & ::= & \star \mid M \mid M \to \mathbb{A}
\end{array}$$

We use to highlight the parts of the grammar that are similar to those in Definition 4.1, and note that $\mathbb{A}$ expands to $S \Rightarrow (A \times Q)$, for some type A, and some $S, Q \in \mathcal{S}$. Moreover, operations over list types are defined as for the CBN and CBV $\lambda^{\mathcal{G}}$-calculi (see Definition 7.45 or Definition 7.67, respectively).

Intuitively, CBPV types combine the CBN view (computations as monadic values) with the CBV view (values as first-class entities), which explains why both types and multi-types are monadic.

Definition 7.88 (CBPV Typing Environments)**.** *CBPV typing environments* are denoted by Γ, Δ and *defined as functions from variables to multi-monadic types* that assign the empty multi-monadic type $\{\!\{\,\}\!\}$ to all but a finite set of variables.

The *domain of a typing environment*, the *union of typing environments*, and any other properties of typing environments, are defined as in Definition 4.2 for the λ-calculus.

Definition 7.89 (CBPV Typing Judgments)**.** There are two kinds of typing judgments:

- *CBPV typing judgments for terms* are denoted by $\Gamma \vdash t : \mathbb{A}$ (resp. M) and, whenever Γ is empty, we may write $\vdash t : \mathbb{A}$ (resp. M) instead of $\emptyset \vdash t : \mathbb{A}$ (resp. M).
- *CBPV typing judgments for configurations* are denoted by $\Gamma \vdash (t, s) : P$ and, whenever Γ is empty, we may write $\vdash (t, s) : P$ instead of $\emptyset \vdash (t, s) : P$.

CBPV typing rules are presented just as in Definition 4.4 for the λ-calculus, along with any additional notation and definitions.

Definition 7.90 (CBPV Type System)**.** The *CBPV Type System* $\mathcal{P}_{\mathcal{G}}$ assigns CBPV (multi-monadic) types to $\lambda!^{\mathcal{G}}$-terms and consists of the following translations of the typing rules in Definition 4.21 for the $\lambda!$-calculus:

$$\frac{}{x : M \vdash x : M}\ \texttt{Ax} \qquad \frac{\Gamma \vdash t : \mathbb{A}}{\Gamma \setminus\setminus x \vdash \lambda x.t : S \Rightarrow ((\Gamma(x) \to \mathbb{A}) \times S)}\ \texttt{Abs}$$

$$\frac{}{\emptyset \vdash \lambda x.t : S \Rightarrow (\star \times S)}\ \texttt{Ans}$$

$$\frac{\Gamma \vdash t : S \Rightarrow ((M \to (R \Rightarrow P)) \times Q) \qquad \Delta \vdash u : Q \Rightarrow (M \times R)}{\Gamma + \Delta \vdash t\, u : S \Rightarrow P}\ \texttt{App}$$

$$\frac{(\Gamma_i \vdash t : \mathbb{A}_i)_{i \in \mathrm{I}}}{+_{i \in \mathrm{I}} \Gamma_i \vdash\ !t : \{\!\{\mathbb{A}_i\}\!\}_{i \in \mathrm{I}}}\ \texttt{Bg} \qquad \frac{\Gamma \vdash t : S \Rightarrow (\{\!\{Q \Rightarrow P\}\!\} \times Q)}{\Gamma \vdash \texttt{der}(t) : S \Rightarrow P}\ \texttt{Der}$$

Along with the following seven additional rules:

$$\frac{\Gamma \vdash v : M}{\Gamma \vdash v : S \Rightarrow (M \times S)}\ \texttt{Lift}$$

$$\frac{\Gamma; x : \texttt{head}(S\,(l)) \vdash t : S\{l := \texttt{tail}(S\,(l))\} \Rightarrow P}{\Gamma \vdash \texttt{get}(l, \lambda x.t) : S \Rightarrow P}\ \texttt{Get} \qquad \frac{\Gamma \vdash v : L \qquad \Delta \vdash t : S\{l := L\} \Rightarrow P}{\Gamma + \Delta \vdash \texttt{set}((l, v), t) : S \Rightarrow P}\ \texttt{Set}$$

$$\frac{\Gamma \vdash v : M \qquad \Delta \vdash v : L}{\Gamma + \Delta \vdash v : M \cdot L}\ \texttt{List} \qquad \frac{}{\emptyset \vdash v : [\,]}\ \texttt{EList}$$

$$\frac{(\Gamma_l \vdash v_l : L_l)_{l \in \mathcal{L}}}{+_{l \in \mathcal{L}} \Gamma_l \vdash \{l \mapsto v_l\}_{l \in \mathcal{L}} : \{l \mapsto L_l\}_{l \in \mathcal{L}}}\ \texttt{State} \qquad \frac{\Gamma \vdash t : S \Rightarrow P \qquad \Delta \vdash s : S}{\Gamma \vdash (t, s) : P}\ \texttt{Conf}$$

We use to highlight the parts of the first set of rules that are similar to those in Definition 4.21, and note that $\mathbb{A}$ expands to $Q \Rightarrow (A \times R)$, and P to $A \times T$, for some type A and some $Q, R, T \in \mathcal{S}$.

Typing Rules. Rules `Ax` and `Bg`, are simple adaptations of the corresponding rule in Definition 4.21. Rules `Abs` and `Ans` are very similar to those in Definition 4.7. Rules `App` and `Lift` are very similar to those in Definition 4.8. Rule `Der` can be explained as follows. The top-level decoration of the premise of the rule denotes the fact that t is evaluated to a value of type $\{\!\{Q \Rightarrow (A \times T)\}\!\}$ and that the (current) state (of type S) is updated to a state of type Q. Moreover, since only bang-terms can be assigned multiset-types, this means that t is a bang-term. Therefore, the bang is going to be removed by the dereliction, and this corresponds to removing the multiset notation from $Q \Rightarrow (A \times T)$. Then, the term inside the bang is evaluated to a term of type A, and the (current) state (of type Q) is updated to a state of type T. The decoration on the type of the conclusion is obtained by (implicitly) applying the $\mu_A^{S,Q,T}$ transformation to the dereliction. The six (excluding rule `Lift`) new additional rules `Get`, `Set`, `List`, `EList`, `State`, and `Conf` are very similar to the ones in CBN and CBV (see Definitions 7.49 and 7.70).

Remark 7.91 (On rules `Abs`, `Ans`, and `Lift`...). In the CBPV system, rules `Abs`, `Ans`, and `Lift` are parametric in the input and output state types. Consequently, any type derivation for a abstraction ending in `Abs` or `Ans`, and any type derivation for a bang-term ending in `Lift`, may be instantiated with arbitrary (but equal) input and output state types. This is key in Lemmas 7.101 and 7.106.

Just like in CBN and CBV (see Remark 7.51), to consider only minimal typings, it is necessary to add the condition $S(l) = [\,]$ to the premises of rule `Set`.

Definition 7.92 (Tight Typings for CBPV)**.** Let $t \in !\Lambda^{\mathcal{G}}_{\circ}$ and $s \in \mathcal{S}_{\circ}$. A type derivation Φ is tight if and only if it is of the form $\Phi \rhd_{\mathcal{P}_{\mathcal{G}}} \emptyset \vdash (t,s) : A \times \{l \mapsto \{\!\{\,\}\!\}\}_{l \in \mathcal{L}}$, such that $A \in \{\star, \{\!\{\,\}\!\}\}$. This reflects the fact that clash-free CBPV normal forms are either values or suspended computations. Moreover, each occurrence of rule `Set` requires $S(l) = \{\!\{\,\}\!\}$.

We now move towards exact quantitative soundness and completeness by stating that multiset-typing can be split and merged.

Lemma 7.93 (CBPV Multiset-Typings Split and Merge)**.** *Let $t \in !\Lambda^{\mathcal{G}}$. Then, there is $\Phi \rhd_{\mathcal{P}_{\mathcal{G}}} \Gamma \vdash t : M$, such that $M = \sqcup_{i \in \mathtt{I}} M_i$, if and only if there are $\Phi_i \rhd_{\mathcal{P}_{\mathcal{G}}} \Gamma_i \vdash t : M_i$, such that $\Gamma = +_{i \in \mathtt{I}} \Gamma_i$, and $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) = +_{i \in \mathtt{I}} \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi_i)$.*

That typing contexts are necessarily empty whenever terms are closed.

Lemma 7.94 (Contexts and Free Variables for CBPV)**.** *If $\Phi \rhd_{\mathcal{P}_{\mathcal{G}}} \Gamma \vdash t : \mathbb{A}$ (resp. M), then $\mathtt{dom}(\Gamma) \subseteq \mathtt{fv}(t)$.*

And that the type system reflects the fact that values are not effectful.

Lemma 7.95 (CBPV Values Are Not Effectful)**.** *Let $t \in !\Lambda^{\mathcal{G}}$ and $\Phi \rhd_{\mathcal{P}_{\mathcal{G}}} \emptyset \vdash t : S \Rightarrow (A \times Q)$. If $t \in !\mathrm{V}^{\mathcal{G}}$, then $Q = S$ and Φ ends with rule `Lift`.*

The fact that having a tight type derivation with zero weight is a characteristic of normal forms is formalized as follows.

Lemma 7.96. *Let $t \in !\Lambda^{\mathcal{G}}_{\circ}$ and $s \in \mathcal{S}_{\circ}$. If $\Phi \rhd_{\mathcal{P}_{\mathcal{G}}} \emptyset \vdash (t,s) : P$ is tight, then $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) = 0 \cdot \mathtt{App} + 0 \cdot \mathtt{Der} + 0 \cdot \mathtt{Get} + 0 \cdot \mathtt{Set}$ if and only if $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$.*

The following lemma ensures that all clash-free normal forms are tightly typable.

Lemma 7.97 (Tight Typability of Clash-Free `CBPV`-Normal Forms)**.** *Let $t \in !\Lambda^{\mathcal{G}}_{\circ}$. If $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$, then there is a derivation $\Phi \rhd_{\mathcal{P}_{\mathcal{G}}} \emptyset \vdash (t,s) : P$ that is tight.*

The following lemma tells us how to pop multiset types from the list type assigned to some location.

Lemma 7.98. *Let $l \in \mathcal{L}$ and $s \in \mathcal{S}$. There is $\Phi \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash s : S$, such that $S(l) \neq [\,]$, if and only if there are $\Psi \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash s\,(l) : \mathtt{head}(S(l))$ and $\Pi \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash s : S\{l := \mathtt{tail}(S(l))\}$. Moreover, $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Pi)$.*

The following lemma tells us how to update the list type assigned to a location, as long as the initial list type is empty.

Lemma 7.99. *Let $l \in \mathcal{L}$ and $s \in \mathcal{S}$. There are $\Phi \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash s : S$, such that $S(l) = [\,]$, and $\Psi \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash v : L$ if and only if there is $\Pi \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash s\{l := v\} : S\{l := L\}$. Moreover, $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi) = \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Pi)$.*

Now, we proceed by showing that $\mathcal{P}_\mathcal{G}$-typability is quantitatively sound and complete with respect to CBPV-normalization. Moreover, we also show that type derivations provide exact measures for the number of $\beta_{\mathtt{v}}$-, $\beta_!$-, `get`-, and `set`-steps.

7.3.3.1 Quantitative Soundness

We proceed by proving a weighted version of the usual substitution lemma.

Lemma 7.100 (Weighted Substitution for CBPV)**.** *Let $t \in\, !\Lambda^{\mathcal{G}}$ and $v \in\, !\mathrm{V}^{\mathcal{G}}$. If $\Phi \rhd_{\mathcal{P}_\mathcal{G}} \Gamma; x : M \vdash t : \mathbb{A}$ (resp. N or L) and $\Psi \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash v : M$, then $\Pi \rhd_{\mathcal{P}_\mathcal{G}} \Gamma \vdash t\{x \setminus v\} : \mathbb{A}$ (resp. N or L), such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Pi) = \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi)$.*

The quantitative version of the subject reduction property follow from the above lemma, but only if $S(l) = \{\!\{\,\}\!\}$ is added as a requirement for rule `Set`. This is to be assumed from now on, but no other tightness criteria are to be assumed.

Lemma 7.101 (Exact Subject Reduction for CBPV)**.** *Let $t \in\, !\Lambda^{\mathcal{G}}_{\circ}$, $s \in \mathcal{S}_{\circ}$, and $S \in \{\beta_{\mathtt{v}}, \beta_!, \mathtt{get}, \mathtt{set}\}$. If $(t,s) \Rightarrow_{\mathtt{CBPV}(S)} (u,q)$ and $\Phi \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash (t,s) : P$, then there is $\Psi \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash (u,q) : P$, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi) + 1 \cdot R$, where*

- *$R = \mathtt{App}$, if $S = \beta_{\mathtt{v}}$;*
- *$R = \mathtt{Der}$, if $S = \beta_!$;*
- *$R = \mathtt{Get}$, if $S = \mathtt{get}$;*
- *and $R = \mathtt{Set}$, if $S = \mathtt{set}$.*

Finally, we can show that $\mathcal{P}_\mathcal{G}$-typability is quantitatively sound with respect to clash-free CBPV-normalization.

Theorem 7.102 (Exact Soundness for CBPV)**.** *Let $t \in\, !\Lambda^{\mathcal{G}}_{\circ}$. If there is $\Phi \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash (t,s) : P$ tight, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) = n \cdot \mathtt{App} + m \cdot \mathtt{Der} + g \cdot \mathtt{Get} + s \cdot \mathtt{Set}$, then $(t,s) \overset{n\cdot\beta_{\mathtt{v}} + m\cdot\beta_! + g\cdot\mathtt{get} + s\cdot\mathtt{set}}{\Rightarrow\!\!\!\!\!\rightarrow}_{\mathtt{CBPV}} (u,q)$, such that $u \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$.*

7.3.3.2 Quantitative Completeness

We start by showing that $\mathcal{P}_{\mathcal{G}}$-typable configurations are necessarily clash-free, *i.e.* normalize to a clash-free CBPV normal form.

Lemma 7.103 (Typability Implies Clash-Freeness)**.** *Let $t \in !\Lambda^{\mathcal{G}}_{\circ}$ and $s \in \mathcal{S}_{\circ}$. If $\Phi \rhd_{\mathcal{P}_{\mathcal{G}}} \emptyset \vdash (t,s) : P$, then t is clash-free.*

In fact, we can show that $\mathcal{P}_{\mathcal{G}}$-typability *captures* the clash-freeness of CBPV-normal forms.

Lemma 7.104 (Typability Characterizes CBPV Clash-Free Normal Forms)**.** *Let $t \in !\Lambda^{\mathcal{G}}_{\circ}$ and $s \in \mathcal{S}_{\circ}$. Then, $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$ if and only if $t \in \mathtt{NO}_{\mathtt{CBPV}}$ and (t,s) is $\mathcal{P}_{\mathcal{G}}$-typable.*

Now, we proceed by proving a weighted version of the usual anti-substitution lemma.

Lemma 7.105 (Weighted Anti-Substitution for CBPV)**.** *Let $t \in !\Lambda^{\mathcal{G}}$, and $v \in !\mathrm{V}^{\mathcal{G}}_{\circ}$. If $\Phi \rhd_{\mathcal{P}_{\mathcal{G}}} \Gamma \vdash t\{x \setminus v\} : \mathbb{A}$ (resp. N or L), then $\Psi \rhd_{\mathcal{P}_{\mathcal{G}}} \Gamma; x : M \vdash t : \mathbb{A}$ (resp. N or L) and $\Pi \rhd_{\mathcal{P}_{\mathcal{G}}} \emptyset \vdash v : M$, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Pi)$.*

The quantitative version of the usual subject expansion property follows from the above lemma, but only if $S(l) = \{\!\{\ \}\!\}$ is added as a requirement for rule Set. This is to be assumed from now on, but no other tightness criteria are to be assumed.

Lemma 7.106 (Exact Subject Expansion for CBPV)**.** *Let $t \in !\Lambda^{\mathcal{G}}_{\circ}$, $s \in \mathcal{S}_{\circ}$, and $S \in \{\beta_{\mathtt{v}}, \beta_{!}, \mathtt{get}, \mathtt{set}\}$. If $(t,s) \Rightarrow_{\mathtt{CBPV}(S)} (u,q)$ and $\Phi \rhd_{\mathcal{P}_{\mathcal{G}}} \emptyset \vdash (u,q) : P$, then there is $\Psi \rhd_{\mathcal{P}_{\mathcal{G}}} \emptyset \vdash (t,s) : P$, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) + 1 \cdot R = \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi)$, where*

- $R = \mathtt{App}$, *if* $S = \beta_{\mathtt{v}}$;
- $R = \mathtt{Der}$, *if* $S = \beta_{!}$;
- $R = \mathtt{Get}$, *if* $S = \mathtt{get}$;
- *and* $R = \mathtt{Set}$, *if* $S = \mathtt{set}$.

Finally, we can show that $\mathcal{P}_{\mathcal{G}}$-typability is quantitatively complete with respect to clash-free CBPV-normalization.

Theorem 7.107 (Exact Completeness for CBPV)**.** *Let $t \in \Lambda^{\mathcal{G}}_{\circ}$. If $(t,s) \overset{n\cdot\beta_{\mathtt{v}}+m\cdot\beta_{!}+g\cdot\mathtt{get}+s\cdot\mathtt{set}}{\Rightarrow\!\!\!\!\!\!\rightarrow}_{\mathtt{CBPV}} (u,q)$, such that $u \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$, then there is $\Phi \rhd_{\mathcal{P}_{\mathcal{G}}} \emptyset \vdash (t,s) : P$ tight, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) = n \cdot \mathtt{App} + m \cdot \mathtt{Der} + g \cdot \mathtt{Get} + s \cdot \mathtt{Set}$.*

7.3.3.3 Soundness and Completeness of Translations

To show that the translations in Definition 7.32 are also sound and complete with respect to $\mathcal{P}_{\mathcal{G}}$-typability, we must first define the translation of CBV types and typing environments to CBPV types and typing environments. Notice that, while the CBV translation in Section 3.2 preserves typings (see Theorem 4.27), the one in Section 7.2.2 does not. This is due to the fact that CBV multi-types (see Definition 7.67) are only allowed to contain types, but CBPV multi-types (see Definition 7.87) are only allowed to contain monadic types. To fix this, we simply need to do the following. Let $\{\!\{A_i\}\!\}_{i\in\mathtt{I}}$ be some CBV multi-type. If a term t is assigned type $\{\!\{A_i\}\!\}_{i\in\mathtt{I}}$, we know that it will normalize to a value v that is going to be copied $|\mathtt{I}|$-many times. Moreover, we also know that each copy of v will assigned a (possibly) different type A_i. In fact, we actually know that each copy of v will be assigned a monadic type of the form $S_i \Rightarrow (A_i \times S_i)$, where S_i is the type of the current state at the moment that particular copy is used. But, then this means that a $\eta^{S_i}_{A_i}$ transformation is (implicitly) applied to each copy of v. Therefore, in order to translate CBV multi-types, one needs to decorate each A_i with the appropriate state type.

Definition 7.108 (Translations of CBV Typings)**.** Let S_{CBV} and S_{CBPV} be the sets of state types for the CBV $\lambda^{\mathcal{G}}$-calculus and the $\lambda!^{\mathcal{G}}$-calculus, respectively. The translations of types, states, and typing environments are as follows:

$$
\begin{array}{rcl}
(\cdot)^{\mathsf{CBV}} : \mathcal{T}y^{\mathcal{G}}_{\mathsf{CBV}} & \to & \mathcal{T}y^{\mathcal{G}}_{\mathsf{CBPV}} \\
M \to \mathbb{M} & \mapsto & (M)^{\mathsf{CBV}} \to (\mathbb{M})^{\mathsf{CBV}} \\
\{\!\{A_i\}\!\}_{i\in\mathtt{I}} & \mapsto & \{\!\{S_i \Rightarrow ((A_i)^{\mathsf{CBV}} \times S_i)\}\!\}_{i\in\mathtt{I}} \\
S \Rightarrow P & \mapsto & (S)^{\mathsf{CBV}} \Rightarrow (P)^{\mathsf{CBV}} \\
M \times S & \mapsto & (M)^{\mathsf{CBV}} \times (S)^{\mathsf{CBV}}
\end{array}
\qquad
\begin{array}{rcl}
(\cdot)^{\mathsf{CBV}} : S_{\mathsf{CBV}} & \to & S_{\mathsf{CBPV}} \\
\{l \mapsto L_l\}_{l\in\mathcal{L}} & \mapsto & \{l \mapsto (L_l)^{\mathsf{CBV}}\}_{l\in\mathcal{L}} \\
M \cdot L & \mapsto & (M)^{\mathsf{CBV}} \cdot (L)^{\mathsf{CBV}}
\end{array}
$$

$$
\begin{array}{rcl}
(\cdot)^{\mathsf{CBV}} : (\mathcal{X} \to \mathcal{T}y^{\mathcal{G}}_{\mathsf{CBV}}) & \to & (\mathcal{X} \to \mathcal{T}y^{\mathcal{G}}_{\mathsf{CBPV}}) \\
\Gamma; x : M & \mapsto & (\Gamma)^{\mathsf{CBV}}; x : (M)^{\mathsf{CBV}}
\end{array}
$$

Although $(\cdot)^{\mathsf{CBV}}$ is (a priori) *non-deterministic* (due to the translation of multi-types to multi-monadic types), it *is* nevertheless injective, *i.e.* each CBPV type in the image of the translation corresponds to a unique CBV type in the preimage of $(\cdot)^{\mathsf{CBV}}$.

In summary, CBV multi-types are translated by eagerly inserting the monadic structure that CBPV would otherwise reconstruct dynamically.

Example 7.109 (Tight Type Derivation for CBN Translation)**.** *Let* $\mathbb{A}_0 = \{l \mapsto [\,]\} \Rightarrow ((\{\!\{\mathbb{A}_1\}\!\} \to \mathbb{A}_1) \times \{l \mapsto [\,]\})$, $\mathbb{A}_1 = \{l \mapsto [\,]\} \Rightarrow ((\{\!\{\mathbb{A}_2\}\!\} \to \mathbb{A}_2) \times \{l \mapsto [\,]\})$, *and* $\mathbb{A}_2 = \{l \mapsto [\,]\} \Rightarrow (\star \times \{l \mapsto [\,]\})$, *as in Example 7.58. Then, we can type the CBN translation of the configuration from Example 7.14 with the following* tight *type derivation* Ψ. *Since* Ψ *is quite big, we are going to split it into the following pieces.*

Let Ψ_s be the following type derivation for the state:

$$\dfrac{\dfrac{}{\emptyset \vdash q\,(l) : []}\ \texttt{EList}}{\emptyset \vdash q : \{l \mapsto []\}}\ \texttt{State}$$

The parameter of the `set` *operations is going to be typed twice. Let the following type derivation Ψ_{p1} be one of those type derivations:*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : \{\!\{\mathbb{A}_1\}\!\} \vdash x : \{\!\{\mathbb{A}_1\}\!\}}\ \texttt{Ax}}{x : \{\!\{\mathbb{A}_1\}\!\} \vdash x : \{l \mapsto []\} \Rightarrow (\{\!\{\mathbb{A}_1\}\!\} \times \{l \mapsto []\})}\ \texttt{Lift}}{x : \{\!\{\mathbb{A}_1\}\!\} \vdash \texttt{der}(x) : \mathbb{A}_1}\ \texttt{Der}}{\emptyset \vdash \lambda x.\texttt{der}(x) : \mathbb{A}_0}\ \texttt{Abs} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{y : \{\!\{\mathbb{A}_2\}\!\} \vdash y : \{\!\{\mathbb{A}_2\}\!\}}\ \texttt{Ax}}{y : \{\!\{\mathbb{A}_2\}\!\} \vdash y : \{l \mapsto []\} \Rightarrow (\{\!\{\mathbb{A}_2\}\!\} \times \{l \mapsto []\})}\ \texttt{Lift}}{y : \{\!\{\mathbb{A}_2\}\!\} \vdash \texttt{der}(y) : \mathbb{A}_2}\ \texttt{Der}}{\emptyset \vdash \lambda y.\texttt{der}(y) : \mathbb{A}_1}\ \texttt{Abs}}{\emptyset \vdash\, !\lambda y.\texttt{der}(y) : \{\!\{\mathbb{A}_1\}\!\}}\ \texttt{Bg}}{\emptyset \vdash\, !\lambda y.\texttt{der}(y) : \{l \mapsto []\} \Rightarrow (\{\!\{\mathbb{A}_1\}\!\} \times \{l \mapsto []\})}\ \texttt{Lift}}{\emptyset \vdash (\lambda x.\texttt{der}(x))\,!\lambda y.\texttt{der}(y) : \mathbb{A}_1}\ \texttt{App}$$

And the following type derivation Ψ_{p2} be the other one:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : \{\!\{\mathbb{A}_2\}\!\} \vdash x : \{\!\{\mathbb{A}_2\}\!\}}\ \texttt{Ax}}{x : \{\!\{\mathbb{A}_2\}\!\} \vdash x : \{l \mapsto []\} \Rightarrow (\{\!\{\mathbb{A}_2\}\!\} \times \{l \mapsto []\})}\ \texttt{Lift}}{x : \{\!\{\mathbb{A}_2\}\!\} \vdash \texttt{der}(x) : \mathbb{A}_2}\ \texttt{Der}}{\emptyset \vdash \lambda x.\texttt{der}(x) : \mathbb{A}_1}\ \texttt{Abs} \qquad \dfrac{\dfrac{\dfrac{}{\emptyset \vdash \lambda y.\texttt{der}(y) : \mathbb{A}_2}\ \texttt{Ans}}{\emptyset \vdash\, !\lambda y.\texttt{der}(y) : \{\!\{\mathbb{A}_2\}\!\}}\ \texttt{Bg}}{\emptyset \vdash\, !\lambda y.\texttt{der}(y) : \{l \mapsto []\} \Rightarrow (\{\!\{\mathbb{A}_2\}\!\} \times \{l \mapsto []\})}\ \texttt{Lift}}{\emptyset \vdash (\lambda x.\texttt{der}(x))\,!\lambda y.\texttt{der}(x) : \mathbb{A}_2}\ \texttt{App}$$

Finally, let Ψ_c be the type derivation for the continuation of the `set` *operation:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{z : \{\!\{\mathbb{A}_1\}\!\} \vdash z : \{\!\{\mathbb{A}_1\}\!\}}\ \texttt{Ax}}{z : \{\!\{\mathbb{A}_1\}\!\} \vdash z : \{l \mapsto []\} \Rightarrow (\{\!\{\mathbb{A}_1\}\!\} \times \{l \mapsto []\})}\ \texttt{Lift}}{z : \{\!\{\mathbb{A}_1\}\!\} \vdash \texttt{der}(z) : \mathbb{A}_1}\ \texttt{Der} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{z : \{\!\{\mathbb{A}_2\}\!\} \vdash z : \{\!\{\mathbb{A}_2\}\!\}}\ \texttt{Ax}}{z : \{\!\{\mathbb{A}_2\}\!\} \vdash z : \{l \mapsto []\} \Rightarrow (\{\!\{\mathbb{A}_2\}\!\} \times \{l \mapsto []\})}\ \texttt{Lift}}{z : \{\!\{\mathbb{A}_2\}\!\} \vdash \texttt{der}(z) : \mathbb{A}_2}\ \texttt{Der}}{z : \{\!\{\mathbb{A}_2\}\!\} \vdash\, !\texttt{der}(z) : \{\!\{\mathbb{A}_2\}\!\}}\ \texttt{Bg}}{z : \{\!\{\mathbb{A}_2\}\!\} \vdash\, !\texttt{der}(z) : \{l \mapsto []\} \Rightarrow (\{\!\{\mathbb{A}_2\}\!\} \times \{l \mapsto []\})}\ \texttt{Lift}}{z : \{\!\{\mathbb{A}_1, \mathbb{A}_2\}\!\} \vdash \texttt{der}(z)\,!\texttt{der}(z) : \mathbb{A}_2}\ \texttt{App}}{\emptyset \vdash \texttt{get}(l, \lambda z.\texttt{der}(z)\,!\texttt{der}(z)) : \{l \mapsto \{\!\{\mathbb{A}_1, \mathbb{A}_2\}\!\} \cdot []\} \Rightarrow (\star \times \{l \mapsto []\})}\ \texttt{Get}$$

Then, Ψ is as follows:

$$\dfrac{\dfrac{\dfrac{\Psi_{p1} \quad \Psi_{p2}}{\emptyset \vdash\, !((\lambda x.\texttt{der}(x))\,!\lambda y.\texttt{der}(y)) : \{\!\{\mathbb{A}_1, \mathbb{A}_2\}\!\}}\ \texttt{Bg} \qquad \dfrac{}{\emptyset \vdash\, !((\lambda x.\texttt{der}(x))\,!\lambda y.\texttt{der}(y)) : []}\ \texttt{EList}}{\emptyset \vdash\, !((\lambda x.\texttt{der}(x))\,!\lambda y.\texttt{der}(y)) : \{\!\{\mathbb{A}_1, \mathbb{A}_2\}\!\} \cdot []}\ \texttt{List} \qquad \Psi_c}{\emptyset \vdash \texttt{set}((l, !((\lambda x.\texttt{der}(x))\,!\lambda y.\texttt{der}(y))), \texttt{get}(l, \lambda z.\texttt{der}(z)\,!\texttt{der}(z))) : \{l \mapsto []\} \Rightarrow (\star \times \{l \mapsto []\})}\ \texttt{Set} \qquad \Psi_s}{\emptyset \vdash (\texttt{set}((l, !((\lambda x.\texttt{der}(x))\,!\lambda y.\texttt{der}(y))), \texttt{get}(l, \lambda z.\texttt{der}(z)\,!\texttt{der}(z))), q) : \star \times \{l \mapsto []\}}\ \texttt{Conf}$$

Notice that $\#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Psi) = 3 \cdot \texttt{App} + 1 \cdot \texttt{Get} + 1 \cdot \texttt{Set} = \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Phi)$, where Φ is the type derivation in Example 7.58. This means that the translations preserve the number of uses of rules `App`, `Get`, *and* `Set`, *and thus of β-,* `get`*-, and* `set`*-steps.*

Example 7.110 (Tight Type Derivation for CBV Translation)**.** *Let $A_1 = \{\!\{\,\}\!\} \to (\{l \mapsto []\} \Rightarrow (\{\!\{\,\}\!\} \times \{l \mapsto []\}))$, as in Example 7.78. Then, we can type the CBV translation of the configuration*

from Example 7.16 with the following tight *type derivation* Ψ*. Since* Ψ *is quite big, we are going to split it into the following pieces.*

Let Ψ_s *be the following type derivation for the state:*

$$\dfrac{\dfrac{}{\varnothing \vdash q\,(l) : [\,]}\ \texttt{EList}}{\varnothing \vdash q : \{l \mapsto [\,]\}}\ \texttt{State}$$

The parameter of the `set` *operation is only typed once. Let* Ψ_p *be such a type derivation:*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\varnothing \vdash x : \{\!\{\,\}\!\}}\ \texttt{Ax}}{\varnothing \vdash x : \{l \mapsto [\,]\} \Rightarrow (\{\!\{\,\}\!\} \times \{l \mapsto [\,]\})}\ \texttt{Lift}}{\varnothing \vdash \lambda x.x : \{l \mapsto [\,]\} \Rightarrow (A_1 \times \{l \mapsto [\,]\})}\ \texttt{Abs}}{\varnothing \vdash\, !\lambda x.x : \{\!\{l \mapsto [\,]\} \Rightarrow (A_1 \times \{l \mapsto [\,]\})\}\!\}}\ \texttt{Bg}$$

Let Ψ_l *be the following type derivation for the operator appearing in the* `get` *operation:*

$$\dfrac{\dfrac{\dfrac{}{y : \{\!\{l \mapsto [\,]\} \Rightarrow (A_1 \times \{l \mapsto [\,]\})\}\!\} \vdash y : \{\!\{l \mapsto [\,]\} \Rightarrow (A_1 \times \{l \mapsto [\,]\})\}\!\}}\ \texttt{Ax}}{y : \{\!\{l \mapsto [\,]\} \Rightarrow (A_1 \times \{l \mapsto [\,]\})\}\!\} \vdash y : \{l \mapsto [\,]\} \Rightarrow (\{\!\{l \mapsto [\,]\} \Rightarrow (A_1 \times \{l \mapsto [\,]\})\}\!\} \times \{l \mapsto [\,]\})}\ \texttt{Lift}}{y : \{\!\{l \mapsto [\,]\} \Rightarrow (A_1 \times \{l \mapsto [\,]\})\}\!\} \vdash \texttt{der}(y) : \{l \mapsto [\,]\} \Rightarrow (A_1 \times \{l \mapsto [\,]\})}\ \texttt{Der}$$

Let Ψ_r *be the following type derivation for the term at the right of the application corresponding to the continuation of* `get` *operation:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\varnothing \vdash z : \{\!\{\,\}\!\}}\ \texttt{Ax}}{\varnothing \vdash z : \{l \mapsto [\,]\} \Rightarrow (\{\!\{\,\}\!\} \times \{l \mapsto [\,]\})}\ \texttt{Lift}}{\varnothing \vdash \lambda z.z : \{l \mapsto [\,]\} \Rightarrow (A_1 \times \{l \mapsto [\,]\})}\ \texttt{Abs}}{\varnothing \vdash\, !\lambda z.z : \{\!\{l \mapsto [\,]\} \Rightarrow (A_1 \times \{l \mapsto [\,]\})\}\!\}}\ \texttt{Bg}}{\varnothing \vdash\, !\lambda z.z : \{l \mapsto [\,]\} \Rightarrow (\{\!\{l \mapsto [\,]\} \Rightarrow (A_1 \times \{l \mapsto [\,]\})\}\!\} \times \{l \mapsto [\,]\})}\ \texttt{Lift}}{\varnothing \vdash \texttt{der}(!\lambda z.z) : \{l \mapsto [\,]\} \Rightarrow (A_1 \times \{l \mapsto [\,]\})}\ \texttt{Der} \qquad \dfrac{\dfrac{}{\varnothing \vdash y : \{\!\{\,\}\!\}}\ \texttt{Ax}}{\varnothing \vdash y : \{l \mapsto [\,]\} \Rightarrow (\{\!\{\,\}\!\} \times \{l \mapsto [\,]\})}\ \texttt{Lift}}{\varnothing \vdash \texttt{der}(!(\lambda z.z))\,y : \{l \mapsto [\,]\} \Rightarrow (\{\!\{\,\}\!\} \times \{l \mapsto [\,]\})}\ \texttt{App}$$

Finally, let Ψ_c *be the following type derivation for the continuation of the* `get` *operation:*

$$\dfrac{\dfrac{\Psi_l \quad \Psi_r}{y : \{\!\{l \mapsto [\,]\} \Rightarrow (A_1 \times \{l \mapsto [\,]\})\}\!\} \vdash \texttt{der}(y)\,(\texttt{der}(!(\lambda z.z))\,y) : \{l \mapsto [\,]\} \Rightarrow (\{\!\{\,\}\!\} \times \{l \mapsto [\,]\})}\ \texttt{App}}{\varnothing \vdash \texttt{get}(l, \lambda y.\texttt{der}(y)\,(\texttt{der}(!(\lambda z.z))\,y)) : \{l \mapsto \{\!\{\{l \mapsto [\,]\} \Rightarrow (A_1 \times \{l \mapsto [\,]\})\}\!\} \cdot [\,]\} \Rightarrow (\{\!\{\,\}\!\} \times \{l \mapsto [\,]\})}\ \texttt{Get}$$

Then, Ψ *is as follows:*

$$\dfrac{\dfrac{\dfrac{\Psi_p \quad \dfrac{}{\varnothing \vdash\, !\lambda x.x : [\,]}\ \texttt{EList}}{\varnothing \vdash\, !\lambda x.x : \{\!\{l \mapsto [\,]\} \Rightarrow (A_1 \times \{l \mapsto [\,]\})\}\!\} \cdot [\,]}\ \texttt{List} \quad \Psi_c}{\varnothing \vdash \texttt{set}((l, !\lambda x.x), \texttt{get}(l, \lambda y.\texttt{der}(y)\,(\texttt{der}(!(\lambda z.z))\,y))) : \{l \mapsto [\,]\} \Rightarrow (\{\!\{\,\}\!\} \times \{l \mapsto [\,]\})}\ \texttt{Set} \quad \Psi_s}{\varnothing \vdash (\texttt{set}((l, !\lambda x.x), \texttt{get}(l, \lambda \texttt{der}(y).y\,(\texttt{der}(!(\lambda z.z))\,y))), s) : \{\!\{\,\}\!\} \times \{l \mapsto [\,]\}}\ \texttt{Conf}$$

Notice that $\#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Phi) = 2 \cdot \texttt{App} + 1 \cdot \texttt{Get} + 1 \cdot \texttt{Set} = \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Phi)$*, where* Φ *is the type derivation in Example 7.78. This means that the translations preserve the number of uses of rules* `App`*,* `Get`*, and* `Set`*, and thus of* β_{v}*-,* `get`*-, and* `set`*-steps.*

Since we have seen that the translations of terms preserve the number of β-, β_v-, `get`-, and `set`-steps we can show that $\mathcal{P}$-typability is not only sound and complete with respect to the translations, but also preserves the number of steps.

Lemma 7.111 (Soundness of Term Translations)**.** *Let $t \in \Lambda^{\mathcal{G}}$.*

1. *If $\Phi \rhd_{\mathcal{N}_{\mathcal{G}}} \Gamma \vdash t : \mathbb{A}$ (resp. M or L), then $\Psi \rhd_{\mathcal{P}_{\mathcal{G}}} \Gamma \vdash (t)^{\mathtt{CBN}}$ (resp. $!(t)^{\mathtt{CBN}}$ or $!(t)^{\mathtt{CBN}}$) : $\mathbb{A}$ (resp. $\mathbb{M}$ or $\mathbb{L}$). Moreover, $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi)$.*
2. *If $\Phi \rhd_{\mathcal{V}_{\mathcal{G}}} \Gamma \vdash t : \mathbb{M}$ (resp. M or L), then $\Psi \rhd_{\mathcal{P}_{\mathcal{G}}} (\Gamma)^{\mathtt{CBV}} \vdash (t)^{\mathtt{CBV}} : (\mathbb{M})^{\mathtt{CBV}}$ (resp. $(M)^{\mathtt{CBV}}$ or $(L)^{\mathtt{CBV}}$). Moreover, $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi)$.*

As it turns out, while the completeness of the CBN translation can be easily shown, the completeness of the CBV translation, besides requiring $S(l) = \{\!\{\ \}\!\}$ to be added as a requirement for rule `Set`, can not be shown by a simple induction on typing derivations. This asymmetry in difficulty between showing soundness and completeness is not unusual. As such, we will focus on showing completeness for the CBN translations, and leave the completeness of the CBV translations as a conjecture.

Lemma 7.112 (Completeness of CBN Term Translations)**.** *Let $t \in \Lambda^{\mathcal{G}}$.*
If $\Psi \rhd_{\mathcal{P}_{\mathcal{G}}} \Gamma \vdash (t)^{\mathtt{CBN}}$ (resp. $!(t)^{\mathtt{CBN}}$or $!(t)^{\mathtt{CBN}}$) : $\mathbb{A}$ (resp. M or L), then $\Phi \rhd_{\mathcal{N}_{\mathcal{G}}} \Gamma \vdash t : \mathbb{A}$ (resp. M or L). Moreover, $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi)$.

Finally, we show that CBN state translations are sound and complete.

Lemma 7.113 (Soundness and Completeness of CBN State Translations)**.** *Let $s \in \mathcal{S}$. Then $\Phi \rhd_{\mathcal{N}_{\mathcal{G}}} \Gamma \vdash s : S$ if and only if $\Psi \rhd_{\mathcal{P}_{\mathcal{G}}} \Gamma \vdash (s)^{\mathtt{CBN}} : S$. Moreover, $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi)$.*

And that CBV state translations are sound.

Lemma 7.114 (Soundness of CBV State Translations)**.** *Let $s \in \mathcal{S}$. If $\Phi \rhd_{\mathcal{V}_{\mathcal{G}}} \Gamma \vdash s : S$, $\Psi \rhd_{\mathcal{P}_{\mathcal{G}}} (\Gamma)^{\mathtt{CBV}} \vdash (s)^{\mathtt{CBV}} : (S)^{\mathtt{CBV}}$. Moreover, $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi)$.*

Proof. The proofs follow smoothly by induction on the structure of $s \in \mathcal{S}$ and Lemma 7.111. □

The soundness and completeness of the CBN translation of configurations then follows from the above statements.

Lemma 7.115 (Soundness and Completeness of CBN Configuration Translations)**.** *Let $t \in \Lambda^{\mathcal{G}}$ and $s \in \mathcal{S}$. Then, $\Phi \rhd_{\mathcal{N}_{\mathcal{G}}} \Gamma \vdash (t, s) : P$ if and only if $\Psi \rhd_{\mathcal{P}_{\mathcal{G}}} \Gamma \vdash ((t, s))^{\mathtt{CBN}} : P$. Moreover, $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi)$.*

The soundness of the CBV translation of configurations follows from the soundness of the CBV translations of terms and states.

Lemma 7.116 (Soundness of CBV Configuration Translations)**.** *Let $t \in \Lambda^{\mathcal{G}}$ and $s \in \mathcal{S}$. If $\Phi \rhd_{\mathcal{V}_{\mathcal{G}}} \Gamma \vdash (t,s) : P$, then $\Psi \rhd_{\mathcal{P}_{\mathcal{G}}} (\Gamma)^{\mathtt{CBV}} \vdash ((t,s))^{\mathtt{CBV}} : (P)^{\mathtt{CBV}}$. Moreover, $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi)$.*

The completeness of the CBV translation of configurations is left as a conjecture, since it can not be shown by a simple induction on typing derivations.

Conjecture 7.117 (Completeness of CBV Configuration Translations). Let $t \in \Lambda^{\mathcal{G}}$ and $s \in \mathcal{S}$. If $\Psi \rhd_{\mathcal{P}_{\mathcal{G}}} (\Gamma)^{\mathtt{CBV}} \vdash ((t,s))^{\mathtt{CBV}} : (P)^{\mathtt{CBV}}$, then $\Phi \rhd_{\mathcal{V}_{\mathcal{G}}} \Gamma \vdash (t,s) : P$. Moreover, $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi)$.

This concludes the quantitative study of global state. The techniques developed here will be applied in the next chapter to a notion of state mapping locations to infinite stacks.

Chapter 8

A Quantitative Study of Infinite Stacks

In Chapter 7 we developed a state-passing non-idempotent type system for global state. Recall that the key point of Remark 7.2 was that even though `get` operations behave *non-linearly* with respect to the state, they must behave *linearly* with respect to the type of the state. As was summarized in Table 7.1, this leads to the following mismatch between the notion of state and state type: while a *state* is a mapping from locations to *values*, a *state type* is a mapping from locations to *lists of types*. In this section, we are going to repair this mismatch by introducing an algebraic theory that directly captures the notion of state required by the type systems in Chapter 7: the algebraic theory for infinite stacks. The algebraic theory for infinite stacks is a particular instance of the algebraic theory for global state, where the state is assumed to be a mapping from locations to infinite stacks. In this setting, the `get` and `set` operations are replaced by `pop` and `push`, which behave as follows: the `pop` operation retrieves and removes the value at the head of the infinite stack assigned to a location; the `push` operation places a new value at the head of that stack. In sum, the operational behavior of the `pop` and `push` operations matches the denotational interpretation of operations `get` and `set` in Table 7.1. Thus, as we shall see, unlike global state, the operational and denotational behavior of `pop` and `push` coincide. In other words, infinite stacks make explicit–at the level of state–the linear consumption that was previously only visible at the level of types. Moreover, as it turns out, we will be able to encode the algebraic theory of global state (where the state is a mapping from locations to values) into the algebraic theory of infinite stacks, as well as many of the algebraic theories describing state-passing computations introduced in Chapter 5. Let us start by introducing the algebraic theory for infinite stacks and show that it can be captured by the more general algebraic theory for global state where the notion of state is unspecified. All the proofs for this chapter are collected in Appendix F.

8.1 The Algebraic Theory of Infinite Stacks

We introduce the algebraic theory for infinite stacks, and show how it can be captured by the algebraic theory for global state where the state is a mapping from locations to *infinite lists*. We refer to these lists as *stacks* because we only interact with them through $\texttt{head}()$ and $\texttt{tail}()$, via the usual pop and push operations: the pop operation *pops a value* from the head (top) of the infinite list assigned to a particular location; and the push operation *pushes a new value* to the head (top) of the infinite list assigned to a particular location. Let us now finally introduce the algebraic theory for infinite stacks.

Example 8.1 (Infinite Stacks)**.** *Let* $\mathtt{v},\mathtt{w} \in \mathcal{V}$*, and* $l, l' \in \mathcal{L}$*, such that* $l \neq l'$*. The algebraic theory* $\mathcal{S}$ *for infinite stacks is defined as follows:*

$$\Sigma_{\mathcal{S}} := \{\mathtt{pop} : \mathcal{L} \rightsquigarrow \mathcal{V}, \mathtt{push} : \mathcal{L} \times \mathcal{V} \rightsquigarrow 1\}$$

$$Eq_{\mathcal{S}} := \left\{ \begin{array}{rcl} \mathtt{pop}(l, \lambda \mathtt{a}.\mathtt{push}((l,\mathtt{a}), \lambda \mathtt{b}.\mathtt{t})) & \sim_{\mathcal{S}} & \mathtt{t} \textit{ if } \mathtt{a} \notin \mathtt{fv}(\mathtt{t}) \\ \mathtt{push}((l,\mathtt{v}), \lambda \mathtt{a}.\mathtt{pop}(l, \lambda \mathtt{b}.\mathtt{t})) & \sim_{\mathcal{S}} & \mathtt{t}\{\mathtt{b} \setminus \mathtt{v}\} \\ \mathtt{pop}(l, \lambda \mathtt{a}.\mathtt{pop}(l', \lambda \mathtt{b}.\mathtt{t})) & \sim_{\mathcal{S}} & \mathtt{pop}(l', \lambda \mathtt{b}.\mathtt{pop}(l, \lambda \mathtt{a}.\mathtt{t})) \\ \mathtt{pop}(l, \lambda \mathtt{a}.\mathtt{push}((l',\mathtt{v}), \lambda \mathtt{b}.\mathtt{t})) & \sim_{\mathcal{S}} & \mathtt{push}((l',\mathtt{v}), \lambda \mathtt{b}.\mathtt{pop}(l, \lambda \mathtt{a}.\mathtt{t})) \\ \mathtt{push}((l,\mathtt{v}), \lambda \mathtt{a}.\mathtt{push}((l',\mathtt{w}), \lambda \mathtt{b}.\mathtt{t})) & \sim_{\mathcal{S}} & \mathtt{push}((l',\mathtt{w}), \lambda \mathtt{b}.\mathtt{push}((l,\mathtt{v}), \lambda \mathtt{a}.\mathtt{t})) \end{array} \right\}.$$

Notice that the two first equations in $\mathrm{Eq}_{\mathcal{S}}$ are the key ones: the first equation states that *popping* a value from a particular location and then *pushing* that same value to the same location does nothing to the state; the second equation states that *pushing* a new value to a particular location and then *popping* it back from that same location does nothing to the state. The remaining equations state that operations on distinct locations commute. This is something common to the algebraic theory for global state (see Example 5.19).

Now, we are going to show that the algebraic theory for infinite stacks can be captured by the algebraic theory for global state. To do this, we are going to show that the pop and push operations can be encoded as sequences of get and set operations in such a way that all the equations in $\mathrm{Eq}_{\mathcal{S}}$ are validated in $\mathcal{G}$. That is, we want to show that computations in $\mathcal{S}$ can be transformed into computations in $\mathcal{G}$ that preserve the algebraic structure of $\mathcal{S}$. In general, this means that we need to define a structure preserving transformation from $\mathtt{F}_{\mathcal{S}}X$ (the free model of $\mathcal{S}$) to $\mathtt{F}_{\mathcal{G}}Y$ (the free model of $\mathcal{G}$) (see Section 5.3). This is precisely the notion of *handler*, first introduced by Plotkin and Pretnar in [88]. We finish this subsection with the general definition of a handler.

Definition 8.2 (Handlers)**.** A *handler* from computations $\mathtt{F}_{\mathcal{T}}X$ to computations $\mathtt{F}_{\mathcal{T}'}Y$ is given by the following data:

- a map $f : X \to |\mathtt{F}_{\mathcal{T}'}Y|$;

- for every operation $\mathtt{op}_i : \mathcal{P}_i \rightsquigarrow \mathcal{A}_i \in \Sigma_{\mathcal{T}}$, a map

$$h_{\mathtt{op}_i} : \mathcal{P}_i \times (\mathcal{A} \to |\mathtt{F}_{\mathcal{T}'} Y|) \to |\mathtt{F}_{\mathcal{T}'} Y|$$

such that

- the maps $h_{\mathtt{op}_i}$ form a $\mathcal{T}$-model on $|\mathtt{F}_{\mathcal{T}'} Y|$, *i.e.* validate the equations in $\mathrm{Eq}_{\mathcal{T}}$.

The map $H : |\mathtt{F}_{\mathcal{T}} X| \to |\mathtt{F}_{\mathcal{T}'} Y|$ induced by the above data is the unique one satisfying:

$$\begin{array}{rcl} H([\mathtt{return}\, x]) & = & f(x) \\ H([\mathtt{op}_i(p, \lambda \mathtt{a}.[\mathtt{t}])]) & = & h_{\mathtt{op}_i}(p, \lambda \mathtt{a}. H([\mathtt{t}])). \end{array}$$

In the next subsection we are going to present a handler from $\mathtt{F}_{\mathcal{S}} X$ to $\mathtt{F}_{\mathcal{G}} Y$.

8.2 Handling Infinite Stacks using Global State

Recall the algebraic theory for infinite stacks in Example 8.1 and the algebraic theory for global state in Example 5.19. We want to define a handler H from $|\mathtt{F}_{\mathcal{S}} X|$ to $|\mathtt{F}_{\mathcal{G}} Y|$. Notice that both theories assume the existence of some set of values and locations. Regarding the sets of locations: let $\mathcal{L}_{\mathcal{S}}$ be the set of locations for $\mathcal{S}$, $\mathcal{L}_{\mathcal{G}}$ be the set of locations for $\mathcal{G}$, and $\phi_{\mathcal{L}} : \mathcal{L}_{\mathcal{S}} \to \mathcal{L}_{\mathcal{G}}$ be an injective function between them. Regarding the sets of values: let $\mathcal{V}_{\mathcal{S}}$ be the set of values for $\mathcal{S}$, $\mathcal{V}_{\mathcal{G}}$ be the set of values for $\mathcal{G}$, and $\phi_{\mathcal{V}} : \mathcal{V}_{\mathcal{S}} \to \mathcal{V}_{\mathcal{G}}$ be an injective function between them. Let $\mathtt{pop} : \mathcal{L}_{\mathcal{S}} \rightsquigarrow \mathcal{V}_{\mathcal{S}}$ and $\mathtt{push} : \mathcal{L}_{\mathcal{S}} \times \mathcal{V}_{\mathcal{S}} \rightsquigarrow 1$. As injective functions, the inverses of $\phi_{\mathcal{L}}$ and $\phi_{\mathcal{V}}$ are not necessarily total functions. For locations, this is enough because we do not need to translate locations from $\mathcal{L}_{\mathcal{G}}$. However, as we shall see below, we will need to translate values from $\mathcal{V}_{\mathcal{G}}$. That being said, this will only be necessary for `pop` operations since these are the only ones that retrieve values from the state. In particular, if a `pop` operation is the first operation that is run by a computation, we will only be able to translate the value that is retrieved from the state if that value is in the image of $\phi_{\mathcal{G}}$. But this means that, as long as the starting state for any computation is only allowed to contain values that appear at the image of $\phi_{\mathcal{V}}$, then $\phi_{\mathcal{V}}^{-1}$ is enough. Moreover, notice that this will always be the case whenever we want to encode a computation in $\mathcal{G}$ into a computation in $\mathcal{S}$. Therefore, we can define the following maps:

$$\begin{array}{rcl} h_{\mathtt{pop}} : \mathcal{L}_{\mathcal{S}} \times (\mathcal{V}_{\mathcal{S}} \to |\mathtt{F}_{\mathcal{G}} Y|) & \to & |\mathtt{F}_{\mathcal{G}} Y| \\ (l, \lambda \mathtt{a}. H([\mathtt{t}])) & = & [\mathtt{get}(\phi_{\mathcal{L}}(l), \lambda \mathtt{b}. \\ & & \quad [\mathtt{set}((\phi_{\mathcal{L}}(l), \mathtt{tail}(\mathtt{b})), H([\mathtt{t}\{\mathtt{a} \setminus \phi_{\mathcal{V}}^{-1}(\mathtt{head}(\mathtt{b}))\}]))])] \\ h_{\mathtt{push}} : (\mathcal{L}_{\mathcal{S}} \times \mathcal{V}_{\mathcal{S}}) \times |\mathtt{F}_{\mathcal{G}} Y| & \to & |\mathtt{F}_{\mathcal{G}} Y| \\ ((l, \mathtt{v}), H([\mathtt{t}])) & = & [\mathtt{get}(\phi_{\mathcal{L}}(l), \lambda \mathtt{a}. [\mathtt{set}((\phi_{\mathcal{L}}(l), \phi_{\mathcal{V}}(\mathtt{v}) \cdot \mathtt{a}), H([\mathtt{t}]))])]. \end{array}$$

Finally, in order to be sure that H is indeed a handler, we need to show that $h_{\mathtt{pop}}$ and $h_{\mathtt{push}}$ form an $\mathcal{S}$-model, *i.e.* that the equations in $\mathrm{Eq}_{\mathcal{S}}$ are validated in $\mathcal{G}$. Recall the equations in $\mathrm{Eq}_{\mathcal{S}}$:

$$\begin{aligned}
\mathtt{pop}(l, \lambda \mathtt{a}.\mathtt{push}((l,\mathtt{a}), \lambda \mathtt{b}.\mathtt{t})) &\sim_{\mathcal{S}} \mathtt{t} \text{ if } \mathtt{a} \notin \mathtt{fv}(\mathtt{t}) \\
\mathtt{push}((l,\mathtt{v}), \lambda \mathtt{a}.\mathtt{pop}(l, \lambda \mathtt{b}.\mathtt{t})) &\sim_{\mathcal{S}} \mathtt{t}\{\mathtt{b} \setminus \mathtt{v}\} \\
\mathtt{pop}(l, \lambda \mathtt{a}.\mathtt{pop}(l', \lambda \mathtt{b}.\mathtt{t})) &\sim_{\mathcal{S}} \mathtt{pop}(l', \lambda \mathtt{b}.\mathtt{pop}(l, \lambda \mathtt{a}.\mathtt{t})) \\
\mathtt{pop}(l, \lambda \mathtt{a}.\mathtt{push}((l',\mathtt{v}), \lambda \mathtt{b}.\mathtt{t})) &\sim_{\mathcal{S}} \mathtt{push}((l',\mathtt{v}), \lambda \mathtt{b}.\mathtt{pop}(l, \lambda \mathtt{a}.\mathtt{t})) \\
\mathtt{push}((l,\mathtt{v}), \lambda \mathtt{a}.\mathtt{push}((l',\mathtt{w}), \lambda \mathtt{b}.\mathtt{t})) &\sim_{\mathcal{S}} \mathtt{push}((l',\mathtt{w}), \lambda \mathtt{b}.\mathtt{push}((l,\mathtt{v}), \lambda \mathtt{a}.\mathtt{t})).
\end{aligned}$$

After applying H to the each side of the above equations, we get the following set of equalities:

$$\begin{aligned}
&1. \quad H([\mathtt{pop}(l, \lambda \mathtt{a}.[\mathtt{push}((l,\mathtt{a}),[\mathtt{t}])])]) &&= H([\mathtt{t}]) \text{ if } \mathtt{a} \notin \mathtt{fv}(\mathtt{t}) \\
&2. \quad H([\mathtt{push}((l,\mathtt{v}),[\mathtt{pop}(l, \lambda \mathtt{a}.[\mathtt{t}])])]) &&= H([\mathtt{t}\{\mathtt{a} \setminus \mathtt{v}\}]) \\
&3. \quad H([\mathtt{pop}(l, \lambda \mathtt{a}.[\mathtt{pop}(l', \lambda \mathtt{b}.[\mathtt{t}])])]) &&= H([\mathtt{pop}(l', \lambda \mathtt{b}.[\mathtt{pop}(l, \lambda \mathtt{a}.[\mathtt{t}])])]) \\
&4. \quad H([\mathtt{pop}(l, \lambda \mathtt{a}.[\mathtt{push}((l',\mathtt{v}),[\mathtt{t}])])]) &&= H([\mathtt{push}((l',\mathtt{v}),[\mathtt{pop}(l, \lambda \mathtt{a}.[\mathtt{t}])])]) \\
&5. \quad H([\mathtt{push}((l,\mathtt{v}),[\mathtt{push}((l',\mathtt{w}),[\mathtt{t}])])]) &&= H([\mathtt{push}((l',\mathtt{w}),[\mathtt{push}((l,\mathtt{v}),[\mathtt{t}])])])
\end{aligned}$$

Now, we must check that they hold in $\mathcal{G}$. Intuitively, the key point is that the `head()-tail()` behavior of `pop` and `push` is faithfully simulated by appropriate sequences of `get-set`.

To help the reader, we use the color *red* to identify structures related to $\mathcal{S}$ and the color *blue* to identify structures related to $\mathcal{G}$:

Equation 1:

Let $\mathtt{a} \notin \mathtt{fv}([\mathtt{t}])$:

$$\begin{aligned}
& H([\mathtt{pop}(l, \lambda \mathtt{a}.[\mathtt{push}((l,\mathtt{a}),[\mathtt{t}])])]) \\
=\ & [\mathtt{get}(\phi_{\mathcal{L}}\,(l), \lambda \mathtt{b}.[\mathtt{set}((\phi_{\mathcal{L}}\,(l), \mathtt{tail}(\mathtt{b})), [\mathtt{get}(\lambda \mathtt{c}.[\mathtt{set}((\phi_{\mathcal{L}}\,(l), \phi_{\mathcal{V}}\,(\phi_{\mathcal{V}}^{-1}\,(\mathtt{head}(\mathtt{b}))) \cdot \mathtt{c}), H([\mathtt{t}]))])])])] \\
=\ & [\mathtt{get}(\phi_{\mathcal{L}}\,(l), \lambda \mathtt{b}.[\mathtt{set}((\phi_{\mathcal{L}}\,(l), \mathtt{tail}(\mathtt{b})), [\mathtt{get}(\lambda \mathtt{c}.[\mathtt{set}((\phi_{\mathcal{L}}\,(l), \mathtt{head}(\mathtt{b}) \cdot \mathtt{c}), H([\mathtt{t}]))])])])] \\
=\ & [\mathtt{get}(\phi_{\mathcal{L}}\,(l), \lambda \mathtt{b}.[\mathtt{set}((\phi_{\mathcal{L}}\,(l), \mathtt{tail}(\mathtt{b})), [\mathtt{set}((\phi_{\mathcal{L}}\,(l), \mathtt{head}(\mathtt{b}) \cdot \mathtt{tail}(\mathtt{b})), H([\mathtt{t}]))])])] \\
=\ & [\mathtt{get}(\phi_{\mathcal{L}}\,(l), \lambda \mathtt{b}.[\mathtt{set}((\phi_{\mathcal{L}}\,(l), \mathtt{tail}(\mathtt{b})), [\mathtt{set}((\phi_{\mathcal{L}}\,(l), \mathtt{b}), H([\mathtt{t}]))])])] \\
=\ & [\mathtt{get}(\phi_{\mathcal{L}}\,(l), \lambda \mathtt{b}.[\mathtt{set}((\phi_{\mathcal{L}}\,(l), \mathtt{b}), H([\mathtt{t}]))])] \\
=\ & H([\mathtt{t}]).
\end{aligned}$$

Equation 2:

$$
\begin{aligned}
& H([\mathtt{push}((l,\mathtt{v}),\mathtt{pop}(l,\lambda\mathtt{a}.[\mathtt{t}]))]) \\
= \;& [\mathtt{get}(\phi_{\mathcal{L}}(l),\lambda\mathtt{b}. \\
& \quad [\mathtt{set}((\phi_{\mathcal{L}}(l),\phi_{\mathcal{V}}(\mathtt{v})\cdot\mathtt{b}),[\mathtt{get}(\phi_{\mathcal{L}}(l),\lambda\mathtt{c}.[\mathtt{set}((\phi_{\mathcal{L}}(l),\mathtt{tail}(\mathtt{c})),H([\mathtt{t}\{\mathtt{a}\setminus\phi_{\mathcal{V}}^{-1}(\mathtt{head}(\mathtt{c}))\}]))])])])] \\
= \;& [\mathtt{get}(\phi_{\mathcal{L}}(l),\lambda\mathtt{b}. \\
& \quad [\mathtt{set}((\phi_{\mathcal{L}}(l),\phi_{\mathcal{V}}(\mathtt{v})\cdot\mathtt{b}),[\mathtt{set}((\phi_{\mathcal{L}}(l),\mathtt{tail}(\phi_{\mathcal{V}}(\mathtt{v})\cdot\mathtt{b})),H([\mathtt{t}\{\mathtt{a}\setminus\phi_{\mathcal{V}}^{-1}(\mathtt{head}(\phi_{\mathcal{V}}(\mathtt{v})\cdot\mathtt{b}))\}]))])])] \\
= \;& [\mathtt{get}(\phi_{\mathcal{L}}(l),\lambda\mathtt{b}.\mathtt{set}((\phi_{\mathcal{L}}(l),\phi_{\mathcal{V}}(\mathtt{v})\cdot\mathtt{b}),[\mathtt{set}((\phi_{\mathcal{L}}(l),\mathtt{b}),H([\mathtt{t}\{\mathtt{a}\setminus\phi_{\mathcal{V}}^{-1}(\phi_{\mathcal{V}}(\mathtt{v}))\}]))]))] \\
= \;& [\mathtt{get}(\phi_{\mathcal{L}}(l),\lambda\mathtt{b}.[\mathtt{set}((\phi_{\mathcal{L}}(l),\phi_{\mathcal{V}}(\mathtt{v})\cdot\mathtt{b}),[\mathtt{set}((\phi_{\mathcal{L}}(l),\mathtt{b}),H([\mathtt{t}\{\mathtt{a}\setminus\mathtt{v}\}]))])])] \\
= \;& [\mathtt{get}(\phi_{\mathcal{L}}(l),\lambda\mathtt{b}.[\mathtt{set}((\phi_{\mathcal{L}}(l),\mathtt{b}),H([\mathtt{t}\{\mathtt{a}\setminus\mathtt{v}\}]))])] \\
= \;& H([\mathtt{t}\{\mathtt{a}\setminus\mathtt{v}\}]).
\end{aligned}
$$

Equation 3:

$$
\begin{aligned}
& H([\mathtt{pop}(l,\lambda\mathtt{a}.[\mathtt{pop}(l',\lambda\mathtt{b}.[\mathtt{t}])])]) \\
= \;& [\mathtt{get}(\phi_{\mathcal{L}}(l),\lambda\mathtt{c}.[\mathtt{set}((\phi_{\mathcal{L}}(l),\mathtt{tail}(\mathtt{c})),[\mathtt{get}(\phi_{\mathcal{L}}(l'),\lambda\mathtt{d}. \\
& \quad [\mathtt{set}((\phi_{\mathcal{L}}(l'),\mathtt{tail}(\mathtt{d})),H([(\mathtt{t}\{\mathtt{a}\setminus\phi_{\mathcal{V}}^{-1}(\mathtt{head}(\mathtt{c}))\})\{\mathtt{b}\setminus\phi_{\mathcal{V}}^{-1}(\mathtt{head}(\mathtt{d}))\}]))])])])] \\
= \;& [\mathtt{get}(\phi_{\mathcal{L}}(l'),\lambda\mathtt{d}.[\mathtt{set}((\phi_{\mathcal{L}}(l'),\mathtt{tail}(\mathtt{d})),[\mathtt{get}(\phi_{\mathcal{L}}(l),\lambda\mathtt{c}. \\
& \quad [\mathtt{set}((\phi_{\mathcal{L}}(l'),\mathtt{tail}(\mathtt{c})),H([(\mathtt{t}\{\mathtt{b}\setminus\phi_{\mathcal{V}}^{-1}(\mathtt{head}(\mathtt{d}))\})\{\mathtt{a}\setminus\phi_{\mathcal{V}}^{-1}(\mathtt{head}(\mathtt{c}))\}]))])])])] \\
= \;& H([\mathtt{pop}(l',\lambda\mathtt{b}.[\mathtt{pop}(l,\lambda\mathtt{a}.[\mathtt{t}])])]).
\end{aligned}
$$

Equation 4:

$$
\begin{aligned}
& H([\mathtt{pop}(l,\lambda\mathtt{a}.[\mathtt{push}((l',\mathtt{v}),[\mathtt{t}])])]) \\
= \;& [\mathtt{get}(\phi_{\mathcal{L}}(l),\lambda\mathtt{b}.[\mathtt{set}((\phi_{\mathcal{L}}(l),\mathtt{tail}(\mathtt{b})),[\mathtt{get}(\phi_{\mathcal{L}}(l'),\lambda\mathtt{c}. \\
& \quad [\mathtt{set}((\phi_{\mathcal{L}}(l'),\phi_{\mathcal{V}}(\mathtt{v})\cdot\mathtt{c}),H([\mathtt{t}\{\mathtt{a}\setminus\phi_{\mathcal{V}}^{-1}\mathtt{head}(\mathtt{b})\}]))])])])] \\
= \;& [\mathtt{get}(\phi_{\mathcal{L}}(l'),\lambda\mathtt{c}.[\mathtt{set}((\phi_{\mathcal{L}}(l'),\phi_{\mathcal{V}}(\mathtt{v})\cdot\mathtt{c}),[\mathtt{get}(\phi_{\mathcal{L}}(l'),\lambda\mathtt{c}. \\
& \quad [\mathtt{set}((\phi_{\mathcal{L}}(l),\mathtt{tail}(\mathtt{b})),H([\mathtt{t}\{\mathtt{a}\setminus\phi_{\mathcal{V}}^{-1}\mathtt{head}(\mathtt{b})\}]))])])])] \\
= \;& [\mathtt{get}(\phi_{\mathcal{L}}(l'),\lambda\mathtt{c}.[\mathtt{set}((\phi_{\mathcal{L}}(l'),\phi_{\mathcal{V}}(\mathtt{v})\cdot\mathtt{c}),([\mathtt{get}(\phi_{\mathcal{L}}(l'),\lambda\mathtt{c}. \\
& \quad [\mathtt{set}((\phi_{\mathcal{L}}(l),\mathtt{tail}(\mathtt{b})),H([\mathtt{t}]))])])\{\mathtt{a}\setminus\phi_{\mathcal{V}}^{-1}\mathtt{head}(\mathtt{b})\})])] \\
= \;& H([\mathtt{push}((l',\mathtt{v}),[\mathtt{pop}(l,\lambda\mathtt{a}.\mathtt{t})])]).
\end{aligned}
$$

Equation 5:

$$
\begin{aligned}
& H([\mathtt{push}((l,\mathtt{v}),[\mathtt{push}((l',\mathtt{w}),[\mathtt{t}])])]) \\
= \;& [\mathtt{get}(\phi_{\mathcal{L}}(l),\lambda\mathtt{a}.[\mathtt{set}((\phi_{\mathcal{L}}(l),\phi_{\mathcal{V}}(\mathtt{v})\cdot\mathtt{a}),[\mathtt{get}(\phi_{\mathcal{L}}(l'),\lambda\mathtt{b}.[\mathtt{set}((\phi_{\mathcal{L}}(l'),\phi_{\mathcal{V}}(\mathtt{w})\cdot\mathtt{b}),H([\mathtt{t}]))])])])] \\
= \;& [\mathtt{get}(\phi_{\mathcal{L}}(l'),\lambda\mathtt{b}.[\mathtt{set}((\phi_{\mathcal{L}}(l'),\phi_{\mathcal{V}}(\mathtt{w})\cdot\mathtt{b}),[\mathtt{get}(\phi_{\mathcal{L}}(l),\lambda\mathtt{a}.[\mathtt{set}((\phi_{\mathcal{L}}(l),\phi_{\mathcal{V}}(\mathtt{v})\cdot\mathtt{a}),H([\mathtt{t}]))])])])] \\
= \;& H([\mathtt{push}((l',\mathtt{w}),[\mathtt{push}((l,\mathtt{v}),[\mathtt{t}])])]).
\end{aligned}
$$

Having shown that $h_{\mathtt{pop}}$ and $h_{\mathtt{push}}$ do form a $\mathcal{S}$-model, we can now conclude that $\mathcal{S}$ is indeed a subtheory of $\mathcal{G}$.

8.2.1 The Comodel for Infinite Stacks

Having proved that $\mathcal{S}$ can be captured by (*i.e.* is a *subtheory* of) $\mathcal{G}$, we can now work out what is the carrier of the comodel of $\mathcal{S}$.

Example 8.3 (Infinite Stacks)**.** *Let $\mathcal{S}$ be the algebraic theory for infinite stacks. A comodel $\mathtt{W}$ for $\mathcal{S}$ is a set $|\mathtt{W}|$ together with the following functions:*

$$[\![\mathtt{pop}]\!]^{\mathtt{W}} : \mathcal{L} \times |\mathtt{W}| \to \mathcal{V} \times |\mathtt{W}|$$
$$[\![\mathtt{push}]\!]^{\mathtt{W}} : (\mathcal{L} \times \mathcal{V}) \times |\mathtt{W}| \to 1 \times |\mathtt{W}|$$

subject to the following equalities:

$$\begin{aligned}
[\![\mathtt{pop}(l, \lambda \mathtt{a}.\mathtt{push}((l,\mathtt{a}), \lambda \mathtt{b}.\mathtt{t}))]\!]^{\mathtt{W}} &= [\![\mathtt{t}]\!]^{\mathtt{W}} \text{ if } \mathtt{a} \notin \mathtt{fv}(\mathtt{t}) \\
[\![\mathtt{push}((l,\mathtt{v}), \lambda \mathtt{a}.\mathtt{pop}(l, \lambda \mathtt{b}.\mathtt{t}))]\!]^{\mathtt{W}} &= [\![\mathtt{t}\{\mathtt{b} \setminus \mathtt{v}\}]\!]^{\mathtt{W}} \\
[\![\mathtt{pop}(l, \lambda \mathtt{a}.\mathtt{pop}(l', \lambda \mathtt{b}.\mathtt{t}))]\!]^{\mathtt{W}} &= [\![\mathtt{pop}(l', \lambda \mathtt{b}.\mathtt{pop}(l, \lambda \mathtt{a}.\mathtt{t}))]\!]^{\mathtt{W}} \\
[\![\mathtt{pop}(l, \lambda \mathtt{a}.\mathtt{push}((l',\mathtt{v}), \lambda \mathtt{b}.\mathtt{t}))]\!]^{\mathtt{W}} &= [\![\mathtt{push}((l',\mathtt{v}), \lambda \mathtt{b}.\mathtt{pop}(l, \lambda \mathtt{a}.\mathtt{t}))]\!]^{\mathtt{W}} \\
[\![\mathtt{push}((l,\mathtt{v}), \lambda \mathtt{a}.\mathtt{push}((l',\mathtt{w}), \lambda \mathtt{b}.\mathtt{t}))]\!]^{\mathtt{W}} &= [\![\mathtt{push}((l',\mathtt{w}), \lambda \mathtt{b}.\mathtt{push}((l,\mathtt{v}), \lambda \mathtt{a}.\mathtt{t}))]\!]^{\mathtt{W}}
\end{aligned}$$

Let $\mathcal{V}^\omega$ be the set of infinite sequences over $\mathcal{V}$. Then, $|\mathtt{W}|$ is a set of states $\mathcal{S} = (\mathcal{V}^\omega)^{\mathcal{L}}$, and

$$\begin{aligned}
[\![\mathtt{pop}]\!]^{\mathtt{W}} : \mathcal{L} \times \mathcal{S} &\to \mathcal{V} \times \mathcal{S} \\
(l, s) &\mapsto (\mathtt{head}(s(l)), s\{l := \mathtt{tail}(s(l))\}), \\
[\![\mathtt{push}]\!]^{\mathtt{W}} : (\mathcal{L} \times \mathcal{V}) \times \mathcal{S} &\to 1 \times \mathcal{S} \\
((l, \mathtt{v}), s) &\mapsto (*, s\{l := \mathtt{v} \cdot s(l)\}).
\end{aligned}$$

Having worked out the carrier of the comodel of $\mathcal{S}$, the difference between the operational and the semantical behavior of operations can be summarized in Table 8.1. In particular, notice that there is no mismatch between the operational and denotational behaviors of the pop and push operations.

	Operational Behavior (State $s \in (\mathcal{V}^\omega)^{\mathcal{L}}$)	**Denotational Behavior** (State $\mathcal{S} \in (M^\omega)^{\mathcal{L}}$)
$\mathtt{pop}(l, \lambda \mathtt{a}.\mathtt{t})$	$[\![\mathtt{pop}]\!]^{(\mathcal{V}^\omega)^{\mathcal{L}}}(l,s) = (\mathtt{head}(s\,(l)), s\{l := \mathtt{tail}(s\,(l))\})$ Variable x is replaced with $\mathtt{head}(s\,(l))$. The updated state is $s\{l := \mathtt{tail}(s\,(l))\}$.	$[\![\mathtt{pop}]\!]^{(M^\omega)^{\mathcal{L}}}(l,\mathcal{S}) = (\mathtt{head}(\mathcal{S}\,(l)), \mathcal{S}\{l := \mathtt{tail}(\mathcal{S}\,(l))\})$ Variable x has type $\mathtt{head}(\mathcal{S}\,(l))$. The updated state has type $\mathcal{S}\{l := \mathtt{tail}(\mathcal{S}\,(l))\}$.
$\mathtt{push}((l, \mathtt{v} \in \mathcal{V}), t)$	$[\![\mathtt{push}]\!]^{(\mathcal{V}^\omega)^{\mathcal{L}}}((l, \mathtt{v} \in \mathcal{V}), s) = (\star, s\{l := \mathtt{v} \cdot s\,(l)\})$ The updated state is $s\{l := \mathtt{v} \cdot s\,(l)\}$	$[\![\mathtt{push}]\!]^{(M^\omega)^{\mathcal{L}}}((l, M), \mathcal{S}) = (\star, \mathcal{S}\{l := M \cdot \mathcal{S}\,(l)\})$ The updated state is assigned type $\mathcal{S}\{l := M \cdot \mathcal{S}\,(l)\}$.

Table 8.1: Comparison between the operational and denotational behavior of operations interacting with infinite stacks.

8.3 The λ-Calculus with Infinite Stacks

This section extends the λ-calculus with primitive operations for manipulating infinite stacks, which we call the $\lambda^{\mathcal{S}}$-calculus.

Most definitions and properties of the $\lambda^{\mathcal{S}}$-calculus can be obtained from those in Section 7.2.2 for the $\lambda^{\mathcal{G}}$-calculus by simply replacing operations `get` and `set` with operations `pop` and `push`, respectively. As such, we are mainly going to focus on presenting the syntax of the language and its operational semantics.

8.3.1 Syntax and Operational Semantics

The syntax of the $\lambda^{\mathcal{S}}$-calculus is the extension of the syntax of the λ-calculus with a notion of state corresponding to a mapping from locations to infinite stacks, algebraic operations for interacting with the infinite stacks (according to Example 8.1), and configurations.

Definition 8.4 (States and $\lambda^{\mathcal{S}}$-Terms)**.** Let $\mathcal{L}$ be a finite set of location names, $\mathcal{V}$ be a countable set of arguments, and $\mathcal{X}$ be a countable *set of variables*. The sets of terms $\Lambda^{\mathcal{S}}$, infinite stacks, states $\mathcal{S}$, and configurations $\Lambda^{\mathcal{S}} \times \mathcal{S}$ of the $\lambda^{\mathcal{S}}$-calculus are generated by the following four grammars, respectively:

$$
\begin{array}{lrcl}
\text{Terms:} & t,u,r,e & ::= & x \in \mathcal{X} \mid \lambda x.t \mid t\,u \mid \mathtt{pop}(l, \lambda x.t) \mid \mathtt{push}((l, u \in \mathcal{V}), t) \\
\text{Infinite Stacks:} & f,g & ::= & (t \in \mathcal{V}) \cdot f \text{ (taken coinductively)} \\
\text{States:} & s,q & ::= & \{l \mapsto f_l\}_{l \in \mathcal{L}} \\
\text{Configurations:} & c,c' & ::= & (t,s).
\end{array}
$$

States are defined according to Section 8.2.1.

The operations over infinite stacks are defined as follows.

Definition 8.5 (Operations On Infinite Stacks)**.** Let $f := (t \in \mathcal{V}) \cdot g$. Then

- we write $\mathtt{head}(f)$ for the argument at the head of f, *i.e.* $\mathtt{head}(f) = t \in \mathcal{V}$;
- and $\mathtt{tail}(f)$ for the infinite stack corresponding to the tail of f, *i.e.* $\mathtt{tail}(f) = g$.

Definition 8.6 (Well-Formed $\lambda^{\mathcal{S}}$-Terms)**.** We say that a $\lambda^{\mathcal{S}}$-term $t \in \Lambda^{\mathcal{S}}$ is *well-formed* if and only if for every subterm $\mathtt{push}((l, r \in \mathcal{V}), u)$, $(\mathtt{fv}(r) \cap \mathtt{cv}(t)) \subseteq \mathtt{cv}_l(t)$. This ensures that pushed arguments do not introduce free variables into unrelated continuations.

Notice that Definition 8.6 is identical to that of Definition 7.4 for the $\lambda^{\mathcal{G}}$-calculus, but mentions `push` instead of `set` operations.

Call-By-Name. The set of arguments $\mathcal{V}$ is the set of $\lambda^{\mathcal{S}}$-terms $\Lambda^{\mathcal{S}}$. CBN contexts (in Definition 2.11) are naturally lifted to closed configurations. CBN reduction is defined for closed configurations as follows.

Definition 8.7 (CBN Reduction)**.** The *CBN reduction strategy* is denoted by $\Rightarrow_{\mathtt{CBN}}$ and defined over closed configurations as the closure by CBN contexts of the following three rewriting steps:

$$\begin{array}{rcl}
((\lambda x.t)\, u, s) & \mapsto_{\beta} & (t\{x \setminus u\}, s) \\
(\mathtt{pop}(l, \lambda x.t), s) & \mapsto_{\mathtt{pop}} & (t\{x \setminus \mathtt{head}(s\,(l))\}, s\{l := \mathtt{tail}(s\,(l))\}) \\
(\mathtt{push}((l, u), t), s) & \mapsto_{\mathtt{push}} & (t, s\{l := u \cdot s\,(l)\}).
\end{array}$$

Example 8.8 (CBN Reduction)**.** *The following example is obtained from Example 7.14 by replacing the* `get` *and* `set` *operations with* `pop` *and* `push` *operations, respectively:*

$$\begin{array}{rcl}
(\mathtt{push}((l, (\lambda x.x)\, \lambda y.y), \mathtt{pop}(l, \lambda z.z\, z)), s) & \Rightarrow_{\mathtt{CBN}(\mathtt{push})} & (\mathtt{pop}(l, \lambda z.z\, z), s\{l := (\lambda x.x)\, \lambda y.y\}) \\
& \Rightarrow_{\mathtt{CBN}(\mathtt{pop})} & (((\lambda x.x)\, \lambda y.y)\, ((\lambda x.x)\, \lambda y.y), s) \\
& \Rightarrow_{\mathtt{CBN}(\beta)} & (\lambda y.y\, ((\lambda x.x)\, \lambda y.y), s) \\
& \Rightarrow_{\mathtt{CBN}(\beta)} & ((\lambda x.x)\, \lambda y.y, s) \\
& \Rightarrow_{\mathtt{CBN}(\beta)} & (\lambda y.y, s).
\end{array}$$

Notice how the `pop` *operation undoes the effect of the* `push` *operation on the state.*

Call-By-Value. The set of ($\lambda^{\mathcal{S}}$-)values of the $\lambda^{\mathcal{S}}$-calculus is denoted by $\mathrm{V}^{\mathcal{S}} \subseteq \Lambda^{\mathcal{S}}$ and generated by the same grammar that generates the set of values of the $\lambda^{\mathcal{G}}$-calculus (see Section 7.1.1).

The set of arguments $\mathcal{V}$ is the set of values $\mathrm{V}^{\mathcal{S}}$. CBV contexts (see Definition 2.14) are naturally lifted to closed configurations. CBV reduction is defined for closed configurations as follows.

Definition 8.9 (CBV Reduction)**.** The *CBV reduction strategy* is denoted by $\Rightarrow_{\mathtt{CBV}}$ and defined over closed configurations as the closure by CBV contexts of the following three rewriting steps:

$$\begin{array}{rcl}
((\lambda x.t)\, v, s) & \mapsto_{\beta_{\mathrm{v}}} & (t\{x \setminus v\}, s) \\
(\mathtt{pop}(l, \lambda x.t), s) & \mapsto_{\mathtt{pop}} & (t\{x \setminus \mathtt{head}(s\,(l))\}, s\{l := \mathtt{tail}(s\,(l))\}) \\
(\mathtt{push}((l, v), t), s) & \mapsto_{\mathtt{push}} & (t, s\{l := v \cdot s\,(l)\}).
\end{array}$$

Observe that the operational rules for stack operations are induced by the comodel (see Section 8.2.1 and Remark 5.39).

Example 8.10 (CBV Reduction)**.** *The following example is obtained from Example 7.16 by replacing the* `get` *and* `set` *operations with* `pop` *and* `push`*, respectively:*

$$\begin{aligned}(\mathtt{push}((l,\lambda x.x),\mathtt{pop}(l,\lambda y.y\,((\lambda z.z)\,y))),s) &\Rightarrow_{\mathtt{CBV}(\mathtt{push})} (\mathtt{pop}(l,\lambda y.y\,((\lambda z.z)\,y)),s\{l := \lambda x.x\})\\ &\Rightarrow_{\mathtt{CBV}(\mathtt{pop})} ((\lambda x.x)\,((\lambda z.z)\,\lambda x.x),s)\\ &\Rightarrow_{\mathtt{CBV}(\beta_{\mathtt{v}})} ((\lambda x.x)\,\lambda x.x,s)\\ &\Rightarrow_{\mathtt{CBV}(\beta_{\mathtt{v}})} (\lambda x.x,s).\end{aligned}$$

The set of `CBN`- and `CBV`-normal forms are the same as those for the $\lambda^{\mathcal{G}}$-calculus (see Lemma 7.18).

8.4 The $\lambda!$-Calculus with Infinite Stacks

We now extend the $\lambda!$-calculus introduced in Chapter 3 with primitive operations for interacting with infinite stacks. We will call this new calculus the $\lambda!^{\mathcal{S}}$-calculus.

In a parallel to the previous section, most definitions and properties of the $\lambda!^{\mathcal{S}}$-calculus can be obtained from those in Section 7.2.1 for the $\lambda!^{\mathcal{G}}$-calculus by simply replacing operations `get` and `set` with operations `pop` and `push`, respectively. However, while in the previous section we skipped most of the details, here we are going to restate some of the definitions and properties common to those of the $\lambda!^{\mathcal{G}}$-calculus, whenever it is important to highlight any key differences.

8.4.1 Syntax and Operational Semantics

The syntax of the $\lambda!^{\mathcal{S}}$-calculus is the extension of the syntax of the λ-calculus with a notion of state corresponding to a mapping from locations to infinite stacks, algebraic operations for interacting with the infinite stacks (according to Example 8.1), and configurations.

Definition 8.11 (States and $\lambda!^{\mathcal{S}}$-Terms)**.** Let $\mathcal{L}$ be a finite set of location names and $\mathcal{X}$ be a countable set of variables. The sets of values $!\mathrm{V}^{\mathcal{S}}$, terms $!\Lambda^{\mathcal{S}}$, infinite stacks $(!\mathrm{V}^{\mathcal{S}})^{\omega}$, states $\mathcal{S}$, and configurations $!\Lambda^{\mathcal{S}} \times \mathcal{S}$ of the $\lambda!^{\mathcal{S}}$-calculus are generated by the following four grammars, respectively:

Values:	v, w	::=	$x \in \mathcal{X} \mid !t$
Terms:	t, u, r, e	::=	$v \mid \lambda x.t \mid t\,u \mid \mathtt{der}(t) \mid \mathtt{pop}(l, \lambda x.t) \mid \mathtt{push}((l, v), t)$
Stacks:	f, g	::=	$v \cdot f$ (taken coinductively)
States:	s, q	::=	$\{l \mapsto f_l\}_{l \in \mathcal{L}}$
Configurations:	c, c'	::=	(t, s).

The notions of free and bound variables for the $\lambda!^{\mathcal{G}}$-calculus are as expected, as are those of α-congruence, substitution, context, and closed and open states and $\lambda!^{\mathcal{S}}$-terms.

CBPV contexts (in Definition 3.3) are naturally lifted to closed configurations, just as was done for the $\lambda!^{\mathcal{G}}$-calculus. CBPV reduction is defined for closed configurations as follows.

Definition 8.12 (CBPV Reduction)**.** The *CBPV reduction strategy* is denoted by $\Rightarrow_{\mathtt{CBPV}}$ and defined over closed configurations as the closure by CBPV contexts of the following three rewriting steps:

$$\begin{array}{rcl}
((\lambda x.t)\, v, s) & \mapsto_{\beta_{\mathsf{v}}} & (t\{x \setminus v\}, s) \\
(\mathtt{der}(!t), s) & \mapsto_{\beta_!} & (t, s) \\
(\mathtt{pop}(l, \lambda x.t), s) & \mapsto_{\mathtt{pop}} & (t\{x \setminus \mathtt{head}(s\,(l))\}, s\{l := \mathtt{tail}(s\,(l))\}) \\
(\mathtt{push}((l, v), t), s) & \mapsto_{\mathtt{push}} & (t, s\{l := v \cdot s\,(l)\}).
\end{array}$$

Observe that the rewriting steps for operations are defined according to Section 8.2.1 and Remark 5.39.

Example 8.13 (CBPV Reduction)**.** *We are going to present two examples of CBPV reduction, both starting at (syntactically) different, but $\mathcal{S}$-equivalent, configurations. Accordingly, they have the same* CBPV*-normal form. However, the former is shorter than the latter:*

$$\begin{array}{rl}
 & (\mathtt{push}((l, !\lambda x.x), \mathtt{pop}(l, \lambda y.\mathtt{der}(y)\, y)), s) \\
\Rightarrow_{\mathtt{push}} & (\mathtt{pop}(l, \lambda y.\mathtt{der}(y)\, y), s\{l := (!\lambda x.x) \cdot s\,(l)\}) \\
\Rightarrow_{\mathtt{pop}} & (\mathtt{der}((!\lambda x.x))\, !\lambda x.x, s) \\
\Rightarrow_{\beta_!} & ((\lambda x.x)\, !\lambda x.x, s) \\
\Rightarrow_{\beta_{\mathsf{v}}} & (!\lambda x.x, s).
\end{array}$$

$$\begin{array}{rl}
 & (\mathtt{push}((l, !\lambda x.x), \mathtt{push}((l, !\lambda x.x), \mathtt{pop}(l, \lambda y.\mathtt{pop}(l, \lambda z.\mathtt{der}(y)\, z)))), s) \\
\Rightarrow_{\mathtt{push}} & (\mathtt{push}((l, !\lambda x.x), \mathtt{pop}(l, \lambda y.\mathtt{pop}(l, \lambda z.\mathtt{der}(y)\, z))), s\{l := (!\lambda x.x) \cdot s\,(l)\}) \\
\Rightarrow_{\mathtt{push}} & (\mathtt{pop}(l, \lambda y.\mathtt{pop}(l, \lambda z.\mathtt{der}(y)\, z)), s\{l := (!\lambda x.x) \cdot (!\lambda x.x) \cdot s\,(l)\}) \\
\Rightarrow_{\mathtt{pop}} & (\mathtt{pop}(l, \lambda z.\mathtt{der}((!\lambda x.x))\, z), s\{l := (!\lambda x.x) \cdot s\,(l)\}) \\
\Rightarrow_{\mathtt{pop}} & (\mathtt{der}((!\lambda x.x))\, !\lambda x.x, s) \\
\Rightarrow_{\beta_!} & ((\lambda x.x)\, !\lambda x.x, s) \\
\Rightarrow_{\beta_{\mathsf{v}}} & (!\lambda x.x, s).
\end{array}$$

The set of (closed) syntactic CBPV-normal forms is identical to that of the $\lambda!^{\mathcal{G}}$-calculus (see Proposition 7.25).

Proposition 8.14 (Characterization of CBPV-Normal Forms)**.** *Let $(t, s) \in !\Lambda^{\mathcal{S}} \times \mathcal{S}_\circ$. Then, $(t, s) \not\Rightarrow_{\mathtt{CBPV}}$ if and only if $t \in \mathtt{NO}_{\mathtt{CBPV}}$.*

The sets of *clashes* and *clash-free configurations* are defined similarly to those for the $\lambda!^{\mathcal{G}}$-calculus (see Proposition 7.29).

Definition 8.15 (Syntactic Characterization of Clash-Free CBPV-Normal Forms)**.** The set of *clash-free* CBPV-normal forms of the $\lambda!^{\mathcal{S}}$-calculus is $\{(t,s) \mid t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}, s \in \mathcal{S}_\circ\}$, where $\mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}} := !\mathrm{V}^{\mathcal{S}}_\circ \cup \{\lambda x.t \mid \lambda x.t \in !\Lambda^{\mathcal{S}}_\circ\}$.

Proposition 8.16 (Characterization of Clash-Free CBPV-Normal Forms)**.** *Let* $(t,s) \in !\Lambda^{\mathcal{S}}$*. Then,* $(t,s) \not\to_{\mathtt{CBPV}}$ *and* t *is clash-free if and only if* $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$*.*

As we did for the $\lambda!^{\mathcal{G}}$-calculus, we will show how *clash-free* terms can be characterized via a lifting of the non-idempotent intersection type system presented in Chapter 3.

8.4.2 Subsuming Call-By-Name and Call-By-Value

The encodings of Section 3.2 can also be extended to the $\lambda!^{\mathcal{S}}$-calculus.

Terms from the $\lambda^{\mathcal{S}}$-calculus are translated to terms of the $\lambda!^{\mathcal{S}}$-calculus much in the same way as term from the $\lambda^{\mathcal{G}}$-calculus were translated to terms of the $\lambda!^{\mathcal{G}}$-calculus (recall Definition 7.30).

Definition 8.17 (Translations of Terms)**.** Consider the following inductively defined translation of terms of the $\lambda^{\mathcal{S}}$-calculus to terms of the $\lambda!^{\mathcal{S}}$-calculus:

$$
\begin{array}{rcl}
(\cdot)^{\mathtt{CBN}} : \Lambda^{\mathcal{S}} & \to & !\Lambda^{\mathcal{S}} \\
x & \mapsto & \mathtt{der}(x) \\
\lambda x.t & \mapsto & \lambda x.(t)^{\mathtt{CBN}} \\
t\,u & \mapsto & (t)^{\mathtt{CBN}}\,!(u)^{\mathtt{CBN}} \\
\mathtt{pop}(l, \lambda x.t) & \mapsto & \mathtt{pop}(l, \lambda x.(t)^{\mathtt{CBN}}) \\
\mathtt{push}((l,u),t) & \mapsto & \mathtt{push}((l, !(u)^{\mathtt{CBN}}), (t)^{\mathtt{CBN}})
\end{array}
\qquad
\begin{array}{rcl}
(\cdot)^{\mathtt{CBV}} : \Lambda^{\mathcal{S}} & \to & !\Lambda^{\mathcal{S}} \\
x & \mapsto & x \\
\lambda x.t & \mapsto & !\lambda x.(t)^{\mathtt{CBV}} \\
t\,u & \mapsto & \mathtt{der}((t)^{\mathtt{CBV}})\,(u)^{\mathtt{CBV}} \\
\mathtt{pop}(l, \lambda x.t) & \mapsto & \mathtt{pop}(l, \lambda x.(t)^{\mathtt{CBV}}) \\
\mathtt{push}((l,v),t) & \mapsto & \mathtt{push}((l, (v)^{\mathtt{CBV}}), (t)^{\mathtt{CBV}}).
\end{array}
$$

The same can be said about the translation of configurations from the $\lambda^{\mathcal{S}}$-calculus to configurations of the $\lambda!^{\mathcal{S}}$-calculus. However, since states are now mappings from locations to *stacks* of terms instead of values, we first define the translations of stacks from the $\lambda^{\mathcal{S}}$-calculus to stacks of the $\lambda!^{\mathcal{S}}$-calculus.

Definition 8.18 (Translations of Stacks)**.** Consider the following coinductively defined translation of stacks of the $\lambda^{\mathcal{S}}$-calculus to stacks of the $\lambda!^{\mathcal{S}}$-calculus:

$$
\begin{array}{rcl}
(\cdot)^{\mathtt{CBN}} : (\Lambda^{\mathcal{S}})^{\omega} & \to & (!\Lambda^{\mathcal{S}})^{\omega} \\
t \cdot f & \mapsto & !(t)^{\mathtt{CBN}} \cdot (f)^{\mathtt{CBN}}
\end{array}
\qquad
\begin{array}{rcl}
(\cdot)^{\mathtt{CBV}} : (\mathrm{V}^{\mathcal{S}})^{\omega} & \to & (!\mathrm{V}^{\mathcal{S}})^{\omega} \\
v \cdot f & \mapsto & (v)^{\mathtt{CBV}} \cdot (f)^{\mathtt{CBV}}.
\end{array}
$$

Now, we can can define the translation of configurations in terms of the above translations.

Definition 8.19 (Translations of Configurations)**.** Consider the following inductively defined translation of states of the $\lambda^{\mathcal{S}}$-calculus to states of the $\lambda!^{\mathcal{S}}$-calculus:

$$\begin{array}{rcl} (\cdot)^{\mathtt{CBN}} : \mathcal{S} & \to & \mathcal{S} \\ \{l \mapsto f_l\}_{l \in \mathcal{L}} & \mapsto & \{l \mapsto (f_l)^{\mathtt{CBN}}\}_{l \in \mathcal{L}} \end{array} \qquad \begin{array}{rcl} (\cdot)^{\mathtt{CBV}} : \mathcal{S} & \to & \mathcal{S} \\ \{l \mapsto f_l\}_{l \in \mathcal{L}} & \mapsto & \{l \mapsto (f_l)^{\mathtt{CBV}}\}_{l \in \mathcal{L}}. \end{array}$$

The translation of configurations of the $\lambda^{\mathcal{S}}$-calculus to configurations of the $\lambda!^{\mathcal{S}}$-calculus are then defined as follows:

$$\begin{array}{rcl} (\cdot)^{\mathtt{CBN}} : \Lambda^{\mathcal{S}} \times \mathcal{S} & \to & !\Lambda^{\mathcal{S}} \times \mathcal{S} \\ (t, s) & \mapsto & ((t)^{\mathtt{CBN}}, (s)^{\mathtt{CBN}}) \end{array} \qquad \begin{array}{rcl} (\cdot)^{\mathtt{CBV}} : \Lambda^{\mathcal{S}} \times \mathcal{S} & \to & !\Lambda^{\mathcal{S}} \times \mathcal{S} \\ (t, s) & \mapsto & ((t)^{\mathtt{CBV}}, (s)^{\mathtt{CBV}}). \end{array}$$

The correctness of full simulations for the case of the $\lambda!^{\mathcal{S}}$-calculus can be shown by following the same overall recipe as for the $\lambda!^{\mathcal{G}}$-calculus. For that reason, we are just going to state the main result of this section.

Theorem 8.20 (Correctness of Full Simulations)**.** *Let $s \in \mathcal{S}_\circ$, and $t \in \Lambda^{\mathcal{S}}_\circ$.*

1. *If $(t, s) \overset{n\cdot\beta + o\cdot\mathtt{pop} + u\cdot\mathtt{push}}{\Rightarrow_{\mathtt{CBN}}} (u, q)$, then $((t, s))^{\mathtt{CBN}} \overset{n\cdot\beta_{\mathtt{v}} + m\cdot\beta_! + o\cdot\mathtt{pop} + u\cdot\mathtt{push}}{\Rightarrow_{\mathtt{CBPV}\cup\mathtt{ADMIN}}} ((u, q))^{\mathtt{CBN}}$, for some $m \geq 0$.*
2. *If $(t, q) \overset{n\cdot\beta_{\mathtt{v}} + o\cdot\mathtt{pop} + u\cdot\mathtt{push}}{\Rightarrow_{\mathtt{CBV}}} (u, q)$, then $((t, s))^{\mathtt{CBV}} \overset{n\cdot\beta_{\mathtt{v}} + m\cdot\beta_! + o\cdot\mathtt{pop} + u\cdot\mathtt{push}}{\Rightarrow_{\mathtt{CBPV}}} ((u, q))^{\mathtt{CBV}}$, for some $m \geq 0$.*

This establishes that the $\lambda!^{\mathcal{S}}$-calculus accurately reproduces the behavior of both the CBN and CBV $\lambda^{\mathcal{S}}$-calculi, forming the foundation for a unified quantitative analysis.

Example 8.21 (Full Simulations)**.** *Recall the CBN reduction sequence for the configuration in Example 8.8:*

$$(\mathtt{push}((l, (\lambda x.x)\, \lambda y.y), \mathtt{pop}(l, \lambda z.z\, z)), s) \overset{3\cdot\beta + 1\cdot\mathtt{pop} + 1\cdot\mathtt{push}}{\Rightarrow_{\mathtt{CBN}}} (\lambda y.y, s\{l := (\lambda x.x)\, \lambda y.y\});$$

and the CBV reduction sequence for the configuration in Example 8.10:

$$(\mathtt{push}((l, \lambda x.x), \mathtt{pop}(l, \lambda y.y\, ((\lambda z.z)\, y))), s) \overset{2\cdot\beta_{\mathtt{v}} + 1\cdot\mathtt{pop} + 1\cdot\mathtt{push}}{\Rightarrow_{\mathtt{CBV}}} (\lambda x.x, s\{l := \lambda x.x\}).$$

In Example 8.13, we can see how Lemma 8.20 holds for the translation of the above configurations.

The following is the CBPV reduction sequence corresponding to the CBN translation of the first configuration:

$$(\mathtt{push}((l, !((\lambda x.\mathtt{der}(x))\, !\lambda y.\mathtt{der}(y))), \mathtt{pop}(l, \lambda z.\mathtt{der}(z)\, !\mathtt{der}(z))), q)$$

$$\overset{3\cdot\beta_{\mathtt{v}} + 5\cdot\beta_! + 1\cdot\mathtt{pop} + 1\cdot\mathtt{push}}{\Rightarrow_{\mathtt{CBPV}}}$$

$$(\lambda y.\mathtt{der}(y), q\{l := !(\lambda x.\mathtt{der}(x))\, !\lambda y.\mathtt{der}(y)\})$$

The following is the CBPV reduction sequence corresponding to the CBV translation of the second configuration:

$$(\mathtt{push}((l, !\lambda x.x), \mathtt{pop}(l, \lambda y.\mathtt{der}(y)\, ((\lambda z.z)\, y))), q) \overset{2\cdot\beta_{\mathtt{v}} + 2\cdot\beta_! + 1\cdot\mathtt{pop} + 1\cdot\mathtt{push}}{\Rightarrow_{\mathtt{CBPV}}} (!\lambda x.x, q\{l := !\lambda x.x\})$$

8.5 State-Passing Non-Idempotent Intersection Types

Perhaps surprisingly, the quantitative systems introduced in this section are very similar to those introduced in Section 7.3 for global state. Indeed, at a syntactic level, they can mostly be obtained by respectively replacing the occurrences of the `get` and `set` operations with occurrences of the `pop` and `push` operations. However, we have already seen how they deeply differ in how closely they follow the operational semantics of the languages to which they pertain. Accordingly, the sets of types both for the CBN and CBV $\lambda^{\mathcal{S}}$-calculi are generated by the same grammar as those for the CBN and CBV $\lambda^{\mathcal{G}}$-calculus (see Definitions 7.45 and 7.67). However, *list types* are now called *stack types* and have a different meaning: list types assign different multi(-monadic) types to the same value across different uses; stack types assign (possibly) different multi(-monadic) types to distinct values occurring at different depths of an infinite stack. Moreover, *values* were assigned the *empty list type* whenever they were not going to be used anymore; *infinite stacks of values* are assigned the *empty stack type* whenever they are not going to be used anymore. Notice that this means that the set of state types is a mapping from locations to *finite* stacks of multi(-monadic) types, *i.e.* $\mathcal{S} = (M^*)^{\mathcal{L}}$. At first glance, this may seem at odds with Table 8.1, where $\mathcal{S}$ was described as $(M^\omega)^{\mathcal{L}}$. This discrepancy is intentional: we are only interested in characterizing finite computations, and finite computations can consume only a finite amount of resources. More generally, any infinite object can only be semantically meaningful through the sum of its finite parts. Since infinite stacks can only ever be partially consumed, it is natural for their quantitative semantics to reflect this fact as well∗.

8.5.1 Call-By-Name

We now present the language of types for the CBN $\lambda^{\mathcal{S}}$-calculus.

Definition 8.22 (CBN Types)**.** The sets of types $\mathcal{T}y^{\mathcal{S}}_{\mathtt{CBN}}$ for the CBN $\lambda^{\mathcal{S}}$-calculus are generated by the following grammar:

$$
\begin{array}{lrcl}
\text{Stack Types:} & L, L' & ::= & [\,] \mid M \cdot L' \\
\text{State Types:} & S, Q & ::= & \{l \mapsto L_l\}_{l \in \mathcal{L}} \\
\text{Monadic Types:} & \mathbb{A} & ::= & S \Rightarrow P \\
\text{Product Types:} & P & ::= & A \times S \\
\text{Multi-Monadic Types} & M, N & ::= & \{\!\{ \mathbb{A}_i \}\!\}_{i \in \mathtt{I}} \\
\text{Types:} & A & ::= & \star \mid M \to P
\end{array}
$$

Note that *stack types* are *finite*, contrary to *stacks*, which are *infinite*.

∗Note that this does not mean that infinite stack types are meaningless. Indeed, infinite stack types are key if one wants instead to characterize *infinite* computations, which is ongoing work of the author of this thesis.

The type system for the CBN $\lambda^{\mathcal{S}}$-calculus is as follows.

Definition 8.23 (CBN Type System)**.** The *CBN Type System* $\mathcal{N}_{\mathcal{S}}$ assigns CBN (multi-monadic) types to $\lambda^{\mathcal{S}}$-terms, and consists of the following translations of the typing rules in Definition 4.7 for the CBN λ-calculus:

$$\frac{}{x : \{\!\{ \mathbb{A} \}\!\} \vdash x : \mathbb{A}}\ \texttt{Ax}$$

$$\frac{\Gamma \vdash t : \mathbb{A}}{\Gamma \setminus\!\setminus x \vdash \lambda x.t : S \Rightarrow ((\Gamma(x) \to \mathbb{A}) \times S)}\ \texttt{Abs} \qquad \frac{}{\emptyset \vdash \lambda x.t : S \Rightarrow (\star \times S)}\ \texttt{Ans}$$

$$\frac{\Gamma \vdash t : S \Rightarrow (M \to (Q \Rightarrow P) \times Q) \qquad \Delta \vdash u : M}{\Gamma + \Delta \vdash t\,u : S \Rightarrow P}\ \texttt{App} \qquad \frac{(\Gamma_i \vdash t : \mathbb{A}_i)_{i \in \mathtt{I}}}{+_{i \in \mathtt{I}} \Gamma_i \vdash t : \{\!\{ \mathbb{A}_i \}\!\}_{i \in \mathtt{I}}}\ \texttt{Many}$$

Along with the following six additional rules:

$$\frac{\Gamma; x : \mathtt{head}(S\,(l)) \vdash t : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P}{\Gamma \vdash \mathtt{pop}(l, \lambda x.t) : S \Rightarrow P}\ \texttt{Pop}$$

$$\frac{\Gamma \vdash u : M \qquad \Delta \vdash t : S\{l := M \cdot S\,(l)\} \Rightarrow P}{\Gamma + \Delta \vdash \mathtt{push}((l, u), t) : S \Rightarrow P}\ \texttt{Push}$$

$$\frac{\Gamma \vdash t : M \qquad \Delta \vdash f : L}{\Gamma + \Delta \vdash t \cdot f : M \cdot L}\ \texttt{Stack} \qquad \frac{}{\emptyset \vdash f : [\,]}\ \texttt{EStack}$$

$$\frac{(\Gamma_l \vdash f_l : L_l)_{l \in \mathcal{L}}}{+_{l \in \mathcal{L}} \Gamma_l \vdash \{l \mapsto f_l\}_{l \in \mathcal{L}} : \{l \mapsto L_l\}_{l \in \mathcal{L}}}\ \texttt{State} \qquad \frac{\Gamma \vdash t : S \Rightarrow P \qquad \Delta \vdash s : S}{\Gamma \vdash (t, s) : P}\ \texttt{Conf}$$

Typing Rules. Rules `Ax`, `Abs`, `Ans`, `App`, `Many`, `State`, and `Conf` are just like those in Definition 7.90 for the $\lambda^{\mathcal{G}}$-calculus. Rules `Pop`, `Push`, `Stack`, and `EStack`, are used to type operations and infinite stacks. Notice that, unlike rules `Get` and `Set`, rules `Pop` and `Push` manipulate the input state type S in exactly the same way as reduction rules `pop` and `push` in Definition 8.7 manipulate the input state s. This is due to the fact that, while the algebraic theory for global state is *non-linear*, the algebraic theory for infinite stacks is *linear*, *i.e.* operations `get` and `set` have non-linear behavior with respect to the state (mapping of locations to arguments), but operations `pop` and `push` behave linearly with respect to the state (mapping of locations to infinite stacks of arguments). Moreover, while rules `List` and `EStack` in Definition 8.23 are all used to type the *same* $\lambda^{\mathcal{S}}$-term, rules `Stack` and `EStack` are used to type infinite stacks consisting of (potentially) *different* $\lambda^{\mathcal{S}}$-terms. Indeed, according to rule `Stack`, an infinite stack $t \cdot f$ has type $M \cdot L$ if t has type M and f has type L. Now, L is also an infinite stack, so rule `Stack` can be used to type L, and this process can continue forever. Thus, in order to type an infinite stacks, we require an infinitely long stack type and consequently type derivation. However, we are only interested in *terminating* (*i.e.* finite) computations, and finite computation can only ever use a finite amount of information. Therefore, instead of typing infinite stacks completely, we are only going to use rule `Stack` to type an infinite stack up to a finite

point, and then type the rest of the (infinite) stack using rule `EStack`. Notice, however, that any occurrences of rule `EStack` in a type derivation can (potentially) be replaced by a use of rule `Stack`. Indeed, from a *qualitative* point of view, as long as enough elements of any particular infinite stack are typed, it is inconsequential to type more elements than necessary. However, from a *quantitative* point of view, this is no longer the case, *i.e.* we want to consider only minimal typings, *i.e.* the minimal information that can be extracted from a state.

Remark 8.24 (Minimal Typings for Infinite Stacks). The minimal (or *tight*) state types are those that assign the empty stack type to all stacks. Moreover, notice that the `push` operation does not *overwrite* the state type. This is contrary to what happens with the `set` operation (see Remark 7.51). Consequently, no *global* restrictions need to be imposed to type derivations.

Definition 8.25 (Tight Typings for CBN)**.** Let $t \in \Lambda_{\circ}^{\mathcal{S}}$ and $s \in \mathcal{S}_{\circ}$. A type derivation Φ is tight if and only if it is of the form $\Phi \rhd_{\mathcal{N}_S} \emptyset \vdash (t,s) : \star \times \{l \mapsto [\,]\}_{l \in \mathcal{L}}$.

We now start moving towards exact quantitative soundness and completeness by showing that multiset-typings can be split and merged.

Lemma 8.26 (CBN Multiset-Typings Split and Merge)**.** *Let $t \in \Lambda^{\mathcal{S}}$. Then, there is $\Phi \rhd_{\mathcal{N}_S} \Gamma \vdash t : M$, such that $M = \sqcup_{i \in \mathtt{I}} M_i$ if and only if there are $\Phi_i \rhd_{\mathcal{N}_S} \Gamma_i \vdash t : M_i$, such that $\Gamma = +_{i \in \mathtt{I}} \Gamma_i$, and $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = +_{i \in \mathtt{I}} \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_i)$.*

That typing contexts are necessarily empty whenever terms are closed.

Lemma 8.27 (Contexts and Free Variables for CBN)**.** *If $\Phi \rhd_{\mathcal{N}_S} \Gamma \vdash t : \mathbb{A}$ (resp. M), then $\mathtt{dom}(\Gamma) \subseteq \mathtt{fv}(t)$.*

And that having a tight type derivation with zero weight is a characteristic of normal forms is formalized as follows.

Lemma 8.28 (Zero Weight Tight Type Derivations Characterize `CBN`-Normal Forms)**.** *Let $t \in \Lambda_{\circ}^{\mathcal{S}}$ and $s \in \mathcal{S}_{\circ}$. If $\Phi \rhd_{\mathcal{N}_S} \emptyset \vdash (t,s) : P$ is tight, then $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = 0 \cdot \mathtt{App} + 0 \cdot \mathtt{Pop} + 0 \cdot \mathtt{Push}$ if and only if $t \in \mathtt{NO}_{\mathtt{CBN}}$.*

The following lemma ensures that all normal forms are tightly typable.

Lemma 8.29 (Tight Typability of `CBN`-Normal Forms)**.** *Let $s \in \mathcal{S}_{\circ}$. If $t \in \mathtt{NO}_{\mathtt{CBN}}$, then there is a derivation $\Phi \rhd_{\mathcal{N}_S} \emptyset \vdash (t,s) : P$ that is tight.*

The following lemma tells us how to update the list type assigned to a location.

Lemma 8.30. *Let $l \in \mathcal{L}$ and $s \in \mathcal{S}$. There are $\Phi \rhd_{\mathcal{N}_S} \emptyset \vdash s : S$ and $\Psi \rhd_{\mathcal{N}_S} \emptyset \vdash t : M$ if and only if there is $\Pi \rhd_{\mathcal{N}_S} \emptyset \vdash s\{l := t \cdot s\,(l)\} : S\{l := M \cdot S\,(l)\}$. Moreover, $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi)$.*

Now, we proceed by proving the quantitative soundness and completeness of typability in type system $\mathcal{N}_{\mathcal{S}}$.

8.5.1.1 Quantitative Soundness

We proceed by proving a weighted version of the usual substitution lemma.

Lemma 8.31 (Weighted Substitution for CBN). *Let $t, u \in \Lambda^{\mathcal{S}}$. If $\Phi \rhd_{\mathcal{N}_{\mathcal{S}}} \Gamma; x : M \vdash t : \mathbb{A}$ (resp. N) and $\Psi \rhd_{\mathcal{N}_{\mathcal{S}}} \emptyset \vdash u : M$, then $\Pi \rhd_{\mathcal{N}_{\mathcal{S}}} \Gamma \vdash t\{x \setminus u\} : \mathbb{A}$ (resp. N), such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi)$.*

The exact version of the subject reduction property follows from the above lemma, without any need to include any additional conditions to the premises of rule `Push`.

Lemma 8.32 (Exact Subject Reduction for CBN). *Let $t \in \Lambda^{\mathcal{S}}_{\circ}$, $s \in \mathcal{S}_{\circ}$, and $S \in \{\beta, \mathtt{pop}, \mathtt{push}\}$. If $(t,s) \Rightarrow_{\mathtt{CBN}(S)} (u,q)$ and $\Phi \rhd_{\mathcal{N}_{\mathcal{S}}} \emptyset \vdash (t,s) : P$, then there is $\Psi \rhd_{\mathcal{N}_{\mathcal{S}}} \emptyset \vdash (u,q) : P$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi) + 1 \cdot R$, where*

- *$R = \mathtt{App}$, if $S = \beta$;*
- *$R = \mathtt{Pop}$, if $S = \mathtt{pop}$;*
- *and $R = \mathtt{Push}$, if $S = \mathtt{push}$.*

Finally, we can show that $\mathcal{N}_{\mathcal{S}}$-typability is exactly quantitatively sound with respect to `CBN`-normalization.

Theorem 8.33 (Exact Soundness for CBN). *Let $t \in \Lambda^{\mathcal{S}}_{\circ}$ and $s \in \mathcal{S}_{\circ}$. If there is $\Phi \rhd_{\mathcal{N}_{\mathcal{S}}} \emptyset \vdash (t,s) : P$ tight, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = n \cdot \mathtt{App} + o \cdot \mathtt{Pop} + u \cdot \mathtt{Push}$, then $(t,s)^{n\cdot\beta + o\cdot\mathtt{pop} + u\cdot\mathtt{push}} \Rightarrow_{\mathtt{CBN}} (u,q)$, such that $u \in \mathtt{NO}_{\mathtt{CBN}}$.*

8.5.1.2 Quantitative Completeness

We proceed by proving a weighted version of the usual anti-substitution lemma.

Lemma 8.34 (Weighted Anti-Substitution for CBN). *Let $t \in \Lambda^{\mathcal{S}}$ and $u \in \Lambda^{\mathcal{S}}_{\circ}$. If $\Phi \rhd_{\mathcal{N}_{\mathcal{S}}} \Gamma \vdash t\{x \setminus u\} : \mathbb{A}$ (resp. N), then there exist $\Psi \rhd_{\mathcal{N}_{\mathcal{S}}} \Gamma; x : M \vdash t : \mathbb{A}$ (resp. N) and $\Pi \rhd_{\mathcal{N}_{\mathcal{S}}} \emptyset \vdash u : M$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi)$.*

The exact version of the subject expansion property follows from the above lemma, also without any need to include any additional conditions to the premises of rule `Push`.

Lemma 8.35 (Exact Subject Expansion for CBN). *Let* $t \in \Lambda^{\mathcal{S}}_{\circ}$, $s \in \mathcal{S}_{\circ}$, *and* $S \in \{\beta, \mathtt{pop}, \mathtt{push}\}$. *If* $(t,s) \Rightarrow_{\mathtt{CBN}(S)} (u,q)$ *and* $\Phi \rhd_{\mathcal{N}_{\mathcal{S}}} \emptyset \vdash (u,q) : P$, *then there is* $\Psi \rhd_{\mathcal{N}_{\mathcal{S}}} \emptyset \vdash (t,s) : P$, *such that* $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi) + 1 \cdot R = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi)$, *where*

- $R = \mathtt{App}$, *if* $S = \beta$;
- $R = \mathtt{Pop}$, *if* $S = \mathtt{pop}$;
- *and* $R = \mathtt{Push}$, *if* $S = \mathtt{push}$.

Finally, we can conclude that $\mathcal{N}_{\mathcal{S}}$-typability is quantitatively complete with respect to CBN-normalization.

Theorem 8.36 (Exact Completeness for CBN). *Let* $t \in \Lambda^{\mathcal{S}}_{\circ}$ *and* $s \in \mathcal{S}_{\circ}$. *If* $(t,s) \overset{n\cdot\beta + o\cdot\mathtt{pop} + u\cdot\mathtt{push}}{\Rightarrow\!\!\!\!\Rightarrow}_{\mathtt{CBN}} (u,q)$, *such that* $u \in \mathtt{NO}_{\mathtt{CBN}}$, *then there is* $\Phi \rhd_{\mathcal{N}_{\mathcal{S}}} \emptyset \vdash (t,s) : P$ *tight, such that* $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = n \cdot \mathtt{App} + o \cdot \mathtt{Pop} + u \cdot \mathtt{Push}$.

8.5.2 Call-By-Value

We now present the language of types for the CBV $\lambda^{\mathcal{S}}$-calculus.

Definition 8.37 (CBV Types). The sets of types $\mathcal{T}y^{\mathcal{S}}_{\mathtt{CBV}}$ for the CBV $\lambda^{\mathcal{S}}$-calculus are generated by the following grammar:

$$\begin{array}{lrcl}
\text{Stack Types:} & L, L' & ::= & [\,] \mid M \cdot L' \\
\text{State Types:} & S, Q & ::= & \{l \mapsto L_l\}_{l \in \mathcal{L}} \\
\text{Monadic Multi-Types:} & \mathbb{M} & ::= & S \Rightarrow P \\
\text{Product Types:} & P & ::= & M \times Q \\
\text{Multi-Types} & M, N & ::= & \{\!\{ A_i \}\!\}_{i \in \mathtt{I}} \\
\text{Types:} & A & ::= & M \to \mathbb{M}
\end{array}$$

Note that, while stacks are infinite, stack types are finitely generated.

The type system for the CBV $\lambda^{\mathcal{S}}$-calculus is as follows.

Definition 8.38 (CBV Type System). The *CBV Type System* $\mathcal{V}_{\mathcal{S}}$ assigns CBV (multi-monadic) types to $\lambda^{\mathcal{S}}$-terms, and consists of the following translations of the typing rules in Definition 4.8 for the CBV λ-calculus:

$$\frac{}{x : M \vdash x : M}\ \mathtt{Ax} \qquad \frac{(\Gamma_i \vdash t : \mathbb{M}_i)_{i \in \mathtt{I}}}{+_{i \in \mathtt{I}} \Gamma_i \setminus\!\setminus x \vdash \lambda x.t : \{\!\{ \Gamma_i(x) \to \mathbb{M}_i \}\!\}_{i \in \mathtt{I}}}\ \mathtt{Abs}$$

$$\frac{\Gamma \vdash t : S \Rightarrow (\{\!\{ N \to (R \Rightarrow P) \}\!\} \times Q) \qquad \Delta \vdash u : Q \Rightarrow (N \times R)}{\Gamma + \Delta \vdash t\,u : S \Rightarrow P}\ \mathtt{App}$$

Along with the following six additional rules:

$$\frac{\Gamma \vdash v : M}{\Gamma \vdash v : S \Rightarrow (M \times S)}\ \texttt{Lift}$$

$$\frac{\Gamma; x : \texttt{head}(S\,(l)) \vdash t : S\{l := \texttt{tail}(S\,(l))\} \Rightarrow P}{\Gamma \vdash \texttt{pop}(l, \lambda x.t) : S \Rightarrow P}\ \texttt{Pop}$$

$$\frac{\Gamma \vdash v : M \qquad \Delta \vdash t : S\{l := M \cdot S\,(l)\} \Rightarrow P}{\Gamma + \Delta \vdash \texttt{push}((l, v), t) : S \Rightarrow P}\ \texttt{Push}$$

$$\frac{\Gamma \vdash v : M \qquad \Delta \vdash f : L}{\Gamma + \Delta \vdash v \cdot f : M \cdot L}\ \texttt{Stack} \qquad \frac{}{\emptyset \vdash f : [\,]}\ \texttt{EStack}$$

$$\frac{(\Gamma_l \vdash f_l : L_l)_{l \in \mathcal{L}}}{+_{l \in \mathcal{L}} \Gamma_l \vdash \{l \mapsto f_l\}_{l \in \mathcal{L}} : \{l \mapsto L_l\}_{l \in \mathcal{L}}}\ \texttt{State} \qquad \frac{\Gamma \vdash t : S \Rightarrow P \qquad \Delta \vdash s : S}{\Gamma \vdash (t, s) : P}\ \texttt{Conf}$$

Typing Rules. Rules `Ax`, `Abs`, `App`, `Lift`, `State`, and `Conf` are just like those in Definition 7.70 for the $\lambda^{\mathcal{G}}$-calculus. Rules `Pop`, `Push`, `Stack`, and `EStack` are very similar to the ones in CBN (see Definition 8.23).

Definition 8.39 (Tight Typings for CBV)**.** Let $t \in \Lambda^{\mathcal{S}}_{\circ}$ and $s \in \mathcal{S}_{\circ}$. A type derivation Φ is tight if and only if it is of the form $\Phi \rhd_{\mathcal{V}_{\mathcal{S}}} \emptyset \vdash (t, s) : \{\!\{\,\}\!\} \times \{l \mapsto [\,]\}_{l \in \mathcal{L}}$.

We start by showing that multiset-typings can be split and merged, as expected.

Lemma 8.40 (CBV Multiset-Typings Split and Merge)**.** *Let $t \in \Lambda^{\mathcal{S}}$. Then, there is $\Phi \rhd_{\mathcal{V}_{\mathcal{S}}} \Gamma \vdash v : M$, such that $M = \sqcup_{i \in \mathtt{I}} M_i$, if and only if there are $\Phi_i \rhd_{\mathcal{V}_{\mathcal{S}}} \Gamma_i \vdash v : M_i$, such that $\Gamma = +_{i \in \mathtt{I}} \Gamma_i$, and $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = +_{i \in \mathtt{I}} \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_i)$.*

That typing contexts are necessarily empty whenever terms are closed.

Lemma 8.41 (Contexts and Free Variables for CBV)**.** *If $\Phi \rhd_{\mathcal{V}_{\mathcal{S}}} \Gamma \vdash t : \mathbb{M}$ (resp. M), then $\texttt{dom}(\Gamma) \subseteq \texttt{fv}(t)$.*

And that the type system reflects the fact that values are not effectful.

Lemma 8.42 (CBV Values Are Not Effectful)**.** *Let $t \in \Lambda^{\mathcal{S}}$ and $\Phi \rhd_{\mathcal{V}_{\mathcal{S}}} \Gamma \vdash t : S \Rightarrow (M \times Q)$. If $t \in \mathrm{V}^{\mathcal{S}}$, then $Q = S$ and Φ ends with rule* `Lift`.

The fact that having a tight type derivation with zero weight is a characteristic of normal forms is formalized as follows.

Lemma 8.43 (Zero Weight Tight Type Derivations Characterize CBV-Normal Forms)**.** *Let $t \in \Lambda^{\mathcal{S}}_{\circ}$ and $s \in \mathcal{S}_{\circ}$. If $\Phi \rhd_{\mathcal{V}_{\mathcal{S}}} \emptyset \vdash (t, s) : P$ is tight, then $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = 0 \cdot \mathtt{App} + 0 \cdot \mathtt{Pop} + 0 \cdot \mathtt{Push}$ if and only if $t \in \mathtt{NO}_{\mathtt{CBV}}$.*

The following lemma ensures that all normal forms are tightly typable.

Lemma 8.44 (Tight Typability of CBV-Normal Forms)**.** *Let $s \in \mathcal{S}_\circ$. If $t \in \mathtt{NO}_{\mathtt{CBV}}$, then there is a derivation $\Phi \rhd_{\mathcal{V}_\mathcal{S}} \emptyset \vdash (t,s) : P$ that is tight.*

The following lemma tells us how to update the list type assigned to a location.

Lemma 8.45. *Let $l \in \mathcal{L}$ and $s \in \mathcal{S}$. There are $\Phi \rhd_{\mathcal{V}_\mathcal{S}} \emptyset \vdash s : S$ and $\Psi \rhd_{\mathcal{V}_\mathcal{S}} \emptyset \vdash v : M$ if and only if there is $\Pi \rhd_{\mathcal{V}_\mathcal{S}} \emptyset \vdash s\{l := v \cdot s(l)\} : S\{l := M \cdot S(l)\}$. Moreover, $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi)$.*

Now, we proceed by proving the quantitative soundness and completeness of typability in type system $\mathcal{V}_\mathcal{S}$.

8.5.2.1 Quantitative Soundness

We proceed by proving a weighted version of the usual substitution lemma.

Lemma 8.46 (Weighted Substitution for CBV)**.** *Let $t \in \Lambda^{\mathcal{S}}_\circ$, and $v \in \mathrm{V}^{\mathcal{S}}$. If $\Phi \rhd_{\mathcal{V}_\mathcal{S}} \Gamma; x : M \vdash t : \mathbb{M}$ (resp. N) and $\Psi \rhd_{\mathcal{V}_\mathcal{S}} \emptyset \vdash v : M$, then $\Pi \rhd_{\mathcal{V}_\mathcal{S}} \Gamma \vdash t\{x \setminus v\} : \mathbb{M}$ (resp. N), such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi)$.*

The exact version of the subject reduction property follows from the above lemma.

Lemma 8.47 (Exact Subject Reduction for CBV)**.** *Let $t \in \Lambda^{\mathcal{S}}_\circ$, $s \in \mathcal{S}_\circ$, and $S \in \{\beta_{\mathtt{v}}, \mathtt{pop}, \mathtt{push}\}$. If $(t,s) \Rightarrow_{\mathtt{CBV}(S)} (u,q)$ and $\Phi \rhd_{\mathcal{V}_\mathcal{S}} \emptyset \vdash (t,s) : P$, then there is $\Psi \rhd_{\mathcal{V}_\mathcal{S}} \emptyset \vdash (u,q) : P$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi) + 1 \cdot R$, where*

- *$R = \mathtt{App}$, if $S = \beta_{\mathtt{v}}$;*
- *$R = \mathtt{Pop}$, if $S = \mathtt{pop}$;*
- *and $R = \mathtt{Push}$, if $S = \mathtt{push}$.*

Finally, we can show that $\mathcal{V}_\mathcal{S}$-typability is exactly quantitatively sound with respect to CBV-normalization.

Theorem 8.48 (Exact Soundness for CBV)**.** *Let $t \in \Lambda^{\mathcal{S}}_\circ$. If there is $\Phi \rhd_{\mathcal{V}_\mathcal{S}} \emptyset \vdash (t,s) : P$ tight, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = n \cdot \mathtt{App} + o \cdot \mathtt{Pop} + u \cdot \mathtt{Push}$, then $(t,s) \overset{n\cdot\beta_{\mathtt{v}} + o\cdot\mathtt{pop} + u\cdot\mathtt{push}}{\Rightarrow\!\!\!\!\!\!\Rightarrow}_{\mathtt{CBV}} (u,q)$, such that $u \in \mathtt{NO}_{\mathtt{CBV}}$.*

8.5.2.2 Quantitative Completeness

We proceed by proving a weighted version of the usual anti-substitution lemma.

Lemma 8.49 (Weighted Anti-Substitution for CBV). *Let $t \in \Lambda^{\mathcal{S}}$ and $v \in \mathrm{V}^{\mathcal{S}}_{\circ}$. If $\Phi \rhd_{\mathcal{V}_{\mathcal{S}}} \Gamma \vdash t\{x \setminus v\} : \mathbb{M}$ (resp. N), then $\Psi \rhd_{\mathcal{V}_{\mathcal{S}}} \Gamma; x : M \vdash t : \mathbb{M}$ (resp. N) and $\Pi \rhd_{\mathcal{V}_{\mathcal{S}}} \varnothing \vdash v : M$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi)$.*

The exact version of the subject expansion properties follows from the above lemma.

Lemma 8.50 (Exact Subject Expansion for CBV). *Let $t, u \in \Lambda^{\mathcal{S}}_{\circ}$, $s \in \mathcal{S}_{\circ}$, and $S \in \{\beta_{\mathtt{v}}, \mathtt{pop}, \mathtt{push}\}$. If $(t,s) \Rightarrow_{\mathtt{CBV}(S)} (u,q)$ and $\Phi \rhd_{\mathcal{V}_{\mathcal{S}}} \varnothing \vdash (u,q) : P$, then there is $\Psi \rhd_{\mathcal{V}_{\mathcal{S}}} \varnothing \vdash (t,s) : P$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) + 1 \cdot R = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi)$, where*

- *$R = \mathtt{App}$, if $S = \beta_{\mathtt{v}}$;*
- *$R = \mathtt{Pop}$, if $S = \mathtt{pop}$;*
- *and $R = \mathtt{Push}$, if $S = \mathtt{push}$.*

Finally, we can conclude that $\mathcal{V}_{\mathcal{S}}$-typability is quantitatively complete with respect to CBV-normalization.

Theorem 8.51 (Exact Completeness for CBV). *Let $t \in \Lambda^{\mathcal{S}}_{\circ}$. If $(t,s) \overset{n\cdot\beta_{\mathtt{v}} + o\cdot\mathtt{pop} + u\cdot\mathtt{push}}{\Rightarrow\!\!\!\!\!\Rightarrow_{\mathtt{CBV}}} (u,q)$, such that $u \in \mathtt{NO}_{\mathtt{CBV}}$, then there is $\Phi \rhd_{\mathcal{V}_{\mathcal{S}}} \varnothing \vdash (t,s) : P$ tight, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = n \cdot \mathtt{App} + o \cdot \mathtt{Pop} + u \cdot \mathtt{Push}$.*

8.5.3 Subsuming Call-By-Name and Call-By-Value

As expected, the CBPV system subsumes both the CBN and CBV quantitative systems introduced earlier. In particular, the linear behavior of the stack operations ensures that the typing rules for pop and push are uniform across all evaluation strategies, without requiring additional side conditions or restrictions.

The set of types for the $\lambda!^{\mathcal{S}}$-calculus is also generated by the same grammar as the one for the $\lambda!^{\mathcal{G}}$-calculus. *List types* are now reinterpreted as *stack types* following the same reasoning as for the CBN and CBV $\lambda^{\mathcal{S}}$-calculi (see Section 7.2.2).

Definition 8.52 (CBPV Types)**.** The sets of types $\mathcal{T}y^{\mathcal{S}}_{\text{CBPV}}$ for the $\lambda!^{\mathcal{S}}$-calculus is generated by the following grammar:

$$\begin{array}{lrcl}
\text{Stack Types:} & L, L' & ::= & [\,] \mid M \cdot L' \\
\text{State Types:} & S, Q & ::= & \{l \mapsto L_l\}_{l \in \mathcal{L}} \\
\text{Monadic Types:} & \mathbb{A} & ::= & S \Rightarrow P \\
\text{Product Types:} & P & ::= & A \times S \\
\text{Multi-Monadic Types} & M, N & ::= & \{\!\{\mathbb{A}_i\}\!\}_{i \in \mathtt{I}} \\
\text{Types:} & A & ::= & \star \mid M \mid M \to \mathbb{A}
\end{array}$$

The type system for the $\lambda!^{\mathcal{S}}$-calculus is as follows.

Definition 8.53 (CBPV Type System)**.** The *CBPV Type System* $\mathcal{P}_{\mathcal{S}}$ assigns CBPV (multi-monadic) types to $\lambda!^{\mathcal{S}}$-terms, and consists of the following translations of the typing rules in Definition 4.21 for the $\lambda!$-calculus:

$$\frac{}{x : M \vdash x : M}\ \mathtt{Ax} \qquad \frac{\Gamma \vdash t : \mathbb{A}}{\Gamma \setminus\!\setminus x \vdash \lambda x.t : S \Rightarrow ((\Gamma(x) \to \mathbb{A}) \times S)}\ \mathtt{Abs}$$

$$\frac{}{\emptyset \vdash \lambda x.t : S \Rightarrow (\star \times S)}\ \mathtt{Ans}$$

$$\frac{\Gamma \vdash t : S \Rightarrow ((M \to (R \Rightarrow P)) \times Q) \qquad \Delta \vdash u : Q \Rightarrow (M \times R)}{\Gamma + \Delta \vdash t\,u : S \Rightarrow P}\ \mathtt{App}$$

$$\frac{(\Gamma_i \vdash t : \mathbb{A}_i)_{i \in \mathtt{I}}}{+_{i \in \mathtt{I}} \Gamma_i \vdash !t : \{\!\{\mathbb{A}_i\}\!\}_{i \in \mathtt{I}}}\ \mathtt{Bg} \qquad \frac{\Gamma \vdash t : S \Rightarrow (\{\!\{Q \Rightarrow P\}\!\} \times Q)}{\Gamma \vdash \mathtt{der}(t) : S \Rightarrow P}\ \mathtt{Der}$$

Along with the following seven additional rules:

$$\frac{\Gamma \vdash v : M}{\Gamma \vdash v : S \Rightarrow (M \times S)}\ \mathtt{Lift}$$

$$\frac{\Gamma; x : \mathtt{head}(S\,(l)) \vdash t : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P}{\Gamma \vdash \mathtt{pop}(l, \lambda x.t) : S \Rightarrow P}\ \mathtt{Pop}$$

$$\frac{\Gamma \vdash v : M \qquad \Delta \vdash t : S\{l := M \cdot S\,(l)\} \Rightarrow P}{\Gamma + \Delta \vdash \mathtt{push}((l, v), t) : S \Rightarrow P}\ \mathtt{Push}$$

$$\frac{\Gamma \vdash v : M \qquad \Delta \vdash f : L}{\Gamma + \Delta \vdash v \cdot f : M \cdot L}\ \mathtt{Stack} \qquad \frac{}{\emptyset \vdash f : [\,]}\ \mathtt{EStack}$$

$$\frac{(\Gamma_l \vdash f_l : L_l)_{l \in \mathcal{L}}}{+_{l \in \mathcal{L}} \Gamma_l \vdash \{l \mapsto f_l\}_{l \in \mathcal{L}} : \{l \mapsto L_l\}_{l \in \mathcal{L}}}\ \mathtt{State} \qquad \frac{\Gamma \vdash t : S \Rightarrow P \qquad \Delta \vdash s : S}{\Gamma \vdash (t, s) : P}\ \mathtt{Conf}$$

Typing Rules. Rules `Ax`, `Abs`, `Ans`, `App`, `Bg`, `Der`, `Lift`, `State`, and `Conf` are just like those in Definition 7.90 for the $\lambda!^{\mathcal{G}}$-calculus. Rules `Pop`, `Push`, `Stack`, and `EStack`, are very similar to the ones in CBN and CBV (see Definitions 7.49 and 7.70).

Definition 8.54 (Tight Typings for CBPV)**.** Let $t \in !\Lambda^{\mathcal{S}}_{\circ}$ and $s \in \mathcal{S}_{\circ}$. A type derivation Φ is tight if and only if it is of the form $\Phi \rhd_{\mathcal{P}_{\mathcal{S}}} \emptyset \vdash (t,s) : A \times \{l \mapsto \{\!\{\,\}\!\}_{l \in \mathcal{L}}$, such that $A \in \{\star, \{\!\{\,\}\!\}\}$. That is, tight CBPV typings produce either a value type or a multiset of computations, together with an empty stack type.

We start by showing that multiset-typing can be split and merged, as expected.

Lemma 8.55 (CBPV Multiset-Typings Split and Merge)**.** *Let $t \in !\Lambda^{\mathcal{S}}$. Then, there is $\Phi \rhd_{\mathcal{P}_{\mathcal{S}}} \Gamma \vdash t : M$, such that $M = \sqcup_{i \in \mathtt{I}} M_i$, if and only if there are $\Phi_i \rhd_{\mathcal{P}_{\mathcal{S}}} \Gamma_i \vdash t : M_i$, such that $\Gamma = +_{i \in \mathtt{I}} \Gamma_i$, and $\#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Phi) = +_{i \in \mathtt{I}} \#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Phi_i)$.*

That typing contexts are necessarily empty whenever terms are closed.

Lemma 8.56 (Contexts and Free Variables for CBPV)**.** *If $\Phi \rhd_{\mathcal{P}_{\mathcal{S}}} \Gamma \vdash t : \mathbb{A}$ (resp. M), then $\mathtt{dom}(\Gamma) \subseteq \mathtt{fv}(t)$.*

And that the type system reflects the fact that values are not effectful.

Lemma 8.57 (CBPV Values Are Not Effectful)**.** *Let $t \in !\Lambda^{\mathcal{S}}$ and $\Phi \rhd_{\mathcal{P}_{\mathcal{S}}} \emptyset \vdash t : S \Rightarrow (A \times Q)$. If $t \in !\mathrm{V}^{\mathcal{S}}$, then $Q = S$ and Φ ends with rule* `Lift`*.*

The fact that having a tight type derivation with zero weight is a characteristic of clash-free normal forms is formalized as follows.

Lemma 8.58 (Zero Weight Tight Type Derivations Characterize `CBPV`-Normal Forms)**.** *Let $t \in \Lambda^{\mathcal{S}}_{\circ}$ and $s \in \mathcal{S}_{\circ}$. If $\Phi \rhd_{\mathcal{P}_{\mathcal{S}}} \emptyset \vdash (t,s) : P$ is tight, then $\#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Phi) = 0 \cdot \mathtt{App} + 0 \cdot \mathtt{Der} + 0 \cdot \mathtt{Pop} + 0 \cdot \mathtt{Push}$ if and only if $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$.*

The following lemma ensures that all clash-free normal-forms are tightly typable.

Lemma 8.59 (Tight Typability of `CBPV`-Normal Forms)**.** *Let $t \in \Lambda^{\mathcal{S}}_{\circ}$. If $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$, then there is a derivation $\Phi \rhd_{\mathcal{P}_{\mathcal{S}}} \emptyset \vdash (t,s) : P$ that is tight.*

The following lemma tells us how to update the list type assigned to a location.

Lemma 8.60. *Let $l \in \mathcal{L}$ and $s \in \mathcal{S}$. There are $\Phi \rhd_{\mathcal{P}_{\mathcal{S}}} \emptyset \vdash s : S$ and $\Psi \rhd_{\mathcal{P}_{\mathcal{S}}} \emptyset \vdash f : L$ if and only if there is $\Pi \rhd_{\mathcal{P}_{\mathcal{S}}} \emptyset \vdash s\{l := f\} : S\{l := L\}$. Moreover, $\#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Phi) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Psi) = \#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Pi)$.*

Now, we proceed by proving the quantitative soundness and completeness of typability in type system $\mathcal{P}_{\mathcal{S}}$.

8.5.3.1 Quantitative Soundness

We proceed by proving a weighted version of the usual substitution lemma.

Lemma 8.61 (Weighted Substitution for CBPV)**.** *Let $t \in !\Lambda^{\mathcal{S}}$ and $v \in !\mathrm{V}^{\mathcal{S}}$. If $\Phi \rhd_{\mathcal{P}_{\mathcal{S}}} \Gamma; x : M \vdash t : \mathbb{A}$ (resp. N) and $\Psi \rhd_{\mathcal{P}_{\mathcal{S}}} \varnothing \vdash v : M$, then $\Pi \rhd_{\mathcal{P}_{\mathcal{S}}} \Gamma \vdash t\{x \setminus v\} : \mathbb{A}$ (resp. N), such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Pi) = \#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Phi) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Psi)$.*

The quantitative version of the subject reduction property follow from the above lemma. Note, however, that it is not necessary to add $s(l) = \{\!\{\ \}\!\}$ as a requirement for rule `Push`.

Lemma 8.62 (Exact Subject Reduction for CBPV)**.** *Let $t \in !\Lambda^{\mathcal{S}}_{\circ}$, $s \in \mathcal{S}_{\circ}$, and $S \in \{\beta_{\mathrm{v}}, \beta_{!}, \mathtt{pop}, \mathtt{push}\}$. If $(t,s) \Rightarrow_{\mathtt{CBPV}(S)} (u,q)$ and $\Phi \rhd_{\mathcal{P}_{\mathcal{S}}} \varnothing \vdash (t,s) : P$, then there is $\Psi \rhd_{\mathcal{P}_{\mathcal{S}}} \varnothing \vdash (u,q) : P$, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Phi) = \#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Psi) + 1 \cdot R$, where*

- *$R = \mathtt{App}$, if $S = \beta_{\mathrm{v}}$;*
- *$R = \mathtt{Der}$, if $S = \beta_{!}$;*
- *$R = \mathtt{Pop}$, if $S = \mathtt{pop}$;*
- *and $R = \mathtt{Push}$, if $S = \mathtt{push}$.*

Finally, we can show that $\mathcal{P}_{\mathcal{S}}$-typability is quantitatively sound with respect to clash-free `CBPV`-normalization.

Theorem 8.63 (Exact Soundness for CBPV)**.** *Let $t \in \Lambda^{\mathcal{S}}_{\circ}$. If there is $\Phi \rhd_{\mathcal{P}_{\mathcal{S}}} \varnothing \vdash (t,s) : P$ tight, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Phi) = n \cdot \mathtt{App} + m \cdot \mathtt{Der} + o \cdot \mathtt{Pop} + u \cdot \mathtt{Push}$, then $(t,s) \overset{n\cdot\beta_{\mathrm{v}} + m\cdot\beta_{!} + o\cdot\mathtt{pop} + u\cdot\mathtt{push}}{\Rightarrow\!\!\!\!\!\rightarrow}_{\mathtt{CBPV}} (u,q)$, such that $u \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$.*

8.5.3.2 Quantitative Completeness

We start by showing that $\mathcal{P}_{\mathcal{S}}$-typable configurations are necessarily clash-free, *i.e.* normalize to a clash-free CBPV normal form.

Lemma 8.64 (Typability Implies Clash-Freeness)**.** *Let $t \in !\Lambda^{\mathcal{S}}_{\circ}$ and $s \in \mathcal{S}_{\circ}$. If $\Phi \rhd_{\mathcal{P}_{\mathcal{S}}} \varnothing \vdash (t,s) : P$, then t is clash-free.*

In fact, we can show that $\mathcal{P}_{\mathcal{S}}$-typability *captures* the clash-freeness of `CBPV`-normal forms.

Lemma 8.65 (Typability Characterizes CBPV Clash-Free Normal Forms)**.** *Let $t \in !\Lambda^{\mathcal{S}}_{\circ}$ and $s \in \mathcal{S}_{\circ}$. Then, $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$ if and only if $t \in \mathtt{NO}_{\mathtt{CBPV}}$ and (t,s) is $\mathcal{P}_{\mathcal{S}}$-typable.*

Lemma 8.66 (Weighted Anti-Substitution for CBPV). *Let* $t \in !\Lambda^{\mathcal{S}}$, *and* $v \in !\mathrm{V}^{\mathcal{S}}_{\circ}$. *If* $\Phi \triangleright_{\mathcal{P}_{\mathcal{S}}} \Gamma \vdash t\{x \setminus v\} : \mathbb{A}$ *(resp. N), then* $\Psi \triangleright_{\mathcal{P}_{\mathcal{S}}} \Gamma; x : M \vdash t : \mathbb{A}$ *(resp. N) and* $\Pi \triangleright_{\mathcal{P}_{\mathcal{S}}} \emptyset \vdash v : M$, *such that* $\#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Phi) = \#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Psi) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Pi)$.

The quantitative version of the subject expansion property follow from the above lemma. Again, it is not necessary to add $s\,(l) = \{\!\{\ \}\!\}$ as a requirement for rule `Push`.

Lemma 8.67 (Exact Subject Expansion for CBPV). *Let* $t \in !\Lambda^{\mathcal{S}}_{\circ}$, $s \in \mathcal{S}_{\circ}$, *and* $S \in \{\beta_{\mathtt{v}}, \beta_{!}, \mathtt{pop}, \mathtt{push}\}$. *If* $(t,s) \Rightarrow_{\mathtt{CBPV}(S)} (u,q)$ *and* $\Phi \triangleright_{\mathcal{P}_{\mathcal{S}}} \emptyset \vdash (u,q) : P$, *then there is* $\Psi \triangleright_{\mathcal{P}_{\mathcal{S}}} \emptyset \vdash (t,s) : P$, *such that* $\#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Phi) + 1 \cdot R = \#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Psi)$, *where*

- $R = \mathtt{App}$, *if* $S = \beta_{\mathtt{v}}$;
- $R = \mathtt{Der}$, *if* $S = \beta_{!}$;
- $R = \mathtt{Pop}$, *if* $S = \mathtt{pop}$;
- *and* $R = \mathtt{Push}$, *if* $S = \mathtt{push}$.

Finally, we can show that $\mathcal{P}_{\mathcal{S}}$-typability is quantitatively complete with respect to clash-free `CBPV`-normalization.

Theorem 8.68 (Exact Completeness for CBPV). *Let* $t \in \Lambda^{\mathcal{S}}_{\circ}$. *If* $(t,s) \overset{n\cdot\beta_{\mathtt{v}}+m\cdot\beta_{!}+o\cdot\mathtt{pop}+u\cdot\mathtt{push}}{\Rightarrow\!\!\!\!\!\!\Rightarrow}_{\mathtt{CBPV}} (u,q)$, *such that* $u \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$, *then there is* $\Phi \triangleright_{\mathcal{P}_{\mathcal{S}}} \emptyset \vdash (t,s) : P$ *tight, such that* $\#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Phi) = n \cdot \mathtt{App} + m \cdot \mathtt{Der} + o \cdot \mathtt{Pop} + u \cdot \mathtt{Push}$.

8.5.3.3 Soundness and Completeness of Translations

The translation of CBV typings for the $\lambda!^{\mathcal{S}}$-calculus is identical to that of the $\lambda!^{\mathcal{G}}$-calculus (see Definition 7.108).

Since we have seen that the translations on terms preserve the number of β-, $\beta_{\mathtt{v}}$-, `pop`-, and `push`-steps we can show that $\mathcal{P}$-typability is not only sound and complete with respect to the translations, but also preserves the number of steps. Moreover, even though stack translations are defined coinductively, $\mathcal{P}$-typability is a *finitary* process and thus the proofs of soundness and completeness of $\mathcal{P}$-typability with respect to the translations do not require any coinductive reasoning. They are both shown by induction of the structure of the type derivations, and the proofs follow the same reasoning as those of the $\lambda!^{\mathcal{G}}$-calculus (see Section 7.3.3.3).

Lemma 8.69 (Soundness of Term and Stack Translations). *Let* $t \in \Lambda^{\mathcal{S}}$.

1. *If* $\Phi \triangleright_{\mathcal{N}_{\mathcal{S}}} \Gamma \vdash t$ *(resp. t or f)* $: \mathbb{A}$ *(resp. M or L), then* $\Psi \triangleright_{\mathcal{P}_{\mathcal{S}}} \Gamma \vdash (t)^{\mathtt{CBN}}$ *(resp.* $!(t)^{\mathtt{CBN}}$ *or* $(f)^{\mathtt{CBN}}$*)* $:$ $\mathbb{A}$ *(resp. M or L). Moreover,* $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi)$.

2. *If* $\Phi \rhd_{\mathcal{V}_S} \Gamma \vdash t$ *(resp. t or f)* $: \mathbb{M}$ *(resp. M or L), then* $\Psi \rhd_{\mathcal{P}_S} (\Gamma)^{G_{CBV}} \vdash (t)^{CBV}$ *(resp.* $(t)^{CBV}$ *or* $(f)^{CBV}$*)* $: (\mathbb{M})^{G_{CBV}}$ *(resp.* $(M)^{G_{CBV}}$ *or* $(L)^{G_{CBV}}$*). Moreover,* $\#_{App,Pop,Push}(\Phi) = \#_{App,Pop,Push}(\Psi)$.

Again, while the completeness of the CBN translation can be easily shown, the completeness of the CBV translation, can not be shown by a simple induction on typing derivations. As such, we will also focus on showing completeness for the CBN translations, and leave the completeness of the CBV translations as a conjecture.

Lemma 8.70 (Completeness of CBN Term and Stack Translations)**.** *Let* $t \in \Lambda^{\mathcal{S}}$*. If* $\Psi \rhd_{\mathcal{P}_S} \Gamma \vdash (t)^{CBN}$ *(resp.* $!(t)^{CBN}$ *or* $(f)^{CBN}$*)* $: \mathbb{A}$ *(resp. M or L), then* $\Phi \rhd_{\mathcal{N}_S} \Gamma \vdash t$ *(resp. t or f)* $: \mathbb{A}$ *(resp. M or L). Moreover,* $\#_{App,Pop,Push}(\Phi) = \#_{App,Pop,Push}(\Psi)$.

Finally, we show that state translations are sound and complete.

Lemma 8.71 (Soundness and Completeness of CBN State Translations)**.** *Let* $s \in \mathcal{S}$*. Then,* $\Phi \rhd_{\mathcal{N}_S} \Gamma \vdash s : S$ *if and only if* $\Psi \rhd_{\mathcal{P}_S} \Gamma \vdash (s)^{CBN} : S$*. Moreover,* $\#_{App,Pop,Push}(\Phi) = \#_{App,Pop,Push}(\Psi)$.

And that CBV state translations are sound.

Lemma 8.72 (Soundness of CBV State Translations)**.** *Let* $s \in \mathcal{S}$*. If* $\Phi \rhd_{\mathcal{V}_S} \Gamma \vdash s : S$*, then* $\Psi \rhd_{\mathcal{P}_S} (\Gamma)^{CBV} \vdash (s)^{CBV} : (S)^{CBV}$*. Moreover,* $\#_{App,Pop,Push}(\Phi) = \#_{App,Pop,Push}(\Psi)$.

The soundness and completeness of the CBN translation of configurations then follows from the above statements.

Theorem 8.73 (Soundness and Completeness of CBN Configuration Translations)**.** *Let* $t \in \Lambda^{\mathcal{S}}$ *and* $s \in \mathcal{S}$*. Then,* $\Phi \rhd_{\mathcal{N}_S} \Gamma \vdash (t, s) : A \times Q$ *if and only if* $\Psi \rhd_{\mathcal{P}_S} \Gamma \vdash ((t, s))^{CBN} : A \times Q$*. Moreover,* $\#_{App,Pop,Push}(\Phi) = \#_{App,Pop,Push}(\Psi)$.

The soundness of the CBV translation of configurations follows from the soundness of the CBV translations of terms and states.

Theorem 8.74 (Soundness of CBV Configuration Translations)**.** *Let* $t \in \Lambda^{\mathcal{S}}$ *and* $s \in \mathcal{S}$*. If* $\Phi \rhd_{\mathcal{V}_S} \Gamma \vdash (t, s) : P$*, then* $\Psi \rhd_{\mathcal{P}_S} \Gamma \vdash ((t, s))^{CBV} : (P)^{G_{CBV}}$*. Moreover,* $\#_{App,Pop,Push}(\Phi) = \#_{App,Pop,Push}(\Psi)$.

The completeness of the CBV translation of configurations is left as a conjecture, since it can not be shown by a simple induction on typing derivations.

Conjecture 8.75 (Completeness of CBV Configuration Translations). Let $t \in \Lambda^{\mathcal{S}}$ and $s \in \mathcal{S}$. If $\Psi \rhd_{\mathcal{P}_S} (\Gamma)^{G_{CBV}} \vdash ((t, s))^{CBV} : (M)^{G_{CBV}} \times (Q)^{G_{CBV}}$, then $\Phi \rhd_{\mathcal{V}_S} \Gamma \vdash (t, s) : M \times Q$. Moreover, $\#_{App,Pop,Push}(\Phi) = \#_{App,Pop,Push}(\Psi)$.

This concludes the quantitative study of infinite stacks. Taken together, the CBN, CBV, and CBPV systems show that infinite stacks provide a uniform, linear, and quantitatively precise foundation for state-passing computations across evaluation strategies.

In the next section we show how infinite stacks can be used to encode other state-passing algebraic effects.

8.6 Handling State-Passing Computations using Infinite Stacks

The goal of this section is to show that infinite stacks form a sufficiently expressive target algebraic theory to encode a wide range of state-passing effects. The encodings proceed in three steps: (i) defining handlers from the source algebraic theory into infinite stacks, (ii) deriving the induced operational semantics, and (iii) deriving the corresponding quantitative CBPV type systems. We illustrate this methodology on global state, interactive input/output, finite nondeterminism, and their combinations.

8.6.1 Computing the Handlers

Let us start by computing handlers for each of the algebraic theories in Section 5.3. In the following, we are going to follows the same approach and reasoning as in Section 8.1. Moreover, all handlers defined in this section follow the same pattern: operations that observe the environment are interpreted as $\mathtt{pop}$ on a distinguished location, while operations that emit information are interpreted as $\mathtt{push}$. Moreover, all handlers are tail-resumptive one-shot handlers, *i.e.* *runners* (in the sense of [38]).

8.6.1.1 Global State

Recall the algebraic theory for global state in Example 5.19. We want to define a handler H from $|\mathtt{F}_{\mathcal{G}}X|$ to $|\mathtt{F}_{\mathcal{S}}Y|$. Let $\mathcal{L}_{\mathcal{G}}$ be the set of locations for $\mathcal{G}$, $\mathcal{L}_{\mathcal{S}}$ be the set of location for $\mathcal{S}$, $\mathcal{V}_{\mathcal{G}}$ be the set of values for $\mathcal{G}$, $\mathcal{V}_{\mathcal{S}}$ be the set of values for $\mathcal{S}$, $\phi_{\mathcal{L}} : \mathcal{L}_{\mathcal{G}} \to \mathcal{L}_{\mathcal{S}}$ be an injective function between locations, and $\phi_{\mathcal{V}} : \mathcal{V}_{\mathcal{G}} \to \mathcal{V}_{\mathcal{S}}$ be an injective function between values. Then, we can define the following maps:

$$\begin{array}{rcl}
h_{\mathtt{get}} : \mathcal{L}_{\mathcal{G}} \times (\mathcal{V}_{\mathcal{G}} \to |\mathtt{F}_{\mathcal{S}}Y|) & \to & |\mathtt{F}_{\mathcal{S}}Y| \\
(l, \lambda \mathtt{a}.H([\mathtt{t}])) & := & [\mathtt{pop}(\phi_{\mathcal{L}}\,(l), \lambda \mathtt{b}.[\mathtt{push}((\phi_{\mathcal{L}}\,(l), \mathtt{b}), H([\mathtt{t}\{\mathtt{a} \setminus \phi_{\mathcal{V}}^{-1}\,(\mathtt{b})\}]))])] \\
h_{\mathtt{set}} : (\mathcal{L}_{\mathcal{G}} \times \mathcal{V}_{\mathcal{G}}) \times (|\mathtt{F}_{\mathcal{S}}Y|) & \to & |\mathtt{F}_{\mathcal{S}}Y| \\
((l, v), H([\mathtt{t}])) & := & [\mathtt{pop}(\phi_{\mathcal{L}}\,(l), \lambda \mathtt{a}.[\mathtt{push}((\phi_{\mathcal{L}}\,(l), \phi_{\mathcal{V}}\,(v)), H([\mathtt{t}]))])].
\end{array}$$

To confirm that H is indeed a handler, we need to show that $h_{\mathtt{get}}$ and $h_{\mathtt{set}}$ form a $\mathcal{G}$-model. Recall the equations in $\mathrm{Eq}_{\mathcal{G}}$:

$$\begin{aligned}
\mathtt{get}(l, \lambda \mathtt{a}.\mathtt{t}) &\sim_{\mathcal{G}} \mathtt{t} \text{ if } \mathtt{a} \notin \mathtt{fv}(\mathtt{t}) \\
\mathtt{get}(l, \lambda \mathtt{a}.\mathtt{set}((l, \mathtt{a}), \lambda \mathtt{b}.\mathtt{t})) &\sim_{\mathcal{G}} \mathtt{t} \text{ if } \mathtt{a} \notin \mathtt{fv}(\mathtt{t}) \\
\mathtt{get}(l, \lambda \mathtt{b}.\mathtt{get}(l, \lambda \mathtt{a}.\mathtt{t})) &\sim_{\mathcal{G}} \mathtt{get}(l, \lambda \mathtt{a}.\mathtt{t}\{\mathtt{b} \setminus \mathtt{a}\}) \\
\mathtt{set}((l, v), \lambda \mathtt{a}.\mathtt{get}(l, \lambda \mathtt{b}.\mathtt{t})) &\sim_{\mathcal{G}} \mathtt{set}((l, v), \lambda \mathtt{a}.\mathtt{t}\{\mathtt{b} \setminus v\}) \\
\mathtt{set}((l, v), \lambda \mathtt{a}.\mathtt{set}((l, w), \lambda \mathtt{b}.\mathtt{t})) &\sim_{\mathcal{G}} \mathtt{set}((l, w), \lambda \mathtt{b}.\mathtt{t}) \\
\mathtt{get}(l, \lambda \mathtt{a}.\mathtt{get}(l', \lambda \mathtt{b}.\mathtt{t})) &\sim_{\mathcal{G}} \mathtt{get}(l', \lambda \mathtt{b}.\mathtt{get}(l, \lambda \mathtt{a}.\mathtt{t})) \\
\mathtt{get}(l, \lambda \mathtt{a}.\mathtt{set}((l', v), \lambda \mathtt{b}.\mathtt{t})) &\sim_{\mathcal{G}} \mathtt{set}((l', v), \lambda \mathtt{b}.\mathtt{get}(l, \lambda \mathtt{a}.\mathtt{t})) \\
\mathtt{set}((l, v), \lambda \mathtt{a}.\mathtt{set}((l', w), \lambda \mathtt{b}.\mathtt{t})) &\sim_{\mathcal{G}} \mathtt{set}((l', w), \lambda \mathtt{b}.\mathtt{set}((l, v), \lambda \mathtt{a}.\mathtt{t}))
\end{aligned}$$

After applying H to each side of the above equations, we get the following set of equalities:

$$\begin{aligned}
&1. && H([\mathtt{get}(l, \lambda \mathtt{a}.[\mathtt{t}])]) &&= H([\mathtt{t}]) \text{ if } \mathtt{a} \notin \mathtt{fv}(\mathtt{t}) \\
&2. && H([\mathtt{get}(l, \lambda \mathtt{a}.[\mathtt{set}((l, \mathtt{a}), \lambda \mathtt{b}.[\mathtt{t}])])]) &&= H([\mathtt{t}]) \text{ if } \mathtt{a} \notin \mathtt{fv}(\mathtt{t}) \\
&3. && H([\mathtt{get}(l, \lambda \mathtt{b}.[\mathtt{get}(l, \lambda \mathtt{a}.[\mathtt{t}])])]) &&= H([\mathtt{get}(l, \lambda \mathtt{a}.[\mathtt{t}\{\mathtt{b} \setminus \mathtt{a}\}])]) \\
&4. && H([\mathtt{set}((l, v), \lambda \mathtt{a}.[\mathtt{get}(l, \lambda \mathtt{b}.[\mathtt{t}])])]) &&= H([\mathtt{set}((l, v), \lambda \mathtt{a}.[\mathtt{t}\{\mathtt{b} \setminus v\}])]) \\
&5. && H([\mathtt{set}((l, v), \lambda \mathtt{a}.[\mathtt{set}((l, w), \lambda \mathtt{b}.[\mathtt{t}])])]) &&= H([\mathtt{set}((l, w), \lambda \mathtt{b}.[\mathtt{t}])]) \\
&6. && H([\mathtt{get}(l, \lambda \mathtt{a}.[\mathtt{get}(l', \lambda \mathtt{b}.[\mathtt{t}])])]) &&= H([\mathtt{get}(l', \lambda \mathtt{b}.[\mathtt{get}(l, \lambda \mathtt{a}.[\mathtt{t}])])]) \\
&7. && H([\mathtt{get}(l, \lambda \mathtt{a}.[\mathtt{set}((l', v), \lambda \mathtt{b}.[\mathtt{t}])])]) &&= H([\mathtt{set}((l', v), \lambda \mathtt{b}.[\mathtt{get}(l, \lambda \mathtt{a}.[\mathtt{t}])])]) \\
&8. && H([\mathtt{set}((l, v), \lambda \mathtt{a}.[\mathtt{set}((l', w), \lambda \mathtt{b}.[\mathtt{t}])])]) &&= H([\mathtt{set}((l', w), \lambda \mathtt{b}.[\mathtt{set}((l, v), \lambda \mathtt{a}.[\mathtt{t}])])])
\end{aligned}$$

Now, we must check that they hold in $\mathcal{S}$. We go through equations 1-5:

Equation 1:

Let $\mathtt{a} \notin \mathtt{fv}(\mathtt{t})$:

$$\begin{aligned}
& H([\mathtt{get}(l, \lambda \mathtt{a}.[\mathtt{t}])]) \\
=\ & [\mathtt{pop}(\phi_{\mathcal{L}}\,(l), \lambda \mathtt{b}.[\mathtt{push}((\phi_{\mathcal{L}}\,(l), \mathtt{b}), H([\mathtt{t}]))])] \\
=\ & H([\mathtt{t}]).
\end{aligned}$$

Equation 2:

Let $\mathtt{a} \notin \mathtt{fv}(\mathtt{t})$:

$$
\begin{aligned}
& H([\mathtt{get}(l, \lambda \mathtt{a}.[\mathtt{set}((l,\mathtt{a}), \lambda \mathtt{b}.[\mathtt{t}])])]) \\
= \;& [\mathtt{pop}(\phi_{\mathcal{L}}\,(l), \lambda \mathtt{c}.[\mathtt{push}((\phi_{\mathcal{L}}\,(l), \mathtt{c}), [\mathtt{pop}(\phi_{\mathcal{L}}\,(l), \lambda \mathtt{d}.[\mathtt{push}((\phi_{\mathcal{L}}\,(l), \phi_{\mathcal{V}}\,(\phi_{\mathcal{V}}^{-1}\,(\mathtt{c}))), H([\mathtt{t}]))])])])] \\
= \;& [\mathtt{pop}(\phi_{\mathcal{L}}\,(l), \lambda \mathtt{c}.[\mathtt{push}((\phi_{\mathcal{L}}\,(l), \mathtt{c}), [\mathtt{pop}(\phi_{\mathcal{L}}\,(l), \lambda \mathtt{d}.[\mathtt{push}((\phi_{\mathcal{L}}\,(l), \mathtt{c}), H([\mathtt{t}]))])])])] \\
= \;& [\mathtt{pop}(\phi_{\mathcal{L}}\,(l), \lambda \mathtt{c}.[\mathtt{push}((\phi_{\mathcal{L}}\,(l), \mathtt{c}), H([\mathtt{t}]))])] \\
= \;& H([\mathtt{t}]).
\end{aligned}
$$

Equation 3:

$$
\begin{aligned}
& H([\mathtt{get}(l, \lambda \mathtt{b}.[\mathtt{get}(l, \lambda \mathtt{a}.[\mathtt{t}])])]) \\
= \;& [\mathtt{pop}(\phi_{\mathcal{L}}\,(l), \lambda \mathtt{c}.[\mathtt{push}((\phi_{\mathcal{L}}\,(l), \mathtt{c}), [\mathtt{pop}(\Phi_{\mathcal{L}}\,(l), \lambda \mathtt{d}. \\
& \quad [\mathtt{push}((\phi_{\mathcal{L}}\,(l), \mathtt{d}), H([(\mathtt{t}\{\mathtt{b} \setminus \phi_{\mathcal{V}}^{-1}\,(\mathtt{c})\})\{\mathtt{a} \setminus \phi_{\mathcal{V}}^{-1}\,(\mathtt{d})\}])])])])] \\
= \;& [\mathtt{pop}(\phi_{\mathcal{L}}\,(l), \lambda \mathtt{c}.[\mathtt{push}((\phi_{\mathcal{L}}\,(l), \mathtt{c}), H([(\mathtt{t}\{\mathtt{b} \setminus \phi_{\mathcal{V}}^{-1}\,(\mathtt{c})\})\{\mathtt{a} \setminus \phi_{\mathcal{V}}^{-1}\,(\mathtt{c})\}])])] \\
= \;& [\mathtt{pop}(\phi_{\mathcal{L}}\,(l), \lambda \mathtt{c}.[\mathtt{push}((\phi_{\mathcal{L}}\,(l), \mathtt{c}), H([(\mathtt{t}\{\mathtt{b} \setminus \mathtt{a}\})\{\mathtt{a} \setminus \phi_{\mathcal{V}}^{-1}\,(\mathtt{c})\}])])] \\
= \;& H([\mathtt{get}(l, \lambda \mathtt{a}.\mathtt{t}\{\mathtt{b} \setminus \mathtt{a}\})]).
\end{aligned}
$$

Equation 4:

$$
\begin{aligned}
& H([\mathtt{set}((l, v), \lambda \mathtt{a}.[\mathtt{get}(l, \lambda \mathtt{b}.[\mathtt{t}])])]) \\
= \;& [\mathtt{pop}(\phi_{\mathcal{L}}\,(l), \lambda \mathtt{c}.[\mathtt{push}((\phi_{\mathcal{L}}\,(l), \phi_{\mathcal{V}}\,v), [\mathtt{pop}(\phi_{\mathcal{L}}\,(l), \lambda \mathtt{d}. \\
& \quad [\mathtt{push}((\phi_{\mathcal{L}}\,(l), \mathtt{d}), H([\mathtt{t}\{\mathtt{b} \setminus \phi_{\mathcal{V}}^{-1}\,(\mathtt{d})\}])])])])] \\
= \;& [\mathtt{pop}(\phi_{\mathcal{L}}\,(l), \lambda \mathtt{c}.[\mathtt{push}((\phi_{\mathcal{L}}\,(l), \phi_{\mathcal{V}}\,(v)), H([\mathtt{t}\{\mathtt{b} \setminus \phi_{\mathcal{V}}^{-1}\,(\phi_{\mathcal{V}}\,(v))\}])])] \\
= \;& [\mathtt{pop}(\phi_{\mathcal{L}}\,(l), \lambda \mathtt{c}.[\mathtt{push}((\phi_{\mathcal{L}}\,(l), \phi_{\mathcal{V}}\,(v)), H([\mathtt{t}\{\mathtt{b} \setminus v\}])])] \\
= \;& H([\mathtt{set}((l, v), \lambda \mathtt{a}.[\mathtt{t}\{\mathtt{b} \setminus v\}])]).
\end{aligned}
$$

Equation 5:

$$
\begin{aligned}
& H([\mathtt{set}((l, v), \lambda \mathtt{a}.[\mathtt{set}((l, w), \lambda \mathtt{b}.[\mathtt{t}])])]) \\
= \;& [\mathtt{pop}(\phi_{\mathcal{L}}\,(l), \lambda \mathtt{c}.[\mathtt{push}((\phi_{\mathcal{L}}\,(l), \phi_{\mathcal{V}}\,(v)), [\mathtt{pop}(\phi_{\mathcal{L}}\,(l), \lambda \mathtt{d}.[\mathtt{push}((\phi_{\mathcal{L}}\,(l), \phi_{\mathcal{V}}\,(w)), H([\mathtt{t}])])])])] \\
= \;& [\mathtt{pop}(\phi_{\mathcal{L}}\,(l), \lambda \mathtt{c}.[\mathtt{push}((\phi_{\mathcal{L}}\,(l), \phi_{\mathcal{V}}\,(w)), H([\mathtt{t}])])] \\
= \;& H([\mathtt{set}((l, w), \lambda \mathtt{b}.[\mathtt{t}])]).
\end{aligned}
$$

The remaining equations correspond to commutation of operations at distinct locations and follow by a similar, but more verbose, calculation.

8.6.1.2 Interactive Input

Recall the algebraic theory for interactive input in Example 5.16. We want to define a handler H from $|\mathtt{F}_{\mathcal{I}} X|$ to $|\mathtt{F}_{\mathcal{S}} Y|$. Let $\mathcal{L} \cup \{\mathtt{input}\}$ be the set of locations for $\mathcal{S}$, $\mathcal{V}_{\mathcal{I}}$ be the set of values for $\mathcal{I}$

(to be understood as the input alphabet), $\mathcal{V}_{\mathcal{S}}$ be the set of values for $\mathcal{S}$, and $\phi_{\mathcal{V}} : \mathcal{V}_{\mathcal{I}} \to \mathcal{V}_{\mathcal{S}}$ be an injective function between values. Then, we can define the following map:

$$\begin{array}{rcl} h_{\mathtt{input}} : (\mathcal{V}_{\mathcal{I}} \to |\mathtt{F}_{\mathcal{S}} Y|) & \to & |\mathtt{F}_{\mathcal{S}} Y| \\ (\lambda \mathtt{a}.H([\mathtt{t}])) & := & [\mathtt{pop}(\mathtt{input}, \lambda \mathtt{b}.H([\mathtt{t}\{\mathtt{a} \setminus \phi_{\mathcal{V}}^{-1}(\mathtt{b})\}]))]. \end{array}$$

To confirm that H is indeed a handler, we need to show that $h_{\mathtt{input}}$ forms a $\mathcal{I}$-model. Since the set of equations $\mathrm{Eq}_{\mathcal{I}}$ is empty, we can immediately conclude.

8.6.1.3 Interactive Output

Recall the algebraic theory for interactive output in Example 5.17. We want to define a handler H from $|\mathtt{F}_{\mathcal{O}} X|$ to $|\mathtt{F}_{\mathcal{S}} Y|$. Let $\mathcal{L} \cup \{\mathtt{output}\}$ be the set of locations for $\mathcal{S}$, $\mathcal{V}_{\mathcal{O}}$ be the set of values for $\mathcal{O}$ (to be understood as the output alphabet), $\mathcal{V}_{\mathcal{S}}$ be the set of values for $\mathcal{S}$, and $\phi_{\mathcal{V}} : \mathcal{V}_{\mathcal{O}} \to \mathcal{V}_{\mathcal{S}}$ be an injective function between values. Then, we can define the following map:

$$\begin{array}{rcl} h_{\mathtt{output}} : \mathcal{V}_{\mathcal{O}} \times |\mathtt{F}_{\mathcal{S}} Y| & \to & |\mathtt{F}_{\mathcal{S}} Y| \\ (v, H([\mathtt{t}])) & := & [\mathtt{push}((\mathtt{output}, \phi_{\mathcal{V}}(v)), H([\mathtt{t}]))]. \end{array}$$

To confirm that H is indeed a handler, we need to show that $h_{\mathtt{output}}$ forms a $\mathcal{O}$-model. Since the set of equations $\mathrm{Eq}_{\mathcal{O}}$ is empty, we can immediately conclude.

8.6.1.4 Nondeterminism

Recall the algebraic theory for finite non-empty nondeterminism in Example 5.18. We want to define a handler H from $|\mathtt{F}_{\mathcal{N}} X|$ to $|\mathtt{F}_{\mathcal{N}} X|$. Let $\mathcal{L} \cup \{\mathtt{choose}\}$ be the set of locations for $\mathcal{S}$, $\mathcal{V}_{\mathcal{N}}$ be the set of values for $\mathcal{N}$ (to be understood as the set of booleans), $\mathcal{V}_{\mathcal{S}}$ be the set of values for $\mathcal{S}$, and $\phi_{\mathcal{V}} : \mathcal{V}_{\mathcal{N}} \to \mathcal{V}_{\mathcal{S}}$ be an injective function between values. Then, we can define the following map:

$$\begin{array}{rcl} h_{\mathtt{choose}} : (\mathcal{V}_{\mathcal{N}} \to |\mathtt{F}_{\mathcal{S}} Y|) & \to & |\mathtt{F}_{\mathcal{S}} Y| \\ (\lambda \mathtt{a}.H(\mathtt{t})) & := & [\mathtt{pop}(\mathtt{choose}, \lambda \mathtt{b}.H(\mathtt{t}\{\mathtt{a} \setminus \mathtt{b}\}))]. \end{array}$$

To confirm that H is indeed a handler, we need to show that $h_{\mathtt{choose}}$ forms a $\mathcal{N}$-model. Since the set of equations $\mathrm{Eq}_{\mathcal{N}}$ is empty, we can immediately conclude.

Intuitively, nondeterministic choice is implemented by popping a Boolean from a distinguished stack, whose contents represent the branching structure of the computation.

8.6.1.5 Interactive IO

As we have seen in Proposition 5.40, it is possible to combine different algebraic theories together. As an example, the algebraic theory for interactive IO $\mathcal{IO}$ can be understood as the sum of $\mathcal{I}$

and $\mathcal{O}$. Then, $\Sigma_{IO} = \Sigma_I \cup \Sigma_O$, and $\mathrm{Eq}_{IO} = \mathrm{Eq}_I \cup \mathrm{Eq}_O$. We want to define a handler H from $|\mathtt{F}_{\mathcal{IO}} X| = |\mathtt{F}_{\mathbb{I}+\mathbb{O}} X|$ to $|\mathtt{F}_{\mathcal{S}} Y|$. Let $\mathcal{L} \cup \{\mathtt{input}, \mathtt{output}\}$ be the set of locations for $\mathcal{S}$, $\mathcal{V}_{\mathcal{IO}} = \mathcal{V}_{\mathcal{I}} \times \mathcal{V}_{\mathcal{O}}$ be the set of values for $\mathcal{IO}$ (to be understood as the cartesian product of the sets of values for $\mathcal{I}$ and $\mathcal{O}$), $\mathcal{V}_{\mathcal{S}}$ be the set of values for $\mathcal{S}$, and $\phi_{\mathcal{V}} : \mathcal{V}_{\mathcal{I}} \to \mathcal{V}_{\mathcal{S}}$ and $\psi_{\mathcal{V}} : \mathcal{V}_{\mathcal{O}} \to \mathcal{V}_{\mathcal{S}}$ be injective functions between values. Then, we can define the following maps:

$$\begin{array}{rcl} h_{\mathtt{input}} : (\mathcal{V}_{\mathcal{I}} \to |\mathtt{F}_{\mathcal{S}} Y|) & \to & |\mathtt{F}_{\mathcal{S}} Y| \\ (\lambda \mathtt{a}.H([\mathtt{t}])) & := & [\mathtt{pop}(\mathtt{input}, \lambda \mathtt{b}.H([\mathtt{t}\{\mathtt{a} \setminus \phi_{\mathcal{V}}^{-1}(\mathtt{b})\}]))] \\ h_{\mathtt{output}} : \mathcal{V}_{\mathcal{O}} \times |\mathtt{F}_{\mathcal{S}} Y| & \to & |\mathtt{F}_{\mathcal{S}} Y| \\ (v, H([\mathtt{t}])) & := & [\mathtt{push}((\mathtt{output}, \psi_{\mathcal{V}}(v)), H([\mathtt{t}]))]. \end{array}$$

To confirm that H is indeed a handler, we need to show that $h_{\mathtt{input}}$ and $h_{\mathtt{output}}$ form a $\mathcal{IO}$-model. Since the set of equations $\mathrm{Eq}_{\mathcal{IO}}$ is empty, we can conclude immediately.

8.6.2 Examples of Using the Handlers

Now we are going to illustrate how the handlers defined in the previous subsection can be used to extend the $\lambda!$-calculus with algebraic operations for dealing with global state and interactive IO. The same can be done for the CBN and CBV λ-calculi and for finite nondeterminism, but these are subsumed by the $\lambda!$-calculus and interactive IO, respectively. Let $\mathcal{T}$ be some algebraic theory, H be a handler from $|\mathtt{F}_{\mathcal{T}} X|$ to $|\mathtt{F}_{\mathcal{S}} Y|$, $\mathtt{op}_i : \mathcal{P}_i \rightsquigarrow \mathcal{A}_i \in \Sigma_{\mathcal{T}}$, and $p \in \mathcal{P}_i$.

$\lambda!$-Terms. Let $\mathcal{L}$ be a finite set of location names and $\mathcal{X}$ be a countable set of variables. The sets of values $!\mathrm{V}^{\mathcal{T}}$ and terms $!\Lambda^{\mathcal{T}}$ of the $\lambda!^{\mathcal{T}}$-calculus are generated by the two following template grammars, respectively:

$$\begin{array}{lrcl} \text{Values:} & v, w & ::= & x \in \mathcal{X} \mid !t \\ \text{Terms:} & t, u, r, e & ::= & v \mid \lambda x.t \mid t\, u \mid \mathtt{der}(t) \mid \mathtt{op}_i(p, \lambda x.t). \end{array}$$

The set of states $\mathcal{S}$ for the $\lambda!^{\mathcal{T}}$-calculus is the one for the $\lambda!^{\mathcal{S}}$-calculus, and the set of configurations is $!\Lambda^{\mathcal{T}} \times \mathcal{S}$.

CBPV Reduction. The *CBPV reduction strategy* $\Rightarrow_{\mathtt{CBPV}}$ is defined over closed configurations as the closure by CBPV contexts of the usual $\beta_{\mathtt{v}}$- and $\beta_{!}$-steps, plus the set of reduction steps that results from instantiating the following template reduction step:

$$(\mathtt{op}_i(p, \lambda x.t), s) \quad \mapsto_{\mathtt{op}_i} \quad (t\{x \setminus ?\}, ?)$$

CBPV Types. The set of types for the $\lambda!^{\mathcal{T}}$-calculus is the same as for the $\lambda!^{\mathcal{S}}$-calculus.

CBPV Type System. The *CBPV Type System* $\mathcal{P}_{\mathcal{S}}$ assigns CBPV (multi-monadic) types to $\lambda!$-terms, and consists of all the typing rules in $\mathcal{P}_{\mathcal{S}}$ with the exception of rules `Pop` and `Push`, which are replaced by the set of typing rules that result from instantiating the following template typing rule:

$$\frac{\Gamma \vdash p : M \qquad \Delta; x : ? \vdash t : ? \Rightarrow P}{\Gamma + \Delta \vdash \mathtt{op}_i(p, \lambda x.t) : S \Rightarrow P}\ \mathtt{Op}_i$$

Now, we are going to use the handlers defined in the previous section in order to instantiate the above templates according to the state-passing algebraic theory $\mathcal{T}$.

8.6.2.1 Handling Global State Using Infinite Stacks

Recall the set of operations of the algebraic theory for global state:

$$\Sigma_{\mathcal{G}} \quad := \quad \{\mathtt{get} : \mathcal{L} \rightsquigarrow \mathcal{V}, \mathtt{set} : \mathcal{L} \times \mathcal{V} \rightsquigarrow 1\}.$$

$\lambda!$-Terms. As expected, the sets of values $!\mathrm{V}^{\mathcal{G}}$ and terms $!\Lambda^{\mathcal{G}}$ of the $\lambda!^{\mathcal{G}}$-calculus are generated by instantiating the template grammars above as follows:

$$\begin{array}{llll} \text{Values:} & v, w & ::= & \cdots \\ \text{Terms:} & t, u, r, e & ::= & \cdots \mid \mathtt{get}(l, \lambda x.t) \mid \mathtt{set}((l, v), t). \end{array}$$

CBPV Reduction. The reduction steps for the `get` and `set` operations are obtained by instantiating the template reduction rules.

Let us start with the template reduction rule for the `get` operation:

$$(\mathtt{get}(l, \lambda x.t), s) \quad \mapsto_{\mathtt{get}} \quad (t\{x \setminus ?\}, ?).$$

Recall mapping $h_{\mathtt{get}}$ for the `get` operation:

$$\begin{array}{rcl} h_{\mathtt{get}} : \mathcal{L}_{\mathcal{G}} \times (\mathcal{V}_{\mathcal{G}} \to |\mathtt{F}_{\mathcal{S}} Y|) & \to & |\mathtt{F}_{\mathcal{S}} Y| \\ (l, \lambda \mathtt{a}.H([\mathtt{t}])) & := & [\mathtt{pop}(\phi_{\mathcal{L}}(l), \lambda \mathtt{b}.[\mathtt{push}((\phi_{\mathcal{L}}(l), \mathtt{b}), H([\mathtt{t}\{\mathtt{a} \setminus \phi_{\mathcal{V}}^{-1}(\mathtt{b})\}]))])] \end{array}$$

If we ignore the equivalence classes and understand $h_{\mathtt{get}}$ as a transformation from $\mathtt{Tree}_{\Sigma_{\mathcal{G}}} X$ to $\mathtt{Tree}_{\Sigma_{\mathcal{S}}} Y$, we can consider the cointerpretation of the operations appearing in tree resulting from the transformation. That is, given some location $l \in \mathcal{L}_{\mathcal{G}}$ and state $s \in (\mathcal{V}_{\mathcal{S}}^{\omega})^{\mathcal{L}_{\mathcal{S}}}$, running operation `get` in $\mathtt{F}_{\mathcal{G}} X$ is the same as running the following sequence of `pop` and `push` operations in $\mathtt{F}_{\mathcal{S}} Y$:

$$\begin{array}{l} [\![\mathtt{pop}]\!]^{(\mathcal{V}_{\mathcal{S}}^{\omega})^{\mathcal{L}_{\mathcal{S}}}}(\phi_{\mathcal{L}}(l), s) = (\mathtt{head}(s(\phi_{\mathcal{L}}(l))), s\{\phi_{\mathcal{L}}(l) := \mathtt{tail}(s(\phi_{\mathcal{L}}(l)))\}) \\ [\![\mathtt{push}]\!]^{(\mathcal{V}_{\mathcal{S}}^{\omega})^{\mathcal{L}_{\mathcal{S}}}}((\phi_{\mathcal{L}}(l), \mathtt{head}(s(\phi_{\mathcal{L}}(l)))), s\{\phi_{\mathcal{L}}(l) := \mathtt{tail}(s(\phi_{\mathcal{L}}(l)))\}) = (*, s). \end{array}$$

This means that $\phi_{\mathcal{V}}^{-1}(\mathtt{head}(s(\phi_{\mathcal{L}}(l))))$ is the value assigned to x and s is the starting state for $t\{x \setminus \phi_{\mathcal{V}}^{-1}(\mathtt{head}(s(\phi_{\mathcal{L}}(l))))\}$. Therefore, assuming that injective functions $\phi_{\mathcal{L}} : \mathcal{L}_{\mathcal{G}} \to \mathcal{L}_{\mathcal{S}}$ and $\phi : \mathcal{V}_{\mathcal{G}} \to \mathcal{V}_{\mathcal{S}}$ exist, the reduction rule for the `get` operation is as follows:

$$(\mathtt{get}(l, \lambda x.t), s) \quad \mapsto_{\mathtt{get}} \quad (t\{x \setminus \phi_{\mathcal{V}}^{-1}(\mathtt{head}(s(\phi_{\mathcal{L}}(l))))\}, s).$$

Now, let us consider the template reduction rule for the `set` operation:

$$(\mathtt{set}((l, v), t), s) \quad \mapsto_{\mathtt{set}} \quad (t, ?).$$

Recall mapping $h_{\mathtt{set}}$ for the `set` operation:

$$\begin{array}{rcl} h_{\mathtt{set}} : (\mathcal{L}_{\mathcal{G}} \times \mathcal{V}_{\mathcal{G}}) \times (|\mathtt{F}_{\mathcal{S}} Y|) & \to & |\mathtt{F}_{\mathcal{S}} Y| \\ ((l, v), H([\mathtt{t}])) & := & [\mathtt{pop}(\phi_{\mathcal{L}}(l), \lambda \mathtt{a}.[\mathtt{push}((\phi_{\mathcal{L}}(l), \phi_{\mathcal{V}}(v)), H([\mathtt{t}]))])]. \end{array}$$

Given some location $l \in \mathcal{L}_{\mathcal{G}}$, value $v \in \mathcal{V}_{\mathcal{G}}$, and state $s \in (\mathcal{V}_{\mathcal{S}}^{\omega})^{\mathcal{L}_{\mathcal{S}}}$, running operation `set` in $\mathtt{F}_{\mathcal{G}} X$ is the same as running the following sequences of `pop` and `push` operations in $\mathtt{F}_{\mathcal{S}} Y$:

$$\begin{array}{l} [\![\mathtt{pop}]\!]^{((\mathcal{V}_{\mathcal{S}})^{\omega})^{\mathcal{L}_{\mathcal{S}}}}(\phi_{\mathcal{L}}(l), s) = (\mathtt{head}(s(\phi_{\mathcal{L}}(l))), s\{\phi_{\mathcal{L}}(l) := \mathtt{tail}(s(\phi_{\mathcal{L}}(l)))\}) \\ [\![\mathtt{push}]\!]^{((\mathcal{V}_{\mathcal{S}})^{\omega})^{\mathcal{L}_{\mathcal{S}}}}((\phi_{\mathcal{L}}(l), \phi_{\mathcal{V}}(v)), s\{\phi_{\mathcal{L}}(l) := \mathtt{tail}(s(\phi_{\mathcal{L}}(l)))\}) = \\ \qquad (*, s\{\phi_{\mathcal{L}}(l) := \phi_{\mathcal{V}}(v) \cdot \mathtt{tail}(s(\phi_{\mathcal{L}}(l)))\}). \end{array}$$

This means that value $\mathtt{head}(s(\phi_{\mathcal{L}}(l)))$ is ignored and that $s\{\phi_{\mathcal{L}}(l) := \phi_{\mathcal{V}}(v) \cdot \mathtt{tail}(s(\phi_{\mathcal{L}}(l)))\}$ is the starting state for t. Therefore, the reduction rule for the `set` operation is as follows:

$$(\mathtt{set}((l, v), t), s) \quad \mapsto_{\mathtt{set}} \quad (t, s\{\phi_{\mathcal{L}}(l) := \phi_{\mathcal{V}}(v) \cdot \mathtt{tail}(s(\phi_{\mathcal{L}}(l)))\}).$$

Example 8.76 (Handling Global State Using Infinite Stacks, Operationally...). *Let $\mathcal{L}_{\mathcal{G}} = \mathcal{L}_{\mathcal{S}} = \{l\}$ consist of the same unique location. Then, $\phi_{\mathcal{L}} : \mathcal{L}_{\mathcal{G}} \to \mathcal{L}_{\mathcal{S}}$ is the identity on $\{l\}$. Now, consider the following translation:*

$$\begin{array}{rcl} (\cdot)^{\mathcal{G}} : !\Lambda^{\mathcal{G}} & \to & !\Lambda^{\mathcal{S}} \\ x & \mapsto & x \\ !t & \mapsto & !(t)^{\mathcal{G}} \\ \lambda x.t & \mapsto & \lambda x.(t)^{\mathcal{G}} \\ t\,u & \mapsto & (t)^{\mathcal{G}}\,(u)^{\mathcal{G}} \\ \mathtt{der}(t) & \mapsto & \mathtt{der}((t)^{\mathcal{G}}) \\ \mathtt{get}(l, \lambda x.t) & \mapsto & \mathtt{pop}(l, \lambda y.\mathtt{push}((l, y), (t\{x \setminus y\})^{\mathcal{G}})) \\ \mathtt{set}((l, v), t) & \mapsto & \mathtt{pop}(l, \lambda y.\mathtt{push}((l, (v)^{\mathcal{G}}), (t)^{\mathcal{G}})). \end{array}$$

Notice that $(\cdot)^{\mathcal{G}}$ *translates operations* `get` *and* `set` *according to the mappings* $h_{\texttt{get}}$ *and* $h_{\texttt{set}}$*. Moreover, it is easy to see that* $(\cdot)^{\mathcal{G}}$ *is injective. Therefore, we can take* $\phi_{\mathcal{V}} : \mathcal{V}_{\mathcal{G}} \to \mathcal{V}_{\mathcal{S}}$ *to be defined as follows:*

$$\begin{array}{rcl} \phi_{\mathcal{V}} : \mathcal{V}_{\mathcal{G}} & \to & \mathcal{V}_{\mathcal{S}} \\ x & \mapsto & x \\ !t & \mapsto & !(t)^{\mathcal{G}}. \end{array}$$

Also, notice that we can extend $\phi_{\mathcal{V}}$ *to states as follows:*

$$\begin{array}{rcl} \hat{\phi}_{\mathcal{V}} : (\mathcal{V}_{\mathcal{G}})^{\mathcal{L}_{\mathcal{G}}} & \to & (\mathcal{V}_{\mathcal{S}})^{\mathcal{L}_{\mathcal{S}}} \\ \{l \mapsto v\} & \mapsto & \{l \mapsto \phi_{\mathcal{V}}(v) \cdot \texttt{repeat}(!\lambda x.x)\}, \end{array}$$

where $\texttt{repeat} : \mathcal{V}_{\mathcal{S}} \to (\mathcal{V}_{\mathcal{S}})^{\omega}$ *is an infinitary function coinductively defined as follows:*

$$\begin{array}{rcl} \texttt{repeat} : \mathcal{V}_{\mathcal{S}} & \to & (\mathcal{V}_{\mathcal{S}})^{\omega} \\ v & \mapsto & v \cdot \texttt{repeat}(v). \end{array}$$

Now, consider the following configuration from Example 7.33:

$$(\texttt{set}((l, !\lambda x.x), \texttt{get}(l, \lambda y.\texttt{der}(y)\,(\texttt{der}(!(\lambda z.z))\,y))), (s)^{\texttt{CBV}}).$$

Let us assume that $(s)^{\texttt{CBV}} = \{l \mapsto v\}$*, so that* $\phi_{\mathcal{V}}(s) = \{l \mapsto \phi_V(v) \cdot \texttt{repeat}(!\lambda x.x)\}$*. Then*

$$\begin{array}{l} (\texttt{set}((l, !\lambda x.x), \texttt{get}(l, \lambda y.\texttt{der}(y)\,(\texttt{der}(!(\lambda z.z))\,y))), \{l \mapsto \phi_{\mathcal{V}}(v) \cdot \texttt{repeat}(!\lambda x.x)\}) \\ \Rightarrow_{\texttt{set}} (\texttt{get}(l, \lambda y.\texttt{der}(y)\,(\texttt{der}(!(\lambda z.z))\,y)), \{l \mapsto (\phi_{\mathcal{V}}(!\lambda x.x)) \cdot \texttt{repeat}(!\lambda x.x)\}) \\ \Rightarrow_{\texttt{get}} (\texttt{der}((!\lambda x.x))\,(\texttt{der}(!(\lambda z.z))\,(!\lambda x.x)), \{l \mapsto (\phi_{\mathcal{V}}(!\lambda x.x)) \cdot \texttt{repeat}(!\lambda x.x)\}) \\ \Rightarrow_{\beta_!} ((\lambda x.x)\,(\texttt{der}(!(\lambda z.z))\,(!\lambda x.x)), \{l \mapsto (\phi_{\mathcal{V}}(!\lambda x.x)) \cdot \texttt{repeat}(!\lambda x.x)\}) \\ \Rightarrow_{\beta_!} ((\lambda x.x)\,((\lambda z.z)\,(!\lambda x.x)), \{l \mapsto (\phi_{\mathcal{V}}(!\lambda x.x)) \cdot \texttt{repeat}(!\lambda x.x)\}) \\ \Rightarrow_{\beta_{\texttt{v}}} ((\lambda x.x)\,(!\lambda x.x), \{l \mapsto (\phi_{\mathcal{V}}(!\lambda x.x)) \cdot \texttt{repeat}(!\lambda x.x)\}) \\ \Rightarrow_{\beta_{\texttt{v}}} (!\lambda x.x, \{l \mapsto (\phi_{\mathcal{V}}(!\lambda x.x)) \cdot \texttt{repeat}(!\lambda x.x)\}). \end{array}$$

That is,

$$\begin{array}{c} (\texttt{set}((l, !\lambda x.x), \texttt{get}(l, \lambda y.\texttt{der}(y)\,(\texttt{der}(!(\lambda z.z))\,y))), \{l \mapsto \phi_{\mathcal{V}}(v) \cdot \texttt{repeat}(!\lambda x.x)\}) \\ \overset{1\cdot\texttt{set}+1\cdot\texttt{get}+2\cdot\beta_!+2\cdot\beta_{\texttt{v}}}{\Rightarrow_{\texttt{CBPV}}} \\ (!\lambda x.x, \{l \mapsto (\phi_{\mathcal{V}}(!\lambda x.x)) \cdot \texttt{repeat}(!\lambda x.x)\}). \end{array}$$

Finally, notice that $\hat{\phi}_{\mathcal{V}}^{-1}(\{l \mapsto (\phi_{\mathcal{V}}(!\lambda x.x)) \cdot \texttt{repeat}(!\lambda x.x)\}) = \{l \mapsto !\lambda x.x\}$*. Therefore, the above configuration corresponds to the following one:*

$$(!\lambda x.x, \{l \mapsto !\lambda x.x\}).$$

CBPV Type System. The *CBPV Type System* $\mathcal{P}_{\mathcal{G}}$ is obtained by instantiating the template type rules above.

Let us start with the template type rule for the `get` operation:

$$\frac{\Gamma; x : ? \vdash t : ? \Rightarrow P}{\Gamma \vdash \mathtt{get}(l, \lambda x.t) : S \Rightarrow P}\ \mathtt{Get}$$

To figure out the type of that should be assigned to variable x and the initial state type for t we must proceed similarly to how we did to determine the reduction rules for the `get` operation. However, there is an important caveat due to `get` operations not being linear in the sense of Remark 7.2. Indeed, given some location $l \in \mathcal{L}_{\mathcal{G}}$ and state type $S \in (M^*)^{\mathcal{L}_S}$, we have the following:

$$[\![\mathtt{pop}]\!]^{(M^*)^{\mathcal{L}_S}}(\phi_{\mathcal{L}}\,(l), S) = (\mathtt{head}(S\,(\phi_{\mathcal{L}}\,(l))), S\{\phi_{\mathcal{L}}\,(l) := \mathtt{tail}(S\,(\phi_{\mathcal{L}}\,(l)))\})$$
$$[\![\mathtt{push}]\!]^{(M^*)^{\mathcal{L}_S}}((\phi_{\mathcal{L}}\,(l), \mathtt{head}(S\,(\phi_{\mathcal{L}}\,(l)))), S\{\phi_{\mathcal{L}}\,(l) := \mathtt{tail}(S\,(\phi_{\mathcal{L}}\,(l)))\}) = (*, S).$$

Therefore, we would have the following typing rule for the `get` operation:

$$\frac{\Gamma; x : \mathtt{head}(S\,(\phi_{\mathcal{L}}\,(l))) \vdash t : S \Rightarrow P}{\Gamma \vdash \mathtt{get}(l, \lambda x.t) : S \Rightarrow P}\ \mathtt{Get}$$

But this rule is *wrong*: if t is assigned the same initial state type as $\mathtt{get}(l, \lambda x.t)$, the type system cannot be quantitatively sound. This is precisely the mismatch already identified in Remark 7.2: operationally, `get` is non-linear in the state, but quantitatively it must behave linearly with respect to resource usage. To solve this problem, we must first consider what happens with the template type rule for the `set` operation:

$$\frac{\Gamma \vdash v : M \qquad \Delta \vdash t : ? \Rightarrow P}{\Gamma + \Delta \vdash \mathtt{set}((l, v), t) : S \Rightarrow P}\ \mathtt{Set}$$

To figure out the initial state type for t we proceed exactly how we did to determine the reduction rule for the `set` operation. Given some location $l \in \mathcal{L}_{\mathcal{G}}$ and state type $S \in (M^*)^{\mathcal{L}_S} n$, we have the following:

$$[\![\mathtt{pop}]\!]^{(M^*)^{\mathcal{L}_S}}(\phi_{\mathcal{L}}\,(l), S) = (\mathtt{head}(S\,(\phi_{\mathcal{L}}\,(l))), S\{\phi_{\mathcal{L}}\,(l) := \mathtt{tail}(S\,(\phi_{\mathcal{L}}\,(l)))\})$$
$$[\![\mathtt{push}]\!]^{(M^*)^{\mathcal{L}_S}}((\phi_{\mathcal{L}}\,(l), M), S\{\phi_{\mathcal{L}}\,(l) := \mathtt{tail}(S\,(\phi_{\mathcal{L}}\,(l)))\}) =$$
$$(*, S\{\phi_{\mathcal{L}}\,(l) := M \cdot \mathtt{tail}(S\,(\phi_{\mathcal{L}}\,(l)))\}).$$

Therefore, we have the following typing rule for the `set` operation:

$$\frac{\Gamma \vdash v : M \qquad \Delta \vdash t : S\{\phi_{\mathcal{L}}\,(l) := M \cdot \mathtt{tail}(S\,(\phi_{\mathcal{L}}\,(l)))\} \Rightarrow P}{\Gamma + \Delta \vdash \mathtt{set}((l, v), t) : S \Rightarrow P}\ \mathtt{Set}$$

Now that we have determined the typing rule for the `set` operations, we can observe the following. Notice that, contrary to rule `Set` in Section 7.3.3, value v is typed with a *multiset type*, not a *list type*. However, as we have also discussed in the last paragraph of Remark 7.2, we can translate

any list type into an equivalent multiset type. Therefore, the $\mathtt{Get}$ that we have obtained for the $\mathtt{get}$ operation should take this into consideration. That is, it should be understood that $\mathtt{head}(S(\phi_{\mathcal{L}}(l)))$ is the multiset union of a list of multiset types, each corresponding to a different use of the value assigned to location l. Thus, instead of assigning $\mathtt{head}(S(\phi_{\mathcal{L}}(l)))$ to x, we must assign it only a multiset subset of $\mathtt{head}(S(\phi_{\mathcal{L}}(l)))$. Let $N \sqsubseteq \mathtt{head}(S(\phi_{\mathcal{L}}(l)))$. Then, we have the following *correct* typing rule for the $\mathtt{get}$ operation:

$$\frac{\Gamma; x : N \vdash t : S\{\phi_{\mathcal{L}}(l) := \mathtt{head}(S(\phi_{\mathcal{L}}(l))) \setminus N \cdot \mathtt{tail}(S(\phi_{\mathcal{L}}(l)))\} \Rightarrow P}{\Gamma \vdash \mathtt{get}(l, \lambda x.t) : S \Rightarrow P}\ \mathtt{Get}$$

And this comes from the following interpretation of the transformation of the $\mathtt{get}$ operation. Given some location $l \in \mathcal{L}_{\mathcal{G}}$ and state type $S \in (M^*)^{\mathcal{L}_S}$, we have the following:

$$\begin{aligned}
&[\![\mathtt{pop}]\!]^{(M^*)^{\mathcal{L}_S}}(\phi_{\mathcal{L}}(l), S) = (\mathtt{head}(S(\phi_{\mathcal{L}}(l))), S\{\phi_{\mathcal{L}}(l) := \mathtt{tail}(S(\phi_{\mathcal{L}}(l)))\}) \\
&\qquad\qquad N \sqsubseteq \mathtt{head}(S(\phi_{\mathcal{L}}(l))) \\
&[\![\mathtt{push}]\!]^{(M^*)^{\mathcal{L}_S}}((\phi_{\mathcal{L}}(l), \mathtt{head}(S(\phi_{\mathcal{L}}(l))) \setminus N), S\{\phi_{\mathcal{L}}(l) := \mathtt{tail}(S(\phi_{\mathcal{L}}(l)))\}) \\
&= (*, S\{\phi_{\mathcal{L}}(l) := \mathtt{head}(S(\phi_{\mathcal{L}}(l))) \setminus N \cdot \mathtt{tail}(S(\phi_{\mathcal{L}}(l)))\}).
\end{aligned}$$

As it turns out, if one ignores everything but the heads of the infinite stacks, one obtains a type system whose behavior is equivalent to that in [43].

Example 8.77 (Handling Global State Using Infinite Stacks, Denotationally...). *Let* $S = \{l \mapsto \{\!\{\,\}\!\} \cdot [\,]\}$ *and* $A_1 = \{\!\{\,\}\!\} \to (S \Rightarrow (\{\!\{\,\}\!\} \times S))$, *as in Example 7.78. Then, we can type the from Example 8.76 with the following* tight *type derivation* Ψ. *Since* Ψ *is quite big, we are going to split it into the following pieces.*

Let Ψ_s *be the following type derivation for the state:*

$$\dfrac{\dfrac{}{\emptyset \vdash \phi_{\mathcal{V}}(v) : \{\!\{\,\}\!\}}\ \mathtt{Ax/Bg} \qquad \dfrac{}{\emptyset \vdash \mathtt{repeat}(!\lambda x.x) : [\,]}\ \mathtt{EStack}}{\dfrac{\emptyset \vdash \phi_{\mathcal{V}}(v) \cdot \mathtt{repeat}(!\lambda x.x) : \{\!\{\,\}\!\} \cdot [\,]}{\emptyset \vdash \phi_{\mathcal{V}}(v) \cdot \mathtt{repeat}(!\lambda x.x) : S}\ \mathtt{State}}\ \mathtt{Stack}$$

where $\mathtt{Ax}$ *or* $\mathtt{Bg}$ *are used, depending on whether* $\phi_{\mathcal{V}}(v)$ *is a variable or a bang-term, respectively.*

The parameter of the $\mathtt{set}$ *operation is only typed once. Let* Ψ_p *be such a type derivation:*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\emptyset \vdash x : \{\!\{\,\}\!\}}\ \mathtt{Ax}}{\emptyset \vdash x : S \Rightarrow (\{\!\{\,\}\!\} \times S)}\ \mathtt{Lift}}{\emptyset \vdash \lambda x.x : S \Rightarrow (A_1 \times S)}\ \mathtt{Abs}}{\emptyset \vdash !\lambda x.x : \{\!\{S \Rightarrow (A_1 \times S)\}\!\}}\ \mathtt{Bg}$$

Let Ψ_l be the following type derivation for the operator appearing in the `get` *operation:*

$$\dfrac{\dfrac{\dfrac{}{y : \{\!\{S \Rightarrow (A_1 \times S)\}\!\} \vdash y : \{\!\{S \Rightarrow (A_1 \times S)\}\!\}}\,\mathtt{Ax}}{y : \{\!\{S \Rightarrow (A_1 \times S)\}\!\} \vdash y : S \Rightarrow (\{\!\{S \Rightarrow (A_1 \times S)\}\!\} \times S)}\,\mathtt{Lift}}{y : \{\!\{S \Rightarrow (A_1 \times S)\}\!\} \vdash \mathtt{der}(y) : S \Rightarrow (A_1 \times S)}\,\mathtt{Der}$$

Let Ψ_r be the following type derivation for the term at the right of the application corresponding to the continuation of `get` *operation:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\varnothing \vdash z : \{\!\{\,\}\!\}}\,\mathtt{Ax}}{\varnothing \vdash z : S \Rightarrow (\{\!\{\,\}\!\} \times S)}\,\mathtt{Lift}}{\varnothing \vdash \lambda z.z : S \Rightarrow (A_1 \times S)}\,\mathtt{Abs}}{\varnothing \vdash\, !\lambda z.z : \{\!\{S \Rightarrow (A_1 \times S)\}\!\}}\,\mathtt{Bg}}{\varnothing \vdash\, !\lambda z.z : S \Rightarrow (\{\!\{S \Rightarrow (A_1 \times S)\}\!\} \times S)}\,\mathtt{Lift} \qquad \dfrac{\dfrac{}{\varnothing \vdash y : \{\!\{\,\}\!\}}\,\mathtt{Ax}}{\varnothing \vdash y : S \Rightarrow (\{\!\{\,\}\!\} \times S)}\,\mathtt{Lift}$$
$$\dfrac{\varnothing \vdash \mathtt{der}(!\lambda z.z) : S \Rightarrow (A_1 \times S) \qquad \varnothing \vdash y : S \Rightarrow (\{\!\{\,\}\!\} \times S)}{\varnothing \vdash \mathtt{der}(!(\lambda z.z))\, y : S \Rightarrow (\{\!\{\,\}\!\} \times S)}\,\mathtt{Der}\ \mathtt{App}$$

Finally, let Ψ_c be the following type derivation for the continuation of the `get` *operation:*

$$\dfrac{\dfrac{\Psi_l \qquad \Psi_r}{y : \{\!\{S \Rightarrow (A_1 \times S)\}\!\} \vdash \mathtt{der}(y)\,(\mathtt{der}(!(\lambda z.z))\, y) : S \Rightarrow (\{\!\{\,\}\!\} \times S)}\,\mathtt{App}}{\varnothing \vdash \mathtt{get}(l, \lambda y.\mathtt{der}(y)\,(\mathtt{der}(!(\lambda z.z))\, y)) : \{l \mapsto \{\!\{S \Rightarrow (A_1 \times S)\}\!\} \cdot [\,]\} \Rightarrow (\{\!\{\,\}\!\} \times S)}\,\mathtt{Get}$$

where $\{\!\{A_1\}\!\} \sqsubseteq \{\!\{A_1\}\!\}$ and $\{\!\{A_1\}\!\} \setminus \{\!\{A_1\}\!\} = \{\!\{\,\}\!\}$.

Then, Ψ is as follows:

$$\dfrac{\dfrac{\dfrac{\Psi_p \qquad \dfrac{}{\varnothing \vdash\, !\lambda x.x : [\,]}\,\mathtt{EList}}{\varnothing \vdash\, !\lambda x.x : \{\!\{S \Rightarrow (A_1 \times S)\}\!\} \cdot [\,]}\,\mathtt{List} \qquad \Psi_c}{\varnothing \vdash \mathtt{set}((l, !\lambda x.x), \mathtt{get}(l, \lambda y.\mathtt{der}(y)\,(\mathtt{der}(!(\lambda z.z))\, y))) : S \Rightarrow (\{\!\{\,\}\!\} \times S)}\,\mathtt{Set} \qquad \Psi_s}{\varnothing \vdash (\mathtt{set}((l, !\lambda x.x), \mathtt{get}(l, \lambda \mathtt{der}(y).y\,(\mathtt{der}(!(\lambda z.z))\, y))), s) : \{\!\{\,\}\!\} \times S}\,\mathtt{Conf}$$

Notice that the structure of Ψ here is identical to that of Ψ in Example 7.110. Moreover, recall that

$$(\mathtt{set}((l, !\lambda x.x), \mathtt{get}(l, \lambda y.\mathtt{der}(y)\,(\mathtt{der}(!(\lambda z.z))\, y))), \{l \mapsto \phi_{\mathcal{V}}\,(v) \cdot \mathtt{repeat}(!\lambda x.x)\})$$
$$\overset{2\cdot\beta_{\mathtt{v}}+2\cdot\beta_{!}+1\cdot\mathtt{get}+1\cdot\mathtt{set}}{\Rightarrow_{\mathtt{CBPV}}}$$
$$(!\lambda x.x, \{l \mapsto (\phi_{\mathcal{V}}\,(!\lambda x.x)) \cdot \mathtt{repeat}(!\lambda x.x)\}).$$

And notice that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Set},\mathtt{Get}}(tydthree) = 2 \cdot \mathtt{App} + 2 \cdot \mathtt{Der} + 1 \cdot \mathtt{Get} + 1 \cdot \mathtt{Set}$.

8.6.2.2 Handling Interactive IO using Infinite Stacks

Recall the set of operations of the algebraic theory for interaction IO:

$$\Sigma_{IO} =: \{\mathtt{input} : 1 \rightsquigarrow \mathcal{A}, \mathtt{output} : \mathcal{A} \rightsquigarrow 1\}.$$

$\lambda!$-Terms. As expected, the sets of values $!\mathrm{V}^{\mathcal{IO}}$ and terms $!\Lambda^{\mathcal{IO}}$ of the $\lambda!^{\mathcal{IO}}$-calculus are generated by instantiating the template grammars above as follows:

$$\begin{array}{llll} \text{Values:} & v, w & ::= & \cdots \\ \text{Terms:} & t, u, r, e & ::= & \cdots \mid \mathtt{input}(\lambda x.t) \mid \mathtt{output}(v, t). \end{array}$$

CBPV Reduction. The reduction steps for the `input` and `output` operations are obtained by instantiating the template reduction rules.

Let us start with the template reduction rule for the `get` operation:

$$(\mathtt{input}(\lambda x.t), s) \quad \mapsto_{\mathtt{input}} \quad (t\{x \setminus ?\}, ?).$$

Recall mapping $h_{\mathtt{input}}$ for the `get` operation:

$$\begin{array}{rcl} h_{\mathtt{input}} : (\mathcal{V}_{\mathcal{I}} \to |\mathtt{F}_{\mathcal{S}} Y|) & \to & |\mathtt{F}_{\mathcal{S}} Y| \\ (\lambda \mathtt{a}.H([\mathtt{t}])) & := & [\mathtt{pop}(\mathtt{input}, \lambda \mathtt{b}.H([\mathtt{t}\{\mathtt{a} \setminus \phi_{\mathcal{V}}^{-1}(\mathtt{b})\}]))] \end{array}$$

Given some $s \in (\mathcal{V}_{\mathcal{S}}^{\omega})^{\mathcal{L}_{\mathcal{S}}}$, running operation `input` in $\mathtt{F}_{\mathcal{IO}} X$ is the same as running the following `pop` operation in $\mathtt{F}_{\mathcal{S}} Y$:

$$[\![\mathtt{pop}]\!]^{(\mathcal{V}_{\mathcal{S}}^{\omega})^{\mathcal{L}_{\mathcal{S}}}}(\mathtt{input}, s) = (\mathtt{head}(s\,(\mathtt{input})), s\{\mathtt{input} := \mathtt{tail}(s\,(\mathtt{input}))\}).$$

This means that $\phi_{\mathcal{V}}^{-1}(\mathtt{head}(s\,(\mathtt{input})))$ is the value assigned to x and $s\{\mathtt{input} := \mathtt{tail}(s\,(\mathtt{input}))\}$ is the starting state for $t\{x \setminus \phi_{\mathcal{V}}^{-1}(\mathtt{head}(s\,(\mathtt{input})))\}$. Therefore, assuming that injective function $\phi : \mathcal{V}_{\mathcal{G}} \to \mathcal{V}_{\mathcal{S}}$ exists, the reduction rule for the `input` operation is as follows:

$$(\mathtt{input}(\lambda x.t), s) \quad \mapsto_{\mathtt{input}} \quad (t\{x \setminus \phi_{\mathcal{V}}^{-1}(\mathtt{head}(s\,(\mathtt{input})))\}, s\{\mathtt{input} := \mathtt{tail}(s\,(\mathtt{input}))\}).$$

Now, let us consider the template reduction rule for the `output` operation:

$$(\mathtt{output}(v, t), s) \quad \mapsto_{\mathtt{output}} \quad (t, ?).$$

Recall mapping $h_{\mathtt{output}}$ for the `output` operation:

$$\begin{array}{rcl} h_{\mathtt{output}} : \mathcal{V}_{\mathcal{O}} \times |\mathtt{F}_{\mathcal{S}} Y| & \to & |\mathtt{F}_{\mathcal{S}} Y| \\ (v, H([\mathtt{t}])) & := & [\mathtt{push}((\mathtt{output}, \psi_{\mathcal{V}}(v)), H([\mathtt{t}]))]. \end{array}$$

Given some value $v \in \mathcal{V}_{\mathcal{IO}}$ and state $s \in (\mathcal{V}_{\mathcal{S}}^{\omega})^{\mathcal{L}_{\mathcal{S}}}$, running operation `output` in $\mathtt{F}_{\mathcal{IO}} X$ is the same as running the following `push` operation in $\mathtt{F}_{\mathcal{S}} Y$:

$$[\![\mathtt{push}]\!]^{((\mathcal{V}_{\mathcal{S}})^{\omega})^{\mathcal{L}_{\mathcal{S}}}}((\mathtt{output}, \phi_{\mathcal{V}}(v)), s) = (*, s\{\mathtt{output} := \phi_{\mathcal{V}}(v) \cdot s\}).$$

This means that $s\{\phi_{\mathcal{L}}\,(l) := \phi_{\mathcal{V}}\,(v) \cdot s\}$ is the starting state for t. Therefore, the reduction rule for the `set` operation is as follows:

$$(\mathtt{output}(v,t),s) \quad \Mapsto_{\mathtt{output}} \quad (t, s\{\mathtt{output} := \phi_{\mathcal{V}}\,(v) \cdot \mathtt{output}\}).$$

Example 8.78 (Handling Interactive IO Using Infinite Stacks, Operationally...)**.** *Consider* $\mathcal{L}_{\mathcal{S}} = \{\mathtt{input}, \mathtt{output}\}$. *Now, consider the following translation:*

$$\begin{array}{rcl}
(\cdot)^{\mathcal{IO}} : !\Lambda^{\mathcal{IO}} & \to & !\Lambda^{\mathcal{S}} \\
x & \mapsto & x \\
!t & \mapsto & !(t)^{\mathcal{IO}} \\
\lambda x.t & \mapsto & \lambda x.(t)^{\mathcal{IO}} \\
t\,u & \mapsto & (t)^{\mathcal{IO}}\,(u)^{\mathcal{IO}} \\
\mathtt{der}(t) & \mapsto & \mathtt{der}((t)^{\mathcal{IO}}) \\
\mathtt{input}(\lambda x.t) & \mapsto & \mathtt{pop}(\mathtt{input}, \lambda y.(t\{x \setminus y\})^{\mathcal{IO}}) \\
\mathtt{output}(v,t) & \mapsto & \mathtt{push}((\mathtt{output}, (v)^{\mathcal{IO}}), (t)^{\mathcal{IO}}).
\end{array}$$

Notice that $(\cdot)^{\mathcal{IO}}$ *translates operations* `input` *and* `output` *according to the mappings* $h_{\mathtt{input}}$ *and* $h_{\mathtt{output}}$. *Moreover, it is easy to see that* $(\cdot)^{\mathcal{G}}$ *is injective. Therefore, we can take* $\phi_{\mathcal{V}} : \mathcal{V}_{\mathcal{IO}} \to \mathcal{V}_{\mathcal{S}}$ *to be defined as follows:*

$$\begin{array}{rcl}
\phi_{\mathcal{V}} : \mathcal{V}_{\mathcal{IO}} & \to & \mathcal{V}_{\mathcal{S}} \\
x & \mapsto & x \\
!t & \mapsto & !(t)^{\mathcal{IO}}.
\end{array}$$

Also, notice that we can extend $\phi_{\mathcal{V}}$ *to input and output streams according to following coinductively defined infinitary function:*

$$\begin{array}{rcl}
\phi^{\omega}_{\mathcal{V}} : (\mathcal{V}_{\mathcal{IO}})^{\omega} & \to & (\mathcal{V}_{\mathcal{S}})^{\omega} \\
v \cdot vs & \mapsto & \phi_{\mathcal{V}}\,(v) \cdot \phi^{\omega}_{\mathcal{V}}\,(vs).
\end{array}$$

Then, we can extend $\phi_{\mathcal{V}}$ *to states (pairs of input streams and output lists) as follows:*

$$\begin{array}{rcl}
\hat{\phi}_V : (\mathcal{V}_{\mathcal{IO}})^{\omega} \times (\mathcal{V}_{\mathcal{IO}})^{*} & \to & (\mathcal{V}_{\mathcal{S}})^{\mathcal{L}_{\mathcal{S}}} \\
(ins, outs) & \mapsto & \{\mathtt{input} \mapsto \phi^{\omega}_{\mathcal{V}}\,(ins), \mathtt{output} \mapsto \phi^{\omega}_{\mathcal{V}}\,(outs)\}.
\end{array}$$

Now, consider the following interactive IO state:

$$(!\lambda x.x \cdot ins, outs)$$

And its translation:

$$\begin{array}{rcl}
\hat{\phi}_{\mathcal{V}}\,((!\lambda x.x \cdot ins, outs)) & = & \{\mathtt{input} \mapsto \phi_{\mathcal{V}}\,(!\lambda x.x) \cdot \phi^{\omega}_{\mathcal{V}}\,(ins), \mathtt{output} \mapsto \phi^{\omega}_{\mathcal{V}}\,(outs)\} \\
& = & \{\mathtt{input} \mapsto !\lambda x.x \cdot \phi^{\omega}_{\mathcal{V}}\,(ins), \mathtt{output} \mapsto \phi^{\omega}_{\mathcal{V}}\,(outs)\}.
\end{array}$$

Now, consider the following configuration:

$$(\mathtt{input}(\lambda x.\mathtt{output}(x, \mathtt{der}(x)\, x)), \{\mathtt{input} \mapsto\, !\lambda x.x \cdot \phi_{\mathcal{V}}^{\omega}\,(ins), \mathtt{output} \mapsto \phi_{\mathcal{V}}^{\omega}\,(outs)\}).$$

Then

$$\begin{aligned}
&(\mathtt{input}(\lambda x.\mathtt{output}(x, \mathtt{der}(x)\, x)), \{\mathtt{input} \mapsto\, !\lambda x.x \cdot \phi_{\mathcal{V}}^{\omega}\,(ins), \mathtt{output} \mapsto \phi_{\mathcal{V}}^{\omega}\,(outs)\}) \\
&\Rightarrow_{\mathtt{input}} (\mathtt{output}(!\lambda x.x, \mathtt{der}((!\lambda x.x))\, !\lambda x.x), \{\mathtt{input} \mapsto \phi_{\mathcal{V}}^{\omega}\,(ins), \mathtt{output} \mapsto \phi_{\mathcal{V}}^{\omega}\,(outs)\}) \\
&\Rightarrow_{\mathtt{output}} (\mathtt{der}((!\lambda x.x))\, !\lambda x.x, \{\mathtt{input} \mapsto \phi_{\mathcal{V}}^{\omega}\,(ins), \mathtt{output} \mapsto\, !\lambda x.x \cdot \phi_{\mathcal{V}}^{\omega}\,(outs)\}) \\
&\Rightarrow_{\beta_!} ((\lambda x.x)\, !\lambda x.x, \{\mathtt{input} \mapsto \phi_{\mathcal{V}}^{\omega}\,(ins), \mathtt{output} \mapsto\, !\lambda x.x \cdot \phi_{\mathcal{V}}^{\omega}\,(outs)\}) \\
&\Rightarrow_{\beta_{\mathsf{v}}} (!\lambda x.x, \{\mathtt{input} \mapsto \phi_{\mathcal{V}}^{\omega}\,(ins), \mathtt{output} \mapsto\, !\lambda x.x \cdot \phi_{\mathcal{V}}^{\omega}\,(outs)\}).
\end{aligned}$$

That is,

$$\begin{gathered}
(\mathtt{input}(\lambda x.\mathtt{output}(x, \mathtt{der}(x)\, x)), \{\mathtt{input} \mapsto\, !\lambda x.x \cdot \phi_{\mathcal{V}}^{\omega}\,(ins), \mathtt{output} \mapsto \phi_{\mathcal{V}}^{\omega}\,(outs)\}) \\
\overset{1\cdot\beta_{\mathsf{v}}+1\cdot\beta_!+1\cdot\mathtt{input}+1\cdot\mathtt{output}}{\Rightarrow_{\mathtt{CBPV}}} \\
(!\lambda x.x, \{\mathtt{input} \mapsto \phi_{\mathcal{V}}^{\omega}\,(ins), \mathtt{output} \mapsto\, !\lambda x.x \cdot \phi_{\mathcal{V}}^{\omega}\,(outs)\}).
\end{gathered}$$

CBPV Type System. The *CBPV Type System* $\mathcal{P}_{\mathcal{IO}}$ is obtained by instantiating the template type rules above.

Let us start with the template type rule for the `input` operation:

$$\frac{\Gamma; x : ? \vdash t : ? \Rightarrow P}{\Gamma \vdash \mathtt{input}(\lambda x.t) : S \Rightarrow P}\ \mathtt{Input}$$

To figure out the type of that should be assigned to variable x and the initial state type for t we must proceed similarly to how we did to determine the reduction rules for the `input` operation. Given some location state type $S \in (M^*)^{\mathcal{L}_S}$, we have the following:

$$[\![\mathtt{input}]\!]^{(M^*)^{\mathcal{L}_S}}(\mathtt{input}, S) = (\mathtt{head}(S\,(\mathtt{input})), S\{\mathtt{input} := \mathtt{tail}(S\,(\mathtt{input}))\}).$$

Therefore, we would have the following typing rule for the `get` operation:

$$\frac{\Gamma; x : \mathtt{head}(S\,(\mathtt{input})) \vdash t : S\{\mathtt{input} := \mathtt{tail}(S\,(\mathtt{input}))\} \Rightarrow P}{\Gamma \vdash \mathtt{input}(\lambda x.t) : S \Rightarrow P}\ \mathtt{Input}$$

Now, we consider what happens with the template type rule for the `output` operation:

$$\frac{\Gamma \vdash v : M \qquad \Delta \vdash t : ? \Rightarrow P}{\Gamma + \Delta \vdash \mathtt{output}(v, t) : S \Rightarrow P}\ \mathtt{Output}$$

To figure out the initial state type for t we proceed exactly how we did to determine the reduction rule for the `output` operation. Given some state type $S \in (M^*)^{\mathcal{L}_S}$, we have the following:

$$[\![\mathtt{output}]\!]^{(M^*)^{\mathcal{L}_S}}((\mathtt{output}, M), S) = (*, S\{\mathtt{output} := M \cdot S\,(\mathtt{output})\}).$$

Therefore, we have the following typing rule for the output operation:

$$\frac{\Gamma \vdash v : M \qquad \Delta \vdash t : S\{\mathtt{output} := M \cdot S\,(\mathtt{output})\} \Rightarrow P}{\Gamma + \Delta \vdash \mathtt{output}(v,t) : S \Rightarrow P}\ \mathtt{Output}$$

Example 8.79 (Handling Interactive IO Using Infinite Stacks, Denotationally...). *Consider the following set of state types:*

$$\begin{aligned} S &= \{\mathtt{input} \mapsto \{\!\{\mathbb{A}\}\!\} \cdot [\,], \mathtt{output} \mapsto [\,]\} \\ Q &= S\{\mathtt{input} := \mathtt{tail}(S\,(\mathtt{input}))\} = \{\mathtt{input} \mapsto [\,], \mathtt{output} \mapsto [\,]\} \\ R &= Q\{\mathtt{output} := \{\!\{\,\}\!\} \cdot Q\,(\mathtt{output})\} = \{\mathtt{input} \mapsto [\,], \mathtt{output} \mapsto \{\!\{\,\}\!\} \cdot [\,]\}, \end{aligned}$$

where $\mathbb{A} = R \Rightarrow ((\{\!\{\,\}\!\} \to (R \Rightarrow (\{\!\{\,\}\!\} \times R))) \times R)$.

Then, we are able to type the configuration from Example 8.78 with the following tight *type derivation* Ψ. *Since* Ψ *is quite big, we are going to split it into the following pieces.*

Let Ψ_s *be the following type derivation for the state:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x : \{\!\{\,\}\!\} \vdash x : \{\!\{\,\}\!\}}\,\mathtt{Ax}}{x : \{\!\{\,\}\!\} \vdash x : R \Rightarrow (\{\!\{\,\}\!\} \times R)}\,\mathtt{Lift}}{\emptyset \vdash \lambda x.x : \mathbb{A}}\,\mathtt{Abs}}{\emptyset \vdash\, !\lambda x.x : \{\!\{\mathbb{A}\}\!\}}\,\mathtt{Bg} \qquad \dfrac{}{\emptyset \vdash \phi^{\omega}_{\mathcal{V}}\,(ins) : [\,]}\,\mathtt{EStack}}{\emptyset \vdash\, !\lambda x.x \cdot \phi^{\omega}_{\mathcal{V}}\,(ins) : \{\!\{\mathbb{A}\}\!\} \cdot [\,]}\,\mathtt{List} \qquad \dfrac{}{\emptyset \vdash \phi^{\omega}_{\mathcal{V}}\,(outs) : [\,]}\,\mathtt{EStack}}{\emptyset \vdash \{\mathtt{input} \mapsto\, !\lambda x.x \cdot \phi^{\omega}_{\mathcal{V}}\,(ins), \mathtt{output} \mapsto \phi^{\omega}_{\mathcal{V}}\,(outs)\} : S}\,\mathtt{State}$$

Now, notice that $\mathtt{head}(S\,(\mathtt{input})) = \{\!\{\mathbb{A}\}\!\}$. *Then,* Ψ *is as follows:*

$$\dfrac{\dfrac{}{x : \{\!\{\,\}\!\} \vdash x : \{\!\{\,\}\!\}}\,\mathtt{Ax} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{x : \{\!\{\mathbb{A}\}\!\} \vdash x : \{\!\{\mathbb{A}\}\!\}}\,\mathtt{Ax}}{x : \{\!\{\mathbb{A}\}\!\} \vdash x : R \Rightarrow (\{\!\{\mathbb{A}\}\!\} \times R)}\,\mathtt{Lift}}{x : \{\!\{\mathbb{A}\}\!\} \vdash \mathtt{der}(x) : \mathbb{A}}\,\mathtt{Der} \qquad \dfrac{\dfrac{}{x : \{\!\{\,\}\!\} \vdash x : \{\!\{\,\}\!\}}\,\mathtt{Ax}}{x : \{\!\{\,\}\!\} \vdash x : R \Rightarrow (\{\!\{\,\}\!\} \times R)}\,\mathtt{Lift}}{x : \{\!\{\mathbb{A}\}\!\} \vdash \mathtt{der}(x)\,x : R \Rightarrow (\{\!\{\,\}\!\} \times R)}\,\mathtt{App}}{x : \mathtt{head}(S\,(\mathtt{input})) \vdash \mathtt{output}(x, \mathtt{der}(x)\,x) : Q \Rightarrow (\{\!\{\,\}\!\} \times R)}\,\mathtt{Output}}{\emptyset \vdash \mathtt{input}(\lambda x.\mathtt{output}(x, \mathtt{der}(x)\,x)) : S \Rightarrow (\{\!\{\,\}\!\} \times R)}\,\mathtt{Input} \qquad \Phi_s}{\emptyset \vdash (\mathtt{input}(\lambda x.\mathtt{output}(x, \mathtt{der}(x)\,x)), \{\mathtt{input} \mapsto\, !\lambda x.x \cdot \phi^{\omega}_{\mathcal{V}}\,(ins), \mathtt{output} \mapsto \phi^{\omega}_{\mathcal{V}}\,(outs)\}) : \{\!\{\,\}\!\} \times R}\,\mathtt{Conf}$$

Finally, recall that

$$\begin{gathered} (\mathtt{input}(\lambda x.\mathtt{output}(x, \mathtt{der}(x)\,x)), \{\mathtt{input} \mapsto\, !\lambda x.x \cdot \phi^{\omega}_{\mathcal{V}}\,(ins), \mathtt{output} \mapsto \phi^{\omega}_{\mathcal{V}}\,(outs)\}) \\ \underset{\mathtt{CBPV}}{1 \cdot \beta_{\mathtt{v}} + 1 \cdot \beta_{!} + 1 \cdot \mathtt{input} + 1 \cdot \mathtt{output}} \\ (!\lambda x.x, \{\mathtt{input} \mapsto \phi^{\omega}_{\mathcal{V}}\,(ins), \mathtt{output} \mapsto\, !\lambda x.x \cdot \phi^{\omega}_{\mathcal{V}}\,(outs)\}), \end{gathered}$$

and notice that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Input},\mathtt{Output}}(\Psi) = 1 \cdot \mathtt{App} + 1 \cdot \mathtt{Der} + 1 \cdot \mathtt{Input} + 1 \cdot \mathtt{Output}$.

These examples show that infinite stacks provide a uniform semantic substrate for a wide class of state-passing effects, where both operational behavior and quantitative typing information are preserved by construction.

We conclude this chapter by observing that many further algebraic effects can be captured by the same methodology, including effects not considered here (such as concurrency), as well as arbitrary combinations of the effects studied in this thesis. The only essential requirement is that the target algebraic theory induces a notion of state rich enough to subsume those of the source theories.

Part V

Epilogue

Chapter 9

Conclusion & Future Work

9.1 Conclusion

Nowadays, programming languages are mostly used in environments where knowing *how* a computation behaves is just as important as knowing *what* it computes. Features such as higher-order functions, concurrency, interaction, and effects make purely qualitative reasoning—concerned only with termination, safety, or equivalence—insufficient to capture the needs of modern computation. Today, to fully understand programs we need to reason intrinsically about quantities: how many times a resource is used, how many computation steps are performed, how often effects are propagated, and how all these costs accumulate. In this setting, quantitative semantics is no longer an optional refinement, but a foundational component of programming language theory.

This thesis addresses this shift by developing a quantitative semantic framework for higher-order computations with effects, addressing a central limitation of traditional, purely qualitative accounts. By enriching models of effectful higher-order languages with quantitative information, it becomes possible to reason systematically about both the structure and the measurable behavior of programs that combine higher-order functions with a wide range of algebraic effects.

This work has shown how to construct semantic models that integrate higher-order function spaces, algebraic operations, and handlers with quantitative structure in a principled way. These models support standard operations of higher-order semantics—such as abstraction, application, and substitution—while tracking quantities like weights, costs, or probabilities. They also provide a basis for defining and studying program equivalences and preorders that are sensitive to quantitative behavior, connecting semantic notions of approximation with operational intuitions about resource usage or effectful behavior.

Several case studies demonstrate that the framework is sufficiently flexible to capture diverse classes of algebraic effects and their combinations. These examples illustrate how the proposed

semantics recovers familiar well-known qualitative properties, while at the same time enabling new forms of quantitative reasoning that are difficult to express in existing approaches. Moreover, the development reveals systematic relationships between algebraic effects, monadic and graded semantics, and quantitative models drawn from areas such as weighted or probabilistic semantics, clarifying how these strands of research fit into a single, coherent picture.

Concretely, the thesis establishes that non-idempotent intersection types provide exact cost semantics—not merely upper bounds—for a range of computational features that go well beyond the pure λ-calculus. These include:

- pattern-matching mechanisms and stuck computations, enabling a quantitative analysis of control flow and failure;
- exception raising and handling, analyzed via their encoding as pattern-matching constructs, despite their non-algebraic nature;
- global state and its generalization to infinite stacks, yielding a uniform operational and typing framework for a broad class of algebraic effects;
- algebraic operations admitting non-trivial comodels, equipped with comodel-based operational semantics that support step-by-step quantitative reasoning, analyzed via their encoding as infinite stacks.

Across these settings, the thesis demonstrates tight soundness and completeness results, showing that typing derivations correspond exactly to operational behavior, and that evaluation lengths and effectful steps can be read directly from types. The use of the $\lambda!$-calculus as a unifying operational core further shows how call-by-name and call-by-value evaluation, as well as state-passing and effectful computation, can be analyzed within a single quantitative framework.

Taken together, these results substantiate the claim that non-idempotent intersection types are not merely a technical tool, but a robust semantic discipline for understanding higher-order effectful computation quantitatively. They provide a principled bridge between operational semantics, type systems, and resource-aware reasoning, contributing to a broader rethinking of how programming languages should be modeled in a computational landscape where quantities matter.

Beyond its concrete constructions, the thesis highlights conceptual lessons for the design and analysis of higher-order effectful languages. It suggests that quantitative structure can be treated as a first-class component of semantic models, on a par with the algebraic description of effects themselves, and that doing so yields both technical advantages and conceptual clarity. At the same time, the results point to several open directions: extending the framework to richer forms of effects

and interaction, sharpening its connections to type systems and program logics for quantitative reasoning, and exploring its applicability to practical languages and verification tools.

In summary, the thesis advances the semantic understanding of higher-order computation with effects by providing a general, quantitative account that is both mathematically robust and broadly applicable. This lays groundwork for future research at the interface of semantics, language design, and quantitative verification, with the long-term aim of supporting more expressive, modular, and analyzable effectful programming languages.

9.2 Future Work

The results developed in this thesis open several promising directions for further research. While the present work establishes exact quantitative semantics for a range of effectful extensions of the λ-calculus, it also highlights structural questions that remain open, both at the technical and conceptual level.

9.2.1 Completeness of Call-By-Value Translations for Stateful Extensions

A first natural continuation concerns the completeness of call-by-value (CBV) translations for the effectful calculi studied in the thesis.

For the pure λ-calculus and for several pattern-matching calculi, CBV translations into finer-grained calculi (such as the $\lambda!$-calculus or CBPV-inspired systems) are known to preserve and reflect quantitative semantics. In the presence of global state and infinite stacks, however, the situation is more delicate. While the thesis establishes quantitative soundness of CBV encodings—showing that evaluation costs are preserved—the corresponding completeness results remain open.

Future work will aim to prove that CBV translations for the CBV $\lambda^{\mathcal{G}}$- and $\lambda^{\mathcal{S}}$-calculus are quantitatively complete, in the sense that every tight typing in the target calculus corresponds to a tight typing in the source calculus with exactly the same cost interpretation. Establishing such results would provide a robust operational justification for CBV encodings in the presence of stateful effects, and would further consolidate the unifying role of the $\lambda!$-calculus as an operational core language for quantitative semantics.

Technically, this requires a refined analysis of how state interactions are reified by CBV translations, and how the induced duplication and sequencing of effects can be reflected back into source-level typing derivations without loss of quantitative precision.

9.2.2 A General Quantitative Framework for Algebraic Effects

A more ambitious direction concerns the elimination of ad hoc state representations. In this thesis, global state and infinite stacks are introduced as concrete operational models supporting exact quantitative analysis. While this approach is effective, it suggests a deeper question:

Can the operational notion of state be inferred directly from the algebraic theory of effects, rather than fixed in advance?

Future work will aim to develop a fully general quantitative framework for algebraic effects, where:

- the operational semantics is derived systematically from a given algebraic theory admitting non-trivial comodels, and
- the corresponding non-idempotent intersection type system is generated uniformly from this structure.

Such a framework would take comodels as primary semantic objects and extract, from them, both:

- a fine-grained operational transition system, and
- a quantitative typing discipline whose derivations correspond exactly to execution costs.

This would generalize the infinite-stack construction developed in the thesis, isolating it as a particular instance of a broader coalgebraic methodology. Ultimately, this line of work would clarify the precise relationship between algebraic theories, their comodels, and quantitative operational semantics, providing a principled foundation for extending non-idempotent intersection types to new effectful languages.

9.2.3 Combining State-Passing Semantics with Relational Intersection Types

Another promising direction concerns the integration of state-passing computations with the relational interpretation of intersection types developed by Gavazzo, Treglia, and Vanoni in [48].

Their work provides a relational semantics for monadic and effectful computations, offering a powerful logical account of effect interactions that is largely orthogonal to the quantitative, cost-oriented perspective adopted in this thesis. A natural question is whether these two approaches can be meaningfully combined.

Future research could explore:

- a relational semantics for non-idempotent intersection types tailored to comodel-based operational semantics;

- the extension of relational intersection types to capture quantitative properties of stateful computations;
- or the development of a hybrid framework where state-passing operational models are paired with relational interpretations that abstract away from concrete execution traces while retaining quantitative information.

Such a synthesis could lead to type systems that simultaneously:

- track exact resource usage,
- express relational properties of effectful programs,
- and support compositional reasoning about state and effects.

Bridging these frameworks would also help position non-idempotent intersection types within the broader landscape of semantic approaches to effects, connecting operational, coalgebraic, and relational perspectives.

9.2.4 Further Directions

Additional avenues for future work include:

- extending the quantitative framework to other algebraic effects such as probabilistic choice or weighted nondeterminism;
- investigating connections with graded and effect-aware type systems, especially in CBPV settings;
- and studying whether the notion of tight typing can be adapted to capture space usage or amortized cost in effectful calculi.

In summary, the thesis lays the groundwork for a broad research program aimed at unifying quantitative typing, operational semantics, and algebraic effects. The directions outlined above suggest that non-idempotent intersection types can serve not only as a tool for precise cost analysis, but also as a central organizing principle for the semantics of effectful computation.

Part VI

Full Proofs

Appendix A

The λ-Calculus

A.1 Syntax and Operational Semantics

Proposition 2.10 (Characterization of β-Normal Forms)**.** *Let $t \in \Lambda$. Then, $t \not\to_\beta$ if and only if $t \in \mathtt{NO}$.*

Proof. We restate and refine the original statement as follows:

1. $t \not\to_\beta$ and t is not an abstraction if and only if $t \in \mathtt{NE}$.
2. $t \not\to_\beta$ if and only if $t \in \mathtt{NO}$.

($\Rightarrow$) The proof follows by induction on the structure of t:

1. Let $t \not\to_\beta$ and t not be an abstraction:
   - If $t = x \in \mathcal{X}$, then $x \in \mathtt{NE}$, by definition.
   - If $t = u\,r$, then $u\,r \not\to_\beta$. Therefore, u is not an abstraction, $u \not\to_\beta$, and $r \not\to_\beta$ by Definition 2.8. By applying *i.h.* 1 to u, we have $u \in \mathtt{NE}$. By applying *i.h.* 2 to r, we have $r \in \mathtt{NO}$. Therefore, $u\,r \in \mathtt{NE}$, by definition.
2. Let $t \not\to_\beta$:
   - If $t = x \in \mathcal{X}$, then $x \in \mathtt{NE} \subseteq \mathtt{NO}$, by definition.
   - If $t = \lambda y.u$, then $\lambda y.u \in \mathtt{NO}$, by definition.
   - If $t = u\,r$, then $u\,r \not\to_\beta$ and $u\,r$ is not an abstraction. Therefore, $u\,r \in \mathtt{NE} \subseteq \mathtt{NO}$, by *i.h.* 1.

($\Leftarrow$) The proof follows by induction on the definition of $\mathtt{NO}$:

1. Let $t \in \mathtt{NE}$:
   - If $t = x \in \mathcal{X}$, then $x \not\to_\beta$ by Definition 2.8.

- If $t = u\,r$, such that $u \in \mathtt{NE}$ and $r \in \mathtt{NO}$. By applying *i.h.* 1 to u, we have $u \not\to_\beta$ and u is not an abstraction. By applying *i.h.* 2 to r, we have $r \not\to_\beta$. Therefore, $u\,r \not\to_\beta$, by definition.

2. Let $t \in \mathtt{NO}$:
   - If $t = \lambda x.u$, such that $u \in \mathtt{NO}$. By applying *i.h.* 2 to u, we have $u \not\to_\beta$. Therefore, $\lambda x.u \not\to_\beta$, by definition.
   - If $t \in \mathtt{NE}$, then $t \not\to_\beta$ and t is not an abstraction, by *i.h.* 1. Therefore, $t \not\to_\beta$.

□

A.2 Call-By-Name and Call-By-Value

Proposition 2.17 (Characterization of CBN- and CBV-Normal Forms)**.** *The sets of* CBN*-normal forms* $\mathtt{NO}_{\mathtt{CBN}}$ *and* CBV*-normal forms* $\mathtt{NO}_{\mathtt{CBV}}$ *are both the set of closed values* $\mathrm{V}_\circ$.

Proof. We restate and refine the original statement as follows:

1. $t \not\to_{\mathtt{CBN}}$ if and only if t is a *closed* abstraction.
2. $t \not\to_{\mathtt{CBV}}$ if and only if t is a *closed* abstraction.

($\Rightarrow$) The proof follows by induction on the structure of t:

1. Let $t \in \Lambda_\circ$ and $t \not\to_{\mathtt{CBN}}$:
   - If $t = \lambda x.u$, then t is a closed abstraction.
   - If $t = u\,r$, then $u\,r \not\to_{\mathtt{CBN}}$. This means that u is not an abstraction and $u \not\to_{\mathtt{CBN}}$ by Definition 2.12. Moreover, it is easy to see that, in particular, $u\,r \in \Lambda_\circ$ implies that $u \in \Lambda_\circ$, and thus that u is not a closed abstraction. However, by *i.h.* 1, we have that u is a closed abstraction, which is a contradiction. Therefore, t cannot be an application.
2. Let $t \in \Lambda_\circ$ and $t \not\to_{\mathtt{CBV}}$:
   - If $t = \lambda x.u$, then t is a closed abstraction.
   - If $t = u\,r$, then $u\,r \not\to_{\mathtt{CBV}}$. This means that, either u is not a value and $u \not\to_{\mathtt{CBV}}$, or u is a value and $r \not\to_{\mathtt{CBV}}$ by Definition 2.15. Moreover, it is easy to see that, $u\,r \in \Lambda_\circ$ implies that $u, r \in \Lambda_\circ$, and thus either (i.) u is not a closed abstraction and $u \not\to_{\mathtt{CBV}}$, or (ii.) u is a closed abstraction and $r \not\to_{\mathtt{CBV}}$:
     i. By applying the *i.h.* to u, we have that u is a closed abstraction, which is a contradiction. Therefore, t cannot be an application of such a form.

ii. By applying the *i.h.* to r, we have that r is a closed abstraction. Therefore, $u\,r$ is a β-redex, which contradicts the fact that $u\,r \not\to_{\mathsf{CBV}}$. Therefore, t cannot be such an application of such a form.

Since t cannot be an application of any other form, t cannot be an application.

($\Leftarrow$) Trivial, by Definition 2.8.

□

Appendix B

The $\lambda!$-Calculus

B.1 Syntax and Operational Semantics

Proposition 3.7 (Characterization of CBPV-Normal Forms)**.** *Let $t \in !\Lambda_\circ$. Then, $t \not\to_{\mathtt{CBPV}}$ if and only if $t \in \mathtt{NO}_{\mathtt{CBPV}}$.*

Proof. We restate and refine the original statement as follows:

1. $t \not\to_{\mathtt{CBPV}}$ and t is not an abstraction or a bang if and only if $t \in \mathtt{NE}_{\mathtt{CBPV}}$.
2. $t \not\to_{\mathtt{CBPV}}$ and t is not an abstraction if and only if $t \in \mathtt{NA}_{\mathtt{CBPV}}$.
3. $t \not\to_{\mathtt{CBPV}}$ and t is not a bang if and only if $t \in \mathtt{NB}_{\mathtt{CBPV}}$.
4. $t \not\to_{\mathtt{CBPV}}$ if and only if $t \in \mathtt{NO}_{\mathtt{CBPV}}$.

($\Rightarrow$) The proof follows by induction on the structure of t:

1. Let $t \not\to_{\mathtt{CBPV}}$ and t not be an abstraction or a bang:
   - If $t = r\,e$, then $r\,e \not\to_{\mathtt{CBPV}}$. This means that, either r is not an abstraction and $r \not\to_{\mathtt{CBPV}}$, or r is an abstraction ($r \not\to_{\mathtt{CBPV}}$), $e \not\to_{\mathtt{CBPV}}$, and e is not a value by Definition 3.4. Moreover, it is easy to see that $r\,e \in !\Lambda_\circ$ implies that $r, e \in !\Lambda_\circ$, and thus either (i.) r is not an abstraction and $r \not\to_{\mathtt{CBPV}}$, or (ii.) r is an abstraction ($r \not\to_{\mathtt{CBPV}}$), $e \not\to_{\mathtt{CBPV}}$, and e is not a bang:
     i. By applying the *i.h.* Lemma 2 to r, we have that $r \in \mathtt{NA}_{\mathtt{CBPV}}$. Therefore, we have that $r\,e \in \mathtt{NE}_{\mathtt{CBPV}}$.
     ii. By applying the *i.h.* Lemma 3 to e, we have that $e \in \mathtt{NB}_{\mathtt{CBPV}}$. Therefore, we have that $r\,e \in \mathtt{NE}_{\mathtt{CBPV}}$.

- If $t = \mathtt{der}(r)$, then $\mathtt{der}(r) \not\to_{\mathtt{CBPV}}$. This means that $r \not\to_{\mathtt{CBPV}}$ and r is not a bang by Definition 3.4. By applying the *i.h.* Lemma 3 to r, we have that $r \in \mathtt{NB}_{\mathtt{CBPV}}$. Therefore, we have that $\mathtt{der}(r) \in \mathtt{NE}_{\mathtt{CBPV}}$.

2. Let $t \not\to_{\mathtt{CBPV}}$ and t not be an abstraction:
   - If $t = !r$, then $!r \in \mathtt{NA}_{\mathtt{CBPV}}$, by definition.
   - If $t = r\,e$, then $r\,e$ is also not a bang. Therefore, by Lemma 1, we have that $r\,e \in \mathtt{NE}_{\mathtt{CBPV}} \subseteq \mathtt{NA}_{\mathtt{CBPV}}$.
   - If $t = \mathtt{der}(r)$, then $\mathtt{der}(r)$ is also not a bang. Therefore, by Lemma 1, we have that $\mathtt{der}(r) \in \mathtt{NE}_{\mathtt{CBPV}} \subseteq \mathtt{NA}_{\mathtt{CBPV}}$.
3. Let $t \not\to_{\mathtt{CBPV}}$ and t not be a bang:
   - If $t = \lambda x.r$, then $\lambda x.r \in \mathtt{NB}_{\mathtt{CBPV}}$, by definition.
   - If $t = r\,e$, then $r\,e$ is also not an abstraction. Therefore, by Lemma 1, we have that $r\,e \in \mathtt{NE}_{\mathtt{CBPV}} \subseteq \mathtt{NB}_{\mathtt{CBPV}}$.
   - If $t = \mathtt{der}(r)$, then $\mathtt{der}(r)$ is also not a bang. Therefore, by Lemma 1, we have that $\mathtt{der}(r) \in \mathtt{NE}_{\mathtt{CBPV}} \subseteq \mathtt{NB}_{\mathtt{CBPV}}$.
4. Let $t \not\to_{\mathtt{CBPV}}$:
   - If $t = !r$, then $!r \in \mathtt{NA}_{\mathtt{CBPV}} \subseteq \mathtt{NO}_{\mathtt{CBPV}}$, by definition.
   - If $t = \lambda x.r$, then $\lambda x.r \in \mathtt{NB}_{\mathtt{CBPV}} \subseteq \mathtt{NO}_{\mathtt{CBPV}}$, by definition.
   - If $t = r\,e$, then $r\,e$ is not an abstraction or a bang. Therefore, by Lemma 1, we have that $r\,e \in \mathtt{NE}_{\mathtt{CBPV}} \subseteq \mathtt{NO}_{\mathtt{CBPV}}$.
   - If $t = \mathtt{der}(r)$, then $\mathtt{der}(r)$ is not an abstraction or a bang. Therefore, by Lemma 1, we have that $\mathtt{der}(r) \in \mathtt{NE}_{\mathtt{CBPV}} \subseteq \mathtt{NO}_{\mathtt{CBPV}}$.

($\Leftarrow$) The proof follows by induction on the definition of $\mathtt{NO}$:

1. Let $t \in \mathtt{NE}_{\mathtt{CBPV}}$:
   - If $t = (\lambda x.u)\,r$, such that $r \in \mathtt{NB}_{\mathtt{CBPV}}$. By applying *i.h.* Lemma 3 to r, we have that $r \not\to_{\mathtt{CBPV}}$ and r is not a bang. Moreover, $\lambda x.u \not\to_{\mathtt{CBPV}}$, by definition. Therefore, $(\lambda x.u)\,r \not\to_{\mathtt{CBPV}}$ by Definition 3.4. Moreover, $(\lambda x.u)\,r$ is not an abstraction or a bang.
   - If $t = r\,e$, such that $r \in \mathtt{NA}_{\mathtt{CBPV}}$. By applying *i.h.* Lemma 2 to r, we have that $r \not\to_{\mathtt{CBPV}}$ and r is not an abstraction. Therefore, $r\,e \not\to_{\mathtt{CBPV}}$ by Definition 3.4. Moreover, $r\,e$ is not an abstraction or a bang.
   - If $t = \mathtt{der}(r)$, such that $r \in \mathtt{NB}_{\mathtt{CBPV}}$. By applying the *i.h.* Lemma 3 to r, we have that $r \not\to_{\mathtt{CBPV}}$ and r is not a bang. Therefore, $\mathtt{der}(r) \not\to_{\mathtt{CBPV}}$ by Definition 3.4. Moreover, $\mathtt{der}(r)$ is not an abstraction or a bang.

2. Let $t \in \mathtt{NA}_{\mathtt{CBPV}}$:
   - If $t = !r$, then $!r \not\to_{\mathtt{CBPV}}$ by Definition 3.4. Moreover, $!r$ is not an abstraction.
   - If $t = r$, such that $r \in \mathtt{NE}_{\mathtt{CBPV}}$, then $r \not\to_{\mathtt{CBPV}}$ and r is not an abstraction or a bang by Lemma 1. Therefore, $r \not\to_{\mathtt{CBPV}}$ and r is not an abstraction.
3. Let $t \in \mathtt{NB}_{\mathtt{CBPV}}$:
   - If $t = \lambda x.u$, then $\lambda x.u \not\to_{\mathtt{CBPV}}$ by Definition 3.4. Moreover, $\lambda x.u$ is not a bang.
   - If $t = r$, such that $r \in \mathtt{NE}_{\mathtt{CBPV}}$, then $r \not\to_{\mathtt{CBPV}}$ and r is not an abstraction or a bang by Lemma 1. Therefore, $r \not\to_{\mathtt{CBPV}}$ and r is not a bang.
4. Let $t \in \mathtt{NO}_{\mathtt{CBPV}}$:
   - Let $t = r$, such that $r \in \mathtt{NA}_{\mathtt{CBPV}}$. By Lemma 2, $r \not\to_{\mathtt{CBPV}}$ and r is not an abstraction. Therefore, $r \not\to_{\mathtt{CBPV}}$.
   - Let $t = r$, such that $r \in \mathtt{NB}_{\mathtt{CBPV}}$. By Lemma 3, $r \not\to_{\mathtt{CBPV}}$ and r is not a bang. Therefore, $r \not\to_{\mathtt{CBPV}}$.

□

Proposition 3.11 (Characterization of Clash-Free CBPV-Normal Forms). *Let $t \in !\Lambda_{\circ}$. Then, $t \not\to_{\mathtt{CBPV}}$ and t is clash-free if and only if $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$.*

Proof. We first refine the statement as follows.

Let $t \in !\Lambda_{\circ}$. Then, $t \not\to_{\mathtt{CBPV}}$ and t is clash-free if and only if t is a closed bang or a closed abstraction.

($\Rightarrow$) Note that, since $t \not\to_{\mathtt{CBPV}}$, then $t \in \mathtt{NO}_{\mathtt{CBPV}}$ by Proposition 3.7. Therefore, $t \in \mathtt{NA}_{\mathtt{CBPV}}$ or $t \in \mathtt{NB}_{\mathtt{CBPV}}$. However, by Lemma B.1, we have that $t \notin \mathtt{NE}_{\mathtt{CBPV}}$. Therefore, t must either be a closed abstraction ($t \in \mathtt{NB}_{\mathtt{CBPV}}$ and $t \notin \mathtt{NE}_{\mathtt{CBPV}}$) or a closed bang ($t \in \mathtt{NA}_{\mathtt{CBPV}}$ and $t \notin \mathtt{NE}_{\mathtt{CBPV}}$).

($\Leftarrow$) Trivial, by Definition 3.9.

□

B.2 Subsuming Call-By-Name and Call-By-Value

Lemma B.1 (Neutral Forms are not Clashes). *Let $t \in !\Lambda_{\circ}$. If $t \in \mathtt{NE}_{\mathtt{CBPV}}$, then t is not clash-free.*

Proof. The proof follows by induction on the definition of $\mathtt{NE}_{\mathtt{CBPV}}$:

- If $t = (\lambda x.u)\, r$, such that $r \in \mathtt{NB}_{\mathtt{CBPV}}$, then either (i.) $r = \lambda y.e$, for some $\lambda y.e \in !\Lambda_{\circ}$; or (ii.) $r \in \mathtt{NE}_{\mathtt{CBPV}}$:

i. Then, $(\lambda x.u)\,\lambda y.e$ is a clash by Definition 3.8, and thus not clash-free by Definition 3.8.

ii. By applying the *i.h.* to r, r is not clash-free. Therefore, $(\lambda x.u)\,r$ is not clash-free by Definition 3.9.

- If $t = u\,r$, such that $u \in \mathtt{NA}_{\mathtt{CBPV}}$, then either (i.) $u = !e$, for some $!e \in !\Lambda_\circ$; or (ii.) $u \in \mathtt{NE}_{\mathtt{CBPV}}$:

  i. Then, $!e\,r$ is a clash by Definition 3.8, and thus not clash-free by Definition 3.8.

  ii. By applying the *i.h.* to u, u is not clash-free. Therefore, $u\,r$ is not clash-free by Definition 3.9.

- If $t = \mathtt{der}(u)$, such that $u \in \mathtt{NB}_{\mathtt{CBPV}}$, then either (i.) $u = \lambda x.r$, for some $\lambda x.r \in !\Lambda_\circ$; or (ii.) $u \in \mathtt{NE}_{\mathtt{CBPV}}$:

  i. Then, $\mathtt{der}(\lambda x.r)$ is a clash by Definition 3.8, and thus not clash-free by Definition 3.8.

  ii. By applying the *i.h.* to u, u is not clash-free. Therefore, $\mathtt{der}(u)$ is not clash-free by Definition 3.9.

□

Lemma 3.16 (Non-Interference of Single Administrative Steps)**.** *Let $t \in \Lambda_\circ$. If $(t)^{\mathtt{CBN}} \not\to_{\mathtt{CBPV}(\beta_{\mathtt{v}})}$ and $(t)^{\mathtt{CBN}} \to_{\mathtt{ADMIN}} u$, then $u \not\to_{\mathtt{CBPV}(\beta_{\mathtt{v}})}$.*

Proof. The proof follows by induction on the structure of t:

- If $t = x$, then $t \notin \Lambda_\circ$. Thus, this case does not apply.
- If $t = \lambda x.r$, then $(t)^{\mathtt{CBN}} = (\lambda x.r)^{\mathtt{CBN}} = \lambda x.(r)^{\mathtt{CBN}}$. Moreover, $\lambda x.(r)^{\mathtt{CBN}} \not\to_{\mathtt{CBPV}(\beta_{\mathtt{v}})}$, by definition. Let $\lambda x.(r)^{\mathtt{CBN}} \to_{\mathtt{ADMIN}} u$. Then, $u = \lambda x.r_0$ by Lemma B.2. Thus, $\lambda x.r_0 \not\to_{\mathtt{CBPV}(\beta_{\mathtt{v}})}$, by definition.
- If $t = r\,e$, then $(t)^{\mathtt{CBN}} = (r\,e)^{\mathtt{CBN}} = (r)^{\mathtt{CBN}}\,!(e)^{\mathtt{CBN}}$. Moreover, if $(r)^{\mathtt{CBN}}\,!(e)^{\mathtt{CBN}} \not\to_{\mathtt{CBPV}(\beta_{\mathtt{v}})}$, then $(r)^{\mathtt{CBN}} \not\to_{\mathtt{CBPV}(\beta_{\mathtt{v}})}$. Also, if $(t)^{\mathtt{CBN}} \not\to_{\mathtt{CBPV}(\beta_{\mathtt{v}})}$, then $(r)^{\mathtt{CBN}}$ is not an abstraction, since no $\beta_{\mathtt{v}}$-steps are fired. Let $r\,e \to_{\mathtt{ADMIN}} u$. Then, $u = r_0\,e_0$ by Lemma B.2. Moreover, since $(r)^{\mathtt{CBN}}\,!(e)^{\mathtt{CBN}} \to_{\mathtt{ADMIN}} r_0\,e_0$, then (i.) $(r)^{\mathtt{CBN}} \to_{\mathtt{ADMIN}} r_0$ and $e_0 = !(e)^{\mathtt{CBN}}$, or (ii.) $!(e)^{\mathtt{CBN}} \to_{\mathtt{ADMIN}} e_0$ and $r_0 = (r)^{\mathtt{CBN}}$:
  - Case (i.): By Lemma B.2, r_0 must also not be an abstraction. By the *i.h.*, we know $r_0 \not\to_{\mathtt{CBPV}(\beta_{\mathtt{v}})}$. Therefore, $r_0\,e_0 \not\to_{\mathtt{CBPV}(\beta_{\mathtt{v}})}$, by definition.
  - Case (ii.): By Lemma B.2, e_0 must also be a bang-term. By the *i.h.*, we know $e_0 \not\to_{\mathtt{CBPV}(\beta_{\mathtt{v}})}$. Recall that $r_0 = (r)^{\mathtt{CBN}}$ is not an abstraction. Therefore, $r_0\,e_0 \not\to_{\mathtt{CBPV}(\beta_{\mathtt{v}})}$, by definition.

□

Corollary 3.17 (Non-Interference of Administrative Steps). *Let $t \in \Lambda_\circ$. If $(t)^{\mathtt{CBN}} \not\to_{\mathtt{CBPV}(\beta_\mathtt{v})}$ and $(t)^{\mathtt{CBN}} \twoheadrightarrow_{\mathtt{ADMIN}} u$, then $u \not\to_{\mathtt{CBPV}(\beta_\mathtt{v})}$.*

Lemma B.2 (Stability of Administrative Steps). *Let $t \in \Lambda_\circ$ and $t \to_{\mathtt{ADMIN}} u$. Then, u has the same top level constructor symbol (application, abstraction, dereliction, bang-term, or operation) than t.*

Proof. By a simply case analysis on the structure of t. □

Theorem 3.19 (Correctness of Full Simulations). *Let $t, u \in \Lambda_\circ$.*

1. *If $t \twoheadrightarrow^{n}_{\mathtt{CBN}} u$, then $(t)^{\mathtt{CBN}} \twoheadrightarrow^{n\cdot\beta_\mathtt{v}+m\cdot\beta_!}_{\mathtt{CBPV}\cup\mathtt{ADMIN}} (u)^{\mathtt{CBN}}$, for some $m \geq 0$*
2. *If $t \twoheadrightarrow^{n}_{\mathtt{CBV}} u$, then $(t)^{\mathtt{CBV}} \twoheadrightarrow^{n\cdot\beta_\mathtt{v}+m\cdot\beta_!}_{\mathtt{CBPV}} (u)^{\mathtt{CBV}}$, for some $m \geq 0$.*

Proof. The proofs follow by induction on n:

1. Let $t \twoheadrightarrow^{n}_{\mathtt{CBN}(\beta)} u$:
   - If $n = 0$, then $u = t$. Therefore, $(t)^{\mathtt{CBN}} = (u)^{\mathtt{CBN}}$, and $(t)^{\mathtt{CBN}} \twoheadrightarrow^{0\cdot\beta_\mathtt{v}+0\cdot\beta_!}_{\mathtt{CBPV}\cup\mathtt{ADMIN}} (u)^{\mathtt{CBN}}$.
   - If $n = n_0 + 1$, then $t \to_{\mathtt{CBN}(\beta)} r \to^{n_0}_{\mathtt{CBN}(\beta)} u$, for some $r \in \Lambda_\circ$. By applying Lemma B.3 to $t \to_{\mathtt{CBN}(\beta)} r$, we have $(t)^{\mathtt{CBN}} \twoheadrightarrow^{1\cdot\beta_\mathtt{v}+m_0\cdot\beta_!}_{\mathtt{CBPV}\cup\mathtt{ADMIN}} (r)^{\mathtt{CBN}}$, for some $m_0 \geq 0$. Moreover, by applying the *i.h.* to n_0, we have that $(r)^{\mathtt{CBN}} \twoheadrightarrow^{n_0\cdot\beta_\mathtt{v}+m_1\cdot\beta_!}_{\mathtt{CBPV}\cup\mathtt{ADMIN}} (u)^{\mathtt{CBN}}$, for some $m_1 \geq 0$. Therefore, $(t)^{\mathtt{CBN}} \twoheadrightarrow^{1\cdot\beta_\mathtt{v}+m_0\cdot\beta_!}_{\mathtt{CBPV}\cup\mathtt{ADMIN}} (r)^{\mathtt{CBN}} \twoheadrightarrow^{n_0\cdot\beta_\mathtt{v}+m_1\cdot\beta_!}_{\mathtt{CBPV}\cup\mathtt{ADMIN}} (u)^{\mathtt{CBN}}$, which is the same as $(t)^{\mathtt{CBN}} \twoheadrightarrow^{n_0+1\cdot\beta_\mathtt{v}+m\cdot\beta_!}_{\mathtt{CBPV}\cup\mathtt{ADMIN}} (u)^{\mathtt{CBN}}$, for $m = m_0 + m_1 \geq 0$.
2. Let $t \twoheadrightarrow^{n}_{\mathtt{CBV}(\beta_\mathtt{v})} u$:
   - If $n = 0$, then $u = t$. Therefore, $(t)^{\mathtt{CBV}} = (u)^{\mathtt{CBV}}$, and $(t)^{\mathtt{CBV}} \twoheadrightarrow^{0\cdot\beta_\mathtt{v}+0\cdot\beta_!}_{\mathtt{CBPV}} (u)^{\mathtt{CBV}}$.
   - If $n = n_0 + 1$, then $t \to_{\mathtt{CBV}(\beta_\mathtt{v})} r \twoheadrightarrow^{n_0}_{\mathtt{CBV}(\beta_\mathtt{v})} u$, for some $r \in !\Lambda_\circ$. By applying Lemma B.3 to $t \to_{\mathtt{CBV}(\beta_\mathtt{v})} r$, we have $(t)^{\mathtt{CBV}} \twoheadrightarrow^{1\cdot\beta_\mathtt{v}+m_0\cdot\beta_!}_{\mathtt{CBPV}} (r)^{\mathtt{CBV}}$. Moreover, by applying the *i.h.* to n_0, we have that $(r)^{\mathtt{CBV}} \twoheadrightarrow^{n_0\cdot\beta_\mathtt{v}+m_1\cdot\beta_!}_{\mathtt{CBPV}} (u)^{\mathtt{CBV}}$. Therefore, $(t)^{\mathtt{CBV}} \to_{\mathtt{CBPV}(\beta_\mathtt{v})} \twoheadrightarrow^{n_0\cdot\beta_\mathtt{v}+m_0+m_1\cdot\beta_!}_{\mathtt{CBPV}} (u)^{\mathtt{CBV}}$, which is the same as $(t)^{\mathtt{CBV}} \twoheadrightarrow^{n_0+1\cdot\beta_\mathtt{v}+m\cdot\beta_!}_{\mathtt{CBPV}} (u)^{\mathtt{CBV}}$, for $m = m_0 + m_1 \geq 0$.

□

Lemma B.3 (Correctness of One-Step Simulations). *Let $t, u \in \Lambda_\circ$.*

1. *If $t \to_{\mathtt{CBN}(\beta)} u$, then $(t)^{\mathtt{CBN}} \twoheadrightarrow^{1\cdot\beta_\mathtt{v}+m\cdot\beta_!}_{\mathtt{CBPV}\cup\mathtt{ADMIN}} (u)^{\mathtt{CBN}}$, for $m \geq 0$.*
2. *If $t \to_{\mathtt{CBV}(\beta_\mathtt{v})} u$, then $(t)^{\mathtt{CBV}} \twoheadrightarrow^{1\cdot\beta_\mathtt{v}+m\cdot\beta_!}_{\mathtt{CBPV}} (u)^{\mathtt{CBV}}$, for $m \geq 0$.*

Proof. The proofs follow by induction on the definition of the contexts of the reduction strategies:

1. Let $t \to_{\mathtt{CBN}(\beta)} u$:

   - If $\mathtt{N} = \Box$, then $t = (\lambda x.r)\, e \mapsto_\beta r\{x \setminus e\} = u$. Note that $(t)^{\mathtt{CBN}} = ((\lambda x.r)\, e)^{\mathtt{CBN}} = \lambda x.(r)^{\mathtt{CBN}}\, !(e)^{\mathtt{CBN}} \mapsto_{\beta_{\mathtt{v}}} (r)^{\mathtt{CBN}}\{x \setminus !(e)^{\mathtt{CBN}}\}$. Therefore, $(r)^{\mathtt{CBN}}\{x \setminus !(e)^{\mathtt{CBN}}\} \overset{\text{by Lemma B.4}}{\mapsto}_{\beta_!} \twoheadrightarrow_{\mathtt{ADMIN}} (r\{x \setminus e\})^{\mathtt{CBN}} = (u)^{\mathtt{CBN}}$. Thus, $(t)^{\mathtt{CBN}} = ((\lambda x.r)\, e)^{\mathtt{CBN}} \twoheadrightarrow^{1\cdot\beta_{\mathtt{v}}+m\cdot\beta_!}_{\mathtt{CBPV}\cup\mathtt{ADMIN}} (r\{x \setminus e\})^{\mathtt{CBN}} = (u)^{\mathtt{CBN}}$, for some $m \geq 0$.

   - If $\mathtt{N} = \mathtt{N}'\, r$, then $t = \mathtt{N}'[e]\, r \to_{\mathtt{CBN}(\beta)} \mathtt{N}'[e']\, r = u$, such that $e \mapsto_\beta e'$. Note that $(\mathtt{N}'[e]\, r)^{\mathtt{CBN}} = (\mathtt{N}'[e])^{\mathtt{CBN}}\, !(r)^{\mathtt{CBN}}$. By applying the *i.h.* to $\mathtt{N}'$, $(\mathtt{N}'[e])^{\mathtt{CBN}} \twoheadrightarrow^{1\cdot\beta_{\mathtt{v}}+m\cdot\beta_!}_{\mathtt{CBPV}\cup\mathtt{ADMIN}} (\mathtt{N}'[e'])^{\mathtt{CBN}}$, for some $m \geq 0$. Therefore, $(\mathtt{N}'[e])^{\mathtt{CBN}}\, !(r)^{\mathtt{CBN}} \twoheadrightarrow^{1\cdot\beta_{\mathtt{v}}+m\cdot\beta_!}_{\mathtt{CBPV}\cup\mathtt{ADMIN}} (\mathtt{N}'[e'])^{\mathtt{CBN}}\, !(r)^{\mathtt{CBN}} = (\mathtt{N}'[e']\, r)^{\mathtt{CBN}} = (u)^{\mathtt{CBN}}$. Thus, $(t)^{\mathtt{CBN}} = (\mathtt{N}'[e]\, r)^{\mathtt{CBN}} \twoheadrightarrow^{1\cdot\beta_{\mathtt{v}}+m\cdot\beta_!}_{\mathtt{CBPV}\cup\mathtt{ADMIN}} (\mathtt{N}'[e']\, r)^{\mathtt{CBN}} = (u)^{\mathtt{CBN}}$.

2. Let $t \to_{\mathtt{CBV}(\beta_{\mathtt{v}})} u$:

   - If $\mathtt{V} = \Box$, then $t = (\lambda x.r)\, v = u$. Note that $((\lambda x.r)\, v)^{\mathtt{CBV}} = (\lambda x.(r)^{\mathtt{CBV}})\, (v)^{\mathtt{CBV}} \overset{\text{by Lemma B.5}}{\to}_{\mathtt{CBPV}(\beta_{\mathtt{v}})} (r)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\}$. Therefore, $(r)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\} \overset{\text{by Lemma B.4}}{=} (r\{x \setminus v\})^{\mathtt{CBV}} = (u)^{\mathtt{CBV}}$. Thus, $(t)^{\mathtt{CBV}} = ((\lambda x.r)\, v)^{\mathtt{CBV}} \to_{\mathtt{CBPV}(\beta_{\mathtt{v}})} (r\{x \setminus v\})^{\mathtt{CBV}} = (u)^{\mathtt{CBV}}$.

   - If $\mathtt{V} = \mathtt{V}'\, r$, then $t = \mathtt{V}'[e]\, r \to_{\mathtt{CBV}(\beta)} \mathtt{V}'[e']\, r = u$, such that $e \mapsto_{\beta_{\mathtt{v}}} e'$. Therefore, $(\mathtt{V}'[e]\, r)^{\mathtt{CBV}} = \mathtt{der}((\mathtt{V}'[e])^{\mathtt{CBV}})\, (r)^{\mathtt{CBV}}$. By applying the *i.h.* to $\mathtt{V}'$, we know that the following is the case $(\mathtt{V}'[e])^{\mathtt{CBV}} \overset{1\cdot\beta_{\mathtt{v}}+m_0\cdot\beta_!}{\to}_{\mathtt{CBPV}} (\mathtt{V}'[e'])^{\mathtt{CBV}}$. Thus, $(t)^{\mathtt{CBV}} = (\mathtt{V}'[e]\, r)^{\mathtt{CBV}} = (\mathtt{V}'[e])^{\mathtt{CBV}}\, (r)^{\mathtt{CBV}} = \mathtt{der}((\mathtt{V}'[e])^{\mathtt{CBV}})\, (r)^{\mathtt{CBV}} \overset{1\cdot\beta_{\mathtt{v}}+m_0\cdot\beta_!}{\twoheadrightarrow}_{\mathtt{CBPV}} \mathtt{der}((\mathtt{V}'[e'])^{\mathtt{CBV}})\, (r)^{\mathtt{CBV}} = (\mathtt{V}'[e']\, r)^{\mathtt{CBV}} = (u)^{\mathtt{CBV}}$.

   - If $\mathtt{V} = v\, \mathtt{V}'$, then $t = v\, \mathtt{V}'[r] \to_{\mathtt{CBV}(\beta_{\mathtt{v}})} v\, \mathtt{V}'[r'] = u$, such that $r \mapsto_{\beta_{\mathtt{v}}} r'$. Note that, since we know that $t \in \Lambda_\circ$, we have $v \in \mathrm{V}_\circ$. So, $v = \lambda x.e$, and $(v\, \mathtt{V}'[r])^{\mathtt{CBV}} = (\mathtt{der}(!\lambda x.(e)^{\mathtt{CBV}}))\, (\mathtt{V}'[r])^{\mathtt{CBV}} \to_{\mathtt{CBPV}(\beta_!)} (\lambda x.(e)^{\mathtt{CBV}})\, (\mathtt{V}'[r])^{\mathtt{CBV}}$. By applying the *i.h.* to $\mathtt{V}'$, we know that $(\mathtt{V}'[r])^{\mathtt{CBV}} \overset{1\cdot\beta_{\mathtt{v}}+m_0\cdot\beta_!}{\twoheadrightarrow}_{\mathtt{CBPV}} (\mathtt{V}'[r'])^{\mathtt{CBV}}$. Therefore, $(\lambda x.(e)^{\mathtt{CBV}})\, (\mathtt{V}'[r])^{\mathtt{CBV}} \to_{\mathtt{CBPV}(\beta_{\mathtt{v}})} (\lambda x.(e)^{\mathtt{CBV}})\, (\mathtt{V}'[r'])^{\mathtt{CBV}} = (v\, \mathtt{V}'[r'])^{\mathtt{CBV}} = (u)^{\mathtt{CBV}}$. Thus, we can conclude with $(t)^{\mathtt{CBV}} = (v\, \mathtt{V}'[r])^{\mathtt{CBV}} \overset{1\cdot\beta_{\mathtt{v}}+(m_0+1)\cdot\beta_!}{\twoheadrightarrow}_{\mathtt{CBPV}} (v\, \mathtt{V}'[r'])^{\mathtt{CBV}} = (u)^{\mathtt{CBV}}$.

□

Lemma B.4 (Preservation under Substitution). *Let $t \in \Lambda$, $u \in \Lambda_\circ$, and $v \in \mathrm{V}_\circ$.*

1. $(t)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\} \to^{=}_{\beta_!} \twoheadrightarrow_{\mathtt{ADMIN}} (t\{x \setminus u\})^{\mathtt{CBN}}$.
2. $(t)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\} = (t\{x \setminus v\})^{\mathtt{CBV}}$.

Proof. The proofs follow by induction on the structure of t:

1. We start by proving the statement for the CBN translation:

- If $t = y \neq x$, then $(t)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\} = (y)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\} = \mathtt{der}(y)\{x \setminus !(u)^{\mathtt{CBN}}\} = \mathtt{der}(y) = (y)^{\mathtt{CBN}} = (y\{x \setminus u\})^{\mathtt{CBN}} = (t\{x \setminus u\})^{\mathtt{CBN}}$.
- If $t = x$, then we have $(t)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\} = (x)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\} = \mathtt{der}(x)\{x \setminus !(u)^{\mathtt{CBN}}\} = \mathtt{der}(!(u)^{\mathtt{CBN}}) \mapsto_{\beta_!} (u)^{\mathtt{CBN}} = (x\{x \setminus u\})^{\mathtt{CBN}} = (t\{x \setminus u\})^{\mathtt{CBN}}$.
- If $t = \lambda y.r$, then we know $(t)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\} = (\lambda y.r)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\} = (\lambda y.(r)^{\mathtt{CBN}})\{x \setminus !(u)^{\mathtt{CBN}}\} = \lambda y.(r)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}$. By applying the *i.h.* to r, $(r)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\} \to_{\beta_!}^{=} \twoheadrightarrow_{\mathtt{ADMIN}} (r\{x \setminus u\})^{\mathtt{CBN}}$. Thus, $\lambda y.(r)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\} \twoheadrightarrow_{\mathtt{ADMIN}} \lambda y.(r\{x \setminus !u\})^{\mathtt{CBN}} = (\lambda y.r\{x \setminus u\})^{\mathtt{CBN}} = ((\lambda y.r)\{x \setminus u\})^{\mathtt{CBN}} = (t\{x \setminus u\})^{\mathtt{CBN}}$.
- If $t = r\,e$, then $(t)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\} = (r\,e)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\} = ((r)^{\mathtt{CBN}}\,!(e)^{\mathtt{CBN}})\{x \setminus !(u)^{\mathtt{CBN}}\} = (r)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}\,!(e)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\} = (r)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}\,!((e)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\})$. By applying the *i.h.* to r, we have $(r)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\} \to_{\beta_!}^{=} \twoheadrightarrow_{\mathtt{ADMIN}} (r\{x \setminus u\})^{\mathtt{CBN}}$. Therefore, $(r)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}\,!((e)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}) \to_{\beta_!}^{=} \twoheadrightarrow_{\mathtt{ADMIN}} (r\{x \setminus u\})^{\mathtt{CBN}}\,!((e)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\})$. By applying the *i.h.* to e, we have $(e)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\} \to_{\beta_!}^{=} \twoheadrightarrow_{\mathtt{ADMIN}} (e\{x \setminus u\})^{\mathtt{CBN}}$. Therefore, we have that $(r\{x \setminus u\})^{\mathtt{CBN}}\,!((e)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}) \to_{\beta_!}^{=} \twoheadrightarrow_{\mathtt{ADMIN}} (r\{x \setminus u\})^{\mathtt{CBN}}\,!(e\{x \setminus u\})^{\mathtt{CBN}} = (r\{x \setminus u\}\,e\{x \setminus u\})^{\mathtt{CBN}} = ((r\,e)\{x \setminus u\})^{\mathtt{CBN}} = (t\{x \setminus u\})^{\mathtt{CBN}}$.

2. We now prove the statement for the CBV translation:

- If $t = y \neq x$, then we have $(t)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\} = y\{x \setminus (v)^{\mathtt{CBV}}\} \overset{\text{by Lemma B.5}}{=} y = (y)^{\mathtt{CBV}} = (y\{x \setminus v\})^{\mathtt{CBV}} = (t\{x \setminus v\})^{\mathtt{CBV}}$.
- If $t = x$, then $(t)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\} = (x)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\} = x\{x \setminus (v)^{\mathtt{CBV}}\} \overset{\text{by Lemma B.5}}{=} (v)^{\mathtt{CBV}} = (x\{x \setminus v\})^{\mathtt{CBV}} = (t\{x \setminus v\})^{\mathtt{CBV}}$.
- If $t = \lambda y.u$, then we know $(t)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\} = (\lambda y.u)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\} = !\lambda y.(u)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\} = !((\lambda y.(u)^{\mathtt{CBV}})\{x \setminus (v)^{\mathtt{CBV}}\}) = !(\lambda y.(u)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\})$. By applying the *i.h.* to u, we have $(u)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\} = (u\{x \setminus v\})^{\mathtt{CBV}}$. Therefore, also $!(\lambda y.(u)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\}) = !\lambda y.(u\{x \setminus v\})^{\mathtt{CBV}} = (\lambda y.u\{x \setminus v\})^{\mathtt{CBV}} = ((\lambda y.u)\{x \setminus v\})^{\mathtt{CBV}} = (t\{x \setminus v\})^{\mathtt{CBV}}$.
- If $t = u\,r$, $(u\,r)^{\mathtt{CBV}} = \mathtt{der}((u)^{\mathtt{CBV}})\,(r)^{\mathtt{CBV}}$. Therefore, we know $(u\,r)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\} = \mathtt{der}((u)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\})\,(r)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\}$. By applying the *i.h.* to u, we have $(u)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\} = (u\{x \setminus v\})^{\mathtt{CBV}}$. By applying the *i.h.* to r, we have $(r)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\} = (r\{x \setminus v\})^{\mathtt{CBV}}$. Therefore, we have that $\mathtt{der}((u)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\})\,(r)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\} = \mathtt{der}((u\{x \setminus v\})^{\mathtt{CBV}})\,(r\{x \setminus v\})^{\mathtt{CBV}} = (u\{x \setminus v\}\,r\{x \setminus v\})^{\mathtt{CBV}} = ((u\,r)\{x \setminus v\})^{\mathtt{CBV}} = (t\{x \setminus v\})^{\mathtt{CBV}}$.

□

Lemma B.5 (Preservation of Values). *If* $v \in \mathrm{V}$, *then* $(v)^{\mathtt{CBV}} \in !\mathrm{V}$.

Proof. The proof follows by case analysis on the form of $v \in \mathrm{V}$:

- If $v = x$, then $(v)^{\mathtt{CBV}} = (x)^{\mathtt{CBV}} = x \in !\mathrm{V}$.
- If $v = \lambda x.t$, then $(\lambda x.t)^{\mathtt{CBV}} = !\lambda x.(t)^{\mathtt{CBV}} \in !\mathrm{V}$.

□

Appendix C

Non-Idempotent Intersection Types

C.1 Call-By-Name and Call-By-Value

Theorem 4.19 (Soundness and Completeness for CBN and CBV)**.** *Let $t \in \Lambda_\circ$. Then, t is:*

1. *$\mathcal{N}$-typable if and only if t $\mathtt{CBN}$-normalizes to some term $u \in \mathtt{NO}_{\mathtt{CBN}}$. Moreover, if $\Phi \rhd_{\mathcal{N}} \emptyset \vdash t : A$, then $t \twoheadrightarrow^{n}_{\mathtt{CBN}} u$ and $\#_{\mathtt{App}}(\Phi) \geq n$.*

2. *$\mathcal{V}$-typable if and only if t $\mathtt{CBV}$-normalizes to some term $u \in \mathtt{NO}_{\mathtt{CBV}}$. Moreover, if $\Phi \rhd_{\mathcal{V}} \emptyset \vdash t : M$, then $t \twoheadrightarrow^{n}_{\mathtt{CBV}} u$ and $\#_{\mathtt{App}}(\Phi) \geq n$.*

Proof. The proofs for both statements follow the same overall structure. The $(\Rightarrow)$ direction holds by a weighted version of subject reduction (see Lemma 4.13), and the $(\Leftarrow)$ direction holds by the typability of normal forms (see Lemma 4.17) and a weighted version of subject expansion (see Lemma 4.15). The "moreover" statements easily hold by the weighted nature of the subject reduction and expansion statements. □

Lemma 4.9 (Weighted Substitution for CBN and CBV)**.** *Let $t, u \in \Lambda$, and $v \in \mathrm{V}$.*

1. *If $\Phi \rhd_{\mathcal{N}} \Gamma; x : M \vdash t : A$ (resp. N) and $\Psi \rhd_{\mathcal{N}} \emptyset \vdash u : M$, then $\Pi \rhd_{\mathcal{N}} \Gamma \vdash t\{x \setminus u\} : A$ (resp. N), such that $\#_{\mathtt{App}}(\Pi) = \#_{\mathtt{App}}(\Phi) + \#_{\mathtt{App}}(\Psi)$.*

2. *If $\Phi \rhd_{\mathcal{V}} \Gamma; x : M \vdash t : N$ and $\Psi \rhd_{\mathcal{V}} \emptyset \vdash v : M$, then $\Pi \rhd_{\mathcal{V}} \Gamma \vdash t\{x \setminus v\} : N$, such that $\#_{\mathtt{App}}(\Pi) = \#_{\mathtt{App}}(\Phi) + \#_{\mathtt{App}}(\Psi)$.*

Proof. The proofs for both statements follow by induction on the structure of Φ.

1. We proceed by cases, according to the last rule R of Φ:

   - Case $R = \mathtt{Ax}$. Then, t is a variable, and we must consider the two following possibilities:

– If $t = y \neq x$, then Φ must be of the form

$$\frac{}{y : \{\!\{A\}\!\}; x : \{\!\{\ \}\!\} \vdash y : A}\ \texttt{Ax}$$

$\Gamma = y : \{\!\{A\}\!\}$, and $M = \{\!\{\ \}\!\}$. Moreover, $t\{x \setminus u\} = y\{x \setminus u\} = y$, and Ψ must be of the form

$$\frac{}{\emptyset \vdash u : \{\!\{\ \}\!\}}\ \texttt{Many}$$

Therefore, we can take $\Pi = \Phi$, since $\#_{\texttt{App}}(\Psi) = 0$, and thus $\#_{\texttt{App}}(\Pi) = \#_{\texttt{App}}(\Phi) + \#_{\texttt{App}}(\Psi)$.

– If $t = x$, then Φ must be of the form

$$\frac{}{x : \{\!\{A\}\!\} \vdash x : A}\ \texttt{Ax}$$

$\Gamma = \emptyset$, and $M = \{\!\{A\}\!\}$. Moreover, $t\{x \setminus u\} = x\{x \setminus u\} = u$. Therefore, we can take $\Pi = \Psi$, since $\#_{\texttt{App}}(\Phi) = 0$, and thus $\#_{\texttt{App}}(\Pi) = \#_{\texttt{App}}(\Phi) + \#_{\texttt{App}}(\Psi)$.

- Case $R = \texttt{Abs}$. Then, $t = \lambda y.r$, and Φ must be of the form

$$\frac{\Phi' \rhd_{\mathcal{N}} \Delta; x : M \vdash r : B}{(\Delta \setminus\!\setminus y); x : M \vdash \lambda y.r : \Delta(y) \to B}\ \texttt{Abs}$$

$\Gamma = \Delta \setminus\!\setminus y$, and $A = \Delta(y) \to B$. By applying the *i.h.* to Φ', we know there exists $\Pi' \rhd_{\mathcal{N}} \Delta \vdash r\{x \setminus u\} : B$, such that $\#_{\texttt{App}}(\Pi') = \#_{\texttt{App}}(\Phi') + \#_{\texttt{App}}(\Psi)$. Therefore, we can take Π to be

$$\frac{\Pi' \rhd_{\mathcal{N}} \Delta \vdash r\{x \setminus u\} : B}{\Delta \setminus\!\setminus y \vdash \lambda y.r\{x \setminus u\} : \Delta(y) \to B}\ \texttt{Abs}$$

since $\lambda y.r\{x \setminus u\} = (\lambda y.r)\{x \setminus u\} = t\{x \setminus u\}$, and $\#_{\texttt{App}}(\Pi) = \#_{\texttt{App}}(\Pi') = \#_{\texttt{App}}(\Phi') + \#_{\texttt{App}}(\Psi) = \#_{\texttt{App}}(\Phi) + \#_{\texttt{App}}(\Psi)$.

- Case $R = \texttt{Ans}$. Then, $t = \lambda y.r$, and Φ must be of the form

$$\frac{}{\emptyset \vdash \lambda y.r : \star}\ \texttt{Ans}$$

$\Gamma = \emptyset$, $M = \{\!\{\ \}\!\}$, and $A = \star$. Moreover, $t\{x \setminus u\} = (\lambda y.r)\{x \setminus u\} = \lambda y.r\{x \setminus u\}$, and Ψ must be of the form

$$\frac{}{\emptyset \vdash u : \{\!\{\ \}\!\}}\ \texttt{Many}$$

Therefore, we can take Π to be

$$\frac{}{\emptyset \vdash \lambda y.r\{x \setminus u\} : \star}\ \texttt{Ans}$$

since $\#_{\texttt{App}}(\Psi) = 0$ and $\#_{\texttt{App}}(\Pi) = \#_{\texttt{App}}(\Phi)$, and thus $\#_{\texttt{App}}(\Pi) = \#_{\texttt{App}}(\Phi) + \#_{\texttt{App}}(\Psi)$.

- Case $R = \texttt{App}$. Then, $t = r\,e$, and Φ must be of the form

$$\frac{\Phi' \rhd_{\mathcal{N}} \Delta; x : F \vdash r : N' \to A \qquad \Phi'' \rhd_{\mathcal{N}} \Sigma; x : G \vdash e : N'}{(\Delta + \Sigma); x : F \sqcup G \vdash r\,e : A}\ \texttt{App}$$

  $\Gamma = \Delta + \Sigma$, and $M = F \sqcup G$. Since $M = F \sqcup G$, we can apply Lemma C.1 to Ψ to obtain the two following derivations

$$\Psi' \rhd_{\mathcal{N}} \emptyset \vdash u : F \text{ and } \Psi'' \rhd_{\mathcal{N}} \emptyset \vdash u : G,$$

  such that $\#_{\texttt{App}}(\Psi) = \#_{\texttt{App}}(\Psi') + \#_{\texttt{App}}(\Psi'')$. By applying the *i.h.* to Φ', we know there exists $\Pi' \rhd_{\mathcal{N}} \Delta \vdash r\{x \setminus u\} : N' \to A$, such that $\#_{\texttt{App}}(\Pi') = \#_{\texttt{App}}(\Phi') + \#_{\texttt{App}}(\Psi')$. Moreover, by applying the *i.h.* to Φ'', we know there exists $\Pi'' \rhd_{\mathcal{N}} \Sigma \vdash e\{x \setminus u\} : N'$, such that $\#_{\texttt{App}}(\Pi'') = \#_{\texttt{App}}(\Phi'') + \#_{\texttt{App}}(\Psi'')$. Therefore, we can take Π to be

$$\frac{\Pi' \rhd_{\mathcal{N}} \Delta \vdash r\{x \setminus u\} : N' \to A \qquad \Pi'' \rhd_{\mathcal{N}} \Sigma \vdash e\{x \setminus u\} : N'}{\Delta + \Sigma \vdash r\{x \setminus u\}\, e\{x \setminus u\} : A}\ \texttt{App}$$

  since $r\{x \setminus u\}\, e\{x \setminus u\} = (r\,e)\{x \setminus u\} = t\{x \setminus u\}$, and $\#_{\texttt{App}}(\Pi) = \#_{\texttt{App}}(\Pi') + \#_{\texttt{App}}(\Pi'') + 1 = \#_{\texttt{App}}(\Phi') + \#_{\texttt{App}}(\Psi') + \#_{\texttt{App}}(\Phi'') + \#_{\texttt{App}}(\Psi'') + 1 = \#_{\texttt{App}}(\Phi) + \#_{\texttt{App}}(\Psi)$.

- Case $R = \texttt{Many}$. Then, Φ must be of the form

$$\frac{(\Phi_i \rhd_{\mathcal{N}} \Gamma_i; x : M_i \vdash t : A_i)_{i \in \mathtt{I}}}{(+_{i \in \mathtt{I}} \Gamma_i); x : \sqcup_{i \in \mathtt{I}} M_i \vdash t : \{\!\{A_i\}\!\}_{i \in \mathtt{I}}}\ \texttt{Many}$$

  $\Gamma = +_{i \in \mathtt{I}} \Gamma_i$, and $M = \sqcup_{i \in \mathtt{I}} M_i$, and $N = \{\!\{A_i\}\!\}_{i \in \mathtt{I}}$. Since $M = \sqcup_{i \in \mathtt{I}} M_i$, we can apply Lemma C.1 to Ψ to obtain the following derivations

$$\Psi_i \rhd_{\mathcal{N}} \emptyset \vdash u : M_i,$$

  such that $\#_{\texttt{App}}(\Psi) = +_{i \in \mathtt{I}} \#_{\texttt{App}}(\Psi_i)$. By applying the *i.h.* to each Φ_i, we know there exist $\Pi_i \rhd_{\mathcal{N}} \Gamma_i \vdash t\{x \setminus u\} : A_i$, such that $\#_{\texttt{App}}(\Pi_i) = \#_{\texttt{App}}(\Phi_i) + \#_{\texttt{App}}(\Psi_i)$, for each $i \in \mathtt{I}$. Therefore, we can take Π to be

$$\frac{(\Pi_i \rhd_{\mathcal{N}} \Gamma_i \vdash t\{x \setminus u\} : A_i)_{i \in \mathtt{I}}}{+_{i \in \mathtt{I}} \Gamma_i \vdash t\{x \setminus u\} : \{\!\{A_i\}\!\}_{i \in \mathtt{I}}}\ \texttt{Many}$$

  and $\#_{\texttt{App}}(\Pi) = +_{i \in \mathtt{I}} \#_{\texttt{App}}(\Pi_i) = +_{i \in \mathtt{I}} (\#_{\texttt{App}}(\Phi_i) + \#_{\texttt{App}}(\Psi_i)) = \#_{\texttt{App}}(\Phi) + \#_{\texttt{App}}(\Psi)$.

2. We proceed by cases, according to the last rule of Φ. We omit the proof, since it is very similar to the one for CBN.

□

Lemma 4.13 (Weighted Subject Reduction for CBN and CBV). *Let $t, u \in \Lambda_\circ$.*

1. *If $t \to_{\texttt{CBN}} u$ and $\Phi \rhd_{\mathcal{N}} \emptyset \vdash t : A$, then there is $\Psi \rhd_{\mathcal{N}} \emptyset \vdash u : A$, such that $\#_{\texttt{App}}(\Phi) = \#_{\texttt{App}}(\Psi) + 1$.*

2. *If* $t \to_{\mathtt{CBV}} u$ *and* $\Phi \rhd_{\mathcal{V}} \varnothing \vdash t : M$*, then there is* $\Psi \rhd_{\mathcal{V}} \varnothing \vdash u : M$*, such that* $\#_{\mathtt{App}}(\Phi) = \#_{\mathtt{App}}(\Psi) + 1$.

Proof. The proofs follow by induction on the definition of the contexts of the reduction strategies:

1. Let $t = \mathtt{N}[t_0] \to_{\mathtt{CBN}} \mathtt{N}[u_0] = u$, such that $t_0 \mapsto_\beta u_0$:

   - Case $\mathtt{N} = \Box$. Then, $t_0 = (\lambda x.r)\, e \mapsto_\beta r\{x \setminus e\} = u_0$, and Φ must be of the form

   $$\dfrac{\dfrac{\Phi' \rhd_{\mathcal{N}} x : N \vdash r : A}{\varnothing \vdash \lambda x.r : N \to A}\ \mathtt{Abs} \qquad \Phi'' \rhd_{\mathcal{N}} \varnothing \vdash e : N}{\varnothing \vdash (\lambda x.r)\, e : A}\ \mathtt{App}$$

   Given the forms of Φ' and Φ'', we can apply Lemma 4.9 to obtain a derivation $\Pi \rhd_{\mathcal{N}} \varnothing \vdash r\{x \setminus e\} : A$, such that $\#_{\mathtt{App}}(\Pi) = \#_{\mathtt{App}}(\Phi') + \#_{\mathtt{App}}(\Phi'')$. Therefore, we can take $\Psi = \Pi$, since $\#_{\mathtt{App}}(\Phi) = \#_{\mathtt{App}}(\Phi') + \#_{\mathtt{App}}(\Phi'') + 1 = \#_{\mathtt{App}}(\Pi) + 1 = \#_{\mathtt{App}}(\Psi) + 1$.

   - Case $\mathtt{N} = \mathtt{N}'\, r$. Then, $t = \mathtt{N}'[t_0]\, r \to_{\mathtt{CBN}} \mathtt{N}'[u_0]\, r = u$, such that $\mathtt{N}'[t_0] \to_{\mathtt{CBN}} \mathtt{N}'[u_0]$, and Φ must be of the form

   $$\dfrac{\Phi' \rhd_{\mathcal{N}} \varnothing \vdash \mathtt{N}'[t_0] : N \to A \qquad \Phi'' \rhd_{\mathcal{N}} \varnothing \vdash r : N}{\varnothing \vdash \mathtt{N}'[t_0]\, r : A}\ \mathtt{App}$$

   By applying the *i.h.* to $\mathtt{N}'[t_0]$, we know there is $\Psi' \rhd_{\mathcal{N}} \varnothing \vdash \mathtt{N}'[u_0] : N \to A$, such that $\#_{\mathtt{App}}(\Phi') = \#_{\mathtt{App}}(\Psi') + 1$. Therefore, we can take Ψ to be

   $$\dfrac{\Psi' \rhd_{\mathcal{N}} \varnothing \vdash \mathtt{N}'[u_0] : N \to A \qquad \Phi'' \rhd_{\mathcal{N}} \varnothing \vdash r : N}{\varnothing \vdash \mathtt{N}'[u_0]\, r : A}\ \mathtt{App}$$

   since $\#_{\mathtt{App}}(\Phi) = \#_{\mathtt{App}}(\Phi') + \#_{\mathtt{App}}(\Phi'') + 1 = \#_{\mathtt{App}}(\Psi') + \#_{\mathtt{App}}(\Phi'') + 2 = \#_{\mathtt{App}}(\Psi) + 1$.

2. Let $t = \mathtt{V}[t_0] \to_{\mathtt{CBV}} \mathtt{V}[u_0] = u$, such that $t_0 \mapsto_{\beta_{\mathtt{v}}} u_0$:

   - Case $\mathtt{V} = \Box$. We omit the proof since it is very similar to the one for CBN.
   - Case $\mathtt{V} = \mathtt{V}'\, \Box$. We omit the proof since it is very similar to the one for CBN.
   - Case $\mathtt{V} = w\, \mathtt{V}'$. Then, $t = w\, \mathtt{V}'[t_0] \to_{\mathtt{CBV}} w\, \mathtt{V}'[u_0] = u$, such that $\mathtt{V}[t_0] \to_{\mathtt{CBV}} \mathtt{V}[u_0]$, and Φ must be of the form

   $$\dfrac{\Phi' \rhd_{\mathcal{V}} \varnothing \vdash w : N \to M \qquad \Phi'' \rhd_{\mathcal{V}} \varnothing \vdash \mathtt{V}'[t_0] : N}{\varnothing \vdash w\, \mathtt{V}'[t_0] : M}\ \mathtt{App}$$

   By applying the *i.h.* to $\mathtt{V}'[t_0]$, we know there is $\Psi' \rhd_{\mathcal{V}} \varnothing \vdash \mathtt{V}'[u_0] : N$, such that $\#_{\mathtt{App}}(\Phi'') = \#_{\mathtt{App}}(\Psi') + 1$. Therefore, we can take Ψ to be

   $$\dfrac{\Phi' \rhd_{\mathcal{V}} \varnothing \vdash w : N \to M \qquad \Psi' \rhd_{\mathcal{V}} \varnothing \vdash \mathtt{V}'[u_0] : N}{\varnothing \vdash w\, \mathtt{V}'[u_0] : M}\ \mathtt{App}$$

   since $\#_{\mathtt{App}}(\Phi) = \#_{\mathtt{App}}(\Phi') + \#_{\mathtt{App}}(\Phi'') + 1 = \#_{\mathtt{App}}(\Phi') + \#_{\mathtt{App}}(\Psi') + 2 = \#_{\mathtt{App}}(\Psi) + 1$.

□

Lemma 4.11 (Weighted Anti-Substitution for CBN and CBV). *Let $t \in \Lambda$, $u \in \Lambda_\circ$, and $v \in \mathrm{V}_\circ$.*

1. *If $\Phi \rhd_{\mathcal{N}} \Gamma \vdash t\{x \setminus u\} : A$ (resp. M), then $\Psi \rhd_{\mathcal{N}} \Gamma; x : M \vdash t : A$ (resp. M) and $\Pi \rhd_{\mathcal{N}} \emptyset \vdash u : M$, such that $\#_{\mathtt{App}}(\Phi) = \#_{\mathtt{App}}(\Psi) + \#_{\mathtt{App}}(\Pi)$.*

2. *If $\Phi \rhd_{\mathcal{V}} \Gamma \vdash t\{x \setminus v\} : N$, then $\Psi \rhd_{\mathcal{V}} \Gamma; x : M \vdash t : N$ and $\Pi \rhd_{\mathcal{V}} \emptyset \vdash v : M$, such that $\#_{\mathtt{App}}(\Phi) = \#_{\mathtt{App}}(\Psi) + \#_{\mathtt{App}}(\Pi)$.*

Proof.

1. The proof follows by induction on the structure of t.

   - Case $t = y \neq x$. Then, $t\{x \setminus u\} = y\{x \setminus u\} = y$. Therefore, we can take $M = \{\!\{\,\}\!\}$, $\Psi = \Phi$, and Π as the following derivation

     $$\frac{}{\emptyset \vdash u : \{\!\{\,\}\!\}}\ \mathtt{Many}$$

     since $\#_{\mathtt{App}}(\Phi) = \#_{\mathtt{App}}(\Psi) = \#_{\mathtt{App}}(\Psi) + \#_{\mathtt{App}}(\Pi)$.

   - Case $t = x$. Then, $t\{x \setminus u\} = x\{x \setminus u\} = u$, and, since $\mathtt{fv}(u) = \emptyset$, we know that $\Gamma = \emptyset$ by Lemma C.2. Now, we must consider two cases, according to the type assigned to u in Φ:

     – Let $\Phi \rhd_{\mathcal{N}} \emptyset \vdash u : A$. Then, we can take $M = \{\!\{A\}\!\}$, Π to be

     $$\frac{\Phi \rhd_{\mathcal{N}} \emptyset \vdash u : A}{\emptyset \vdash u : \{\!\{A\}\!\}}\ \mathtt{Many}$$

     and Ψ to be

     $$\frac{}{x : \{\!\{A\}\!\} \vdash x : A}\ \mathtt{Ax}$$

     since $\#_{\mathtt{App}}(\Phi) = \#_{\mathtt{App}}(\Pi) = \#_{\mathtt{App}}(\Psi) + \#_{\mathtt{App}}(\Pi)$.

     – Let $\Phi \rhd_{\mathcal{N}} \emptyset \vdash u : N$. Then, we can take $M = N$, $\Pi = \Phi$, assume (without loss of generality) that $M = \{\!\{A_i\}\!\}_{i \in \mathtt{I}}$, and Ψ to be

     $$\frac{\left(\dfrac{}{x : \{\!\{A_i\}\!\} \vdash x : A_i}\ \mathtt{Ax}\right)_{i \in \mathtt{I}}}{x : \{\!\{A_i\}\!\}_{i \in \mathtt{I}} \vdash x : \{\!\{A_i\}\!\}_{i \in \mathtt{I}}}\ \mathtt{Many}$$

     since $\#_{\mathtt{App}}(\Phi) = \#_{\mathtt{App}}(\Pi) = \#_{\mathtt{App}}(\Psi) + \#_{\mathtt{App}}(\Pi)$.

   - Case $t = \lambda y.r$. Then, $t\{x \setminus u\} = (\lambda y.r)\{x \setminus u\} = \lambda y.r\{x \setminus u\}$, since we may assume that $x \neq y$. Now, we must consider three cases, according to the last rule R of Φ:

– Case $R = \texttt{Abs}$. Then, Φ must be of the form

$$\frac{\Phi' \rhd_{\mathcal{N}} \Gamma; y : N \vdash r\{x \setminus u\} : A}{\Gamma \vdash \lambda y.r\{x \setminus u\} : N \to A}\ \texttt{Abs}$$

By applying the *i.h.* to r, there are $\Psi' \rhd_{\mathcal{N}} \Gamma; y : N; x : M \vdash r : A$ and $\Pi' \rhd_{\mathcal{N}} \emptyset \vdash u : M$, such that $\#_{\texttt{App}}(\Phi') = \#_{\texttt{App}}(\Psi') + \#_{\texttt{App}}(\Pi')$. Therefore, we can take $\Pi = \Pi'$, and Ψ to be

$$\frac{\Psi' \rhd_{\mathcal{N}} \Gamma; y : N; x : M \vdash r : A}{\Gamma; x : M \vdash \lambda y.r : N \to A}\ \texttt{Abs}$$

since $\#_{\texttt{App}}(\Phi) = \#_{\texttt{App}}(\Phi') = \#_{\texttt{App}}(\Psi') + \#_{\texttt{App}}(\Pi') = \#_{\texttt{App}}(\Psi) + \#_{\texttt{App}}(\Pi)$.

– Case $R = \texttt{Ans}$. Then, Φ must be of the form

$$\frac{}{\emptyset \vdash \lambda y.r\{x \setminus u\} : \star}\ \texttt{Ans}$$

$\Gamma = \emptyset$, and $A = \star$. Therefore, we can take $M = \{\!\{\ \}\!\}$, Ψ to be

$$\frac{}{\emptyset \vdash \lambda y.r : \star}\ \texttt{Ans}$$

and Π to be

$$\frac{}{\emptyset \vdash u : \{\!\{\ \}\!\}}\ \texttt{Many}$$

since $\#_{\texttt{App}}(\Phi) = 0 = \#_{\texttt{App}}(\Psi) + \#_{\texttt{App}}(\Pi)$.

– Case $R = \texttt{Many}$. Then, Φ must be of the form

$$\frac{(\Phi_i \rhd_{\mathcal{N}} \Gamma_i \vdash \lambda y.r\{x \setminus u\} : A_i)_{i \in \mathrm{I}}}{+_{i \in \mathrm{I}} \Gamma_i \vdash \lambda y.r\{x \setminus u\} : \{\!\{A_i\}\!\}_{i \in \mathrm{I}}}\ \texttt{Many}$$

$\Gamma = +_{i \in \mathrm{I}} \Gamma_i$, and $N = \{\!\{A_i\}\!\}_{i \in \mathrm{I}}$. The rest of the proof follows smoothly from applying the *i.h.* to each Φ_i and Lemma C.1.

- Case $t = r\,e$. Then, $t\{x \setminus u\} = (r\,e)\{x \setminus u\} = r\{x \setminus u\}\,e\{x \setminus u\}$. Now, we must consider two cases, according to the last rule R of Φ:

– Case $R = \texttt{App}$. Then, Φ must be of the form

$$\frac{\Phi' \rhd_{\mathcal{N}} \Delta \vdash r\{x \setminus u\} : N' \to A \qquad \Phi'' \rhd_{\mathcal{N}} \Sigma \vdash e\{x \setminus u\} : N'}{\Delta + \Sigma \vdash r\{x \setminus u\}\,e\{x \setminus u\} : A}\ \texttt{App}$$

$\Gamma = \Delta + \Sigma$. By applying the *i.h.* to r, there are $\Psi' \rhd_{\mathcal{N}} \Delta; x : F \vdash r : N' \to A$ and $\Pi' \rhd_{\mathcal{N}} \emptyset \vdash u : F$, such that $\#_{\texttt{App}}(\Phi') = \#_{\texttt{App}}(\Psi') + \#_{\texttt{App}}(\Pi')$. Moreover, by applying the ih to e, there are $\Psi'' \rhd_{\mathcal{N}} \Sigma; x : G \vdash e : N'$ and $\Pi'' \rhd_{\mathcal{N}} \emptyset \vdash u : G$, such that $\#_{\texttt{App}}(\Phi'') = \#_{\texttt{App}}(\Psi'') + \#_{\texttt{App}}(\Pi'')$. By Lemma C.1, there is a derivation $\Theta \rhd_{\mathcal{N}} \emptyset \vdash u : F \sqcup G$, such that $\#_{\texttt{App}}(\Theta) = \#_{\texttt{App}}(\Pi') + \#_{\texttt{App}}(\Pi'')$. Therefore, we can

take $\Pi = \Theta$, and Ψ to be

$$\dfrac{\Psi' \rhd_{\mathcal{N}} \Delta; x : F \vdash r : N' \to A \qquad \Psi'' \rhd_{\mathcal{N}} \Sigma; x : G \vdash e : N'}{(\Delta + \Sigma); x : F \sqcup G \vdash r\, e : A}\ \texttt{App}$$

since

$$\begin{aligned} \#_{\texttt{App}}(\Phi) &= \#_{\texttt{App}}(\Phi') + \#_{\texttt{App}}(\Phi'') + 1 \\ &= \#_{\texttt{App}}(\Psi') + \#_{\texttt{App}}(\Pi') + \#_{\texttt{App}}(\Psi'') + \#_{\texttt{App}}(\Pi'') + 1 \\ &= \#_{\texttt{App}}(\Psi) + \#_{\texttt{App}}(\Theta) \\ &= \#_{\texttt{App}}(\Psi) + \#_{\texttt{App}}(\Pi). \end{aligned}$$

– Case $R = \texttt{Many}$. Then, Φ must be of the form

$$\dfrac{(\Phi_i \rhd_{\mathcal{N}} \Gamma_i \vdash r\{x \setminus u\}\, e\{x \setminus u\} : A_i)_{i \in \mathtt{I}}}{+_{i \in \mathtt{I}} \Gamma_i \vdash r\{x \setminus u\}\, e\{x \setminus u\} : \{\!\{A_i\}\!\}_{i \in \mathtt{I}}}\ \texttt{Many}$$

$\Gamma = +_{i \in \mathtt{I}} \Gamma_i$, and $N = \{\!\{A_i\}\!\}_{i \in \mathtt{I}}$. The rest of the proof follows smoothly from applying the *i.h.* to each Φ_i and Lemma C.1.

2. The proof follows by induction on the structure of t. We omit the proof, since it is very similar to the one for CBN.

□

Lemma 4.15 (Weighted Subject Expansion for CBN and CBV)**.** *Let $t, u \in \Lambda_\circ$.*

1. *If $t \to_{\texttt{CBN}} u$ and $\Phi \rhd_{\mathcal{N}} \emptyset \vdash u : A$, then there is $\Psi \rhd_{\mathcal{N}} \emptyset \vdash t : A$, such that $\#_{\texttt{App}}(\Phi) + 1 = \#_{\texttt{App}}(\Psi)$.*
2. *If $t \to_{\texttt{CBV}} u$ and $\Phi \rhd_{\mathcal{V}} \emptyset \vdash u : M$, then there is $\Psi \rhd_{\mathcal{V}} \emptyset \vdash t : M$, such that $\#_{\texttt{App}}(\Phi) + 1 = \#_{\texttt{App}}(\Psi)$.*

Proof. The proofs follow by induction on the definition of the contexts of the reduction strategies:

1. Let $t = \mathtt{N}[t_0] \to_{\texttt{CBN}} \mathtt{N}[u_0] = u$, such that $t_0 \mapsto_\beta u_0$:

   - Case $\mathtt{N} = \Box$. Then, $t_0 = (\lambda x.r)\, e \mapsto_\beta r\{x \setminus e\} = u_0$, and $\Phi \rhd_{\mathcal{N}} \emptyset \vdash r\{x \setminus e\} : A$. Given the form of Φ, we can apply Lemma 4.11 to obtain derivations $\Psi' \rhd_{\mathcal{N}} x : M \vdash r : A$ and $\Psi'' \rhd_{\mathcal{N}} \emptyset \vdash e : M$, such that $\#_{\texttt{App}}(\Phi) = \#_{\texttt{App}}(\Psi') + \#_{\texttt{App}}(\Psi'')$. Therefore, we can take Ψ to be

   $$\dfrac{\dfrac{\Psi' \rhd_{\mathcal{N}} x : M \vdash r : A}{\emptyset \vdash \lambda x.r : M \to A}\ \texttt{Abs} \qquad \Psi'' \rhd_{\mathcal{N}} \emptyset \vdash e : M}{\emptyset \vdash (\lambda x.r)\, e : A}\ \texttt{App}$$

   since $\#_{\texttt{App}}(\Phi) + 1 = \#_{\texttt{App}}(\Psi') + \#_{\texttt{App}}(\Psi'') + 1 = \#_{\texttt{App}}(\Psi)$.

   - Case $\mathtt{N} = \mathtt{N}'\, r$. Then, $t = \mathtt{N}'[t_0]\, r \to_{\texttt{CBN}} \mathtt{N}'[u_0]\, r = u$, such that $\mathtt{N}'[t_0] \to_{\texttt{CBN}} \mathtt{N}'[u_0]$, and Φ must be of the form

   $$\dfrac{\Phi' \rhd_{\mathcal{N}} \emptyset \vdash \mathtt{N}'[u_0] : N \to A \qquad \Phi'' \rhd_{\mathcal{N}} \emptyset \vdash r : N}{\emptyset \vdash \mathtt{N}'[u_0]\, r : A}\ \texttt{App}$$

By applying the *i.h.* to $\mathtt{N}'[u_0]$, we know there is $\Psi' \triangleright_{\mathcal{N}} \emptyset \vdash \mathtt{N}'[t_0] : N \to A$, such that $\#_{\mathtt{App}}(\Phi') + 1 = \#_{\mathtt{App}}(\Psi')$. Therefore, we can take Ψ to be

$$\dfrac{\Psi' \triangleright_{\mathcal{N}} \emptyset \vdash \mathtt{N}'[t_0] : N \to A \qquad \Phi'' \triangleright_{\mathcal{N}} \emptyset \vdash r : N}{\emptyset \vdash \mathtt{N}'[t_0]\, r : A}\ \mathtt{App}$$

since $\#_{\mathtt{App}}(\Phi) + 1 = \#_{\mathtt{App}}(\Phi') + \#_{\mathtt{App}}(\Phi'') + 1 = \#_{\mathtt{App}}(\Psi') + \#_{\mathtt{App}}(\Phi'') = \#_{\mathtt{App}}(\Psi)$.

2. Let $t = \mathtt{V}[t_0] \to_{\mathtt{CBV}} \mathtt{V}[u_0] = u$, such that $t_0 \mapsto_{\beta_{\mathtt{v}}} u_0$:

   - Case $\mathtt{V} = \Box$. We omit the proof since it is very similar to the one for CBN.
   - Case $\mathtt{V} = \mathtt{V}'\,\Box$. We omit the proof since it is very similar to the one for CBN.
   - Case $\mathtt{V} = w\,\mathtt{V}'$. Then, $t = w\,\mathtt{V}'[t_0] \to_{\mathtt{CBV}} w\,\mathtt{V}'[u_0] = u$, such that $\mathtt{V}[t_0] \to_{\mathtt{CBV}} \mathtt{V}[u_0]$, and Φ must be of the form

   $$\dfrac{\Phi' \triangleright_{\mathcal{V}} \emptyset \vdash w : N \to M \qquad \Phi'' \triangleright_{\mathcal{V}} \emptyset \vdash \mathtt{V}'[u_0] : N}{\emptyset \vdash w\,\mathtt{V}'[u_0] : M}\ \mathtt{App}$$

   By applying the *i.h.* to $\mathtt{V}'[u_0]$, we know there is $\Psi' \triangleright_{\mathcal{V}} \emptyset \vdash \mathtt{V}'[t_0] : N$, such that $\#_{\mathtt{App}}(\Phi'') + 1 = \#_{\mathtt{App}}(\Psi')$. Therefore, we can take Ψ to be

   $$\dfrac{\Phi' \triangleright_{\mathcal{V}} \emptyset \vdash w : N \to M \qquad \Psi' \triangleright_{\mathcal{V}} \emptyset \vdash \mathtt{V}'[t_0] : N}{\emptyset \vdash w\,\mathtt{V}'[t_0] : M}\ \mathtt{App}$$

   since $\#_{\mathtt{App}}(\Phi) + 1 = \#_{\mathtt{App}}(\Phi') + \#_{\mathtt{App}}(\Phi'') + 1 = \#_{\mathtt{App}}(\Phi') + \#_{\mathtt{App}}(\Psi') = \#_{\mathtt{App}}(\Psi)$.

□

Lemma C.1 (CBN and CBV Multiset-Typings Split and Merge)**.** *Let $t \in \Lambda$ and $v \in \mathrm{V}$.*

1. *There is $\Phi \triangleright_{\mathcal{N}} \Gamma \vdash t : M$, such that $M = \sqcup_{i\in\mathtt{I}} M_i$ if and only if there are $\Phi_i \triangleright_{\mathcal{N}} \Gamma_i \vdash t : M_i$, such that $\Gamma = +_{i\in\mathtt{I}}\Gamma_i$, and $\#_{\mathtt{App}}(\Phi) = +_{i\in\mathtt{I}}\#_{\mathtt{App}}(\Phi_i)$.*
2. *There is $\Phi \triangleright_{\mathcal{V}} \Gamma \vdash v : M$, such that $M = \sqcup_{i\in\mathtt{I}} M_i$ if and only if there are $\Phi_i \triangleright_{\mathcal{V}} \Gamma_i \vdash v : M_i$, such that $\Gamma = +_{i\in\mathtt{I}}\Gamma_i$, and $\#_{\mathtt{App}}(\Phi) = +_{i\in\mathtt{I}}\#_{\mathtt{App}}(\Phi_i)$.*

Lemma C.2 (Contexts and Free Variables for CBN and CBV)**.** *If $\Phi \triangleright_{\mathcal{N}\,(\text{resp. }\mathcal{V})} \Gamma \vdash t : A$ (resp. M), then $\mathtt{dom}(\Gamma) \subseteq \mathtt{fv}(t)$.*

Lemma 4.17 (Typability of CBN- and CBV-Normal Forms)**.** *If $t \in \mathrm{V}_\circ$, then there is a derivation $\Phi \triangleright_{\mathcal{N}\,(\text{resp. }\mathcal{V})} \emptyset \vdash t : A$ (resp. M).*

Proof. We start by noticing that $t \in \mathrm{V}_\circ$ implies that t is a closed value v, *i.e.* an abstraction $\lambda x.u$. Therefore, we can take Φ to be

$$\dfrac{}{\emptyset \vdash \lambda x.u : \star}\ \mathtt{Ans}\left(\text{resp. }\dfrac{}{\emptyset \vdash \lambda x.u : \{\!\{\,\}\!\}}\ \mathtt{Abs}\right)$$

□

C.2 Subsuming Call-By-Name and Call-By-Value

Theorem 4.26 (Soundness and Completeness for CBPV)**.** *Let $t \in !\Lambda_{\circ}$. Then, t is $\mathcal{P}$-typable if and only if t $\mathtt{CBPV}$-normalizes to a term $u \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$. Moreover, if $\Phi \rhd_{\mathcal{P}} \emptyset \vdash t : A$ is a (not necessarily tight) type derivation, then $t \overset{n\cdot\beta_{\mathtt{v}}+m\cdot\beta_{!}}{\twoheadrightarrow_{\mathtt{CBPV}}} u$ and $\#_{\mathtt{App},\mathtt{Der}}(\Phi) \geq n \cdot \mathtt{App} + m \cdot \mathtt{Der}$.*

Proof. The soundness proof ($\Rightarrow$) is straightforward by Proposition 3.7 and Lemmas 4.24 and C.4. The completeness proof ($\Leftarrow$) is obtained by induction on the length of the $\mathtt{CBPV}$-normalizing sequence:

- Case $n + m = 0$. Then, since $u \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$, we know that u is $\mathcal{P}$-typable by Lemma 4.24.
- Case $n + m = k + 1$. Then, $t \overset{n-1\cdot\beta_{\mathtt{v}}+m\cdot\beta_{!}}{\rightarrow_{\mathtt{CBPV}}} r \rightarrow_{\mathtt{CBPV}} u$ or $t \overset{n\cdot\beta_{\mathtt{v}}+m-1\cdot\beta_{!}}{\rightarrow_{\mathtt{CBPV}}} r \rightarrow_{\mathtt{CBPV}} u$. Notice that, by Lemma 4.24, u is $\mathcal{P}$-typable. By applying the *i.h.* to $k = n + m - 1$, we know r is $\mathcal{P}$-typable. Therefore, t is $\mathcal{P}$-typable by Lemma C.6.

The "moreover" statement easily holds by the "weightedness" of the subject reduction and expansion statements. □

Lemma C.3 (Weighted Substitution for CBPV)**.** *Let $t \in !\Lambda$ and $v \in !\mathrm{V}$. If $\Phi \rhd_{\mathcal{P}} \Gamma; x : M \vdash t : A$ and $\Psi \rhd_{\mathcal{P}} \emptyset \vdash v : M$, then $\Pi \rhd_{\mathcal{P}} \Gamma \vdash t\{x \setminus v\} : A$, such that $\#_{\mathtt{App},\mathtt{Der}}(\Pi) = \#_{\mathtt{App},\mathtt{Der}}(\Phi) + \#_{\mathtt{App},\mathtt{Der}}(\Psi)$.*

Proof. The proof follows by induction on the structure of Φ. We proceed by cases, according to the last rule R in Φ:

- Case $R = \mathtt{Ax}$. Then, t is a variable, and we must consider the two following possibilities:
  - If $t = y \neq x$, then Φ must be of the form

$$\frac{}{y : N; x : \{\!\{\,\}\!\} \vdash y : N}\ \mathtt{Ax}$$

    $\Gamma = y : N$, $M = \{\!\{\,\}\!\}$, $A = N$. Moreover, $t\{x \setminus v\} = y\{x \setminus v\} = y$, $v = z$ or $v = !u$, and Ψ must be of the form

$$\frac{}{\emptyset \vdash z : \{\!\{\,\}\!\}}\ \mathtt{Ax} \quad \text{or} \quad \frac{}{\emptyset \vdash\ !u : \{\!\{\,\}\!\}}\ \mathtt{Bg}$$

    Therefore, we can take $\Pi = \Phi$, since $\#_{\mathtt{App},\mathtt{Der}}(\Psi) = 0 \cdot \mathtt{App} + 0 \cdot \mathtt{Der}$, and so $\#_{\mathtt{App},\mathtt{Der}}(\Pi) = \#_{\mathtt{App},\mathtt{Der}}(\Phi) + \#_{\mathtt{App},\mathtt{Der}}(\Psi)$.

– If $t = x$, then Φ must be of the form

$$\frac{}{x : M \vdash x : M}\ \mathtt{Ax}$$

$\Gamma = \emptyset$, and $A = M$. Moreover, $t\{x \setminus v\} = x\{x \setminus v\} = v$. Therefore, we can take $\Pi = \Psi$, since $\#_{\mathtt{App},\mathtt{Der}}(\Phi) = 0 \cdot \mathtt{App} + 0 \cdot \mathtt{Der}$, and thus $\#_{\mathtt{App},\mathtt{Der}}(\Pi) = \#_{\mathtt{App},\mathtt{Der}}(\Psi) = \#_{\mathtt{App},\mathtt{Der}}(\Psi) + \#_{\mathtt{App},\mathtt{Der}}(\Phi)$.

- Case $R = \mathtt{Abs}$. We omit this case because it is very similar to the corresponding case in Lemma 4.9.

- Case $R = \mathtt{Ans}$. We omit this case because it is very similar to the corresponding case in Lemma 4.9 for CBN.

- Case $R = \mathtt{App}$. We omit this case because it is very similar to the corresponding case in Lemma 4.9 (using Lemma C.7).

- Case $R = \mathtt{Bg}$. Then, $t = !u$, and Φ must be of the form

$$\frac{(\Phi_i \rhd_{\mathcal{P}} \Gamma_i; x : M_i \vdash u : A_i)_{i \in \mathtt{I}}}{+_{i \in \mathtt{I}} \Gamma_i; x : \sqcup_{i \in \mathtt{I}} M_i \vdash\ !u : \{\!\{A_i\}\!\}_{i \in \mathtt{I}}}\ \mathtt{Bg}$$

$\Gamma = +_{i \in \mathtt{I}} \Gamma_i$, $M = \sqcup_{i \in \mathtt{I}} M_i$, and $A = \{\!\{A_i\}\!\}_{i \in \mathtt{I}}$. Since $M = \sqcup_{i \in \mathtt{I}} M_i$, we can apply Lemma C.7 to Ψ to obtain derivations

$$\Psi_i \rhd_{\mathcal{P}} \emptyset \vdash v : M_i,$$

such that $\#_{\mathtt{App},\mathtt{Der}}(\Psi) = +_{i \in \mathtt{I}} \#_{\mathtt{App},\mathtt{Der}}(\Psi_i)$. By applying the *i.h.* to each Φ_i, we know there exist $\Pi_i \rhd_{\mathcal{P}} \Gamma_i \vdash u\{x \setminus v\} : A_i$, such that $\#_{\mathtt{App},\mathtt{Der}}(\Pi_i) = \#_{\mathtt{App},\mathtt{Der}}(\Phi_i) + \#_{\mathtt{App},\mathtt{Der}}(\Psi_i)$, for each $i \in \mathtt{I}$. It is easy to see that $u\{x \setminus v\}$ is a value. Therefore, we can take Π to be

$$\frac{(\Pi_i \rhd_{\mathcal{P}} \Gamma_i \vdash u\{x \setminus v\} : A_i)_{i \in \mathtt{I}}}{+_{i \in \mathtt{I}} \Gamma_i \vdash\ !(u\{x \setminus v\}) : \{\!\{A_i\}\!\}_{i \in \mathtt{I}}}\ \mathtt{Bg}$$

since $!(u\{x \setminus v\}) = !u\{x \setminus v\}$, and $\#_{\mathtt{App},\mathtt{Der}}(\Pi) = +_{i \in \mathtt{I}} \#_{\mathtt{App},\mathtt{Der}}(\Pi_i) = +_{i \in \mathtt{I}} (\#_{\mathtt{App},\mathtt{Der}}(\Phi_i) + \#_{\mathtt{App},\mathtt{Der}}(\Psi_i)) = \#_{\mathtt{App},\mathtt{Der}}(\Phi) + \#_{\mathtt{App},\mathtt{Der}}(\Psi)$.

- Case $R = \mathtt{Der}$. Then, $t = \mathtt{der}(u)$, $t\{x \setminus v\} = \mathtt{der}(u)\{x \setminus v\} = \mathtt{der}(u\{x \setminus v\})$, and Φ must be of the form

$$\frac{\Phi' \rhd_{\mathcal{P}} \Gamma; x : M \vdash u : \{\!\{A\}\!\}}{\Gamma; x : M \vdash \mathtt{der}(u) : A}\ \mathtt{Der}$$

By applying the *i.h.* to Φ', we know there exists $\Pi' \rhd_{\mathcal{P}} \Gamma \vdash u\{x \setminus v\} : \{\!\{A\}\!\}$, such that $\#_{\mathtt{App},\mathtt{Der}}(\Pi') = \#_{\mathtt{App},\mathtt{Der}}(\Phi') + \#_{\mathtt{App},\mathtt{Der}}(\Psi)$. Therefore, we can take Π to be

$$\frac{\Pi' \rhd_{\mathcal{P}} \Gamma \vdash u\{x \setminus v\} : \{\!\{A\}\!\}}{\Gamma \vdash \mathtt{der}(u\{x \setminus v\}) : A}\ \mathtt{Der}$$

since $\#_{\mathtt{App},\mathtt{Der}}(\Pi) = \#_{\mathtt{App},\mathtt{Der}}(\Pi') + 0 \cdot \mathtt{App} + 1 \cdot \mathtt{Der} = \#_{\mathtt{App},\mathtt{Der}}(\Phi') + \#_{\mathtt{App},\mathtt{Der}}(\Psi) + 0 \cdot \mathtt{App} + 1 \cdot \mathtt{Der} = \#_{\mathtt{App},\mathtt{Der}}(\Phi) + \#_{\mathtt{App},\mathtt{Der}}(\Psi)$.

□

Lemma C.4 (Weighted Subject Reduction for CBPV). *Let $t, u \in !\Lambda$. If $t \to_{\mathtt{CBPV}} u$ and $\Phi \rhd_{\mathcal{P}} \emptyset \vdash t : A$, then there is $\Psi \rhd_{\mathcal{P}} \emptyset \vdash u : A$, such that*

- $\#_{\mathtt{App},\mathtt{Der}}(\Phi) = \#_{\mathtt{App},\mathtt{Der}}(\Psi) + 1 \cdot \mathtt{App} + 0 \cdot \mathtt{Der}$, *if $t \to_{\mathtt{CBPV}(\beta_{\mathtt{v}})} u$.*
- $\#_{\mathtt{App},\mathtt{Der}}(\Phi) = \#_{\mathtt{App},\mathtt{Der}}(\Psi) + 0 \cdot \mathtt{App} + 1 \cdot \mathtt{Der}$, *if $t \to_{\mathtt{CBPV}(\beta_{!})} u$.*

Proof. The proof follows by induction on the definition of the contexts of the reduction strategy. Let $t = \mathtt{B}[t_0] \to_{\mathtt{CBPV}} \mathtt{B}[u_0] = u$, such that $t_0 \mapsto_{\beta_{\mathtt{v}}/\beta_{!}} u_0$:

- Case $\mathtt{B} = \Box$. Then, we must consider two possibilities:
  - If $t = (\lambda x.r)\, v \mapsto_{\beta_{\mathtt{v}}} r\{x \setminus v\} = u$, then Φ must be of the form
  $$\dfrac{\dfrac{\Phi' \rhd_{\mathcal{P}} x : M \vdash r : A}{\emptyset \vdash \lambda x.r : M \to A}\ \mathtt{Abs} \qquad \Phi'' \rhd_{\mathcal{P}} \emptyset \vdash v : M}{\emptyset \vdash (\lambda x.r)\, v : A}\ \mathtt{App}$$
  Given the forms of Φ' and Φ'', we can apply Lemma C.3 to obtain a derivation $\Pi \rhd_{\mathcal{P}} \emptyset \vdash r\{x \setminus v\} : A$, such that $\#_{\mathtt{App},\mathtt{Der}}(\Pi) = \#_{\mathtt{App},\mathtt{Der}}(\Phi') + \#_{\mathtt{App},\mathtt{Der}}(\Phi'')$. Therefore, we can take $\Psi = \Pi$, since $\#_{\mathtt{App},\mathtt{Der}}(\Phi) = \#_{\mathtt{App},\mathtt{Der}}(\Phi') + \#_{\mathtt{App},\mathtt{Der}}(\Phi'') + 1 \cdot \mathtt{App} + 0 \cdot \mathtt{Der} = \#_{\mathtt{App},\mathtt{Der}}(\Pi) + 1 \cdot \mathtt{App} + 0 \cdot \mathtt{Der} = \#_{\mathtt{App},\mathtt{Der}}(\Psi) + 1 \cdot \mathtt{App} + 0 \cdot \mathtt{Der}$.
  - If $t = \mathtt{der}(!r) \mapsto_{\beta_{!}} r = u$, then Φ must be of the form
  $$\dfrac{\dfrac{\Phi' \rhd_{\mathcal{P}} \emptyset \vdash r : A}{\emptyset \vdash\ !r : \{\!\{A\}\!\}}\ \mathtt{Bg}}{\emptyset \vdash \mathtt{der}(!r) : A}\ \mathtt{Der}$$
  Therefore, we can take $\Psi = \Phi'$, since we have $\#_{\mathtt{App},\mathtt{Der}}(\Phi) = \#_{\mathtt{App},\mathtt{Der}}(\Phi') + 0 \cdot \mathtt{App} + 1 \cdot \mathtt{Der} = \#_{\mathtt{App},\mathtt{Der}}(\Psi) + 0 \cdot \mathtt{App} + 1 \cdot \mathtt{Der}$.
- $\mathtt{B} = \mathtt{B}'\, r$. We omit this case because it is very similar to the corresponding one in Lemma 4.13.
- $\mathtt{B} = (\lambda x.r)\, \mathtt{B}'$. We omit this case because it is very similar to the corresponding one in Lemma 4.13.
- $\mathtt{B} = \mathtt{der}(\mathtt{B}')$. Then, $t = \mathtt{der}(\mathtt{B}'[t_0]) \to_{\mathtt{CBPV}} \mathtt{der}(\mathtt{B}'[u_0]) = u$, such that $\mathtt{B}'[t_0] \to_{\mathtt{CBPV}} \mathtt{B}'[u_0]$, and Φ must be of the form
$$\dfrac{\Phi' \rhd_{\mathcal{P}} \emptyset \vdash \mathtt{B}'[t_0] : \{\!\{A\}\!\}}{\emptyset \vdash \mathtt{der}(\mathtt{B}'[t_0]) : A}\ \mathtt{Der}$$

By applying the *i.h.* to $\mathtt{B}'[t_0]$, we know there is $\Psi' \rhd_{\mathcal{P}} \emptyset \vdash \mathtt{B}'[u_0] : \{\!\{A\}\!\}$, such that

- $\#_{\mathtt{App},\mathtt{Der}}(\Phi') = \#_{\mathtt{App},\mathtt{Der}}(\Psi') + 1 \cdot \mathtt{App} + 0 \cdot \mathtt{Der}$, if $\mathtt{B}'[t_0] \to_{\mathtt{CBPV}} \mathtt{B}'[u_0]$ is a $\beta_{\mathtt{v}}$-step.
- $\#_{\mathtt{App},\mathtt{Der}}(\Phi') = \#_{\mathtt{App},\mathtt{Der}}(\Psi') + 0 \cdot \mathtt{App} + 1 \cdot \mathtt{Der}$, if $\mathtt{B}'[t_0] \to_{\mathtt{CBPV}} \mathtt{B}'[u_0]$ is a $\beta_{!}$-step.

Therefore, we can take Ψ to be

$$\frac{\Psi' \rhd_{\mathcal{P}} \emptyset \vdash \mathtt{B}'[u_0] : \{\!\{A\}\!\}}{\emptyset \vdash \mathtt{der}(\mathtt{B}'[u_0]) : A}\ \mathtt{Der}$$

since

- if $\mathtt{B}'[t_0] \to_{\mathtt{CBPV}} \mathtt{B}'[u_0]$ is a $\beta_{\mathtt{v}}$-step, $\mathtt{der}(\mathtt{B}'[t_0]) \to_{\mathtt{CBPV}(\beta_{\mathtt{v}})} \mathtt{der}(\mathtt{B}'[u_0])$, and thus $\#_{\mathtt{App},\mathtt{Der}}(\Phi) = \#_{\mathtt{App},\mathtt{Der}}(\Phi') + 0 \cdot \mathtt{App} + 1 \cdot \mathtt{Der} = \#_{\mathtt{App},\mathtt{Der}}(\Psi') + 1 \cdot \mathtt{App} + 1 \cdot \mathtt{Der} = \#_{\mathtt{App},\mathtt{Der}}(\Psi) + 1 \cdot \mathtt{App} + 0 \cdot \mathtt{Der}$.
- if $\mathtt{B}'[t_0] \to_{\mathtt{CBPV}} \mathtt{B}'[u_0]$ is a $\beta_{!}$-step, then $\mathtt{der}(\mathtt{B}'[t_0]) \to_{\mathtt{CBPV}(\beta_{!})} \mathtt{der}(\mathtt{B}'[u_0])$, and thus have that $\#_{\mathtt{App},\mathtt{Der}}(\Phi) = \#_{\mathtt{App},\mathtt{Der}}(\Phi') + 0 \cdot \mathtt{App} + 1 \cdot \mathtt{Der} = \#_{\mathtt{App},\mathtt{Der}}(\Psi') + 0 \cdot \mathtt{App} + 2 \cdot \mathtt{Der} = \#_{\mathtt{App},\mathtt{Der}}(\Psi) + 0 \cdot \mathtt{App} + 1 \cdot \mathtt{Der}$.

□

Lemma C.5 (Weighted Anti-Substitution for CBPV). *Let $t \in !\Lambda$, and $v \in !\mathrm{V}_{\circ}$. If $\Phi \rhd_{\mathcal{P}} \Gamma \vdash t\{x \setminus v\} : A$, then $\Psi \rhd_{\mathcal{P}} \Gamma; x : M \vdash t : A$ and $\Pi \rhd_{\mathcal{P}} \emptyset \vdash v : M$, such that $\#_{\mathtt{App},\mathtt{Der}}(\Phi) = \#_{\mathtt{App},\mathtt{Der}}(\Psi) + \#_{\mathtt{App},\mathtt{Der}}(\Pi)$.*

Proof. The proof follows by induction on the structure of t.

- Case $t = y \neq x$. Then, $t\{x \setminus v\} = y\{x \setminus v\} = y$. Moreover, $v = !u$, such that $u \in !\Lambda_{\circ}$. Therefore, we can take $M = \{\!\{\,\}\!\}$, $\Psi = \Phi$, and Π to be

$$\frac{}{\emptyset \vdash !u : \{\!\{\,\}\!\}}\ \mathtt{Bg}$$

since $\#_{\mathtt{App},\mathtt{Der}}(\Phi) = \#_{\mathtt{App},\mathtt{Der}}(\Psi) = \#_{\mathtt{App},\mathtt{Der}}(\Psi) + \#_{\mathtt{App},\mathtt{Der}}(\Pi)$.

- Case $t = x$. Then, $t\{x \setminus v\} = v$. Moreover, $v = !u$, such that $u \in !\Lambda_{\circ}$. Therefore, $\Phi \rhd_{\mathcal{P}} \emptyset \vdash !u : N$, where $\Gamma = \emptyset$ (by Lemma C.8), and $A = N$. Therefore, we can take Ψ to be

$$\frac{}{x : N \vdash x : N}\ \mathtt{Ax}$$

and $\Pi = \Phi$, since $\#_{\mathtt{App},\mathtt{Der}}(\Phi) = 0 \cdot \mathtt{App} + 0 \cdot \mathtt{Der} + \#_{\mathtt{App},\mathtt{Der}}(\Pi) = \#_{\mathtt{App},\mathtt{Der}}(\Psi) + \#_{\mathtt{App},\mathtt{Der}}(\Pi)$.

- Case $t = \lambda x.u$. We omit this case because it is very similar to the corresponding one in Lemma 4.11.

- Case $t = u\,r$. We omit this case because it is very similar to the corresponding one in Lemma 4.11 (using Lemma C.7).

- Case $t = !u$. Then, $t\{x \setminus v\} = !u\{x \setminus v\} = !(u\{x \setminus v\})$. Therefore, Φ must be of the form

$$\frac{(\Phi_i \rhd_{\mathcal{P}} \Gamma_i \vdash u\{x \setminus v\} : A_i)_{i\in \mathtt{I}}}{+_{i\in \mathtt{I}}\Gamma_i \vdash !(u\{x \setminus v\}) : \{\!\{A_i\}\!\}_{i\in \mathtt{I}}}\ \mathtt{Bg}$$

  and $A = \{\!\{A_i\}\!\}_{i\in \mathtt{I}}$. By applying the *i.h.* to each Φ_i, there are $\Psi_i \rhd_{\mathcal{P}} \Gamma_i; x : M_i \vdash u : A_i$ and $\Pi_i \rhd_{\mathcal{P}} \emptyset \vdash v : M_i$, such that $\#_{\mathtt{App},\mathtt{Der}}(\Phi_i) = \#_{\mathtt{App},\mathtt{Der}}(\Psi_i) + \#_{\mathtt{App},\mathtt{Der}}(\Pi_i)$. By Lemma C.7, there is $\Pi' \rhd_{\mathcal{P}} \emptyset \vdash v : \sqcup_{i\in \mathtt{I}} M_i$, such that $\#_{\mathtt{App},\mathtt{Der}}(\Pi') = +_{i\in \mathtt{I}}\#_{\mathtt{App},\mathtt{Der}}(\Pi_i)$. Therefore, we can take $M = \sqcup_{i\in \mathtt{I}} M_i$, $\Pi = \Pi'$, and Ψ to be

$$\frac{(\Psi_i \rhd_{\mathcal{P}} \Gamma_i; x : M_i \vdash u : A_i)_{i\in \mathtt{I}}}{+_{i\in \mathtt{I}}\Gamma_i; x : \sqcup_{i\in \mathtt{I}} M_i \vdash !u : \{\!\{A_i\}\!\}_{i\in \mathtt{I}}}\ \mathtt{Bg}$$

  since we have

$$\begin{aligned}\#_{\mathtt{App},\mathtt{Der}}(\Phi) &= +_{i\in \mathtt{I}}\#_{\mathtt{App},\mathtt{Der}}(\Phi_i)\\ &= +_{i\in \mathtt{I}}\#_{\mathtt{App},\mathtt{Der}}(\Psi_i) +_{i\in \mathtt{I}} \#_{\mathtt{App},\mathtt{Der}}(\Pi_i)\\ &= \#_{\mathtt{App},\mathtt{Der}}(\Psi) + \#_{\mathtt{App},\mathtt{Der}}(\Pi).\end{aligned}$$

- Case $t = \mathtt{der}(u)$. Then, $t\{x \setminus v\} = \mathtt{der}(u)\{x \setminus v\} = \mathtt{der}(u\{x \setminus v\})$. Therefore, Φ must be of the form

$$\frac{\Phi' \rhd_{\mathcal{P}} \Gamma \vdash u\{x \setminus v\} : \{\!\{A\}\!\}}{\Gamma \vdash \mathtt{der}(u\{x \setminus v\}) : A}\ \mathtt{Der}$$

  By applying the *i.h.* to Φ', there are $\Psi' \rhd_{\mathcal{P}} \Gamma; x : M \vdash u : \{\!\{A\}\!\}$ and $\Pi' \rhd_{\mathcal{P}} \emptyset \vdash v : M$, such that $\#_{\mathtt{App},\mathtt{Der}}(\Phi') = \#_{\mathtt{App},\mathtt{Der}}(\Psi') + \#_{\mathtt{App},\mathtt{Der}}(\Pi')$. Therefore, we can take $\Pi = \Pi'$, and Ψ to be

$$\frac{\Psi' \rhd_{\mathcal{P}} \Gamma; x : M \vdash u : \{\!\{A\}\!\}}{\Gamma; x : M \vdash \mathtt{der}(u) : A}\ \mathtt{Der}$$

  since $\#_{\mathtt{App},\mathtt{Der}}(\Phi) = \#_{\mathtt{App},\mathtt{Der}}(\Phi') + 0 \cdot \mathtt{App} + 1 \cdot \mathtt{Der} = \#_{\mathtt{App},\mathtt{Der}}(\Psi') + \#_{\mathtt{App},\mathtt{Der}}(\Pi') + 0 \cdot \mathtt{App} + 1 \cdot \mathtt{Der} = \#_{\mathtt{App},\mathtt{Der}}(\Psi) + \#_{\mathtt{App},\mathtt{Der}}(\Pi)$.

□

Lemma C.6 (Weighted Subject Expansion for CBPV). *Let $t, u \in !\Lambda_{\circ}$. If $t \to_{\mathtt{CBPV}} u$ and $\Phi \rhd_{\mathcal{P}} \emptyset \vdash u : A$, then there is $\Psi \rhd_{\mathcal{P}} \emptyset \vdash t : A$, such that*

- $\#_{\mathtt{App},\mathtt{Der}}(\Phi) + 1 \cdot \mathtt{App} + 0 \cdot \mathtt{Der} = \#_{\mathtt{App},\mathtt{Der}}(\Psi)$, *if* $t \to_{\mathtt{CBPV}(\beta_{\mathtt{v}})} u$.
- $\#_{\mathtt{App},\mathtt{Der}}(\Phi) + 0 \cdot \mathtt{App} + 1 \cdot \mathtt{Der} = \#_{\mathtt{App},\mathtt{Der}}(\Psi)$, *if* $t \to_{\mathtt{CBPV}(\beta_{!})} u$.

Proof. The proofs follow by induction on the definition of the contexts of the reduction strategy. Let $t = \mathtt{B}[t_0] \to_{\mathtt{CBPV}} \mathtt{B}[u_0] = u$, such that $t_0 \mapsto_{\beta_{\mathtt{v}}/\beta_{!}} u_0$:

- Case $\mathtt{B} = \Box$. Then, we must consider two possibilities:
  - If $t = (\lambda x.r)\, v \mapsto_{\beta_v} r\{x \setminus v\} = u$, then $\Phi \rhd_{\mathcal{P}} \emptyset \vdash r\{x \setminus v\} : A$. Given the form of Φ, we can apply Lemma C.5 to obtain derivations $\Psi' \rhd_{\mathcal{P}} x : M \vdash r : A$ and $\Psi'' \rhd_{\mathcal{P}} \emptyset \vdash v : M$, such that $\#_{\mathtt{App},\mathtt{Der}}(\Phi) = \#_{\mathtt{App},\mathtt{Der}}(\Psi') + \#_{\mathtt{App},\mathtt{Der}}(\Psi'')$. Therefore, we can take Ψ to be
$$\dfrac{\dfrac{\Psi' \rhd_{\mathcal{P}} x : M \vdash r : A}{\emptyset \vdash \lambda x.r : M \to A}\,\mathtt{Abs} \qquad \Psi'' \rhd_{\mathcal{P}} \emptyset \vdash v : M}{\emptyset \vdash (\lambda x.r)\, v : A}\,\mathtt{App}$$
    since $\#_{\mathtt{App},\mathtt{Der}}(\Phi) + 1 \cdot \mathtt{App} + 0 \cdot \mathtt{Der} = \#_{\mathtt{App},\mathtt{Der}}(\Psi') + \#_{\mathtt{App},\mathtt{Der}}(\Psi'') + 1 \cdot \mathtt{App} + 0 \cdot \mathtt{Der} = \#_{\mathtt{App},\mathtt{Der}}(\Psi)$.
  - If $t = \mathtt{der}(!r) \mapsto_{\beta_!} r = u$, then we can take Ψ to be
$$\dfrac{\dfrac{\Phi \rhd_{\mathcal{P}} \emptyset \vdash r : A}{\emptyset \vdash\, !r : \{\!\{A\}\!\}}\,\mathtt{Bg}}{\emptyset \vdash \mathtt{der}(!r) : A}\,\mathtt{Der}$$
    since we have $\#_{\mathtt{App},\mathtt{Der}}(\Phi) + 0 \cdot \mathtt{App} + 1 \cdot \mathtt{Der} = \#_{\mathtt{App},\mathtt{Der}}(\Psi)$.
- $\mathtt{B} = \mathtt{B}'\, r$. We omit this case because it is very similar to the corresponding one in Lemma 4.15.
- $\mathtt{B} = (\lambda x.r)\, \mathtt{B}'$. We omit this case because it is very similar to the corresponding one in Lemma 4.15.
- $\mathtt{B} = \mathtt{der}(\mathtt{B}')$. Then, $t = \mathtt{der}(\mathtt{B}'[t_0]) \to_{\mathtt{CBPV}} \mathtt{der}(\mathtt{B}'[u_0]) = u$, such that $\mathtt{B}'[t_0] \to_{\mathtt{CBPV}} \mathtt{B}'[u_0]$, and Φ must be of the form
$$\dfrac{\Phi' \rhd_{\mathcal{P}} \emptyset \vdash \mathtt{B}'[u_0] : \{\!\{A\}\!\}}{\emptyset \vdash \mathtt{der}(\mathtt{B}'[u_0]) : A}\,\mathtt{Der}$$
  By applying the *i.h.* to $\mathtt{B}'[u_0]$, we know there is $\Psi' \rhd_{\mathcal{P}} \emptyset \vdash \mathtt{B}'[t_0] : \{\!\{A\}\!\}$, such that
  - $\#_{\mathtt{App},\mathtt{Der}}(\Phi') + 1 \cdot \mathtt{App} + 0 \cdot \mathtt{Der} = \#_{\mathtt{App},\mathtt{Der}}(\Psi')$, if $\mathtt{B}'[t_0] \to_{\mathtt{CBPV}} \mathtt{B}'[u_0]$ is a β_v-step.
  - $\#_{\mathtt{App},\mathtt{Der}}(\Phi') + 0 \cdot \mathtt{App} + 1 \cdot \mathtt{Der} = \#_{\mathtt{App},\mathtt{Der}}(\Psi')$, if $\mathtt{B}'[t_0] \to_{\mathtt{CBPV}} \mathtt{B}'[u_0]$ is a $\beta_!$-step.

  Therefore, we can take Ψ to be
$$\dfrac{\Psi' \rhd_{\mathcal{P}} \emptyset \vdash \mathtt{B}'[t_0] : \{\!\{A\}\!\}}{\emptyset \vdash \mathtt{der}(\mathtt{B}'[t_0]) : A}\,\mathtt{Der}$$
  since
  - if $\mathtt{B}'[t_0] \to_{\mathtt{CBPV}} \mathtt{B}'[u_0]$ is a β_v-step, then $\mathtt{der}(\mathtt{B}'[t_0]) \to_{\mathtt{CBPV}(\beta_v)} \mathtt{der}(\mathtt{B}'[u_0])$, and thus $\#_{\mathtt{App},\mathtt{Der}}(\Phi) + 1 \cdot \mathtt{App} + 0 \cdot \mathtt{Der} = \#_{\mathtt{App},\mathtt{Der}}(\Phi') + 1 \cdot \mathtt{App} + 1 \cdot \mathtt{Der} = \#_{\mathtt{App},\mathtt{Der}}(\Psi') + 0 \cdot \mathtt{App} + 1 \cdot \mathtt{Der} = \#_{\mathtt{App},\mathtt{Der}}(\Psi)$.

– if $\mathtt{B}'[t_0] \to_{\mathtt{CBPV}} \mathtt{B}'[u_0]$ is a $\beta_!$-step, then $\mathtt{der}(\mathtt{B}'[t_0]) \to_{\mathtt{CBPV}(\beta_!)} \mathtt{der}(\mathtt{B}'[u_0])$, and thus $\#_{\mathtt{App},\mathtt{Der}}(\Phi) + 0 \cdot \mathtt{App} + 1 \cdot \mathtt{Der} = \#_{\mathtt{App},\mathtt{Der}}(\Phi') + 0 \cdot \mathtt{App} + 2 \cdot \mathtt{Der} = \#_{\mathtt{App},\mathtt{Der}}(\Psi') + 0 \cdot \mathtt{App} + 1 \cdot \mathtt{Der} = \#_{\mathtt{App},\mathtt{Der}}(\Psi)$.

□

Lemma C.7 (CBPV Multiset-Typings Split and Merge). *Let $t \in !\Lambda$. Then, there is $\Phi \rhd_{\mathcal{P}} \Gamma \vdash t : M$, such that $M = \sqcup_{i \in \mathtt{I}} M_i$ if and only if there are $\Phi_i \rhd_{\mathcal{P}} \Gamma_i \vdash t : M_i$, such that $\Gamma = +_{i \in \mathtt{I}} \Gamma_i$, and $\#_{\mathtt{App},\mathtt{Der}}(\Phi) = +_{i \in \mathtt{I}} \#_{\mathtt{App},\mathtt{Der}}(\Phi_i)$.*

Lemma C.8 (Contexts and Free Variables for CBPV). *If $\Phi \rhd_{\mathcal{P}} \Gamma \vdash t : A$, then $\mathtt{dom}(\Gamma) \subseteq \mathtt{fv}(t)$.*

Lemma 4.22 (Typability of Clash-Free CBPV-Normal Forms). *If $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$, then there is a derivation $\Phi \rhd_{\mathcal{P}} \emptyset \vdash t : A$.*

Proof. By Proposition 3.11, since $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$, we know that t is either a closed bang-term $!u$ with $u \in !\Lambda_\circ$, or a closed abstraction $\lambda x.u \in !\Lambda_\circ$:

- Case $t = !u$, then we can take Φ to be

$$\frac{}{\emptyset \vdash !u : \{\!\{\,\}\!\}}\ \mathtt{Bg}$$

- Case $t = \lambda x.u$, then we can take Φ to be

$$\frac{}{\emptyset \vdash \lambda x.u : \star}\ \mathtt{Ans}$$

□

Lemma 4.23 (Typability Implies Clash-Freeness). *Let $t \in !\Lambda_\circ$. If $\Phi \rhd_{\mathcal{P}} \emptyset \vdash t : A$, then t is clash-free.*

Proof. Let us assume, towards a contradiction, that t is *not* clash-free. Then, there is a CBPV context $\mathtt{B}$ and a clash term u, such that $t \twoheadrightarrow_{\mathtt{CBPV}} \mathtt{B}[u]$. Then, by Lemma C.4, there is a derivation $\Psi \rhd_{\mathcal{P}} \emptyset \vdash \mathtt{B}[u] : A$. Therefore, if we show that $\mathtt{B}[u]$ is not $\mathcal{P}$-typable, we are done. We do this by induction on the definition of $\mathtt{B}$:

- Case $\mathtt{B} = \Box$. Then, we can easily see that, for every possible form of the clash u, there is a mismatch between its syntactical form and the typing rules of system $\mathcal{P}$:

  – If $u = v\,r$, then v would need to have an arrow type to justify rule App. However, by the definition of $\mathcal{P}$, closed values can only receive multi-types, never arrow types. Hence u is not typable.

  - If $u = \mathtt{der}(\lambda x.r)$, then $\lambda x.r$ would need to have a multi-type to justify rule `Der`. However, by the definition of $\mathcal{P}$, abstractions can only be typed using `Ans` or `Abs`, none of which assign multi-types.
  - If $u = r\,(\lambda x.e)$, then $\lambda x.e$ would need to have a multi-type to justify rule `App` However, by the definition of $\mathcal{P}$, abstractions only be typed using `Ans` or `Abs`, none of which assign multi-types.

- Case $\mathtt{B} = \mathtt{B}'\,r$. We omit this case because it follows smoothly from the *i.h.*, since all typing rules of $\mathcal{P}$ are syntax-directed and compositional, and untypability of the hole propagates to the enclosing context.

- Case $\mathtt{B} = (\lambda x.r)\,\mathtt{B}'$. We omit this case because it follows smoothly from the *i.h.*, for the same reason.

- Case $\mathtt{B} = \mathtt{der}(\mathtt{B}')$. We omit this case because it follows smoothly from the *i.h.*, for the same reason.

□

Lemma 4.24 (Typability Characterizes Clash-Free CBPV-Normal Forms)**.** *Let $t \in !\Lambda_\circ$. Then, $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$ if and only if $t \in \mathtt{NO}_{\mathtt{CBPV}}$ and t is $\mathcal{P}$-typable.*

Proof. $(\Rightarrow)$ This direction follows directly from Proposition 3.11 and Lemma 4.22.

$(\Leftarrow)$ Since $t \in \mathtt{NO}_{\mathtt{CBPV}}$, t is a CBPV-normal form. Moreover, since t is $\mathcal{P}$-typable, it is clash-free by Lemma 4.23. Hence, t is a clash-free CBPV-normal form, and therefore $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$ by Proposition 3.11.

□

Theorem 4.27 (Soundness and Completeness of Translations)**.** *Let $t \in \Lambda$. Then*

1. *$\Phi \rhd_{\mathcal{N}} \Gamma \vdash t : A$ if and only if $\Psi \rhd_{\mathcal{P}} \Gamma \vdash (t)^{\mathtt{CBN}} : A$. Moreover, $\#_{\mathtt{App}}(\Phi) = \#_{\mathtt{App}}(\Psi)$.*
2. *$\Phi \rhd_{\mathcal{V}} \Gamma \vdash t : M$ if and only if $\Psi \rhd_{\mathcal{P}} \Gamma \vdash (t)^{\mathtt{CBV}} : M$. Moreover, $\#_{\mathtt{App}}(\Phi) = \#_{\mathtt{App}}(\Psi)$.*

Proof. 1. $(\Rightarrow)$ By induction on the structure of t:

  - Case $t = x$. Then, $(x)^{\mathtt{CBN}} = \mathtt{der}(x)$. Moreover, Φ must be of the form

$$\frac{}{x : \{\!\{A\}\!\} \vdash x : A}\ \mathtt{Ax}$$

and $\Gamma = x : \{\!\{A\}\!\}$. Therefore, we can take Ψ to be of the form

$$\dfrac{\dfrac{}{x : \{\!\{A\}\!\} \vdash x : \{\!\{A\}\!\}}\ \mathtt{Ax}}{x : \{\!\{A\}\!\} \vdash \mathtt{der}(x) : A}\ \mathtt{Der}$$

since $\#_{\mathtt{App}}(\Phi) = 0 = \#_{\mathtt{App}}(\Psi)$.

– Case $t = \lambda x.u$. Then, $(\lambda x.u)^{\mathtt{CBN}} = \lambda x.(u)^{\mathtt{CBN}}$. Now, we must consider two possibilities, according to the the last rule R of Φ:

  * Case $R = \mathtt{Abs}$, then Φ must be of the form

$$\dfrac{\Phi' \rhd_{\mathcal{N}} \Gamma; x : M \vdash u : B}{\Gamma \vdash \lambda x.u : M \to B}\ \mathtt{Abs}$$

  and $A = M \to B$. By applying the *i.h.* to u, we know there is $\Psi' \rhd_{\mathcal{P}} \Gamma; x : M \vdash (u)^{\mathtt{CBN}} : B$, such that $\#_{\mathtt{App}}(\Phi') = \#_{\mathtt{App}}(\Psi')$. Therefore, we can take Ψ to be

$$\dfrac{\Psi' \rhd_{\mathcal{P}} \Gamma; x : M \vdash (u)^{\mathtt{CBN}} : B}{\Gamma \vdash \lambda x.(u)^{\mathtt{CBN}} : M \to B}\ \mathtt{Abs}$$

  since $\#_{\mathtt{App}}(\Phi) = \#_{\mathtt{App}}(\Phi') = \#_{\mathtt{App}}(\Psi') = \#_{\mathtt{App}}(\Psi)$.

  * Case $R = \mathtt{Ans}$, then Φ must be of the form

$$\dfrac{}{\emptyset \vdash \lambda x.u : \star}\ \mathtt{Ans}$$

  $\Gamma = \emptyset$, and $A = \star$. Therefore, we can take Ψ to be

$$\dfrac{}{\emptyset \vdash \lambda x.(u)^{\mathtt{CBN}} : \star}\ \mathtt{Ans}$$

  since $\#_{\mathtt{App}}(\Phi) = 0 = \#_{\mathtt{App}}(\Psi)$.

– Case $t = u\,r$. Then, $(u\,r)^{\mathtt{CBN}} = (u)^{\mathtt{CBN}}\,!(r)^{\mathtt{CBN}}$. Moreover, Φ must be of the form

$$\dfrac{\Phi' \rhd_{\mathcal{N}} \Gamma' \vdash u : \{\!\{A_i\}\!\}_{i\in \mathtt{I}} \to A \qquad \dfrac{(\Phi_i \rhd_{\mathcal{N}} \Gamma_i \vdash r : A_i)_{i\in\mathtt{I}}}{+_{i\in\mathtt{I}}\Gamma_i \vdash r : \{\!\{A_i\}\!\}_{i\in\mathtt{I}}}\ \mathtt{Many}}{\Gamma' + +_{i\in\mathtt{I}}\Gamma_i \vdash u\,r : A}\ \mathtt{App}$$

  and $\Gamma = \Gamma' + +_{i\in\mathtt{I}}\Gamma_i$. By applying the *i.h.* to u, we know there exists $\Psi' \rhd_{\mathcal{P}} \Gamma' \vdash (u)^{\mathtt{CBN}} : \{\!\{A_i\}\!\}_{i\in\mathtt{I}} \to A$, such that $\#_{\mathtt{App}}(\Phi') = \#_{\mathtt{App}}(\Psi')$. By applying the *i.h.* to each occurrence of r, we know there exist $\Psi_i \rhd_{\mathcal{P}} \Gamma_i \vdash (r)^{\mathtt{CBN}} : A_i$, such that $\#_{\mathtt{App}}(\Phi_i) = \#_{\mathtt{App}}(\Psi_i)$, for each $i \in \mathtt{I}$. Therefore, we can take Ψ to be

$$\dfrac{\Psi' \rhd_{\mathcal{P}} \Gamma' \vdash (u)^{\mathtt{CBN}} : \{\!\{A_i\}\!\}_{i\in\mathtt{I}} \to A \qquad \dfrac{(\Psi_i \rhd_{\mathcal{P}} \Gamma_i \vdash (r)^{\mathtt{CBN}} : A_i)_{i\in\mathtt{I}}}{+_{i\in\mathtt{I}}\Gamma_i \vdash\, !(r)^{\mathtt{CBN}} : \{\!\{A_i\}\!\}_{i\in\mathtt{I}}}\ \mathtt{Bg}}{\Gamma' +_{i\in\mathtt{I}} \Gamma_i \vdash (u)^{\mathtt{CBN}}\,!(r)^{\mathtt{CBN}} : A}\ \mathtt{App}$$

since $\#_{\mathtt{App}}(\Phi) = \#_{\mathtt{App}}(\Phi') +_{i\in\mathtt{I}} \#_{\mathtt{App}}(\Phi_i) + 1 = \#_{\mathtt{App}}(\Psi') +_{i\in\mathtt{I}} \#_{\mathtt{App}}(\Psi_i) + 1 = \#_{\mathtt{App}}(\Psi)$.

($\Leftarrow$) By induction on the structure of t:

- Case $t = x$. Notice that $(x)^{\mathtt{CBN}} = \mathtt{der}(x)$, and Ψ must be of the form

$$\dfrac{\dfrac{}{x : \{\!\{A\}\!\} \vdash x : \{\!\{A\}\!\}}\ \mathtt{Ax}}{x : \{\!\{A\}\!\} \vdash \mathtt{der}(x) : A}\ \mathtt{Der}$$

  Therefore, we can take Φ to be

$$\dfrac{}{x : \{\!\{A\}\!\} \vdash x : A}\ \mathtt{Ax}$$

  since $\#_{\mathtt{App}}(\Phi) = 0 = \#_{\mathtt{App}}(\Psi)$.

- Case $t = \lambda x.u$. Notice that $(\lambda x.u)^{\mathtt{CBN}} = \lambda x.(u)^{\mathtt{CBN}}$. Now, we must consider two possibilities, according to the last rule R of Ψ:

  * Case $R = \mathtt{Abs}$. Then, Ψ must be of the form

$$\dfrac{\Psi' \rhd_{\mathcal{P}} \Gamma; x : M \vdash (u)^{\mathtt{CBN}} : B}{\Gamma \vdash \lambda x.(u)^{\mathtt{CBN}} : M \to B}\ \mathtt{Abs}$$

    By applying the *i.h.* to u, we know there exists $\Phi' \rhd_{\mathcal{N}} \Gamma; x : M \vdash u : B$, such that $\#_{\mathtt{App}}(\Phi') = \#_{\mathtt{App}}(\Psi')$. Therefore, we can take Φ to be

$$\dfrac{\Phi' \rhd_{\mathcal{N}} \Gamma; x : M \vdash u : B}{\Gamma \vdash \lambda x.u : M \to B}\ \mathtt{Abs}$$

    since $\#_{\mathtt{App}}(\Phi) = \#_{\mathtt{App}}(\Phi') = \#_{\mathtt{App}}(\Psi') = \#_{\mathtt{App}}(\Psi)$.

  * Case $R = \mathtt{Ans}$. Then, Ψ must be of the form

$$\dfrac{}{\varnothing \vdash \lambda x.(u)^{\mathtt{CBN}} : \star}\ \mathtt{Ans}$$

    $\Gamma = \varnothing$, and $A = \star$. Therefore, we can take Φ to be

$$\dfrac{}{\varnothing \vdash \lambda x.u : \star}\ \mathtt{Ans}$$

    since $\#_{\mathtt{App}}(\Phi) = 0 = \#_{\mathtt{App}}(\Psi)$.

- Case $t = u\,r$. Notice that $(u\,r)^{\mathtt{CBN}} = (u)^{\mathtt{CBN}}\,!(r)^{\mathtt{CBN}}$. Then, Ψ must be of the form

$$\dfrac{\Psi' \rhd_{\mathcal{P}} \Gamma' \vdash (u)^{\mathtt{CBN}} : M \to A \qquad \Psi'' \rhd_{\mathcal{P}} \Gamma'' \vdash\, !(r)^{\mathtt{CBN}} : M}{\Gamma' + \Gamma'' \vdash (u)^{\mathtt{CBN}}\,!(r)^{\mathtt{CBN}} : A}\ \mathtt{App}$$

  $\Gamma = \Gamma' + \Gamma''$. By applying the *i.h.* to u, we know there exists $\Phi' \rhd_{\mathcal{N}} \Gamma' \vdash u : M \to A$, such that $\#_{\mathtt{App}}(\Phi') = \#_{\mathtt{App}}(\Psi')$. Moreover, Ψ'' must be of the form

$$\dfrac{(\Psi_i \rhd_{\mathcal{P}} \Gamma_i \vdash (r)^{\mathtt{CBN}} : A_i)_{i \in \mathtt{I}}}{+_{i \in \mathtt{I}} \Gamma_i \vdash\, !(r)^{\mathtt{CBN}} : \{\!\{A_i\}\!\}_{i \in \mathtt{I}}}\ \mathtt{Bg}$$

$\Gamma'' = +_{i\in\mathtt{I}}\Gamma_i$, and $\#_{\mathtt{App}}(\Psi'') = +_{i\in\mathtt{I}}\#_{\mathtt{App}}(\Psi_i)$. By applying the *i.h.* to each occurrence of r, we know there exist $\Phi_i \rhd_{\mathcal{N}} \Gamma_i \vdash r : A_i$, such that $\#_{\mathtt{App}}(\Phi_i) = \#_{\mathtt{App}}(\Psi_i)$, for each $i \in \mathtt{I}$. Therefore, we can take Φ to be

$$\dfrac{\Phi' \rhd_{\mathcal{N}} \Gamma' \vdash u : \{\!\{A_i\}\!\}_{i\in\mathtt{I}} \to A \qquad \dfrac{(\Phi_i \rhd_{\mathcal{N}} \Gamma_i \vdash r : A_i)_{i\in\mathtt{I}}}{+_{i\in\mathtt{I}}\Gamma_i \vdash r : \{\!\{A_i\}\!\}_{i\in\mathtt{I}}}\ \mathtt{Many}}{\Gamma' + +_{i\in\mathtt{I}}\Gamma_i \vdash u\, r : A}\ \mathtt{App}$$

since $\#_{\mathtt{App}}(\Phi) = \#_{\mathtt{App}}(\Phi') + +_{i\in\mathtt{I}}\#_{\mathtt{App}}(\Phi_i) + 1 = \#_{\mathtt{App}}(\Psi') + +_{i\in\mathtt{I}}\#_{\mathtt{App}}(\Psi_i) + 1 = \#_{\mathtt{App}}(\Psi') + \#_{\mathtt{App}}(\Psi'') + 1 = \#_{\mathtt{App}}(\Psi)$.

2. ($\Rightarrow$) By induction on the structure of t:

   – Case $t = x$. Then, $(x)^{\mathtt{CBV}} = x$. Moreover, Φ must be of the form

   $$\dfrac{}{x : M \vdash x : M}\ \mathtt{Ax}$$

   and $\Gamma = x : M$. Therefore, we can take Ψ to be

   $$\dfrac{}{x : M \vdash x : M}\ \mathtt{Ax}$$

   since $\#_{\mathtt{App}}(\Phi) = 0 = \#_{\mathtt{App}}(\Psi)$.

   – Case $t = \lambda x.u$. Then, $(\lambda x.u)^{\mathtt{CBV}} = !\lambda x.(u)^{\mathtt{CBV}}$. Moreover, Φ must be of the form

   $$\dfrac{(\,\Phi_i \rhd_{\mathcal{V}} \Gamma_i; x : M_i \vdash u : N_i)_{i\in\mathtt{I}}}{+_{i\in\mathtt{I}}\,\Gamma_i \vdash \lambda x.u : \{\!\{M_i \to N_i\}\!\}_{i\in\mathtt{I}}}\ \mathtt{Abs}$$

   $\Gamma = +_{i\in\mathtt{I}}\Gamma_i$, and $M = \{\!\{M_i \to N_i\}\!\}_{i\in\mathtt{I}}$. By applying the *i.h.* to each occurrence of u, there are $\Psi_i \rhd_{\mathcal{P}} \Gamma_i; x : M_i \vdash (u)^{\mathtt{CBV}} : N_i$, such that $\#_{\mathtt{App}}(\Phi_i) = \#_{\mathtt{App}}(\Psi_i)$, for each $i \in \mathtt{I}$. Therefore, we can take Ψ to be

   $$\dfrac{\left(\dfrac{\Psi_i \rhd_{\mathcal{P}} \Gamma_i; x : M_i \vdash (u)^{\mathtt{CBV}} : N_i}{\Gamma_i \vdash \lambda x.(u)^{\mathtt{CBV}} : M_i \to N_i}\ \mathtt{Abs}\right)_{i\in\mathtt{I}}}{+_{i\in\mathtt{I}}\,\Gamma_i \vdash !\lambda x.(u)^{\mathtt{CBV}} : \{\!\{M_i \to N_i\}\!\}_{i\in\mathtt{I}}}\ \mathtt{Bg}$$

   since $\#_{\mathtt{App}}(\Phi) = +_{i\in\mathtt{I}}\#_{\mathtt{App}}(\Phi_i) = +_{i\in\mathtt{I}}\#_{\mathtt{App}}(\Psi_i) = \#_{\mathtt{App}}(\Psi)$.

   – Case $t = u\,r$. Then, we have that $(u\,r)^{\mathtt{CBV}} = \mathtt{der}((u)^{\mathtt{CBV}})\,(r)^{\mathtt{CBV}}$. Moreover, Φ must be of the form

   $$\dfrac{\Phi' \rhd_{\mathcal{V}} \Gamma' \vdash u : \{\!\{N \to M\}\!\} \qquad \Phi'' \rhd_{\mathcal{V}} \Gamma'' \vdash r : N}{\Gamma' + \Gamma'' \vdash u\, r : M}\ \mathtt{App}$$

   and $\Gamma = \Gamma' + \Gamma''$. By applying the *i.h.* to u, we know there exists a derivation $\Psi' \rhd_{\mathcal{P}} \Gamma' \vdash (u)^{\mathtt{CBV}} : \{\!\{N \to M\}\!\}$, such that $\#_{\mathtt{App}}(\Phi') = \#_{\mathtt{App}}(\Psi')$. By applying the *i.h.* to r, we know there is $\Psi'' \rhd_{\mathcal{P}} \Gamma'' \vdash (r)^{\mathtt{CBV}} : N$, such that $\#_{\mathtt{App}}(\Phi'') = \#_{\mathtt{App}}(\Psi'')$.

Therefore, we can take Ψ to be

$$\dfrac{\dfrac{\Psi' \rhd_{\mathcal{P}} \Gamma' \vdash (u)^{\mathtt{CBV}} : \{\!\{N \to M\}\!\}}{\Gamma' \vdash \mathtt{der}((u)^{\mathtt{CBV}}) : N \to M}\ \mathtt{Der} \qquad \Psi'' \rhd_{\mathcal{P}} \Gamma'' \vdash (r)^{\mathtt{CBV}} : N}{\Gamma' + \Gamma'' \vdash \mathtt{der}((u)^{\mathtt{CBV}})\,(r)^{\mathtt{CBV}} : M}\ \mathtt{App}$$

since $\#_{\mathtt{App}}(\Phi) = \#_{\mathtt{App}}(\Phi') + \#_{\mathtt{App}}(\Phi'') + 1 = \#_{\mathtt{App}}(\Psi') + \#_{\mathtt{App}}(\Psi'') + 1 = \#_{\mathtt{App}}(\Psi)$.

($\Leftarrow$) By induction on the structure of t:

- Case $t = x$. Notice that $(x)^{\mathtt{CBV}} = x$, and Ψ must be of the form

$$\dfrac{}{x : M \vdash x : M}\ \mathtt{Ax}$$

Therefore, we can take Φ to be

$$\dfrac{}{x : M \vdash x : M}\ \mathtt{Ax}$$

since $\#_{\mathtt{App}}(\Phi) = 0 = \#_{\mathtt{App}}(\Psi)$.

- Case $t = \lambda x.u$. Notice that $(\lambda x.u)^{\mathtt{CBV}} = !\lambda x.(u)^{\mathtt{CBV}}$. Then, Ψ must be of the form

$$\dfrac{\left(\dfrac{\Psi_i \rhd_{\mathcal{P}} \Gamma_i; x : N_i \vdash (u)^{\mathtt{CBV}} : M_i}{\Gamma_i \vdash \lambda x.(u)^{\mathtt{CBV}} : N_i \to M_i}\ \mathtt{Abs}\right)_{i \in \mathtt{I}}}{+_{i \in \mathtt{I}}\, \Gamma_i \vdash\ !\lambda x.(u)^{\mathtt{CBV}} : \{\!\{N_i \to M_i\}\!\}_{i \in \mathtt{I}}}\ \mathtt{Bg}$$

By applying the *i.h.* to each occurrence of u, we know there exists $\Phi_i \rhd_{\mathcal{V}} \Gamma_i; x : N_i \vdash u : M_i$, such that $\#_{\mathtt{App}}(\Phi_i) = \#_{\mathtt{App}}(\Psi_i)$, for each $i \in \mathtt{I}$. Therefore, we can take Φ to be

$$\dfrac{(\Phi_i \rhd_{\mathcal{V}} \Gamma_i; x : N_i \vdash u : M_i)_{i \in \mathtt{I}}}{+_{i \in \mathtt{I}} \Gamma_i \vdash \lambda x.u : \{\!\{N_i \to M_i\}\!\}_{i \in \mathtt{I}}}\ \mathtt{Abs}$$

since $\#_{\mathtt{App}}(\Phi) = +_{i \in \mathtt{I}} \#_{\mathtt{App}}(\Phi_i) = +_{i \in \mathtt{I}} \#_{\mathtt{App}}(\Psi_i) = \#_{\mathtt{App}}(\Psi)$.

- Case $t = u\,r$. Notice that $(u\,r)^{\mathtt{CBV}} = \mathtt{der}((u)^{\mathtt{CBV}})\,(r)^{\mathtt{CBV}}$. Then, Ψ must be of the form

$$\dfrac{\dfrac{\Psi' \rhd_{\mathcal{P}} \Gamma' \vdash (u)^{\mathtt{CBV}} : \{\!\{N' \to M\}\!\}}{\Gamma' \vdash \mathtt{der}((u)^{\mathtt{CBV}}) : N' \to M}\ \mathtt{Der} \qquad \Psi'' \rhd_{\mathcal{P}} \Gamma'' \vdash (r)^{\mathtt{CBV}} : N'}{\Gamma' + \Gamma'' \vdash \mathtt{der}((u)^{\mathtt{CBV}})\,(r)^{\mathtt{CBV}} : M}\ \mathtt{App}$$

By applying the *i.h.* to u, we know there is $\Phi' \rhd_{\mathcal{V}} \Gamma' \vdash u : N \to M$, such that $\#_{\mathtt{App}}(\Phi') = \#_{\mathtt{App}}(\Psi')$. By applying the *i.h.* to r, we know there is $\Phi'' \rhd_{\mathcal{V}} \Gamma'' \vdash r : N$, such that $\#_{\mathtt{App}}(\Phi'') = \#_{\mathtt{App}}(\Psi'')$. Therefore, we can take Φ to be

$$\dfrac{\Phi' \rhd_{\mathcal{V}} \Gamma' \vdash u : N \to M \qquad \Phi'' \rhd_{\mathcal{V}} \Gamma'' \vdash r : N}{\Gamma' + \Gamma'' \vdash u\,r : M}\ \mathtt{App}$$

since $\#_{\mathtt{App}}(\Phi) = \#_{\mathtt{App}}(\Phi') + \#_{\mathtt{App}}(\Phi'') + 1 = \#_{\mathtt{App}}(\Psi') + \#_{\mathtt{App}}(\Psi'') + 1 = \#_{\mathtt{App}}(\Psi)$.

□

Appendix D

A Quantitative Study of Pattern-Matching

D.1 The Pattern-Matching Calculus

D.1.1 Syntax and Operational Semantics

Lemma 6.8 (One-Step Diamond). *The reduction relation $\to_{\mathtt{WH}}$ enjoys the one-step diamond property.*

Proof. Let $s_i \in \{\mathtt{dB}, \mathtt{sub}, \mathtt{match}, \mathtt{branch}\}$. We show that, if $t \to_{\mathtt{WH}(s_1)} t_1$ and $t \to_{\mathtt{WH}(s_2)} t_2$, with $t_1 \neq t_2$, there exists t_3 such that $t_1 \to_{\mathtt{WH}(s_2)} t_3$ and $t_2 \to_{\mathtt{WH}(s_1)} t_3$. All reduction steps are assumed to satisfy the freshness conditions required by the definition of $\mathtt{WH}$, which are preserved by contextual closure. The proof follows by induction on t:

- Case $t = x$. We can conclude immediately, since $x \not\to_{\mathtt{WH}}$.
- Case $t = us$. We have to consider four possible diverging cases:
  - Let $u = \mathtt{L}_1(\!|\mathtt{L}_2(\!|\lambda q.r|\!)[x\backslash v]|\!)$, $s_1 = \mathtt{dB}$ and $t_1 = \mathtt{L}_1(\!|\mathtt{L}_2(\!|r[q\backslash s]|\!)[x\backslash v]|\!)$, such that $\mathtt{bv}(\mathtt{L}_2[x\backslash v]\mathtt{L}_1) \cap \mathtt{fv}(s) = \emptyset$ (so in particular $x \notin \mathtt{fv}(s)$), and $s_2 = \mathtt{sub}$ and $t_2 = \mathtt{L}_1(\!|\mathtt{L}_2(\!|\lambda q.r|\!)\{x \setminus v\}|\!)s$. We can take $t_3 = \mathtt{L}_1(\!|\mathtt{L}_2(\!|r[q\backslash s]|\!)\{x \setminus v\}|\!)$ and close the diagram:

$$\begin{array}{ccc} \mathtt{L}_1(\!|\mathtt{L}_2(\!|\lambda q.r|\!)[x\backslash v]|\!)s & \xrightarrow{\mathtt{WH(dB)}} & \mathtt{L}_1(\!|\mathtt{L}_2(\!|r[q\backslash s]|\!)[x\backslash v]|\!) \\ \downarrow{\scriptstyle \mathtt{WH(sub)}} & & \downarrow{\scriptstyle \mathtt{WH(sub)}} \\ \mathtt{L}_1(\!|\mathtt{L}_2(\!|\lambda q.r|\!)\{x \setminus v\}|\!)s & & \mathtt{L}_1(\!|\mathtt{L}_2(\!|r[q\backslash s]|\!)\{x \setminus v\}|\!) \\ = & \xrightarrow{\mathtt{WH(dB)}} & = \\ \mathtt{L}_1(\!|\mathtt{L}_2\{x \setminus v\}(\!|\lambda q.r\{x \setminus v\}|\!)|\!)s & & \mathtt{L}_1(\!|\mathtt{L}_2\{x \setminus v\}(\!|r\{x \setminus v\}[q\backslash s\{x \setminus v\}]|\!)|\!) \end{array}$$

  We can conclude since $x \notin \mathtt{fv}(s)$, hence $s\{x \setminus v\} = s$.

– Let

$$u = \mathtt{L}_1(\!|\mathtt{L}_2(\!|\lambda q.r|\!)[\mathtt{c}(p_i)_n \backslash \mathtt{L}_3(\!|\mathtt{c}(v_i)_n|\!)]|\!|),$$
$$s_1 = \mathtt{dB},$$
$$\text{and } t_1 = \mathtt{L}_1(\!|\mathtt{L}_2(\!|r[q\backslash s]|\!)[\mathtt{c}(p_i)_n \backslash \mathtt{L}_3(\!|\mathtt{c}(v_i)_n|\!)]|\!|),$$

such that $\mathtt{bv}(\mathtt{L}_2[\mathtt{c}(p_i)_n \backslash \mathtt{L}_3(\!|\mathtt{c}(v_i)_n|\!)]\mathtt{L}_1) \cap \mathtt{fv}(s) = \emptyset$, and $s_2 = \mathtt{match}$, and also we have $t_2 = \mathtt{L}_1(\!|\mathtt{L}_3(\!|\mathtt{L}_2(\!|\lambda q.r|\!)[p_1\backslash v_1]\cdots[p_n\backslash v_n]|\!)|\!|)$, such that $\mathtt{bv}(\mathtt{L}_3) \cap \mathtt{fv}(\mathtt{L}_2(\!|\lambda q.r|\!)) = \emptyset$. We can take $t_3 = \mathtt{L}_1(\!|\mathtt{L}_3(\!|\mathtt{L}_2(\!|r[q\backslash s]|\!)[p_1\backslash v_1]\cdots[p_n\backslash v_n]|\!)|\!|)$ and close the diagram as follows:

$$\begin{array}{ccc}
\mathtt{L}_1(\!|\mathtt{L}_2(\!|\lambda q.r|\!)[\mathtt{c}(p_i)_n \backslash \mathtt{L}_3(\!|\mathtt{c}(v_i)_n|\!)]|\!|)s & \xrightarrow{\mathtt{WH}(\mathtt{dB})} & \mathtt{L}_1(\!|\mathtt{L}_2(\!|r[q\backslash s]|\!)[\mathtt{c}(p_i)_n \backslash \mathtt{L}_3(\!|\mathtt{c}(v_i)_n|\!)]|\!|) \\
\downarrow{\scriptstyle \mathtt{WH}(\mathtt{match})} & & \downarrow{\scriptstyle \mathtt{WH}(\mathtt{match})} \\
\mathtt{L}_1(\!|\mathtt{L}_3(\!|\mathtt{L}_2(\!|\lambda q.r|\!)[p_1\backslash v_1]\cdots[p_n\backslash v_n]|\!)|\!|)s & \xrightarrow[\mathtt{WH}(\mathtt{dB})]{} & \mathtt{L}_1(\!|\mathtt{L}_3(\!|\mathtt{L}_2(\!|r[q\backslash s]|\!)[p_1\backslash v_1]\cdots[p_n\backslash v_n]|\!)|\!|)
\end{array}$$

The remaining cases, where both reductions occur strictly inside the same subterm, follow directly by the *i.h.* and are therefore omitted.

- Case $t = u[x\backslash s]$. Then, $t_1 = u\{x \setminus s\}$, $s_1 = \mathtt{sub}$, and $t_2 = u'[x\backslash s]$, such that $u \to_{\mathtt{WH}(s_2)} u'$. We can pick $t_3 = u'\{x \setminus s\}$ and close the diagram using Lemma D.2:

$$\begin{array}{ccc}
u[x\backslash s] & \xrightarrow{\mathtt{WH}(\mathtt{sub})} & u\{x \setminus s\} \\
\downarrow{\scriptstyle \mathtt{WH}(s_2)} & & \downarrow{\scriptstyle \mathtt{WH}(s_2)} \\
u'[x\backslash s] & \xrightarrow[\mathtt{WH}(\mathtt{sub})]{} & u'\{x \setminus s\}
\end{array}$$

- Case $t = u[\hat{p}\backslash s]$. We have to consider four possible diverging cases:

  – Let $\hat{p} = \mathtt{c}(p_i)_n$ and $s = \mathtt{L}_1(\!|\mathtt{L}_2(\!|\mathtt{c}(s_i)_n|\!)[\mathtt{c}'(q_i)_m \backslash \mathtt{L}_3(\!|\mathtt{c}'(r_i)_m|\!)]|\!|)$. Then, we know we have that $t_1 = u[\mathtt{c}(p_i)_n \backslash \mathtt{L}_1(\!|\mathtt{L}_3(\!|\mathtt{L}_2(\!|\mathtt{c}(s_i)_n|\!)[q_1\backslash r_1]\cdots[q_m\backslash r_m]|\!)|\!|)]$ and $s_1 = \mathtt{match}$, such that $\mathtt{bv}(\mathtt{L}_3) \cap \mathtt{fv}(\mathtt{L}_2(\!|\mathtt{c}(s_i)_n|\!)) = \emptyset$, and $t_2 = \mathtt{L}_1(\!|\mathtt{L}_2(\!|u[p_1\backslash s_1]\cdots[p_n\backslash s_n]|\!)[\mathtt{c}'(q_i)_m \backslash \mathtt{L}_3(\!|\mathtt{c}'(r_i)_m|\!)]|\!|)$, and $s_2 = \mathtt{match}$, such that $\mathtt{bv}(\mathtt{L}_2[\mathtt{c}'(q_i)_m \backslash \mathtt{L}_3(\!|\mathtt{c}'(r_i)_m|\!)]) \cap \mathtt{fv}(u) = \emptyset$. We can pick the following term $t_3 = \mathtt{L}_1(\!|\mathtt{L}_3(\!|\mathtt{L}_2(\!|u[p_1\backslash s_1]\cdots[p_n\backslash s_n]|\!)[q_1\backslash r_1]\cdots[q_m\backslash r_m]|\!)|\!|)$ and close the diagram:

$$\begin{array}{ccc}
u[\mathtt{c}(p_i)_n \backslash \mathtt{L}_1(\!|\mathtt{L}_2(\!|\mathtt{c}(s_i)_n|\!)[\mathtt{c}'(q_i)_m \backslash \mathtt{L}_3(\!|\mathtt{c}'(r_i)_m|\!)]|\!|)] & \xrightarrow{\mathtt{WH}(\mathtt{match})} & u[\mathtt{c}(p_i)_n \backslash \mathtt{L}_1(\!|\mathtt{L}_3(\!|\mathtt{L}_2(\!|\mathtt{c}(s_i)_n|\!)[q_1\backslash r_1]\cdots[q_m\backslash r_m]|\!)|\!|)] \\
\downarrow{\scriptstyle \mathtt{WH}(\mathtt{match})} & & \downarrow{\scriptstyle \mathtt{WH}(\mathtt{match})} \\
\mathtt{L}_1(\!|\mathtt{L}_2(\!|u[p_1\backslash s_1]\cdots[p_n\backslash s_n]|\!)[\mathtt{c}'(q_i)_m \backslash \mathtt{L}_3(\!|\mathtt{c}'(r_i)_m|\!)]|\!|) & \xrightarrow[\mathtt{WH}(\mathtt{match})]{} & \mathtt{L}_1(\!|\mathtt{L}_3(\!|\mathtt{L}_2(\!|u[p_1\backslash s_1]\cdots[p_n\backslash s_n]|\!)[q_1\backslash r_1]\cdots[q_m\backslash r_m]|\!)|\!|)
\end{array}$$

  – Let $t_1 = u_1[\hat{p}\backslash s]$, such that $u \to_{\mathtt{WH}(s_1)} u_1$, $t_2 = u_2[\hat{p}\backslash s]$, such that $u \to_{\mathtt{WH}(s_2)} u_2$. By the *i.h.*, we know there exists u_3, such that $u_1 \to_{\mathtt{WH}(s_2)} u_3$ and $u_2 \to_{\mathtt{WH}(s_1)} u_3$. Therefore, we can

take $t_3 = u_3[\hat{p}\backslash s]$ and close the diagram:

$$\begin{array}{ccc} u[\hat{p}\backslash s] & \xrightarrow{\mathtt{WH}(s_1)} & u_1[\hat{p}\backslash s] \\ {\scriptstyle \mathtt{WH}(s_2)}\downarrow & & \downarrow{\scriptstyle \mathtt{WH}(s_2)} \\ u_2[\hat{p}\backslash s] & \xrightarrow[\mathtt{WH}(s_1)]{} & u_3[\hat{p}\backslash s] \end{array}$$

– Let $t_1 = u[\hat{p}\backslash r_1]$, such that $s \to_{\mathtt{WH}(s_1)} r_1$, and $t_2 = u[\hat{p}\backslash r_2]$, such that $s \to_{\mathtt{WH}(s_2)} r_2$. By the *i.h.*, we know there exists r_3, such that $r_1 \to_{\mathtt{WH}(s_2)} r_3$ and $r_2 \to_{\mathtt{WH}(s_1)} r_3$. Therefore, we can take $t_3 = u[\hat{p}\backslash r_3]$ and close the diagram:

$$\begin{array}{ccc} u[\hat{p}\backslash s] & \xrightarrow{\mathtt{WH}(s_1)} & u[\hat{p}\backslash r_1] \\ {\scriptstyle \mathtt{WH}(s_2)}\downarrow & & \downarrow{\scriptstyle \mathtt{WH}(s_2)} \\ u[\hat{p}\backslash r_2] & \xrightarrow[\mathtt{WH}(s_1)]{} & u[\hat{p}\backslash r_3] \end{array}$$

– Let $t_1 = u'[\hat{p}\backslash s]$, such that $u \to_{\mathtt{WH}(s_1)} u'$, and $t_2 = u[\hat{p}\backslash s']$, such that $s \to_{\mathtt{WH}(s_2)} s'$. We can pick $t_3 = u'[\hat{p}\backslash s']$ and close the diagram:

$$\begin{array}{ccc} u[\hat{p}\backslash s] & \xrightarrow{\mathtt{WH}(s_1)} & u'[\hat{p}\backslash s] \\ {\scriptstyle \mathtt{WH}(s_2)}\downarrow & & \downarrow{\scriptstyle \mathtt{WH}(s_2)} \\ u[\hat{p}\backslash s] & \xrightarrow[\mathtt{WH}(s_1)]{} & u'[\hat{p}\backslash s'] \end{array}$$

- Case $t = \mathtt{case}\ u\ \mathtt{of}\ B$. We have to consider two diverging cases:

– Let $u = \mathtt{L}_1(\!|\mathtt{L}_2(\!|\mathtt{c}(s_i)_m|\!)[\mathtt{c}'(p_i)_n\backslash\mathtt{L}_3(\!|\mathtt{c}'(r_i)_n|\!)]|\!)$ and $B = (\ldots, \mathtt{c}(q_i)_m.v, \ldots)$. Then, $t_1 = \mathtt{case}\ \mathtt{L}_1(\!|\mathtt{L}_3(\!|\mathtt{L}_2(\!|\mathtt{c}(s_i)_m|\!)[p_1\backslash r_1]\cdots[p_n\backslash r_n]|\!)|\!)\ \mathtt{of}\ B$ and $s_1 = \mathtt{match}$, such that $\mathtt{bv}(\mathtt{L}_3) \cap \mathtt{fv}(\mathtt{L}_2(\!|\mathtt{c}(s_i)_m|\!)) = \emptyset$, and $t_2 = \mathtt{L}_1(\!|\mathtt{L}_2(\!|v[q_1\backslash s_1]\cdots[q_m\backslash s_m]|\!)[\mathtt{c}'(p_i)_n\backslash\mathtt{L}_3(\!|\mathtt{c}'(r_i)_n|\!)]|\!)$ and $s_2 = \mathtt{branch}$, such that $\mathtt{bv}(\mathtt{L}_2[\mathtt{c}'(p_i)_n\backslash\mathtt{L}_3(\!|\mathtt{c}'(r_i)_n|\!)]\mathtt{L}_1) \cap \mathtt{fv}(v) = \emptyset$. We can pick the following term $t_3 = \mathtt{L}_1(\!|\mathtt{L}_3(\!|\mathtt{L}_2(\!|v[q_1\backslash v_1]\cdots[q_m\backslash v_m]|\!)[p_1\backslash r_1]\cdots[p_n\backslash r_n]|\!)|\!)$ and close the diagram:

$$\begin{array}{ccc} \mathtt{case}\ \mathtt{L}_1(\!|\mathtt{L}_2(\!|\mathtt{c}(s_i)_m|\!)[\mathtt{c}'(p_i)_n\backslash\mathtt{L}_3(\!|\mathtt{c}'(r_i)_n|\!)]|\!)\ \mathtt{of}\ B & \xrightarrow{\mathtt{WH}(\mathtt{match})} & \mathtt{case}\ \mathtt{L}_1(\!|\mathtt{L}_3(\!|\mathtt{L}_2(\!|\mathtt{c}(s_i)_m|\!)[p_1\backslash r_1]\cdots[p_n\backslash r_n]|\!)|\!)\ \mathtt{of}\ B \\ {\scriptstyle \mathtt{WH}(\mathtt{branch})}\downarrow & & \downarrow{\scriptstyle \mathtt{WH}(\mathtt{branch})} \\ \mathtt{L}_1(\!|\mathtt{L}_2(\!|v[q_1\backslash s_1]\cdots[q_m\backslash s_m]|\!)[\mathtt{c}'(p_i)_n\backslash\mathtt{L}_3(\!|\mathtt{c}'(r_i)_n|\!)]|\!) & \xrightarrow[\mathtt{WH}(\mathtt{match})]{} & \mathtt{L}_1(\!|\mathtt{L}_3(\!|\mathtt{L}_2(\!|v[q_1\backslash v_1]\cdots[q_m\backslash v_m]|\!)[p_1\backslash r_1]\cdots[p_n\backslash r_n]|\!)|\!) \end{array}$$

– Let $t_1 = \mathtt{case}\ u_1\ \mathtt{of}\ B$, such that $u \to_{\mathtt{WH}(s_1)} u_1$, $t_2 = \mathtt{case}\ u_2\ \mathtt{of}\ B$, such that $u \to_{\mathtt{WH}(s_2)} u_2$. By the *i.h.*, we know there exists u_3, such that $u_1 \to_{\mathtt{WH}(s_2)} u_3$ and $u_2 \to_{\mathtt{WH}(s_1)} u_3$. Therefore, we can take $t_3 = \mathtt{case}\ u_3\ \mathtt{of}\ B$ and close the diagram:

$$\begin{array}{ccc} \mathtt{case}\ u\ \mathtt{of}\ B & \xrightarrow{\mathtt{WH}(s_1)} & \mathtt{case}\ u_1\ \mathtt{of}\ B \\ {\scriptstyle \mathtt{WH}(s_2)}\downarrow & & \downarrow{\scriptstyle \mathtt{WH}(s_2)} \\ \mathtt{case}\ u_2\ \mathtt{of}\ B & \xrightarrow[\mathtt{WH}(s_1)]{} & \mathtt{case}\ u_3\ \mathtt{of}\ B \end{array}$$

□

Corollary 6.9 (Confluence). *The reduction relation $\to_{\mathtt{WH}}$ is confluent.*

Proof. Let us first prove that $\to_{\mathtt{WH}}$ is *subcommutative* (Def.1.1.8.(iv) of [105]). Take any $t, u, r \in \lambda_{PM}$ such that $u \;{}_{\mathtt{WH}}\!\leftarrow t \to_{\mathtt{WH}} r$. If $u = r$, then we can immediately conclude. Otherwise, $u \neq r$, and we can conclude by Lemma 6.8. Moreover, it is well-known that subcommutativity implies confluence (Prop.1.1.10 of [105]), so we can immediately conclude that $\to_{\mathtt{WH}}$ is confluent. □

Proposition 6.11 (Determinism). *The reduction relation $\to_{\mathtt{PM}}$ is deterministic.*

Proof. Reduction relation $\to_{\mathtt{PM}}$ is deterministic if, for any term t, either $t \not\to_{\mathtt{PM}}$ or there exists exactly one t', such that $t \to_{\mathtt{PM}} t'$. We show this by induction on the structure of t. Note that in all cases, the premises of the reduction rules are mutually exclusive, hence at most one rule can apply at any term:

- Cases $t = x$, $t = \lambda p.u$, and $t = \mathtt{c}T$. No rule applies to t, so we can conclude.
- Case $t = us$. Let us assume that $t \to_{\mathtt{PM}} t'$. Rules `dB` and `appL` may apply to t. However, either u is of the form $\mathtt{L}(\!|\lambda p.r|\!)$ and only rule `dB` applies, or $\neg\mathtt{isabs}(u)$ and only rule `appL` applies. Let us assume that only rule `dB` applies. Then, determinism follows immediately. Now, let us assume that only rule `appL` applies. Then, $us \to_{\mathtt{PM}} u's$, such that $u \to_{\mathtt{PM}} u'$. We can conclude by the *i.h.*.
- Case $t = u[p \backslash s]$. Let us assume that $t \to_{\mathtt{PM}} t'$. Rules `match`, `sub`, `esL`, and `esR` may apply to t. However, either p is a variable and only rule `sub` applies, or p is of the form $\mathtt{c}(q_i)_n$ and only rules `match`, `esL` and `esR` may apply. If rule `sub` applies, then we can trivially conclude. Otherwise, either $\mathtt{isdata_c}(s)$ or $\neg\mathtt{isdata_c}(s)$. If $\mathtt{isdata_c}(s)$ holds, then only rule `match` applies, and determinism follows immediately. If $\neg\mathtt{isdata_c}(s)$ holds, then either $s \not\to_{\mathtt{PM}}$ or $s \to_{\mathtt{PM}} s'$. Indeed, if $s \not\to_{\mathtt{PM}}$ then only rule `esL` applies. In which case $u \to_{\mathtt{PM}} u'$, and we can conclude by the *i.h.* over u. If $s \to_{\mathtt{PM}} s'$, then only rule `esR` applies, and we can conclude by the *i.h.* over s.
- Case $t = \mathtt{case}\ u\ \mathtt{of}\ B$. Let us assume $t \to_{\mathtt{PM}} t'$. Rules `branch` and `caseArg` may apply to t. However, either $u \not\to_{\mathtt{PM}}$ and only rule `branch` applies, or $u \to_{\mathtt{PM}} u'$ and only rule `caseArg` applies. If rule `branch` applies, then we can conclude immediately. If rule `caseArg` applies, then $u \to_{\mathtt{PM}} u'$, and we can conclude by the *i.h.* over u.

□

Lemma D.1 (Substitution Lemma). *Let $s, u, r \in \lambda_{PM}$ and $y \notin \mathtt{fv}(u)$. Then, $s\{x \setminus u\}\{y \setminus r\{x \setminus u\}\} = s\{y \setminus r\}\{x \setminus u\}$.*

Proof. We proceed by induction on the structure of s:

- Case $s = x$. Then

$$x\{x \setminus u\}\{y \setminus r\{x \setminus u\}\} = u\{y \setminus r\{x \setminus u\}\} = u = x\{y \setminus r\}\{x \setminus u\},$$

  where the second equality follows from $y \notin \mathtt{fv}(u)$.

- Case $s = y$. Then

$$y\{x \setminus u\}\{y \setminus r\{x \setminus u\}\} = y\{y \setminus r\{x \setminus u\}\} = r\{x \setminus u\} = y\{y \setminus r\}\{x \setminus u\}.$$

- Case $s = z$, with $z \neq x, y$. Immediate by the definition of substitution.

- The remaining cases follow by the definition of substitution and the *i.h.*.

□

Lemma D.2 (Preservation of Substitutions)**.** *Let $t, u, r \in \Lambda^{PM}$. If $t \to_{\mathtt{WH}(s)} r$ then $t\{x \setminus u\} \to_{\mathtt{WH}(s)} r\{x \setminus u\}$.*

Proof. The proof follows by induction on $t \to_{\mathtt{WH}} t'$:

- Case $t = \mathtt{L}(\!|\lambda p.s|\!)r \to_{\mathtt{WH(dB)}} \mathtt{L}(\!|s[p\backslash r]|\!) = t'$, such that $\mathtt{bv}(\mathtt{L}) \cap \mathtt{fv}(r) = \emptyset$. By α-conversion we can suppose in particular that $\mathtt{vars}(p) \cap \mathtt{fv}(u) = \emptyset$. We proceed by induction on $\mathtt{L}$:

  - Case $\mathtt{L} = \Box$. Then, $t = (\lambda p.s)r \to_{\mathtt{WH(dB)}} s[p\backslash r] = t'$. Clearly, $((\lambda p.s)r)\{x \setminus u\} = (\lambda p.s\{x \setminus u\})(r\{x \setminus u\})$. Moreover, $(\lambda p.s\{x \setminus u\})(r\{x \setminus u\}) \to_{\mathtt{WH(dB)}} s\{x \setminus u\}[p\backslash r\{x \setminus u\}]$. Therefore, we can conclude since $s\{x \setminus u\}[p\backslash r\{x \setminus u\}] = s[p\backslash r]\{x \setminus u\}$ (because $\mathtt{vars}(p) \cap \mathtt{fv}(u) = \emptyset$).

  - Case $\mathtt{L} = \mathtt{L}_1[q\backslash v]$. Then, $t = \mathtt{L}_1(\!|\lambda p.s|\!)[q\backslash v]r \to_{\mathtt{WH(dB)}} \mathtt{L}_1(\!|s[p\backslash r]|\!)[q\backslash v] = t'$, such that $\mathtt{bv}(\mathtt{L}_1[q\backslash v]) \cap \mathtt{fv}(r) = \emptyset$. Notice that $\mathtt{L}_1(\!|\lambda p.s|\!)r \to_{\mathtt{WH(dB)}} \mathtt{L}_1(\!|s[p\backslash r]|\!)$. So, we can apply the *i.h.*, obtaining $(\mathtt{L}_1(\!|\lambda p.s|\!)r)\{x \setminus u\} \to_{\mathtt{WH(dB)}} \mathtt{L}_1(\!|s[p\backslash r]|\!)\{x \setminus u\}$. Therefore, $t_1 = \mathtt{L}_1(\!|\lambda p.s|\!)\{x \setminus u\}[q\backslash v\{x \setminus u\}]r\{x \setminus u\} \to_{\mathtt{WH(dB)}} \mathtt{L}_1\{x \setminus u\}(\!|s\{x \setminus u\}[p\backslash r\{x \setminus u\}]|\!)[q\backslash v\{x \setminus u\}] = t'_1$. And we can conclude since $t_1 = (\mathtt{L}_1(\!|\lambda p.s|\!)[q\backslash v]r)\{x \setminus u\}$ and $t'_1 = (\mathtt{L}_1(\!|s[p\backslash r]|\!)[q\backslash v])\{x \setminus u\}$ (because $\mathtt{vars}(q) \cap \mathtt{fv}(u) = \emptyset$ and $\mathtt{vars}(p) \cap \mathtt{fv}(u) = \emptyset$).

- Case $t = \mathtt{case}\ \mathtt{L}(\!|\mathtt{c}(s_i)_n|\!)\ \mathtt{of}\ (\ldots, \mathtt{c}(p_i)_n.r, \ldots) \to_{\mathtt{WH(branch)}} \mathtt{L}(\!|r[p_1\backslash s_1] \cdots [p_n\backslash s_n]|\!) = t'$. By α-conversion we can assume $\mathtt{vars}(\mathtt{c}(p_i)_n) \cap \mathtt{fv}(u) = \emptyset$. We proceed by induction on $\mathtt{L}$:

  - Case $\mathtt{L} = \Box$. Then, we know that $t = \mathtt{case}\ \mathtt{c}(s_i)_n\ \mathtt{of}\ (\ldots, \mathtt{c}(p_i)_n.r, \ldots) \to_{\mathtt{WH(branch)}} r[p_1\backslash s_1] \cdots [p_n\backslash s_n] = t'$. Clearly, we have $(\mathtt{case}\ \mathtt{c}(s_i)_n\ \mathtt{of}\ (\ldots, \mathtt{c}(p_i)_n.r, \ldots))\{x \setminus u\} =$

$\texttt{case}\ \texttt{c}(s_i\{x \setminus u\})_n\ \texttt{of}\ (\ldots, \texttt{c}(p_i)_n.r\{x \setminus u\}, \ldots)$. Moreover, we also have $\texttt{case}\ \texttt{c}(s_i\{x \setminus u\})_n\ \texttt{of}\ (\ldots, \texttt{c}(p_i)_n.r\{x \setminus u\}, \ldots) \to_{\texttt{WH(branch)}} r\{x \setminus u\}[p_1 \setminus r_1\{x \setminus u\}] \cdots [p_n \setminus r_n\{x \setminus u\}]$. Therefore, we can conclude that $r\{x \setminus u\}[p_1 \setminus r_1\{x \setminus u\}] \cdots [p_n \setminus r_n\{x \setminus u\}] = r[p_1 \setminus r_1] \cdots [p_n \setminus r_n]\{x \setminus u\}$ (because $\texttt{vars}(\texttt{c}(p_i)_n) \cap \texttt{fv}(u) = \emptyset$).

- Case $\texttt{L} = \texttt{L}_1[q \setminus v]$. Then, $t = \texttt{case}\ \texttt{L}_1(\!|\texttt{c}(s_i)_n|\!)[q \setminus v]\ \texttt{of}\ (\ldots, \texttt{c}(p_i)_n.r, \ldots) \to_{\texttt{WH(branch)}} \texttt{L}_1(\!|r[p_1 \setminus s_1] \cdots [p_n \setminus s_n]|\!)[q \setminus v] = t'$. Notice $\texttt{case}\ \texttt{L}_1(\!|\texttt{c}(s_i)_n|\!)\ \texttt{of}\ (\ldots, \texttt{c}(p_i)_n.r, \ldots) \to_{\texttt{WH(branch)}} \texttt{L}_1(\!|r[p_1 \setminus s_1] \cdots [p_n \setminus s_n]|\!)$. By the *i.h.*, we know $(\texttt{case}\ \texttt{L}_1(\!|\texttt{c}(s_i)_n|\!)\ \texttt{of}\ (\ldots, \texttt{c}(p_i)_n.r, \ldots))\{x \setminus u\} \to_{\texttt{WH(branch)}} \texttt{L}_1(\!|r[p_1 \setminus s_1] \cdots [p_n \setminus s_n]|\!)\{x \setminus u\}$. Thus, we have $t_1 = (\texttt{case}\ \texttt{L}_1(\!|\texttt{c}(s_i)_n|\!)\ \texttt{of}\ (\ldots, \texttt{c}(p_i)_n.r, \ldots))\{x \setminus u\}[q \setminus v\{x \setminus u\}] \to_{\texttt{WH(branch)}} \texttt{L}_1(\!|r[p_1 \setminus s_1] \cdots [p_n \setminus s_n]|\!)\{x \setminus u\}[q \setminus v\{x \setminus u\}] = t'_1$. Hence, $t_1 = (\texttt{case}\ \texttt{L}_1(\!|\texttt{c}(s_i)_n|\!)[q \setminus v]\ \texttt{of}\ (\ldots, \texttt{c}(p_i)_n.r, \ldots))\{x \setminus u\}$ and $t'_1 = (\texttt{L}_1(\!|r[p_1 \setminus s_1] \cdots [p_n \setminus s_n]|\!)[q \setminus v])\{x \setminus u\}$ (because $\texttt{vars}(q) \cap \texttt{fv}(u) = \emptyset$ and $\texttt{vars}(\texttt{c}(p_i)_n) \cap \texttt{fv}(u) = \emptyset$).

- Case $t = s[\texttt{c}(p_i)_n \setminus \texttt{L}(\!|\texttt{c}(r_i)_n|\!)] \to_{\texttt{WH(match)}} \texttt{L}(\!|s[p_1 \setminus r_1] \cdots [p_n \setminus r_n]|\!) = t'$, such that $\texttt{bv}(\texttt{L}) \cap \texttt{fv}(s) = \emptyset$. Moreover, by α-conversion we can suppose $\texttt{vars}(\texttt{c}(p_i)_n) \cap \texttt{fv}(u) = \emptyset$. We proceed by induction on $\texttt{L}$:

  - Case $\texttt{L} = \Box$. Then, $t = s[\texttt{c}(p_i)_n \setminus \texttt{c}(r_i)_n] \to_{\texttt{WH(match)}} s[p_1 \setminus r_1] \cdots [p_n \setminus r_n] = t'$. Clearly, we know $(s[\texttt{c}(p_i)_n \setminus \texttt{c}(r_i)_n])\{x \setminus u\} = s\{x \setminus u\}[\texttt{c}(p_i)_n \setminus \texttt{c}(r_i\{x \setminus u\})_n]$. Moreover, $s\{x \setminus u\}[\texttt{c}(p_i)_n \setminus \texttt{c}(r_i\{x \setminus u\})_n] \to_{\texttt{WH(match)}} s\{x \setminus u\}[p_1 \setminus r_1\{x \setminus u\}] \cdots [p_n \setminus r_n\{x \setminus u\}]$. Therefore, we can conclude since $s\{x \setminus u\}[p_1 \setminus r_1\{x \setminus u\}] \cdots [p_n \setminus r_n\{x \setminus u\}] = s[p_1 \setminus r_1] \cdots [p_n \setminus r_n]\{x \setminus u\}$ (because $\texttt{vars}(\texttt{c}(p_i)_n) \cap \texttt{fv}(u) = \emptyset$).

  - Case $\texttt{L} = \texttt{L}_1[q \setminus v]$. Then, we know that the following holds: $t = s[\texttt{c}(p_i)_n \setminus \texttt{L}_1(\!|\texttt{c}(r_i)_n|\!)[q \setminus v]] \to_{\texttt{WH(match)}} \texttt{L}_1(\!|s[p_1 \setminus r_1] \cdots [p_n \setminus r_n]|\!)[q \setminus v] = t'$, such that $\texttt{bv}(\texttt{L}_1[q \setminus v]) \cap \texttt{fv}(u) = \emptyset$. Notice that $s[\texttt{c}(p_i)_n \setminus \texttt{L}_1(\!|\texttt{c}(r_i)_n|\!)] \to_{\texttt{WH(match)}} \texttt{L}_1(\!|s[p_1 \setminus r_1] \cdots [p_n \setminus r_n]|\!)$. So, we can apply the *i.h.*, obtaining $s[\texttt{c}(p_i)_n \setminus \texttt{L}_1(\!|\texttt{c}(r_i)_n|\!)]\{x \setminus u\} \to_{\texttt{WH(match)}} \texttt{L}_1(\!|s[p_1 \setminus r_1] \cdots [p_n \setminus r_n]|\!)\{x \setminus u\}$. Therefore, $t_1 = s[\texttt{c}(p_i)_n \setminus \texttt{L}_1(\!|\texttt{c}(r_i)_n|\!)]\{x \setminus u\}[q \setminus v\{x \setminus u\}] \to_{\texttt{WH(match)}} \texttt{L}_1(\!|s[p_1 \setminus r_1] \cdots [p_n \setminus r_n]|\!)\{x \setminus u\}[q \setminus v\{x \setminus u\}] = t'_1$. And we can conclude with $t_1 = s[\texttt{c}(p_i)_n \setminus \texttt{L}_1(\!|\texttt{c}(r_i)_n|\!)[q \setminus v]]\{x \setminus u\}$ and $t'_1 = \texttt{L}_1(\!|s[p_1 \setminus r_1] \cdots [p_n \setminus r_n]|\!)[q \setminus v]\{x \setminus u\}$ (because $\texttt{vars}(q) \cap \texttt{fv}(u) = \emptyset$ and $\texttt{vars}(\texttt{c}(p_i)_n) \cap \texttt{fv}(u) = \emptyset$).

- Case $t = s[y \setminus r] \to_{\texttt{WH(sub)}} s\{y \setminus r\} = t'$. We assume by α-conversion that $y \notin \texttt{fv}(u)$. Notice that $s[y \setminus r]\{x \setminus u\} = s\{x \setminus u\}[y \setminus r\{x \setminus u\}]$. Moreover, $s\{x \setminus u\}[y \setminus r\{x \setminus u\}] \to_{\texttt{WH(sub)}} s\{x \setminus u\}\{y \setminus r\{x \setminus u\}\}$. And we can conclude since $s\{x \setminus u\}\{y \setminus r\{x \setminus u\}\} = s\{y \setminus r\}\{x \setminus u\}$ by the substitution Lemma D.1.

- Case $t = s[p\backslash r] \to_{\mathtt{WH}(s)} s'[p\backslash r] = t'$, where $s \to_{\mathtt{WH}(s)} s'$. We assume by α-conversion that $\mathtt{vars}(p) \cap \mathtt{fv}(u) = \emptyset$. By the *i.h.*, we know that $s\{x \setminus u\} \to_{\mathtt{WH}(s)} s'\{x \setminus u\}$. Therefore, $s\{x \setminus u\}[p\backslash r\{x \setminus u\}] \to_{\mathtt{WH}(s)} s'\{x \setminus u\}[p\backslash r\{x \setminus u\}]$. Hence, $s\{x \setminus u\}[p\backslash r\{x \setminus u\}] = s[p\backslash r]\{x \setminus u\}$ and $s'\{x \setminus u\}[p\backslash r\{x \setminus u\}] = s'[p\backslash r]\{x \setminus u\}$ (because $\mathtt{vars}(p) \cap \mathtt{fv}(u) = \emptyset$).

- Case $t = sr \to_{\mathtt{WH}(s)} s'r = t'$, where $s \to_{\mathtt{WH}(s)} s'$. By the *i.h.*, we know that $s\{x \setminus u\} \to_{\mathtt{WH}(s)} s'\{x \setminus u\}$. Therefore, $s\{x \setminus u\}(r\{x \setminus u\}) \to_{\mathtt{WH}(s)} s'\{x \setminus u\}(r\{x \setminus u\})$. Therefore, $s\{x \setminus u\}(r\{x \setminus u\}) = (sr)\{x \setminus u\}$ and $s'\{x \setminus u\}(r\{x \setminus u\}) = (s'r)\{x \setminus u\}$.

- Case $t = s[\hat{p}\backslash r] \to_{\mathtt{WH}(s)} s[\hat{p}\backslash r'] = t'$, where $r \to_{\mathtt{WH}(s)} r'$. By the *i.h.*, we know that $r\{x \setminus u\} \to_{\mathtt{WH}(s)} r'\{x \setminus u\}$. We assume by α-conversion that $\mathtt{vars}(\hat{p}) \cap \mathtt{fv}(u) = \emptyset$. Therefore, $s\{x \setminus u\}[\hat{p}\backslash r\{x \setminus u\}] \to_{\mathtt{WH}} s\{x \setminus u\}[\hat{p}\backslash r'\{x \setminus u\}]$. And we can conclude since $s\{x \setminus u\}[\hat{p}\backslash r\{x \setminus u\}] = s[\hat{p}\backslash r]\{x \setminus u\}$ and $s\{x \setminus u\}[\hat{p}\backslash r'\{x \setminus u\}] = s[\hat{p}\backslash r']\{x \setminus u\}$ (because $\mathtt{vars}(\hat{p}) \cap \mathtt{fv}(u) = \emptyset$).

- Case $t = \mathtt{case}\ s\ \mathtt{of}\ (\mathtt{c}_i P_i.r_i)_n \to_{\mathtt{WH}(s)} \mathtt{case}\ s'\ \mathtt{of}\ (\mathtt{c}_i P_i.r_i)_n = t'$, where $s \to_{\mathtt{WH}(s)} s'$. By α-conversion we can assume in particular that $\bigcup_{i \in \{1,\dots,n\}} \mathtt{vars}(\mathtt{c}_i P_i) \cap \mathtt{fv}(u) = \emptyset$. By the *i.h.*, we know that $s\{x \setminus u\} \to_{\mathtt{WH}(s)} s'\{x \setminus u\}$. Therefore, $\mathtt{case}\ s\{x \setminus u\}\ \mathtt{of}\ (\mathtt{c}_i P_i.r_i)_n\{x \setminus u\} \to_{\mathtt{WH}(s)} \mathtt{case}\ s'\{x \setminus u\}\ \mathtt{of}\ (\mathtt{c}_i P_i.r_i)_n\{x \setminus u\}$. Hence, $\mathtt{case}\ s\{x \setminus u\}\ \mathtt{of}\ (\mathtt{c}_i P_i.r_i)_n\{x \setminus u\} = (\mathtt{case}\ s\ \mathtt{of}\ (\mathtt{c}_i P_i.r_i)_n)\{x \setminus u\}$ and $\mathtt{case}\ s'\{x \setminus u\}\ \mathtt{of}\ (\mathtt{c}_i P_i.r_i)_n\{x \setminus u\} = (\mathtt{case}\ s'\ \mathtt{of}\ (\mathtt{c}_i P_i.r_i)_n)\{x \setminus u\}$ (because $\bigcup_{i \in \{1,\dots,n\}} \mathtt{vars}(\mathtt{c}_i P_i) \cap \mathtt{fv}(u) = \emptyset$).

□

Lemma 6.15 (Characterization of Normal Forms)**.** *Let $t \in \Lambda^{PM}$. Then, $t \in \mathtt{NO}$ if and only if $t \not\to_{\mathtt{PM}}$.*

Proof. Simple corollary of Lemma D.3. □

Lemma D.3. *$t \in \mathtt{NO}_\mathtt{c}$ if and only if $t \not\to_{\mathtt{PM}}$, and either $\neg\mathtt{isdata}(t)$ or $\mathtt{isdata}_\mathtt{c}(t)$.*

Proof. In order to show the original statement, we are going to refine it into the three following statements:

1. $t \in \mathtt{NE}$ if and only if $t \not\to_{\mathtt{PM}}$, $\neg\mathtt{isabs}(t)$, and $\neg\mathtt{isdata}(t)$.
2. $t \in \mathtt{NA}_\mathtt{c}$ if and only if $t \not\to_{\mathtt{PM}}$, $\neg\mathtt{isabs}(t)$, and either $\neg\mathtt{isdata}(t)$ or $\mathtt{isdata}_\mathtt{c}(t)$.
3. $t \in \mathtt{NO}_\mathtt{c}$ if and only if $t \not\to_{\mathtt{PM}}$, and either $\neg\mathtt{isdata}(t)$ or $\mathtt{isdata}_\mathtt{c}(t)$.

The proof of the left-to-right implication follows by induction on $t \in \mathtt{NO}_\mathtt{c}$:

1. Let $t \in \mathtt{NE}$:

- Case $t = x$. Then, $x \not\to_{\mathtt{PM}}$, $\neg\mathtt{isabs}(x)$, and $\neg\mathtt{isdata}(x)$ trivially hold.
- Case $t = us$, where $u \in \mathtt{NA}$. Since $\mathtt{NA} = \bigcup_{\mathtt{c}\in\mathtt{C}} \mathtt{NA}_\mathtt{c}$, we can assume, without loss of generality, that $u \in \mathtt{NA}_\mathtt{c}$, for some $\mathtt{c} \in \mathtt{C}$. By the *i.h.* (2) over u, we know $u \not\to_{\mathtt{PM}}$, $\neg\mathtt{isabs}(u)$, and either $\neg\mathtt{isdata}(u)$ or $\mathtt{isdata}_\mathtt{c}(u)$. Since $\neg\mathtt{isabs}(u)$ and $u \not\to_{\mathtt{PM}}$, then $us \not\to_{\mathtt{PM}}$. Moreover, $\neg\mathtt{isabs}(us)$ and $\neg\mathtt{isdata}(us)$ trivially hold.
- Case $t = u[\mathtt{c}'P\backslash s]$, where $u \in \mathtt{NE}$ and $s \in \mathtt{NO}_{\neg\mathtt{c}'}$. By the *i.h.* (1) over u, we know $u \not\to_{\mathtt{PM}}$, $\neg\mathtt{isabs}(u)$, and $\neg\mathtt{isdata}(u)$. Since $s \in \mathtt{NO}_{\neg\mathtt{c}'}$, we can assume, without loss of generality, that $s \in \mathtt{NO}_{\mathtt{c}_0}$ for some $\mathtt{c}_0 \neq \mathtt{c}'$. By the *i.h.* (3) over s, we know $s \not\to_{\mathtt{PM}}$, and either $\neg\mathtt{isdata}(s)$ or $\mathtt{isdata}_{\mathtt{c}_0}(s)$. Clearly, rule (sub) does not apply to t. Additionally, since either $\neg\mathtt{isdata}(s)$ or $\mathtt{isdata}_{\mathtt{c}_0}(s)$ for $\mathtt{c}_0 \neq \mathtt{c}'$, rule (match) also does not apply to t. Finally, since $s \not\to_{\mathtt{PM}}$ and $u \not\to_{\mathtt{PM}}$, then $u[\mathtt{c}'P\backslash s] \not\to_{\mathtt{PM}}$. Moreover, $\neg\mathtt{isabs}(u[\mathtt{c}'P\backslash s])$ and $\neg\mathtt{isdata}(u[\mathtt{c}'P\backslash s])$ hold, since $\neg\mathtt{isabs}(u)$ and $\neg\mathtt{isdata}(u)$ hold.
- Case $t = \mathtt{case}\ u\ \mathtt{of}\ (\mathtt{c}_i P_i.u_i)_n$, where $u \in \mathtt{NO}_{\neg\{\mathtt{c}_1,\ldots,\mathtt{c}_n\}}$. Since $u \in \mathtt{NO}_{\neg\{\mathtt{c}_1,\ldots,\mathtt{c}_n\}}$, we can assume, without loss of generality, that $u \in \mathtt{NO}_{\mathtt{c}_0}$ for some $\mathtt{c}_0 \notin \{\mathtt{c}_1,\ldots,\mathtt{c}_n\}$. By the *i.h.* (3), we know $u \not\to_{\mathtt{PM}}$, and either $\neg\mathtt{isdata}(u)$ or $\mathtt{isdata}_{\mathtt{c}_0}(u)$. Since $\neg\mathtt{isdata}(u)$ or $\mathtt{isdata}_{\mathtt{c}_0}(u)$ for $\mathtt{c}_0 \notin \{\mathtt{c}_1,\ldots,\mathtt{c}_n\}$, then rule (branch) does not apply to t. Additionally, since $u \not\to_{\mathtt{PM}}$, then rule (caseArg) also does not apply to t. Therefore, $\mathtt{case}\ u\ \mathtt{of}\ (\mathtt{c}_i P_i.u_i)_n \not\to_{\mathtt{PM}}$. Moreover, we also have that $\neg\mathtt{isabs}(\mathtt{case}\ u\ \mathtt{of}\ (\mathtt{c}_i P_i.u_i)_n)$ and $\neg\mathtt{isdata}(\mathtt{case}\ u\ \mathtt{of}\ (\mathtt{c}_i P_i.u_i)_n)$ trivially hold.

2. Let $t \in \mathtt{NA}_\mathtt{c}$:
   - Case $t \in \mathtt{NE}$. Then, $t \not\to_{\mathtt{PM}}$, $\neg\mathtt{isabs}(t)$, and $\neg\mathtt{isdata}(t)$ by statement (1). So, we can conclude.
   - Case $t = \mathtt{c}T$. Then, $\mathtt{c}T \not\to_{\mathtt{PM}}$, $\neg\mathtt{isabs}(\mathtt{c}T)$, and $\mathtt{isdata}_\mathtt{c}(\mathtt{c}T)$.
   - Case $t = u[\mathtt{c}'P\backslash s]$, where $u \in \mathtt{NA}_\mathtt{c}$ and $s \in \mathtt{NO}_{\neg\mathtt{c}'}$. This case is very similar to the corresponding case when assuming $t \in \mathtt{NE}$.
3. Let $t \in \mathtt{NO}_\mathtt{c}$:
   - Case $t \in \mathtt{NA}_\mathtt{c}$. Then, $t \not\to_{\mathtt{PM}}$, $\neg\mathtt{isabs}(t)$, and either $\neg\mathtt{isdata}(t)$ or $\mathtt{isdata}_\mathtt{c}(t)$ by statement (2). So, we can conclude.
   - Case $t = \lambda p.u$. Then, $\lambda p.u \not\to_{\mathtt{PM}}$ and $\neg\mathtt{isdata}(\lambda p.u)$.
   - Case $t = u[\mathtt{c}'P\backslash s]$, where $u \in \mathtt{NO}_\mathtt{c}$ and $s \in \mathtt{NO}_{\neg\mathtt{c}'}$. This case is very similar to the corresponding case when assuming $t \in \mathtt{NA}_\mathtt{c}$ or $t \in \mathtt{NE}$.

The right-to-left implication follows by induction on the structure of t:

- Case $t = x$. Then, $x \not\to_{\mathtt{PM}}$, $\neg\mathtt{isabs}(x)$, and $\neg\mathtt{isdata}(x)$, and thus the hypothesis of points (1), (2), and (3) apply. However, since $\mathtt{NE} \subseteq \mathtt{NA_c} \subseteq \mathtt{NO_c}$ for any $\mathtt{c}$, it is enough to show point (1). And we can conclude since $x \in \mathtt{NE}$, by definition.

- Case $t = us$. Then, suppose $us \not\to_{\mathtt{PM}}$. We always have $\neg\mathtt{isabs}(us)$, and $\neg\mathtt{isdata}(us)$. Then, the hypothesis of points (1), (2), and (3) apply. But, $\mathtt{NE} \subseteq \mathtt{NA_c} \subseteq \mathtt{NO_c}$ for any $\mathtt{c}$, it is enough to show point (1). Since $us \not\to_{\mathtt{PM}}$, then in particular $\neg\mathtt{isabs}(u)$ holds (otherwise rule (dB) would apply), and $u \not\to_{\mathtt{PM}}$ (otherwise rule (appL) would apply). Note that it is always the case that either $\neg\mathtt{isdata}(u)$, or there exists $\mathtt{c}' \in \mathtt{C}$, such that $\mathtt{isdata_{c'}}(u)$. By the *i.h.* (2) over u, we know that $u \in \mathtt{NA_{c'}}$ for some $\mathtt{c}' \in \mathtt{C}$. Therefore, $u \in \mathtt{NA_{c'}} \subseteq \mathtt{NA}$ and $us \in \mathtt{NE}$.

- Case $t = \lambda p.u$. Then, $\lambda p.u \not\to_{\mathtt{PM}}$ and $\neg\mathtt{isdata}(\lambda p.u)$. Therefore, the hypothesis of point (3) applies. And we can conclude since $\lambda p.u \in \mathtt{NO_c}$, by definition.

- Case $t = u[p\backslash s]$. Then, suppose $u[p\backslash s] \not\to_{\mathtt{PM}}$. We have $\neg\mathtt{isdata}(u[p\backslash s])$ or $\mathtt{isdata_c}(u[p\backslash s])$ for some $\mathtt{c} \in \mathtt{C}$, which implies either $\neg\mathtt{isdata}(u)$ or $\mathtt{isdata_c}(u)$. In any case, p must not be a variable (otherwise rule (sub) would apply), so let us assume that $p = \mathtt{c}'P$, for some $\mathtt{c}' \in \mathtt{C}$. It is also the case that $\neg\mathtt{isdata_{c'}}(s)$ must hold (otherwise rule (match) would apply), $u \not\to_{\mathtt{PM}}$ (otherwise rule (esL) would apply), and $s \not\to_{\mathtt{PM}}$ (otherwise rule (esR) would apply). Moreover, since $\neg\mathtt{isdata_{c'}}(s)$, then either $\neg\mathtt{isdata}(s)$, or $\mathtt{isdata_{c_0}}(s)$, for some $\mathtt{c}_0 \in \mathtt{C}$ such that $\mathtt{c}_0 \neq \mathtt{c}'$. By the *i.h.* (3) over s, we know $s \in \mathtt{NO_{c_0}}$ *i.e.* that $s \in \mathtt{NO_{\neg c'}}$.

  - If we make no further assumptions regarding the form of u, then only the hypothesis of point (3) applies. By the induction *i.h.* (1) over u, we know $u \in \mathtt{NO_c}$. Therefore, $u[p\backslash s] \in \mathtt{NO_c}$ by definition, and this concludes point (3).
  - If we also assume that $\neg\mathtt{isabs}(u[p\backslash s])$ holds, then $\neg\mathtt{isabs}(u)$ also holds and the hypothesis of point (2) also applies. By the *i.h.* (2) over u, we know $u \in \mathtt{NA_c}$, for some $\mathtt{c} \in \mathtt{C}$. Therefore, $u[p\backslash s] \in \mathtt{NA_c}$ by definition, and this concludes point (2).
  - If we assume both $\neg\mathtt{isabs}(u[p\backslash s])$ and $\neg\mathtt{isdata}(u[p\backslash s])$, then $\neg\mathtt{isdata}(u)$ also holds and the hypothesis of point (1) also applies. By the *i.h.* (1) over u, we know that $u \in \mathtt{NE}$. Therefore, $u[p\backslash s] \in \mathtt{NE}$ by definition, and this concludes point (1).

- Case $t = \mathtt{c}T$. Then, $\mathtt{c}T \not\to_{\mathtt{PM}}$, $\neg\mathtt{isabs}(\mathtt{c}T)$ and $\mathtt{isdata_c}(\mathtt{c}T)$. Therefore, the hypothesis of points (2) and (3) apply. We have $\mathtt{c}T \in \mathtt{NA_c}$ by definition, thus point (2) holds. Therefore, point (3) also holds since $\mathtt{NA_c} \subseteq \mathtt{NO_c}$.

- Case $t = \texttt{case}\ u\ \texttt{of}\ (\texttt{c}_i P_i.s_i)_n$. Then, suppose that $\texttt{case}\ u\ \texttt{of}\ (\texttt{c}_i P_i.s_i)_n \not\to_{\texttt{PM}}$. We also have $\neg\texttt{isabs}(\texttt{case}\ u\ \texttt{of}\ (\texttt{c}_i P_i.s_i)_n)$, and $\neg\texttt{isdata}(\texttt{case}\ u\ \texttt{of}\ (\texttt{c}_i P_i.s_i)_n)$. Therefore, the hypothesis of points (1), (2), and (3) apply. As before, it is enough to show point (1). Since $\texttt{case}\ u\ \texttt{of}\ (\texttt{c}_i P_i.s_i)_n \not\to_{\texttt{PM}}$, then $\neg\texttt{isdata}_{\texttt{c}_0}(u)$, for $\texttt{c}_0 \in \{\texttt{c}_1, \ldots, \texttt{c}_n\}$ must hold (otherwise rule (branch) would apply), and $u \not\to_{\texttt{PM}}$ (otherwise rule (caseArg) would apply). Thus, either $\neg\texttt{isdata}(u)$ or $\texttt{isdata}_{\texttt{c}'}(u)$ for $\texttt{c}' \notin \{\texttt{c}_1, \ldots, \texttt{c}_n\}$. By the *i.h.* (3) over u, we know $u \in \texttt{NO}_{\texttt{c}'}$, *i.e.* $u \in \texttt{NO}_{\neg\{\texttt{c}_1,\ldots,\texttt{c}_n\}}$. Therefore, $\texttt{case}\ u\ \texttt{of}\ (\texttt{c}_i P_i.s_i)_n \in \texttt{NE}$, by definition.

□

Lemma 6.19 (Clash-Free Normal Forms)**.** *Let $t \in \Lambda^{PM}$. Then, $t \in \texttt{NO}^{\texttt{CF}}$ if and only if $t \not\to_{\texttt{PM}}$ and $t \notin \texttt{CL}$*

Proof. To show the original statement, we are going to refine it into the following two statements:

1. $t \in \texttt{NE}^{\texttt{CF}}$ if and only if $t \not\to_{\texttt{PM}}$, $t \notin \texttt{CL}$, $\neg\texttt{isabs}(t)$, and $\neg\texttt{isdata}(t)$.
2. $t \in \texttt{NO}^{\texttt{CF}}$ if and only if $t \not\to_{\texttt{PM}}$ and $t \notin \texttt{CL}$.

We start by showing the left-to-right implication, by induction on $t \in \texttt{NO}^{\texttt{CF}}$:

1. Let $t \in \texttt{NE}^{\texttt{CF}}$:
   - Case $t = x$. Then, $x \not\to_{\texttt{PM}}$, $x \notin \texttt{CL}$, $\neg\texttt{isabs}(x)$, and $\neg\texttt{isdata}(x)$ trivially hold.
   - Case $t = us$, such that $u \in \texttt{NE}^{\texttt{CF}}$. By the *i.h.* (1) over u, we know $u \not\to_{\texttt{PM}}$, $u \notin \texttt{CL}$, $\neg\texttt{isabs}(u)$, and $\neg\texttt{isdata}(u)$. We can easily conclude that $us \not\to_{\texttt{PM}}$ by a simple inspection of the rules. Let us assume $us \in \texttt{CL}$. Then, us must be of one of the following forms:
     - $u = \texttt{L}(\!|\texttt{c}T|\!)$, for some $c \in \texttt{C}$. Contradiction with the fact that $\neg\texttt{isdata}(u)$.
     - $u \in \texttt{CL}$. Contradiction with the fact that $u \notin \texttt{CL}$.

     Thus, we can conclude that $us \notin \texttt{CL}$. Moreover, $\neg\texttt{isabs}(us)$ and $\neg\texttt{isdata}(us)$ trivially hold.
   - Case $t = u[p\backslash s]$, such that $u, s \in \texttt{NE}^{\texttt{CF}}$ and $p = \texttt{c}P$. By the *i.h.* (1) over u (resp. s), we know $u \not\to_{\texttt{PM}}$ (resp. $s \not\to_{\texttt{PM}}$), $u \notin \texttt{CL}$ (resp. $s \notin \texttt{CL}$), $\neg\texttt{isabs}(u)$ (resp. $\neg\texttt{isabs}(s)$), and $\neg\texttt{isdata}(u)$ (resp. $\neg\texttt{isdata}(s)$). We can easily conclude that $u[p\backslash s] \not\to_{\texttt{PM}}$ by a simple inspection of the rules. Let us assume $u[p\backslash s] \in \texttt{CL}$. Then, $u[p\backslash s]$ must be of one of the following forms:
     - $s = \texttt{L}(\!|\lambda q.r|\!)$. Contradiction with the fact that $\neg\texttt{isabs}(s)$.
     - $s = \texttt{L}(\!|\texttt{c}'T|\!)$, such that $\texttt{c} \neq \texttt{c}'$. Contradiction with the fact that $\neg\texttt{isdata}(s)$.
     - $u \in \texttt{CL}$. Contradiction with the fact that $u \notin \texttt{CL}$.

– $s \in \texttt{CL}$. Contradiction with the fact that $s \notin \texttt{CL}$.

Thus, we conclude that $u[p\backslash s] \notin \texttt{CL}$. Moreover, since $\neg\texttt{isabs}(u)$ and $\neg\texttt{isdata}(u)$ hold, then so do $\neg\texttt{isabs}(u[p\backslash s])$ and $\neg\texttt{isdata}(u[p\backslash s])$.

- Case $t = \texttt{case}\ u\ \texttt{of}\ (\texttt{c}_i P_i.u_i)_n$, such that $u \in \texttt{NE}^{\texttt{CF}}$. By the *i.h.* (1), we know $u \not\to_{\texttt{PM}}$, $u \notin \texttt{CL}$, $\neg\texttt{isabs}(u)$, and $\neg\texttt{isdata}(u)$. We can easily conclude that $\texttt{case}\ u\ \texttt{of}\ (\texttt{c}_i P_i.u_i)_n \not\to_{\texttt{PM}}$ by a simple inspection of the rules. Let us assume $\texttt{case}\ u\ \texttt{of}\ (\texttt{c}_i P_i.u_i)_n \in \texttt{CL}$. Then, $\texttt{case}\ u\ \texttt{of}\ (\texttt{c}_i P_i.u_i)_n$ must be of one of the following forms:

  – $u = \texttt{L}(\!|\lambda p.t|\!)$. Contradiction with the fact that $\neg\texttt{isabs}(u)$.

  – $u = \texttt{c}T$, such that $\texttt{c} \notin \{\texttt{c}_1, \ldots, \texttt{c}_n\}$. Contradiction with the fact $\neg\texttt{isdata}(u)$.

  – $u \in \texttt{CL}$. Contradiction with the fact that $u \notin \texttt{CL}$.

  Thus, we can conclude that $\texttt{case}\ u\ \texttt{of}\ (\texttt{c}_i P_i.u_i)_n \notin \texttt{CL}$. Moreover, it is the case that $\neg\texttt{isabs}(\texttt{case}\ u\ \texttt{of}\ (\texttt{c}_i P_i.u_i)_n)$ and $\neg\texttt{isdata}(\texttt{case}\ u\ \texttt{of}\ (\texttt{c}_i P_i.u_i)_n)$ hold.

2. Case $t \in \texttt{NO}^{\texttt{CF}}$:

   - Case $t \in \texttt{NE}^{\texttt{CF}}$. This case follows from statement (1).
   - Case $t = \lambda p.u$. Then, $\lambda p.u \not\to_{\texttt{PM}}$ and $\lambda p.u \notin \texttt{CL}$, by definition.
   - Case $t = \texttt{c}T$. Then, $\texttt{c}T \not\to_{\texttt{PM}}$ and $\texttt{c}T \notin \texttt{CL}$, by definition.
   - Case $t = u[\texttt{c}P\backslash s]$, such that $u \in \texttt{NO}^{\texttt{CF}}$ and $s \in \texttt{NE}^{\texttt{CF}}$. This case is very similar to the corresponding case when $t \in \texttt{NE}^{\texttt{CF}}$.

Now, we show the right-to-left implication, by induction on the structure of t:

- Case $t = x$. Then, $x \in \texttt{NE}^{\texttt{CF}} \subseteq \texttt{NO}^{\texttt{CF}}$ by definition so that both points (1) and (2) hold.
- Case $t = \lambda p.u$. Then, $\lambda p.u \not\to_{\texttt{PM}}$, $\lambda p.u \notin \texttt{CL}$, and $\texttt{isabs}(\lambda p.u)$ and $\neg\texttt{isdata}(\lambda p.u)$. Therefore, only the hypothesis of point (2) applies. And we can conclude since $\lambda p.u \in \texttt{NO}^{\texttt{CF}}$, by definition.
- Case $t = us$. Then, suppose $us \not\to_{\texttt{PM}}$ and $us \notin \texttt{CL}$. We always have $\neg\texttt{isabs}(us)$ and $\neg\texttt{isdata}(us)$. Then, the hypothesis of point (1) and (2) apply. As before, it is enough to show point (1) which implies (2). Since $us \not\to_{\texttt{PM}}$, then $\neg\texttt{isabs}(u)$ must hold (otherwise rule (dB) would apply), and $u \not\to_{\texttt{PM}}$ (otherwise rule (appL) would apply). Additionally, since $us \notin \texttt{CL}$, then $\neg\texttt{isdata}(u)$ and $u \notin \texttt{CL}$. By the *i.h.* (1), we know $u \in \texttt{NE}^{\texttt{CF}}$. Therefore, $us \in \texttt{NE}^{\texttt{CF}}$, by definition.
- Case $t = u[p\backslash s]$. Then, suppose $u[p\backslash s] \not\to_{\texttt{PM}}$, $u[p\backslash s] \notin \texttt{CL}$. Since $u[p\backslash s] \not\to_{\texttt{PM}}$, the following must be the case: p must be of the form $p = \texttt{c}P$ for some $\texttt{c} \in \texttt{C}$ (otherwise rule (sub)

would apply); $\neg\texttt{isdata}_{\texttt{c}}(s)$ (otherwise rule (match) would apply); $u \not\to_{\texttt{PM}}$ (otherwise rule (esL) would apply); and $s \not\to_{\texttt{PM}}$ (otherwise rule (esR) would apply). Additionally, since $u[p\backslash s] \notin \texttt{CL}$, then $\neg\texttt{isabs}(s)$, $\neg\texttt{isdata}_{\texttt{c}'}(s)$ for any $\texttt{c}' \neq \texttt{c}$, $u \notin \texttt{CL}$, and $s \notin \texttt{CL}$. Thus, $\neg\texttt{isdata}(s)$ must necessarily hold. By *i.h.* (1), we know $s \in \texttt{NE}^{\texttt{CF}}$. If we make no further assumptions regarding the form of u, then only the hypothesis of point (2) applies. By *i.h.* (2), $u \in \texttt{NO}^{\texttt{CF}}$. Therefore, $u[p\backslash s] \in \texttt{NO}^{\texttt{CF}}$, by definition. If we also assume that $\neg\texttt{isabs}(u[p\backslash s])$ and $\neg\texttt{isdata}(u[p\backslash s])$ hold, then $\neg\texttt{isabs}(u)$ and $\neg\texttt{isdata}(u)$ also hold and the hypothesis of point (1) also applies. By *i.h.* (1), $u \in \texttt{NE}^{\texttt{CF}}$. Therefore, $u[p\backslash s] \in \texttt{NE}^{\texttt{CF}}$, by definition.

- Case $t = \texttt{c}T$. Then, $\texttt{c}T \not\to_{\texttt{PM}}$, $\texttt{c}T \notin \texttt{CL}$, $\neg\texttt{isabs}(\texttt{c}T)$ and $\texttt{isdata}(\texttt{c}T)$. Therefore, only the hypothesis of point (2) applies. And we can conclude since $\texttt{c}T \in \texttt{NO}^{\texttt{CF}}$, by definition.

- Case $t = \texttt{case}\ u\ \texttt{of}\ (\texttt{c}P_i.s_i)_n$. Then, suppose $\texttt{case}\ u\ \texttt{of}\ (\texttt{c}P_i.s_i)_n \not\to_{\texttt{PM}}$ and $\texttt{case}\ u\ \texttt{of}\ (\texttt{c}P_i.s_i)_n \notin \texttt{CL}$. We always have $\neg\texttt{isabs}(\texttt{case}\ u\ \texttt{of}\ (\texttt{c}P_i.s_i)_n)$ and $\neg\texttt{isdata}(\texttt{case}\ u\ \texttt{of}\ (\texttt{c}P_i.s_i)_n)$. Then, the hypothesis of points (1) and (2) apply. It is enough to show point (1) which implies point (2). Since $\texttt{case}\ u\ \texttt{of}\ (\texttt{c}P_i.s_i)_n \not\to_{\texttt{PM}}$, the following must be the case: either $\neg\texttt{isdata}(u)$ or $\texttt{isdata}_{\texttt{c}'}(u)$ for $\texttt{c}' \notin \{\texttt{c}_1, \ldots, \texttt{c}_n\}$ (otherwise rule (branch) would apply); and $u \not\to_{\texttt{PM}}$ (otherwise rule (caseArg) would apply). Additionally, since $\texttt{case}\ u\ \texttt{of}\ (\texttt{c}P_i.s_i)_n \notin \texttt{CL}$, then $\neg\texttt{isabs}(u)$, $\neg\texttt{isdata}_{\texttt{c}'}(u)$ for $\texttt{c}' \notin \{\texttt{c}_1, \ldots, \texttt{c}_n\}$, and $u \notin \texttt{CL}$. Thus, $\neg\texttt{isdata}(u)$ must necessarily hold. By the *i.h.* (1) over u, we know that $u \in \texttt{NE}^{\texttt{CF}}$. Therefore, $\texttt{case}\ u\ \texttt{of}\ (\texttt{c}P_i.s_i)_n \in \texttt{NE}^{\texttt{CF}}$, by definition.

□

Lemma 6.20 (Closed Clash-Free Normal Forms)**.** *Let $t \in \Lambda^{PM}$ be a closed term. Then, $t \not\to_{\texttt{PM}}$ and $t \notin \texttt{CL}$ if and only if $t = \lambda p.u$ or $t = \texttt{c}T$ for $\texttt{c} \in \texttt{C}$.*

Proof.

⇒) Let $t \not\to_{\texttt{PM}}$ and $t \notin \texttt{CL}$. The proof follows by induction on t:

  - Case $t = x$ does not apply because x is not a closed term.
  - Case $t = \lambda p.u$ or $t = \texttt{c}T$ for some $\texttt{c} \in \texttt{C}$, we can conclude immediately.
  - Case $t = us$. Since $us \not\to_{\texttt{PM}}$, we know that $\neg\texttt{isabs}(u)$ and $u \not\to_{\texttt{PM}}$ by a simple inspection of the reduction rules. Since $us \notin \texttt{CL}$, we know $u \notin \texttt{CL}$. Now, notice that, since us is closed, then so is u. By the *i.h.* over u, we know $u = \lambda p.u$ or $u = \texttt{c}T$ for some $\texttt{c} \in \texttt{C}$. However, we know that $\neg\texttt{isabs}(u)$ must hold. Therefore, $u = \texttt{c}T$. But, then $\texttt{c}T \in \texttt{CL}$, which contradicts the hypothesis. Thus, this case does not apply.

– Case $t = u[p\backslash s]$. Since $u[p\backslash s] \not\to_{\mathtt{PM}}$, we know that p cannot be a variable because of rule (sub). Let us assume, without any loss of generality, that $p = \mathtt{c}P$ for some $\mathtt{c} \in \mathtt{C}$. Then, we also know that $\neg\mathtt{isdata}_{\mathtt{c}}(s)$, $s \not\to_{\mathtt{PM}}$ and $u \not\to_{\mathtt{PM}}$ by a simple inspection of the remaining reduction rules. Since $u[p\backslash s] \notin \mathtt{CL}$, we know $\neg\mathtt{isabs}(s)$, $\neg\mathtt{isdata}_{\mathtt{c}'}(s)$ with $\mathtt{c}' \neq \mathtt{c}$, $u \notin \mathtt{CL}$, and $s \notin \mathtt{CL}$. Therefore, we can immediately conclude that $\neg\mathtt{isdata}(s)$. Note that, since $u[p\backslash s]$ is closed, then so is s. By the *i.h.* over s, we know $s = \lambda p.r$ or $s = \mathtt{c}_0 T$ for some $\mathtt{c}_0$. However, we know that $\neg\mathtt{isabs}(s)$ and $\neg\mathtt{isdata}(s)$ must hold. Thus, this case does not apply.

– Case $t = \mathtt{case}\ u\ \mathtt{of}\ (\mathtt{c}_i P_i.s_i)_n$. Since $\mathtt{case}\ u\ \mathtt{of}\ (\mathtt{c}_i P_i.s_i)_n \not\to_{\mathtt{PM}}$, we know that $\neg\mathtt{isdata}_{\mathtt{c}_i}(u)$ for $i \in \{1,\ldots,n\}$ and $u \not\to_{\mathtt{PM}}$ by a simple inspection of the reduction rules. Since $\mathtt{case}\ u\ \mathtt{of}\ (\mathtt{c}_i P_i.s_i)_n \notin \mathtt{CL}$, we know $\neg\mathtt{isabs}(u)$, $\neg\mathtt{isdata}_{\mathtt{c}_0}(u)$ for some $\mathtt{c}_0 \notin \{\mathtt{c}_1,\ldots,\mathtt{c}_n\}$, and $u \notin \mathtt{CL}$. Therefore, we can immediately conclude that $\neg\mathtt{isdata}(u)$. Note that, since $\mathtt{case}\ u\ \mathtt{of}\ (\mathtt{c}_i P_i.s_i)_n$ is closed, then so is u. Now, notice that, since $\mathtt{case}\ u\ \mathtt{of}\ (\mathtt{c}_i P_i.s_i)_n$ is closed, then so is u. By the *i.h.* over u, we know $u = \lambda x.s$ or $u = \mathtt{c}T$. However, we also know that $\neg\mathtt{isabs}(u)$ and $\neg\mathtt{isdata}(u)$ must hold. Thus, this case does not apply.

$\Leftarrow$) Let $t = \lambda p.u$ or $t = \mathtt{c}T$. Then, $t \not\to_{\mathtt{PM}}$ and $t \notin \mathtt{CL}$ hold trivially by definition.

□

D.1.2 Encoding the CBN and CBV λ-Calculi

Lemma 6.21 (CBN Simulation)**.** *Let $t, u \in \Lambda$. If $t \twoheadrightarrow_{\mathtt{CBN}} u$, then $t \twoheadrightarrow_{\mathtt{WH}} u$.*

Proof. Let $t \twoheadrightarrow_{\mathtt{CBN}}{}^{n} u$, where n is the number of evaluation steps between t and u. The proof follows by induction on n:

- Let $n = 0$. Then, $u = t$ and thus $t \twoheadrightarrow_{\mathtt{WH}} u$.
- Let $n > 0$. Then, there exists t', such that $t \to_{\mathtt{CBN}} t' \twoheadrightarrow_{\mathtt{CBN}} u$. By the *i.h.* $t' \twoheadrightarrow_{\mathtt{WH}} u$, therefore, it is now enough to show that $t \twoheadrightarrow_{\mathtt{WH}} t'$. Moreover, since $\twoheadrightarrow_{\mathtt{WH}}$ is closed under contexts, it suffices to reason by induction on CBN evaluation contexts:
  - Case $\mathtt{N} = \Box$. Then, $t = (\lambda x.r)s \to_{\beta_{\mathtt{n}}} r\{x \setminus s\} = t'$. Then, $(\lambda x.r)s \to_{\mathtt{dB}} r[x\backslash s] \to_{\mathtt{sub}} r\{x \setminus s\}$.
  - Case $\mathtt{N} = \mathtt{N}'r$. Then, $\mathtt{N}(\!|s|\!) = \mathtt{N}'(\!|s|\!)r$ and $\mathtt{N}'(\!|s|\!) \to_{\mathtt{CBN}} \mathtt{N}'(\!|s'|\!)$. By the *i.h.* we know $\mathtt{N}'(\!|s|\!) \twoheadrightarrow_{\mathtt{WH}} \mathtt{N}'(\!|s'|\!)$. Therefore, $t = \mathtt{N}(\!|s|\!) = \mathtt{N}'(\!|s|\!)r \twoheadrightarrow_{\mathtt{WH}} \mathtt{N}'(\!|s'|\!)r = \mathtt{N}(\!|s'|\!) = t'$.

□

Lemma 6.22 (The CBV Translation Preserves Substitutions). *Let* $t, v \in \Lambda$. *Then,* $(t\{x \setminus v\})^\bullet = (t)^\bullet\{x \setminus (v)^\circ\}$.

Proof. By induction on t:

- Case $t = x$. Then, $x\{x \setminus v\} = v$. Therefore, $(t\{x \setminus v\})^\bullet = (v)^\bullet = \mathtt{v}((v)^\circ)$. Thus, $(t)^\bullet\{x \setminus (v)^\circ\} = \mathtt{v}(x)\{x \setminus (v)^\circ\} = \mathtt{v}((v)^\circ) = (v)^\bullet$.
- Case $t = y \neq x$. Then, $y\{x \setminus v\} = y$. Therefore, $(t\{x \setminus v\})^\bullet = (y)^\bullet$. Also, $(t)^\bullet\{x \setminus (v)^\circ\} = (y)^\bullet\{x \setminus (v)^\circ\} = (y)^\bullet$.
- Case $t = \lambda y.t'$. By α-conversion, we assume $y \neq x$ and $y \notin \mathtt{fv}(v)$. Then, $(\lambda y.t')\{x \setminus v\} = \lambda y.t'\{x \setminus v\}$, so that

$$\begin{array}{rcl}((\lambda y.t')\{x \setminus v\})^\bullet & = & \mathtt{v}((\lambda y.t'\{x \setminus v\})^\circ) \\ & = & \mathtt{v}(\lambda y.(t'\{x \setminus v\})^\bullet) \\ & \overset{i.h.}{=} & \mathtt{v}(\lambda y.(t')^\bullet\{x \setminus (v)^\circ\}) \\ & = & \mathtt{v}(\lambda y.(t')^\bullet)\{x \setminus (v)^\circ\} \\ & = & \mathtt{v}((\lambda y.t')^\circ)\{x \setminus (v)^\circ\} \\ & = & (\lambda y.t')^\bullet\{x \setminus (v)^\circ\}.\end{array}$$

- Case $t = ur$. Then, $(ur)\{x \setminus v\} = (u\{x \setminus v\})(r\{x \setminus v\})$, so that

$$\begin{array}{rcl}((ur)\{x \setminus v\})^\bullet & = & (yz)[\mathtt{v}(z)\backslash(r\{x \setminus v\})^\bullet][\mathtt{v}(y)\backslash(u\{x \setminus v\})^\bullet] \\ & \overset{i.h.}{=} & (yz)[\mathtt{v}(z)\backslash(r)^\bullet\{x \setminus (v)^\circ\}][\mathtt{v}(y)\backslash(u)^\bullet\{x \setminus (v)^\circ\}] \\ & = & ((yz)[\mathtt{v}(z)\backslash(r)^\bullet][\mathtt{v}(y)\backslash(u)^\bullet])\{x \setminus (v)^\circ\} \\ & = & (ur)^\bullet\{x \setminus (v)^\circ\}.\end{array}$$

□

Lemma 6.23 (CBV Simulation). *Let* $t, u \in \Lambda$. *If* $t \twoheadrightarrow_{\mathtt{CBV}} u$, *then* $(t)^\bullet \twoheadrightarrow_{\mathtt{WH}} (u)^\bullet$.

Proof. Let $t \twoheadrightarrow_{\mathtt{CBV}}{}^n u$, where n is the number of evaluation steps between t and u. The proof follows by induction on n:

- Let $n = 0$. Then, $u = t$ and thus $(t)^\bullet \twoheadrightarrow_{\mathtt{WH}} (u)^\bullet$.
- Let $n > 0$. Then, there exists t', such that $t \to \mathtt{v}t' \twoheadrightarrow_{\mathtt{CBV}} u$. By the *i.h.* $(t')^\bullet \twoheadrightarrow_{\mathtt{WH}} (u)^\bullet$. Therefore, it is now enough to show that $(t)^\bullet \twoheadrightarrow_{\mathtt{WH}} (t')^\bullet$. Moreover, since $\twoheadrightarrow_{\mathtt{WH}}$ is closed under contexts, it suffices to reason by induction on CBV evaluation contexts:

– Case $\mathtt{V} = \Box$. Then, $t = (\lambda x.r)v \to \beta_{\mathtt{v}} r\{x \setminus v\} = t'$. Therefore,

$$\begin{array}{rcl}
((\lambda x.r)v)^\bullet & = & (yz)[\mathtt{v}(z)\backslash(v)^\bullet][\mathtt{v}(y)\backslash(\lambda x.r)^\bullet] \\
& = & (yz)[\mathtt{v}(z)\backslash\mathtt{v}((v)^\circ)][\mathtt{v}(y)\backslash\mathtt{v}(\lambda x.(r)^\bullet)] \\
& \to_{\mathtt{WH}(\mathtt{match})} & (yz)[\mathtt{v}(z)\backslash\mathtt{v}((v)^\circ)][y\backslash\lambda x.(r)^\bullet] \\
& \to_{\mathtt{WH}(\mathtt{sub})} & ((\lambda x.(r)^\bullet)z)[\mathtt{v}(z)\backslash\mathtt{v}((v)^\circ)] \\
& \to_{\mathtt{WH}(\mathtt{match})} & ((\lambda x.(r)^\bullet)z)[z\backslash(v)^\circ] \\
& \to_{\mathtt{WH}(\mathtt{sub})} & (\lambda x.(r)^\bullet)(v)^\circ \\
& \to_{\mathtt{WH}(\mathtt{dB})} & (r)^\bullet[x\backslash(v)^\circ] \\
& \to_{\mathtt{WH}(\mathtt{sub})} & (r)^\bullet\{x \setminus (v)^\circ\} \\
& \overset{\textit{Lemma } 6.22}{=} & (r\{x \setminus v\})^\bullet.
\end{array}$$

– Case $\mathtt{V} = \mathtt{V}'r$. Then, $t = \mathtt{V}(\!|s|\!) = \mathtt{V}'(\!|s|\!)r$ and $\mathtt{V}'(\!|s|\!) \to \mathtt{v}\mathtt{V}'(\!|s'|\!)$. Moreover,

$$\begin{array}{rcl}
(t)^\bullet & = & (\mathtt{V}(\!|s|\!))^\bullet \\
& = & (\mathtt{V}'(\!|s|\!)\ r)^\bullet \\
& = & (xy)[\mathtt{v}(y)\backslash(r)^\bullet][\mathtt{v}(x)\backslash(\mathtt{V}'(\!|s|\!))^\bullet].
\end{array}$$

By the *i.h.*, we know $(\mathtt{V}'(\!|s|\!))^\bullet \twoheadrightarrow_{\mathtt{WH}} (\mathtt{V}'(\!|s'|\!))^\bullet$. Therefore,

$$\begin{array}{rcl}
(t)^\bullet & = & (xy)[\mathtt{v}(y)\backslash(r)^\bullet][\mathtt{v}(x)\backslash(\mathtt{V}'(\!|s|\!))^\bullet] \\
& \twoheadrightarrow_{\mathtt{WH}} & (xy)[\mathtt{v}(y)\backslash(r)^\bullet][\mathtt{v}(x)\backslash(\mathtt{V}'(\!|s'|\!))^\bullet] \\
& = & (\mathtt{V}'(\!|s'|\!)\ r)^\bullet \\
& = & (\mathtt{V}(\!|s'|\!))^\bullet \\
& = & (t')^\bullet.
\end{array}$$

– Case $\mathtt{V} = v\mathtt{V}'$. Then, $t = \mathtt{V}(\!|r|\!) = v\ \mathtt{V}'(\!|r|\!)$ and $\mathtt{V}'(\!|r|\!) \to \mathtt{v}\mathtt{V}'(\!|r'|\!)$. Moreover,

$$\begin{array}{rcl}
(t)^\bullet & = & (\mathtt{V}(\!|r|\!))^\bullet \\
& = & (v\ \mathtt{V}'(\!|r|\!))^\bullet \\
& = & (xy)[\mathtt{v}(y)\backslash(\mathtt{V}'(\!|r|\!))^\bullet][\mathtt{v}(x)\backslash(v)^\bullet].
\end{array}$$

By the *i.h.*, we know $(\mathtt{V}'(\!|r|\!))^\bullet \twoheadrightarrow_{\mathtt{WH}} (\mathtt{V}'(\!|r'|\!))^\bullet$. Therefore,

$$\begin{array}{rcl}
(t)^\bullet & = & (xy)[\mathtt{v}(y)\backslash(\mathtt{V}'(\!|r|\!))^\bullet][\mathtt{v}(x)\backslash(v)^\bullet] \\
& \twoheadrightarrow_{\mathtt{WH}} & (xy)[\mathtt{v}(y)\backslash(\mathtt{V}'(\!|r'|\!))^\bullet][\mathtt{v}(x)\backslash(v)^\bullet] \\
& = & (v\ \mathtt{V}'(\!|r'|\!))^\bullet \\
& = & (\mathtt{V}(\!|r'|\!))^\bullet \\
& = & (t')^\bullet.
\end{array}$$

Note that although $\to_{\mathtt{WH}}$ is nondeterministic and allows the matching redex to be discharged immediately via sub, it is also possible to first reduce the translated argument. In either case, contextual closure ensures reachability of $(t')^\bullet$.

□

D.2 Pattern-Matching Non-Idempotent Intersection Types

Lemma 6.29 (Split and Merge for Multi-Types). *Let $t \in \Lambda^{PM}$ and $M = \sqcup_{i\in\mathtt{I}} M_i$. Then, there exists $\Phi \triangleright \Gamma \vdash t : M$ if and only if there exist $(\Phi_i \triangleright \Gamma_i \vdash t : M_i)_{i\in\mathtt{I}}$, such that $\Gamma = +_{i\in\mathtt{I}}\Gamma_i$. Moreover, $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi) = +_{i\in\mathtt{I}}\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_i)$.*

Proof. Let $M = \sqcup_{i\in\mathtt{I}} M_i$.

$\Rightarrow$) If there exists $\Phi \triangleright \Gamma \vdash t : M$, then Φ it must be of the following form:

$$\frac{(\,\Phi_k \triangleright \Gamma_k \vdash t : A_k)_{k\in K}}{+_{k\in K}\Gamma_k \vdash t : \{\!\{A_k\}\!\}_{k\in K}}\ \mathtt{Many}$$

where $\Gamma = +_{k\in K}\Gamma_k$ and $M = \{\!\{A_k\}\!\}_{k\in K}$. Since also $M = \sqcup_{i\in\mathtt{I}} M_i$, then there exist disjoint sets $(K_i)_{i\in\mathtt{I}} \subseteq K$ such that $\cup_{i\in\mathtt{I}} K_i = K$ and $M_i = \{\!\{A_k\}\!\}_{k\in K_i}$. One concludes this case by constructing Φ_i and $\Gamma_i = +_{k\in K_i}\Gamma_k$ for any $i \in \mathtt{I}$ as follows:

$$\frac{(\,\Phi_k \triangleright \Gamma_k \vdash t : A_k)_{k\in K_i}}{+_{k\in K_i}\Gamma_k \vdash t : \{\!\{A_k\}\!\}_{k\in K_i}}\ \mathtt{Many}$$

Indeed, one has:

$$\Gamma = +_{k\in K}\Gamma_k = +_{i\in\mathtt{I}} +_{k\in K_i} \Gamma_k = +_{i\in\mathtt{I}}\Gamma_i$$

and

$$\begin{array}{rcl} \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi) & = & +_{k\in K}\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_k) \\ & = & +_{i\in\mathtt{I}} +_{k\in K_i} \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_k) \\ & = & +_{i\in\mathtt{I}}\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_i) \end{array}$$

$\Leftarrow$) If there exist $(\Phi_i \triangleright \Gamma_i \vdash t : M_i)_{i\in\mathtt{I}}$, then each Φ_i must be of the form:

$$\frac{(\,\Phi_k \triangleright \Gamma_k \vdash t : A_k)_{k\in K_i}}{+_{k\in K_i}\Gamma_k \vdash t : \{\!\{A_k\}\!\}_{k\in K_i}}\ \mathtt{Many}$$

where $\Gamma_i = +_{k\in K_i}\Gamma_k$, $M_i = \{\!\{A_k\}\!\}_{k\in K_i}$, and $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_i) = +_{k\in K_i}\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_k)$. Then, we can build Φ as follows:

$$\frac{(\,\Phi_k \triangleright \Gamma_k \vdash t : A_k)_{k\in K_i, i\in\mathtt{I}}}{+_{k\in K_i, i\in\mathtt{I}}\Gamma_k \vdash t : \{\!\{A_k\}\!\}_{k\in K_i, i\in\mathtt{I}}}\ \mathtt{Many}$$

where $\Gamma = +_{k\in K_i, i\in \mathtt{I}}\Gamma_k = +_{i\in\mathtt{I}}\Gamma_i$, and $M = \{\!\{A_k\}\!\}_{k\in K_i,i\in\mathtt{I}} = \{\!\{A_i\}\!\}_{i\in\mathtt{I}}$. And we can conclude with $+_{i\in\mathtt{I}}\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_i) = +_{k\in K_i,i\in\mathtt{I}}\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_k) = \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi)$.

□

Lemma 6.30 (Contexts and Free Variables for Pattern-Matching)**.** *Let $t \in \Lambda^{PM}$ and $\Phi \triangleright \Gamma \vdash t : A$. Then,* $\mathtt{dom}(\Gamma) \subseteq \mathtt{fv}(t)$.

Proof. We prove a slightly more general statement by allowing Φ_t to conclude with either a term type A or a multi-type M.

The proof follows by induction on Φ. We reason according to the last rule:

- Rule `Ax`. Then, $t = x$ and $\Gamma = (x : \{\!\{A\}\!\})$. Therefore, we can conclude immediately.
- Rule `Many`. Suppose $\Phi_i \triangleright \Gamma_i \vdash t : A_i$ are the premises of the rule. Then, $\Gamma = +_{i\in\mathtt{I}}\Gamma_i$. By the *i.h.* over each Φ_i, we know $\mathtt{dom}(\Gamma_i) \subseteq \mathtt{fv}(t)$. Therefore, $\mathtt{dom}(\Gamma) = \bigcup_{i\in\mathtt{I}} \mathtt{dom}(\Gamma_i) \subseteq \bigcup_{i\in\mathtt{I}} \mathtt{fv}(t) = \mathtt{fv}(t)$.
- Rule `Abs`. Then, $t = \lambda p.u$. Suppose $\Phi_t \triangleright \Gamma_u \vdash u : A$ and $\Pi \triangleright \Gamma|_p \Vdash p : M$ are the premises of the rule for u and p, respectively. Then, $\Gamma = \Gamma_u \setminus\!\setminus \mathcal{X}p$. By the *i.h.* over Φ_t, we know $\mathtt{dom}(\Gamma_u) \subseteq \mathtt{fv}(u)$. Therefore, $\mathtt{dom}(\Gamma_u) \setminus \mathcal{X}p \subseteq \mathtt{fv}(u) \setminus \mathcal{X}p$. And we can conclude since $\mathtt{dom}(\Gamma_u) \setminus \mathcal{X}p = \mathtt{dom}(\Gamma_u \setminus\!\setminus \mathcal{X}p) = \mathtt{dom}(\Gamma)$ and $\mathtt{fv}(u) \setminus \mathcal{X}p = \mathtt{fv}(\lambda p.u)$.
- Rule `Ans`. Then, $t = \lambda p.u$ and $\Gamma = \emptyset$. So we can conclude immediately.
- Rule `App`. Then, $t = us$. Suppose $\Phi_u \triangleright \Gamma_u \vdash u : M \to A$ and $\Phi_s \triangleright \Gamma_s \vdash s : M$ are the premises of the rule for u and s, respectively. Then, $\Gamma = \Gamma_u + \Gamma_s$. By the *i.h.* over Φ_u (resp. Φ_s), we know $\mathtt{dom}(\Gamma_u) \subseteq \mathtt{fv}(u)$ (resp. $\mathtt{dom}(\Gamma_s) \subseteq \mathtt{fv}(s)$). Therefore, $\mathtt{dom}(\Gamma) = \mathtt{dom}(\Gamma_u) \cup \mathtt{dom}(\Gamma_s) \subseteq \mathtt{fv}(u) \cup \mathtt{fv}(s) = \mathtt{fv}(us)$.
- Rule `Data`. Then, $t = \mathtt{c}(u_i)_n$. Suppose $\Phi_i \triangleright \Gamma_i \vdash u_i : M_i$ are the premises of the rule for each u_i. Then, $\Gamma = +_{i\in\{1,\dots,n\}}\Gamma_i$. By the *i.h.* over each Φ_i, we know $\mathtt{dom}(\Gamma_i) \subseteq \mathtt{fv}(u_i)$. Therefore, $\mathtt{dom}(\Gamma) = \bigcup_{i\in\{1,\dots,n\}} \mathtt{dom}(\Gamma_i) \subseteq \bigcup_{i\in\{1,\dots,n\}} \mathtt{fv}(u_i) = \mathtt{fv}(\mathtt{c}(u_i)_n)$.
- Rule `Match`. Then, $t = u[p\backslash s]$. Suppose $\Phi_u \triangleright \Gamma_u \vdash u : A$, $\Pi \triangleright \Gamma|_p \Vdash p : M$, and $\Phi_s \triangleright \Gamma_s \vdash s : M$ are the premises of the rule for u, p, and s, respectively. Then, $\Gamma = (\Gamma_u \setminus\!\setminus \mathcal{X}p) + \Gamma_s$. By the *i.h.* over Φ_u (resp. Φ_s), we know $\mathtt{dom}(\Gamma_u) \subseteq \mathtt{fv}(u)$ (resp. $\mathtt{dom}(\Gamma_s) \subseteq \mathtt{fv}(s)$). Therefore, $\mathtt{dom}(\Gamma) = \mathtt{dom}(\Gamma_u \setminus\!\setminus \mathcal{X}p) \cup \mathtt{dom}(\Gamma_s) = (\mathtt{dom}(\Gamma_u) \setminus \mathcal{X}p) \cup \mathtt{dom}(\Gamma_s) \subseteq (\mathtt{fv}(u) \setminus \mathcal{X}p) \cup \mathtt{fv}(s) = \mathtt{fv}(u[p\backslash s])$.
- Rule (`Case`). Then, $t = \mathtt{case}\ u\ \mathtt{of}\ (\hat{p}_i.s_i)_n$. Fix an arbitrary branch $k \in \{1,\dots,n\}$. Now, suppose $\Phi_u \triangleright \Gamma_u \vdash u : M$, $\Pi \triangleright \Gamma_u|_{\hat{p}_k} \Vdash \hat{p}_k : M$ and $\Phi_{s_k} \triangleright \Gamma_{s_k} \vdash s_k : A$ are the premises of the rule for u, $\hat{p}_k$, and s_k, respectively. Then, $\mathtt{dom}(\Gamma) = \mathtt{dom}(\Gamma_{s_k} \setminus\!\setminus \mathcal{X}\hat{p}_k) \cup \mathtt{dom}(\Gamma_u)$. By the

i.h. over Φ_u (resp. Φ_{s_k}), we know $\mathtt{dom}(\Gamma_u) \subseteq \mathtt{fv}(u)$ (resp. $\mathtt{dom}(\Gamma_{s_k}) \subseteq \mathtt{fv}(s_k)$). Therefore, $\mathtt{dom}(\Gamma) = \mathtt{dom}(\Gamma_{s_k} \setminus\!\setminus \mathcal{X}\hat{p}_k) \cup \mathtt{dom}(\Gamma_u) = (\mathtt{dom}(\Gamma_{s_k}) \setminus \mathcal{X}\hat{p}_k) \cup \mathtt{dom}(\Gamma_u) = \mathtt{fv}(u) \cup (\mathtt{fv}(s_k) \setminus \mathcal{X}\hat{p}_k) \subseteq \mathtt{fv}(u) \bigcup_{i\in\{1,\dots,n\}} (\mathtt{fv}(s_i) \setminus \mathcal{X}\hat{p}_i) = \mathtt{fv}(\mathtt{case}\ u\ \mathtt{of}\ (\hat{p}_i.s_i)_n)$.

If t is closed, then $\mathtt{fv}(t) = \emptyset$ by definition. Therefore, $\Gamma = \emptyset$ by the previous induction. □

Lemma 6.31 (Zero Weight Tight Type Derivations Characterize (Clash-Free) Normal Forms)**.** *Let $t \in \Lambda^{PM}_{\circ}$. If $\Phi \rhd_{\mathcal{PM}} \emptyset \vdash t : A$ is tight, then $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi) = 0 \cdot \mathtt{App} + 0 \cdot \mathtt{VPat} + 0 \cdot \mathtt{DPat}$ if and only if $t \in \mathtt{NO}^{\mathtt{CF}}$.*

Proof. We start by noting that $t \in \mathtt{NO}^{\mathtt{CF}}$ if and only if $t = \lambda x.u$ or $t = \mathtt{c}T$ by Lemma 6.20.

(⇒) Let $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi) = 0 \cdot \mathtt{App} + 0 \cdot \mathtt{VPat} + 0 \cdot \mathtt{DPat}$. We proceed by cases, according to the last rule R of Φ:

- Case $R = \mathtt{Ax}$. Then, $t = x$. Contradiction with $t \in \Lambda^{PM}_{\circ}$. Therefore, this case does not apply.
- Case $R = \mathtt{Abs}$. Then, A is an arrow type. Contradiction with Φ being tight. Therefore, this case does not apply.
- Case $R = \mathtt{Ans}$. Then, $t = \lambda x.u$. Therefore, we can conclude immediately.
- Case $R = \mathtt{App}$. Contradiction with $\#_{\mathtt{App}}(\Phi) = 0 \cdot \mathtt{App}$. Therefore, this case does not apply.
- Case $R = \mathtt{Many}$. Contradiction with Φ ending with a type. Therefore, this case does not apply.
- Case $R = \mathtt{Data}$. Then, $t = \mathtt{c}T$. Therefore, we can conclude immediately.
- Case $R = \mathtt{Match}$. Then, $t = r[p\backslash u]$ and Φ must be of the form

$$\frac{\Phi_r \rhd_{\mathcal{PM}} \Gamma \vdash r : A \qquad \Phi_p \rhd_{\mathcal{PM}} \Gamma|_{\mathtt{vars}(p)} \Vdash p : M \qquad \Phi_u \rhd_{\mathcal{PM}} \emptyset \vdash u : M}{\Gamma \setminus\!\setminus \mathtt{vars}(p) \vdash r[p\backslash u] : A}\ \mathtt{Match}$$

  such that $\Gamma \setminus\!\setminus \mathtt{vars}(p) = \emptyset$. Notice that $\#_{\mathtt{VPat},\mathtt{DPat}}(\Phi_p) \neq 0 \cdot \mathtt{VPat} + 0 \cdot \mathtt{DPat}$, since any derivation typing a pattern necessarily uses either $\mathtt{VPat}$ or $\mathtt{DPat}$ at least once. Contradiction with $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi) = 0 \cdot \mathtt{App} + 0 \cdot \mathtt{VPat} + 0 \cdot \mathtt{DPat}$. Therefore, this case does not apply.

- Case $R = \mathtt{Case}$. Then, $t = \mathtt{case}\ r\ \mathtt{of}\ (\hat{p}_i.u_i)_n$, and Φ must be of the form

$$\frac{\Phi_r \rhd_{\mathcal{PM}} \emptyset \vdash r : M \qquad \Phi_{\hat{p}_k} \rhd_{\mathcal{PM}} \Gamma|_{\mathtt{vars}(\hat{p}_k)} \Vdash \hat{p}_k : M \qquad \Phi_{u_k} \rhd_{\mathcal{PM}} \Gamma \vdash u_k : A}{\Gamma \setminus\!\setminus \mathtt{vars}(\hat{p}_i) \vdash \mathtt{case}\ t\ \mathtt{of}\ (\hat{p}_i.u_i)_n : A}\ \mathtt{Case}$$

such that $k \in \{1, \ldots, n\}$, and $\Gamma \backslash\backslash \mathtt{vars}(\hat{p}_i) = \emptyset$. Notice that $\#_{\mathtt{VPat},\mathtt{DPat}}(\Phi_{\hat{p}_k}) \neq 0 \cdot \mathtt{VPat} + 0 \cdot \mathtt{DPat}$. Contradiction with $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi) = 0 \cdot \mathtt{App} + 0 \cdot \mathtt{VPat} + 0 \cdot \mathtt{DPat}$. Therefore, this case does not apply.

($\Leftarrow$) Let $t \in \mathtt{NO}^{\mathtt{CF}}$. Then, $t = \lambda x.u$ or $t = \mathtt{c}T$. Therefore, Φ must end with rule `Ans` or `Data`. If Φ ends with rule `Ans`, then clearly $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi) = 0 \cdot \mathtt{App} + 0 \cdot \mathtt{VPat} + 0 \cdot \mathtt{DPat}$. If Φ ends with rule `Data`, then $t = \mathtt{c}T$. Let $T = (u_1, \ldots, u_n)$, for some $n \in \mathtt{N}$. Then, Φ must be of the form

$$\dfrac{\left(\dfrac{}{\emptyset \vdash u_i : \{\!\{\ \}\!\}}\ \mathtt{Many}\right)_{i \in \{1,\ldots,n\}}}{\emptyset \vdash \mathtt{c}(u_1, \ldots, u_n) : \mathtt{c}(\{\!\{\ \}\!\})_n}\ \mathtt{Data}$$

Therefore, we can conclude that $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi) = 0 \cdot \mathtt{App} + 0 \cdot \mathtt{VPat} + 0 \cdot \mathtt{DPat}$.

□

Lemma 6.32 (Typability Implies Clash-Freeness)**.** *Let $t \in \Lambda^{PM}$. If $\Gamma \vdash t : A$, then t is clash-free.*

Proof. We reason by contraposition. Assume t is *not* clash-free. Then, there exists $t' \in \mathtt{CL}$, such that $t \twoheadrightarrow_{\mathtt{PM}} t'$. Therefore, either $t = t'$, in which case $\Phi \triangleright \Gamma \vdash t' : A$ by hypothesis, or $t \to_{\mathtt{PM}} t'$ and by Lemma 6.37, we can conclude that there exists $\Phi \triangleright \Gamma \vdash t' : A$. Therefore, if we show that t' is not $\mathcal{PM}$-typable, then we are done. The proof follows by induction on $t \in \mathtt{CL}$. For each case, we reason by contradiction, assuming there exist Φ:

1. Let $t \in \mathtt{CL}^0$:

   - Case $t = \mathtt{L}(\!|\mathtt{c}T|\!)s$. Then, Φ must end with rule `App` and $\mathtt{L}(\!|\mathtt{c}T|\!)$ must be assigned an arrow type. However, we can easily conclude by a simple inspection of the rules that $\mathtt{c}T$ can only be assigned a data type. Therefore, we can also conclude by that $\mathtt{L}(\!|\mathtt{c}T|\!)$ can only be assigned a data type. Thus, we can conclude that there is no Φ, by contradiction.
   - Case $t = u[\mathtt{c}P \backslash \mathtt{L}(\!|\lambda q.s|\!)]$. Then, Φ must end with rule `Match` and $\mathtt{L}(\!|\lambda q.s|\!)$ must be assigned the same type as $\mathtt{c}P$. A simple inspection of the rules tells us that $\mathtt{c}P$ can only be assigned a data type. However, it is also clear from the rules that $\lambda q.s$ can only be assigned an arrow type. Therefore, $\mathtt{L}(\!|\lambda q.s|\!)$ can only be assigned an arrow type. Thus, we can conclude that there is no Φ, by contradiction.
   - Case $t = u[\mathtt{c}P \backslash \mathtt{L}(\!|\mathtt{c}'T|\!)]$, where $\mathtt{c} \neq \mathtt{c}'$. Then, Φ must end with rule `Match` and $\mathtt{L}(\!|\mathtt{c}'T|\!)$ must be assigned the same type as $\mathtt{c}P$. A simple inspection of the rules tells us that $\mathtt{c}P$ can only be assigned a data type with top symbol $\mathtt{c}$. Consequently, $\mathtt{L}(\!|\mathtt{c}'T|\!)$ will also have to be assigned a data type with top symbol $\mathtt{c}$. However, it is also clear from the rules that $\mathtt{c}'T$ can only be assigned a data type with top symbol $\mathtt{c}'$. Therefore, $\mathtt{L}(\!|\mathtt{c}'T|\!)$ can only

be assigned a data type starting constructor $\mathtt{c}'$. Thus, we can conclude that there is no Φ, by contradiction.

- Case $t = \mathtt{case}\ \mathtt{L}(\!|\lambda p.u|\!)\ \mathtt{of}\ (\mathtt{c}_i P_i.s_i)_n$. Then, Φ must end with rule $\mathtt{Case}$ and $\mathtt{L}(\!|\lambda p.u|\!)$ must be assigned the same type one of the patterns $\mathtt{c}_i P_i$. A simple inspection of the rules tells us that all data patterns must be typed with data types. However, it is also clear from the rules that $\lambda p.u$ can only be assigned an arrow type. Therefore, $\mathtt{L}(\!|\lambda p.u|\!)$ can only be assigned an arrow type. Thus, we can conclude that there is no Φ, by contradiction.
- Case $t = \mathtt{case}\ \mathtt{L}(\!|\mathtt{c}T|\!)\ \mathtt{of}\ (\mathtt{c}_i P_i.u_i)_n$, where $\mathtt{c} \notin \{\mathtt{c}_1, \ldots, \mathtt{c}_n\}$. Then, Φ must end with rule $\mathtt{Case}$ and $\mathtt{L}(\!|\mathtt{c}T|\!)$ must be assigned the same type as one of the patterns $\mathtt{c}_i P_i$. A simple inspection of the rules tells us that data patterns must be typed with data types starting with matching constructors. However, it is also clear from the rules that $\mathtt{c}T$ must be assigned a data type with top symbol $\mathtt{c}$. Therefore, $\mathtt{L}(\!|\mathtt{c}T|\!)$ can only be assigned a data type with top symbol $\mathtt{c}$. Thus, we can conclude there is no Φ, by contradiction.

2. Let $t \in \mathtt{CL}$:
   - Case $t \in \mathtt{CL}^0$. We can conclude by the previous point.
   - If $t \notin \mathtt{CL}^0$, then t contains a strict subterm $u \in \mathtt{CL}$. By inversion on the typing rules, typability of t would imply typability of u, which contradicts the induction hypothesis.

□

Lemma 6.33 (Patterns are Always Typable)**.** *Given any typing context Γ and pattern p, there exists $\Pi \rhd \Gamma|_p \Vdash p : M$. Moreover, if $p = \mathtt{c}(q_i)_n$, then there exist $\Pi_i \rhd_{\mathcal{PM}} \Gamma|_{q_i} \Vdash q_i : M_i$, such that $M = \{\!\{\mathtt{c}(M_i)_n\}\!\}$.*

Proof. The proof follows by induction on p:

- Case $p = x$. Then, it is always the case that $\Gamma|_x(x) = M$ from some multi-type M (possibly $\{\!\{\ \}\!\}$). So we can build Π using rule $\mathtt{VPat}$.
- Case $p = \mathtt{c}(q_i)_n$. By the *i.h.*, we know there exist $\Pi_i \rhd \Gamma|_{q_i} \Vdash q_i : M_i$ for each q_i. So we can build $\Pi \rhd +_{i\in\{1,\ldots,n\}}\Gamma|_{q_i} \Vdash \mathtt{c}(q_i)_n : \{\!\{\mathtt{c}(M_i)_n\}\!\}$ using rule $\mathtt{DPat}$. We can conclude since $\Gamma|_p = +_{\{1,\ldots,n\}}(\Gamma|_{q_i})$.

□

Lemma 6.34 (Tight Typability of Closed Clash-Free PM-Normal Forms)**.** *Let $t \in \Lambda_\circ^{PM}$. If $t \in \mathtt{NO}^{\mathtt{CF}}$, then there is a derivation $\Phi \rhd_{\mathcal{PM}} \emptyset \vdash t : A$ that is tight.*

Proof. If $t \in \mathtt{NO}^{\mathtt{CF}}$, then $t = \lambda x.u$ or $t = \mathtt{c}T$:

- Let $t = \lambda x.u$. Then, we can build Φ using rule `Ans`. Since `Ans` has no premises and assigns type $\star$ to $\lambda x.u$, the resulting type derivation is tight.
- Let $t = \mathtt{c}T$. Let $T = (u_1, \ldots, u_n)$, for some $n \in \mathtt{N}$. Then, we can build Φ as follows:

$$\dfrac{\left(\dfrac{}{\varnothing \vdash u_i : \{\!\{\ \}\!\}}\ \mathtt{Many}\right)_{i \in \{1,\ldots,n\}}}{\varnothing \vdash \mathtt{c}(u_1, \ldots, u_n) : \mathtt{c}(\{\!\{\ \}\!\})_n}\ \mathtt{Data}$$

  We can conclude since the above type derivation is tight.

□

Lemma 6.35 (Typability Characterizes Closed Clash-Free Normal Forms)**.** *Let $t \in \Lambda_{\circ}^{PM}$. Then, $t \in \mathtt{NO}^{\mathtt{CF}}$ if and only if $t \in \mathtt{NO}$ and t is $\mathcal{PM}$-typable.*

Proof. ($\Rightarrow$) $t \in \mathtt{NO}^{\mathtt{CF}}$, then t is a `PM`-normal form by Lemma 6.19. Moreover, since t is closed and clash-free, it admits a tight typing derivation by Lemma 6.34. In particular, t is $\mathcal{PM}$-typable.

($\Leftarrow$) Since $t \in \mathtt{NO}$, t is a `PM`-normal form. Moreover, since t is $\mathcal{PM}$-typable, it is clash-free by Lemma 6.32. Therefore, t is a closed clash-free `PM`-normal form, and thus $t \in \mathtt{NO}^{\mathtt{CF}}$ by Lemma 6.20.

□

D.2.1 Quantitative Soundness

Lemma 6.36 (Weighted Substitution)**.** *Let $t, u \in \Lambda^{PM}$ and assume $\Phi_t \rhd_{\mathcal{PM}} \Gamma; x : M \vdash t : A$ and $\Phi_u \rhd_{\mathcal{PM}} \Delta \vdash u : M$. Then, there exists $\Phi_{t\{x \setminus u\}} \rhd \Gamma + \Delta \vdash t\{x \setminus u\} : A$, such that $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{t\{x\setminus u\}}) = \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_t) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u)$.*

Proof. We generalize the original statement by allowing Φ_t to conclude with either a term type A or a multiset type M.

If $\Phi_t \rhd_{\mathcal{PM}} \Gamma; x : M \vdash t : T$, where $T \in \{A, M\}$, and $\Phi_u \rhd_{\mathcal{PM}} \Delta \vdash u : M$, then there exists $\Phi_{t\{x\setminus u\}} \rhd_{\mathcal{PM}} \Gamma + \Delta \vdash t\{x \setminus u\} : T$, such that $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{t\{x\setminus u\}}) = \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_t) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u)$.

The proof follows by induction on Φ_t. We reason according to the last rule R of Φ_t:

- Case $R = \mathtt{Ax}$. Then, $t = y$ and we must consider two different cases:
  - If $y = x$, then $t\{x \setminus u\} = u$ and Φ_t must be of the following form:

$$\dfrac{}{x : \{\!\{A\}\!\} \vdash x : A}\ \mathtt{Ax}$$

where $T = A$, $\Gamma = \emptyset$, and $M = \{\!\{A\}\!\}$. Therefore, Φ_u must be of the following form:

$$\frac{\Phi'_u + \Delta \vdash u : A}{\Delta \vdash u : \{\!\{A\}\!\}}\ \texttt{Many}$$

So, we can pick $\Phi_{t\{x\setminus u\}} = \Phi'_u$ and conclude since $\Gamma + \Delta = \Delta$, and

$$\begin{aligned}\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_{t\{x\setminus u\}}) &= \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi'_u)\\ &= \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u)\\ &= \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_t) + \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u).\end{aligned}$$

– If $y \neq x$, then $y\{x \setminus u\} = y$ and Φ_t must be of the following form:

$$\frac{}{y : \{\!\{A\}\!\}; x : \{\!\{\ \}\!\} \vdash y : A}\ \texttt{Ax}$$

where $T = A$, $\Gamma = (y : \{\!\{A\}\!\})$, and $M = \{\!\{\ \}\!\}$. Therefore, Φ_u must be of the following form:

$$\frac{}{\emptyset \vdash u : \{\!\{\ \}\!\}}$$

where $\Delta = \emptyset$. So, we can pick $\Phi_{t\{x\setminus u\}} = \Phi_t$ and conclude since $\Gamma + \Delta = \Gamma$ and

$$\begin{aligned}\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_{t\{x\setminus u\}}) &= \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_t)\\ &= \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_t) + \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u).\end{aligned}$$

- Case $R = \texttt{Many}$. Then, Φ_t must be of the following form:

$$\frac{\left(\Phi^i_t \rhd_{\mathcal{PM}} \Gamma_i; x : M_i \vdash t : A_i\right)_{i\in \mathtt{I}}}{+_{i\in\mathtt{I}}\,(\Gamma_i; x : M_i) \vdash t : \{\!\{A_i\}\!\}_{i\in\mathtt{I}}}\ \texttt{Many}$$

where $T = \{\!\{A_i\}\!\}_{i\in\mathtt{I}}$, $(\Gamma; x : M) = +_{i\in\mathtt{I}}(\Gamma_i; x : M_i)$, and thus $\Gamma = +_{i\in\mathtt{I}}\Gamma_i$ and $M = \sqcup_{i\in\mathtt{I}} M_i$. By Lemma 6.29 over Φ_u, we know there exist $(\Phi^i_u \rhd_{\mathcal{PM}} \Delta_i \vdash u : M_i)_{i\in\mathtt{I}}$, such that $\Delta = +_{i\in\mathtt{I}}\Delta_i$ and $\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u) = +_{i\in\mathtt{I}}\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi^i_u)$. By the induction hypothesis over each Φ^i_t, we know there exists $\Phi^i_{t\{x\setminus u\}} \rhd_{\mathcal{PM}} \Gamma_i + \Delta_i \vdash t\{x \setminus u\} : A_i$, for each $i \in \mathtt{I}$, such that $\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi^i_{t\{x\setminus u\}}) = \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi^i_t) + \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi^i_u)$. Therefore, we can build $\Phi_{t\{x\setminus u\}}$ as follows:

$$\frac{\left(\Phi^i_{t\{x\setminus u\}}\right)_{i\in\mathtt{I}}}{+_{i\in\mathtt{I}}\Gamma_i +_{i\in\mathtt{I}} \Delta_i \vdash t\{x \setminus u\} : \{\!\{A_i\}\!\}_{i\in\mathtt{I}}}\ \texttt{Many}$$

We can conclude since $\Gamma + \Delta = +_{i\in\mathtt{I}}\Gamma_i +_{i\in\mathtt{I}} \Delta_i$ and

$$\begin{aligned}\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_{t\{x\setminus u\}}) &= +_{i\in\mathtt{I}}\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi^i_{t\{x\setminus u\}})\\ &= +_{i\in\mathtt{I}}\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi^i_t + \Phi^i_u)\\ &= +_{i\in\mathtt{I}}\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi^i_t) +_{i\in\mathtt{I}} \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi^i_u)\\ &= \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_t) + \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u).\end{aligned}$$

- Case $R = \texttt{Abs}$. Then, Φ_t must be of the following form:

$$\frac{\Phi_s \triangleright_{\mathcal{PM}} \Gamma_s; x : M \vdash s : A \qquad \Pi \triangleright_{\mathcal{PM}} (\Gamma_s; x : M)|_p \Vdash p : M_0}{(\Gamma_s; x : M) \setminus\!\setminus \texttt{vars}(p) \vdash \lambda p.s : M_0 \to A}\ \texttt{Abs}$$

where $t = \lambda p.s$, $T = M_0 \to A$, and $\Gamma = (\Gamma_s; x : M) \setminus\!\setminus \texttt{vars}(p)$. Moreover, by α-conversion we can assume $\texttt{vars}(p) \cap \{x\} = \emptyset$ so that $(\Gamma_s; x : M) \setminus\!\setminus \texttt{vars}(p) = (\Gamma_s \setminus\!\setminus \texttt{vars}(p)); x : M = (\Gamma; x : M)$, and thus $\Gamma = \Gamma_s \setminus\!\setminus \texttt{vars}(p)$ and $(\Gamma_s; x : M)|_p = \Gamma_s|_p$. By the induction hypothesis over Φ_s, we know there exists $\Phi_{s\{x\setminus u\}} \triangleright_{\mathcal{PM}} \Gamma_s + \Delta \vdash s\{x \setminus u\} : A$, such that $\#_{\texttt{App,VPat,DPat}}(\Phi_{s\{x\setminus u\}}) = \#_{\texttt{App,VPat,DPat}}(\Phi_s) + \#_{\texttt{App,VPat,DPat}}(\Phi_u)$. Therefore, we can build $\Phi_{t\{x\setminus u\}}$ as follows:

$$\frac{\Phi_{s\{x\setminus u\}} \qquad \Pi}{(\Gamma_s + \Delta) \setminus\!\setminus \texttt{vars}(p) \vdash \lambda p.s\{x \setminus u\} : M_0 \to A}\ \texttt{Abs}$$

By α-conversion, we may assume that $\texttt{vars}(p) \cap \texttt{fv}(u) = \emptyset$. Thus, $\lambda p.s\{x \setminus u\} = (\lambda p.s)\{x \setminus u\}$ and $\texttt{dom}(\Delta) \cap \texttt{vars}(p) = \emptyset$ (by Lemma 6.30). Therefore, we know $(\Gamma_s + \Delta) \setminus\!\setminus \texttt{vars}(p) = (\Gamma_s \setminus\!\setminus \texttt{vars}(p)) + \Delta$, and we can conclude since $\Gamma + \Delta = (\Gamma_s \setminus\!\setminus \texttt{vars}(p)) + \Delta$ and

$$\begin{aligned}
\#_{\texttt{App,VPat,DPat}}(\Phi_{t\{x\setminus u\}}) &= \#_{\texttt{App,VPat,DPat}}(\Phi_{s\{x\setminus u\}}) + \#_{\texttt{App,VPat,DPat}}(\Pi) \\
&= \#_{\texttt{App,VPat,DPat}}(\Phi_s) + \#_{\texttt{App,VPat,DPat}}(\Phi_u) + \#_{\texttt{App,VPat,DPat}}(\Pi) \\
&= \#_{\texttt{App,VPat,DPat}}(\Phi_t) + \#_{\texttt{App,VPat,DPat}}(\Phi_u).
\end{aligned}$$

- Case $R = \texttt{Ans}$. Then, Φ_t must be of the following form:

$$\frac{}{\vdash \lambda p.s : \star}\ \texttt{Ans}$$

where $T = \star$, $t = \lambda p.s$, and $(\Gamma; x : M) = \emptyset$, and thus $\Gamma = \emptyset$ and $M = \{\!\{\ \}\!\}$. Therefore, Φ_u must be of the following form:

$$\frac{}{\vdash u : \{\!\{\ \}\!\}}$$

where $\Delta = \emptyset$. So, we can build $\Phi_{t\{x\setminus u\}}$ as follows:

$$\frac{}{\vdash \lambda p.(s\{x \setminus u\}) : \star}\ \texttt{Ans}$$

By α-conversion, we may assume that $\texttt{vars}(p) \cap \texttt{fv}(u) = \emptyset$. Thus, $\lambda p.s\{x \setminus u\} = (\lambda p.s)\{x \setminus u\}$. We can conclude since $\Gamma + \Delta = \emptyset$ and

$$\#_{\texttt{App,VPat,DPat}}(\Phi_{t\{x\setminus u\}}) = 0 = \#_{\texttt{App,VPat,DPat}}(\Phi_t) + \#_{\texttt{App,VPat,DPat}}(\Phi_u).$$

- Case $R = \texttt{App}$. Then, Φ_t must be of the following form:

$$\frac{\Phi_s \triangleright_{\mathcal{PM}} \Gamma_s; x : M_1 \vdash s : M_0 \to A \qquad \Phi_r \triangleright_{\mathcal{PM}} \Gamma_r; x : M_2 \vdash r : M_0}{(\Gamma_s; x : M_1) + (\Gamma_r; x : M_2) \vdash sr : A}\ \texttt{App}$$

where $t = sr$, $T = A$, $(\Gamma; x : M) = (\Gamma_s; x : M_1) + (\Gamma_r; x : M_2) = (\Gamma_s + \Gamma_r); x : M_1 \sqcup M_2$, and thus $\Gamma = \Gamma_s + \Gamma_r$, and $M = M_1 \sqcup M_2$. By Lemma 6.29 over Φ_u, we know there exist $(\Phi_u^i \triangleright_{\mathcal{PM}} \Delta_i \vdash u : M_i)_{i\in\{1,\dots,2\}}$, such that $\Delta = \Delta_1 + \Delta_2$, $M = M_1 \sqcup M_2$ and $\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u) = \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u^1) + \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u^2)$. By the induction hypothesis over Φ_s (resp. Φ_r), we know there exists $\Phi_{s\{x\setminus u\}} \triangleright_{\mathcal{PM}} \Gamma_s + \Delta_1 \vdash s\{x \setminus u\} : M_0 \to A$ (resp. $\Phi_{r\{x\setminus u\}} \triangleright_{\mathcal{PM}} \Gamma_r + \Delta_2 \vdash r\{x \setminus u\} : M_0$), such that $\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_{s\{x\setminus u\}}) = \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_s) + \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u^1)$ (resp. $\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_{r\{x\setminus u\}}) = \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_r) + \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u^2)$). Therefore, we can build $\Phi_{t\{x\setminus u\}}$ as follows:

$$\frac{\Phi_{s\{x\setminus u\}} \qquad \Phi_{r\{x\setminus u\}}}{\Gamma_s + \Delta_1 + \Gamma_r + \Delta_2 \vdash s\{x \setminus u\}r\{x \setminus u\} : A}\ \texttt{App}$$

We can conclude since $s\{x \setminus u\}\, r\{x \setminus u\} = (sr)\{x \setminus u\}$, $\Gamma + \Delta = \Gamma_s + \Gamma_r + \Delta_1 + \Delta_2$, and

$$\begin{aligned}
\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_{t\{x\setminus u\}}) &= \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_{s\{x\setminus u\}}) + \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_{r\{x\setminus u\}}) \\
&+ 1 \cdot \texttt{App} \\
&= (\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_s) + \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_r) + 1 \cdot \texttt{App}) \\
&+ (\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u^1) + \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u^2)) \\
&= \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_t) + \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u).
\end{aligned}$$

- Case $R = \texttt{Data}$. Then, Φ_t must be of the following form:

$$\frac{\left(\Phi_{s_i} \triangleright_{\mathcal{PM}} \Gamma_i; x : M_i \vdash s_i : M_i^0\right)_{i\in\{1,\dots,n\}}}{+_{i\in\{1,\dots,n\}}(\Gamma_i; x : M_i) \vdash \mathtt{c}(s_i)_n : \mathtt{c}(M_i^0)_n}\ \texttt{Data}$$

where $t = \mathtt{c}(s_i)_n$, $T = \mathtt{c}(M_i^0)_n$, $(\Gamma; x : M) = +_{i\in\{1,\dots,n\}}(\Gamma_i; x : M_i) = +_{i\in\{1,\dots,n\}}\Gamma_i; x : \sqcup_{i\in\{1,\dots,n\}}M_i$, and thus $\Gamma = +_{i\in\{1,\dots,n\}}\Gamma_i$ and $M = \sqcup_{i\in\{1,\dots,n\}}M_i$. By Lemma 6.29 over Φ_u, we know there exist $(\Phi_u^i \triangleright_{\mathcal{PM}} \Delta_i \vdash u : M_i)_{\{1,\dots,n\}}$, such that $\Delta = +_{i\in\mathtt{I}}\Delta_i$ and $\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u) = +_{i\in\{1,\dots,n\}}\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u^i)$. By the induction hypothesis over each Φ_{s_i}, we know there exists $\Phi_{s_i\{x\setminus u\}} \triangleright_{\mathcal{PM}} \Gamma_i + \Delta_i \vdash s_i\{x \setminus u\} : M_i^0$, such that $\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_{s_i\{x\setminus u\}}) = \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_{s_i}) + \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u^i)$. Therefore, we can build $\Phi_{t\{x\setminus u\}}$ as follows:

$$\frac{\left(\Phi_{s_i\{x\setminus u\}}\right)_{i\in\{1,\dots,n\}}}{+_{i\in\{1,\dots,n\}}(\Gamma_i + \Delta_i) \vdash \mathtt{c}(s_i\{x \setminus u\})_n : \mathtt{c}(M_i^0)_n}\ \texttt{Data}$$

We can conclude since $\mathtt{c}(s_i\{x \setminus u\})_n = \mathtt{c}(s_i)_n\{x \setminus u\}$, $\Gamma + \Delta = +_{i\in\{1,\dots,n\}}(\Gamma_i + \Delta_i)$, and

$$\begin{aligned}
\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_{t\{x\setminus u\}}) &= +_{i\in\{1,\dots,n\}}\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_{s_i\{x\setminus u\}}) \\
&= +_{i\in\{1,\dots,n\}}(\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_{s_i}) + \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u^i)) \\
&= +_{i\in\{1,\dots,n\}}\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_{s_i}) +_{i\in\{1,\dots,n\}} \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u^i) \\
&= \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_t) + \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u).
\end{aligned}$$

- Case $R = \texttt{Match}$. Then, Φ_t must be of the following form:

$$\frac{\Phi_s \rhd_{\mathcal{PM}} \Gamma_s; x : M_1 \vdash s : A \quad \Pi \rhd_{\mathcal{PM}} (\Gamma_s; x : M_1)|_p \Vdash p : M_0 \quad \Phi_r \rhd_{\mathcal{PM}} \Gamma_r; x : M_2 \vdash r : M_0}{(\Gamma_s; x : M_1) \setminus\setminus \texttt{vars}(p) + \Gamma_r; x : M_2 \vdash s[p \backslash r] : A}\ \texttt{Match}$$

where $t = s[p \backslash r]$, $T = A$. Moreover, by α-conversion, we may assume that $\texttt{vars}(p) \cap \texttt{fv}(u) = \emptyset$ so that $(\Gamma; x : M) = (\Gamma_s; x : M_1) \setminus\setminus \texttt{vars}(p) + \Gamma_r; x : M_2 = ((\Gamma_s \setminus\setminus \texttt{vars}(p)) + \Gamma_r); x : M_1 \sqcup M_2$, and thus $\Gamma = (\Gamma_s \setminus\setminus \texttt{vars}(p)) + \Gamma_r$, $M = M_1 \sqcup M_2$ and $(\Gamma_s; x : M_1)|_p = \Gamma_s|_p$. By Lemma 6.29 over Φ_u, we know there exist $(\Phi_u^i \rhd_{\mathcal{PM}} \Delta_i \vdash u : M_i)_{i \in \{1,\dots,2\}}$, such that $\Delta = +_{i \in \{1,\dots,2\}} \Delta_i$ and $\#_{\texttt{App,VPat,DPat}}(\Phi_u) = \#_{\texttt{App,VPat,DPat}}(\Phi_u^1) + \#_{\texttt{App,VPat,DPat}}(\Phi_u^2)$. By the induction hypothesis over Φ_s (resp. Φ_r), we know there exists $\Phi_{s\{x\setminus u\}} \rhd_{\mathcal{PM}} \Gamma_s + \Delta_1 \vdash s\{x \setminus u\} : A$ (resp. $\Phi_{r\{x\setminus u\}} \rhd_{\mathcal{PM}} \Gamma_r + \Delta_2 \vdash r\{x \setminus u\} : M_0$), such that $\#_{\texttt{App,VPat,DPat}}(\Phi_{s\{x\setminus u\}}) = \#_{\texttt{App,VPat,DPat}}(\Phi_s) + \#_{\texttt{App,VPat,DPat}}(\Phi_u^1)$ (resp. $\#_{\texttt{App,VPat,DPat}}(\Phi_{r\{x\setminus u\}}) = \#_{\texttt{App,VPat,DPat}}(\Phi_r) + \#_{\texttt{App,VPat,DPat}}(\Phi_u^2)$). Therefore, we can build $\Phi_{t\{x\setminus u\}}$ as follows:

$$\frac{\Phi_{s\{x\setminus u\}} \quad \Pi \quad \Phi_{r\{x\setminus u\}}}{((\Gamma_s + \Delta_1) \setminus\setminus \texttt{vars}(p)) + \Gamma_r + \Delta_2 \vdash s\{x \setminus u\}[p \backslash r\{x \setminus u\}] : A}\ \texttt{Match}$$

Thus, $s\{x \setminus u\}[p \backslash r\{x \setminus u\}] = (s[p \backslash r])\{x \setminus u\}$ and $\texttt{dom}(\Delta_1) \cap \texttt{vars}(p) = \emptyset$ (by Lemma 6.30). Therefore, $(\Gamma_s + \Delta_1) \setminus\setminus \texttt{vars}(p) = (\Gamma_s \setminus\setminus \texttt{vars}(p)) + \Delta_1$, and we can conclude since $\Gamma + \Delta = (\Gamma_s \setminus\setminus \texttt{vars}(p)) + \Gamma_r + \Delta_1 + \Delta_2$ and

$$\begin{aligned}
\#_{\texttt{App,VPat,DPat}}(\Phi_{t\{x\setminus u\}}) &= \#_{\texttt{App,VPat,DPat}}(\Phi_{s\{x\setminus u\}}) + \#_{\texttt{App,VPat,DPat}}(\Pi) \\
&+ \#_{\texttt{App,VPat,DPat}}(\Phi_{r\{x\setminus u\}}) \\
&= \#_{\texttt{App,VPat,DPat}}(\Phi_s) + \#_{\texttt{App,VPat,DPat}}(\Phi_u^1) + \#_{\texttt{App,VPat,DPat}}(\Pi) \\
&+ \#_{\texttt{App,VPat,DPat}}(\Phi_r) + \#_{\texttt{App,VPat,DPat}}(\Phi_u^2) \\
&= \#_{\texttt{App,VPat,DPat}}(\Phi_t) + \#_{\texttt{App,VPat,DPat}}(\Phi_u).
\end{aligned}$$

- Case $R = \texttt{Case}$. Then, Φ_t must be of the following form:

$$\frac{\Phi_r \rhd_{\mathcal{PM}} \Gamma_r; x : M_2 \vdash r : M_0 \quad \Pi \rhd_{\mathcal{PM}} (\Gamma_{s_k}; x : M_1)|_{\hat{p}_k} \Vdash \hat{p}_k : M_0 \quad \Phi_{s_k} \rhd_{\mathcal{PM}} \Gamma_{s_k}; x : M_1 \vdash s_k : A}{((\Gamma_{s_k}; x : M_1) \setminus\setminus \texttt{vars}(\hat{p}_k)) + \Gamma_r; x : M_2 \vdash \texttt{case}\ r\ \texttt{of}\ (\hat{p}_i.s_i)_n : A}\ \texttt{Case}$$

where $t = \texttt{case}\ r\ \texttt{of}\ (\hat{p}_i.s_i)_n$, $T = A$, $(\Gamma; x : M) = ((\Gamma_{s_k}; x : M_1) \setminus\setminus \texttt{vars}(\hat{p}_k)) + \Gamma_r; x : M_2 = ((\Gamma_{s_k} \setminus\setminus \texttt{vars}(\hat{p}_k)) + \Gamma_r); x : M_1 \sqcup M_2$, and thus $\Gamma = (\Gamma_{s_k} \setminus\setminus \texttt{vars}(\hat{p}_k)) + \Gamma_r$, $M = M_1 \sqcup M_2$ and $(\Gamma_{s_k}; x : M_1)|_{\hat{p}_k} = \Gamma_{s_k}|_{\hat{p}_k}$. By Lemma 6.29 over Φ_u, we know there exist $(\Phi_u^i \rhd_{\mathcal{PM}} \Delta_i \vdash u : M_i)_{i \in \{1,\dots,2\}}$, such that $\Delta = \Delta_1 + \Delta_2$ and $\#_{\texttt{App,VPat,DPat}}(\Phi_u) = \#_{\texttt{App,VPat,DPat}}(\Phi_u^1) + \#_{\texttt{App,VPat,DPat}}(\Phi_u^2)$. By the induction hypothesis over Φ_{s_k} (resp. Φ_r), we know there exists $\Phi_{s_k\{x\setminus u\}} \rhd_{\mathcal{PM}} \Gamma_{s_k} + \Delta_1 \vdash s_k\{x \setminus u\} : A$ (resp. $\Phi_{r\{x\setminus u\}} \rhd_{\mathcal{PM}} \Gamma_r + \Delta_2 \vdash r\{x \setminus u\} : M_0$), such that $\#_{\texttt{App,VPat,DPat}}(\Phi_{s_k\{x\setminus u\}}) = \#_{\texttt{App,VPat,DPat}}(\Phi_{s_k}) + \#_{\texttt{App,VPat,DPat}}(\Phi_u^1)$ (resp. $\#_{\texttt{App,VPat,DPat}}(\Phi_{r\{x\setminus u\}}) =$

$\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_r) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u^2)$). Therefore, we can build $\Phi_{t\{x\setminus u\}}$ as follows:

$$\frac{\Phi_{r\{x\setminus u\}} \quad \Pi \quad \Phi_{s_k\{x\setminus u\}}}{((\Gamma_{s_k} + \Delta_1) \setminus\!\setminus \mathtt{vars}(\hat{p}_k)) + \Gamma_r + \Delta_2 \vdash \mathtt{case}\ r\{x \setminus u\}\ \mathtt{of}\ (\hat{p}_i.s_i\{x \setminus u\})_n : A}\ \mathtt{Case}$$

By α-conversion, we may assume $\mathtt{vars}(\hat{p}_i) \cap \mathtt{fv}(u) = \emptyset$, for all $i \in \{1,\dots,n\}$. Thus, $\mathtt{case}\ r\{x \setminus u\}\ \mathtt{of}\ (\hat{p}_i.s_i\{x \setminus u\})_n = (\mathtt{case}\ r\ \mathtt{of}\ (\hat{p}_i.s_i)_n)\{x \setminus u\}$ and $\mathtt{dom}(\Delta_1) \cap \mathtt{vars}(\hat{p}_k) = \emptyset$ (by Lemma 6.30). Therefore, $(\Gamma_{s_k} + \Delta_1) \setminus\!\setminus \mathtt{vars}(\hat{p}_k) = (\Gamma_s \setminus\!\setminus \mathtt{vars}(\hat{p}_k)) + \Delta_1$, and we can conclude since $\Gamma + \Delta = (\Gamma_s \setminus\!\setminus \mathtt{vars}(\hat{p}_k)) + \Gamma_r + \Delta_1 + \Delta_2$ and

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{t\{x\setminus u\}}) &= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{s_k\{x\setminus u\}}) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi) \\
&+ \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{r\{x\setminus u\}}) \\
&= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{s_k}) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u^1) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi) \\
&+ \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_r) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u^2) \\
&= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_t) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u).
\end{aligned}$$

□

Lemma 6.37 (Exact Subject Reduction). *Let $t \in \Lambda^{PM}$, and $S \in \{\mathtt{dB}, \mathtt{branch}, \mathtt{match}, \mathtt{sub}\}$. If $t \to_{\mathtt{PM}(S)} t'$ and $\Phi_t \rhd_{\mathcal{PM}} \Gamma \vdash t : A$, then there is $\Phi_{t'} \rhd_{\mathcal{PM}} \Gamma \vdash t' : A$, such that $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_t) = \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{t'}) + 1 \cdot R$, where*

- $R = \mathtt{App}$, *if* $S = \mathtt{dB}$;
- $R = \mathtt{VPat}$, *if* $S = \mathtt{sub}$;
- $R = \mathtt{DPat}$ *if* $S \in \{\mathtt{branch}, \mathtt{match}\}$.

Proof. The proof follows by induction on $t \to_{\mathtt{PM}} t'$:

- Case $t = \mathtt{L}_0(\!|\lambda p.u|\!)s \to_{\mathtt{PM}(\mathtt{dB})} \mathtt{L}_0(\!|u[p\setminus s]|\!) = t'$, such that $\mathtt{bv}(\mathtt{L}_0) \cap \mathtt{fv}(s) = \emptyset$. We proceed by induction on the list matching context $\mathtt{L}_0$:
  - Let $\mathtt{L}_0 = \Box$. Then, Φ_t must be of the following form:

$$\frac{\dfrac{\Phi_u \rhd_{\mathcal{PM}} \Gamma_u \vdash u : A \qquad \Pi \rhd_{\mathcal{PM}} \Gamma_u|_p \Vdash p : M}{\Gamma_u \setminus\!\setminus \mathtt{vars}(p) \vdash \lambda p.u : M \to A}\ \mathtt{Abs} \qquad \Phi_s \rhd_{\mathcal{PM}} \Gamma_s \vdash s : M}{(\Gamma_u \setminus\!\setminus \mathtt{vars}(p)) + \Gamma_s \vdash (\lambda p.u)s : A}\ \mathtt{App}$$

    where $\Gamma = (\Gamma_u \setminus\!\setminus \mathtt{vars}(p)) + \Gamma_s$. Therefore, we can build $\Phi_{t'}$ as follows:

$$\frac{\Phi_u \quad \Pi \quad \Phi_s}{(\Gamma_u \setminus\!\setminus \mathtt{vars}(p)) + \Gamma_s \vdash u[p\setminus s] : A}\ \mathtt{Match}$$

We can conclude with

$$\begin{aligned}\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_t) &= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi)\\ &+ \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_s) + 1\cdot\mathtt{App}\\ &= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{t'}) + 1\cdot\mathtt{App}.\end{aligned}$$

– Let $\mathtt{L}_0 = \mathtt{L}_1[q\backslash r]$. Then, Φ_t must be of the following form:

$$\dfrac{\dfrac{\begin{array}{c}\Phi_{\mathtt{L}_1(\!|\lambda p.u|\!)} \rhd_{\mathcal{PM}} \Gamma_{\mathtt{L}_1(\!|\lambda p.u|\!)} \vdash \mathtt{L}_1(\!|\lambda p.u|\!) : M_p \to A\\ \Pi \rhd_{\mathcal{PM}} \Gamma_{\mathtt{L}_1(\!|\lambda p.u|\!)}|_q \Vdash q : M_q\\ \Phi_r \rhd_{\mathcal{PM}} \Gamma_r \vdash r : M_q\end{array}}{(\Gamma_{\mathtt{L}_1(\!|\lambda p.u|\!)} \setminus\!\setminus \mathtt{vars}(q)) + \Gamma_r \vdash \mathtt{L}_1(\!|\lambda p.u|\!)[q\backslash r] : M_p \to A}\ \mathtt{Match} \qquad \Phi_s \rhd_{\mathcal{PM}} \Gamma_s \vdash s : M_p}{(\Gamma_{\mathtt{L}_1(\!|\lambda p.u|\!)} \setminus\!\setminus \mathtt{vars}(q)) + \Gamma_r + \Gamma_s \vdash ((\mathtt{L}_1(\!|\lambda p.u|\!))[q\backslash r])s : A}\ \mathtt{App}$$

where $\Gamma = (\Gamma_{\mathtt{L}_1(\!|\lambda p.u|\!)} \setminus\!\setminus \mathtt{vars}(q)) + \Gamma_r + \Gamma_s$. Since $\mathtt{bv}(\mathtt{L}_1) \subseteq \mathtt{bv}(\mathtt{L}_0)$, we can do the step $\mathtt{L}_1(\!|\lambda p.u|\!)s \to_{\mathtt{PM}(\mathtt{dB})} \mathtt{L}_1(\!|u[p\backslash s]|\!)$. Moreover, we can build derivation $\Psi_{\mathtt{L}_1(\!|\lambda p.u|\!)s}$ for $\mathtt{L}_1(\!|\lambda p.u|\!)s$ as follows:

$$\dfrac{\Phi_{\mathtt{L}_1(\!|\lambda p.u|\!)} \qquad \Phi_s}{\Gamma_{\mathtt{L}_1(\!|\lambda p.u|\!)} + \Gamma_s \vdash \mathtt{L}_1(\!|\lambda p.u|\!)s : A}\ \mathtt{App}$$

Therefore, by the *i.h.* over $\mathtt{L}_1(\!|\lambda p.u|\!)s \to_{\mathtt{PM}(\mathtt{dB})} \mathtt{L}_1(\!|u[p\backslash s]|\!)$, there is $\Psi_{\mathtt{L}_1(\!|u[p\backslash s]|\!)} \rhd_{\mathcal{PM}} \Gamma_{\mathtt{L}_1(\!|\lambda p.u|\!)} + \Gamma_s \vdash \mathtt{L}_1(\!|u[p\backslash s]|\!) : A$, such that $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Psi_{\mathtt{L}_1(\!|\lambda p.u|\!)s}) = \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Psi_{\mathtt{L}_1(\!|u[p\backslash s]|\!)}) + 1\cdot\mathtt{App}$. Since $\mathtt{bv}(\mathtt{L}_0) \cap \mathtt{fv}(s) = \emptyset$, we know that $\mathtt{fv}(s) \cap \mathtt{vars}(q) = \emptyset$ and thus $\mathtt{vars}(q) \cap \mathtt{dom}(\Gamma_s) = \emptyset$ (Lemma 6.30). Therefore, we know that $\Gamma_{\mathtt{L}_1(\!|\lambda p.u|\!)} + \Gamma_s|_q = \Gamma_{\mathtt{L}_1(\!|\lambda p.u|\!)}|_q$, $(\Gamma_{\mathtt{L}_1(\!|\lambda p.u|\!)} + \Gamma_s) \setminus\!\setminus \mathtt{vars}(q) = (\Gamma_{\mathtt{L}_1(\!|\lambda p.u|\!)}) \setminus\!\setminus \mathtt{vars}(q) + \Gamma_s$, and we can build $\Phi_{t'}$ as follows:

$$\dfrac{\Psi_{\mathtt{L}_1(\!|u[p\backslash s]|\!)} \qquad \Pi \qquad \Phi_r}{((\Gamma_{\mathtt{L}_1(\!|\lambda p.u|\!)} + \Gamma_s) \setminus\!\setminus \mathtt{vars}(q)) + \Gamma_r \vdash (\mathtt{L}_1(\!|u[p\backslash s]|\!))[q\backslash r] : A}\ \mathtt{Match}$$

We can conclude since

$$\Gamma = (\Gamma_{\mathtt{L}_1(\!|\lambda p.u|\!)} \setminus\!\setminus \mathtt{vars}(q)) + \Gamma_r + \Gamma_s = ((\Gamma_{\mathtt{L}_1(\!|\lambda p.u|\!)} + \Gamma_s) \setminus\!\setminus \mathtt{vars}(q)) + \Gamma_r,$$

and

$$\begin{aligned}\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_t) &= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{\mathtt{L}_1(\!|\lambda p.u|\!)}) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi)\\ &+ \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_r) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_s) + 1\cdot\mathtt{App}\\ &= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Psi_{\mathtt{L}_1(\!|\lambda p.u|\!)s}) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi)\\ &+ \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_r)\\ &= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Psi_{\mathtt{L}_1(\!|u[p\backslash s]|\!)}) + 1\cdot\mathtt{App} + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi)\\ &+ \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_r)\\ &= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{t'}) + 1\cdot\mathtt{App}.\end{aligned}$$

- Case $t = u[\mathtt{c}(p_i)_n \backslash \mathtt{L}_0(\!|\mathtt{c}(s_i)_n|\!)] \to_{\mathtt{PM(match)}} \mathtt{L}_0(\!|u[p_1\backslash s_1]\cdots[p_n\backslash s_n]|\!) = t'$, such that $\mathtt{bv}(\mathtt{L}_0) \cap \mathtt{fv}(u) = \emptyset$. We proceed by induction on the list matching context M_0:

  – Let $\mathtt{L}_0 = \Box$. Then, Φ_t must be of the following form:

$$\dfrac{\Phi_u \rhd_{\mathcal{PM}} \Gamma_u \vdash u : A \qquad \dfrac{(\Pi_{p_i} \rhd_{\mathcal{PM}} \Gamma_u|_{p_i} \Vdash p_i : M_i)_{i\in\{1,\ldots,n\}}}{\Gamma_u|_{\mathtt{c}(p_i)_n} \Vdash \mathtt{c}(p_i)_n : \{\!\{\mathtt{c}(M_i)_n\}\!\}}\ \mathtt{DPat} \qquad \dfrac{\dfrac{(\ \Phi_{s_i} \rhd_{\mathcal{PM}} \Gamma_{s_i} \vdash s_i : M_i)_{i\in\{1,\ldots,n\}}}{+_{i\in\{1,\ldots,n\}} \Gamma_{s_i} \vdash \mathtt{c}(s_i)_n : \mathtt{c}(M_i)_n}\ \mathtt{Data}}{+_{i\in\{1,\ldots,n\}}\Gamma_{s_i} \vdash \mathtt{c}(s_i)_n : \{\!\{\mathtt{c}(M_i)_n\}\!\}}\ \mathtt{Many}}{(\Gamma_u \setminus\!\setminus \mathtt{vars}(\mathtt{c}(p_i)_n)) +_{i\in\{1,\ldots,n\}} \Gamma_{s_i} \vdash u[\mathtt{c}(p_i)_n\backslash\mathtt{c}(s_i)_n] : A}\ \mathtt{Match}$$

  where $\Gamma = (\Gamma_u \setminus\!\setminus \mathtt{vars}(\mathtt{c}(p_i)_n)) +_{i\in\{1,\ldots,n\}} \Gamma_{s_i}$, and $\Gamma_u|_{\mathtt{c}(p_i)_n} = +_{i\in\{1,\ldots,n\}}(\Gamma_u|_{p_i})$. By α-conversion, we can assume $\mathtt{fv}(\mathtt{c}(s_i)_n) \cap \mathtt{vars}(\mathtt{c}(p_i)_n) = \emptyset$ and thus $\mathtt{dom}(+_{\{1,\ldots,n\}}\Gamma_{s_i}) \cap \mathtt{vars}(\mathtt{c}(p_i)_n) = \emptyset$ (Lemma 6.30). Therefore, $(+_{\{1,\ldots,n\}}\Gamma_{s_i}) \setminus\!\setminus \mathtt{vars}(\mathtt{c}(p_i)_n) = +_{\{1,\ldots,n\}}\Gamma_{s_i}$, $(+_{\{1,\ldots,n\}}\Gamma_{s_i})|_{\mathtt{c}(p_i)_n} = \emptyset$, and we can build $\Phi_{t'}$ as follows:

$$\dfrac{\dfrac{\Phi_u \quad \Pi_{p_1} \quad \Phi_{s_1}}{\Gamma_u \setminus\!\setminus \mathtt{vars}(p_1) + \Gamma_{s_1} \vdash u[p_1\backslash s_1] : A}\ \mathtt{Match} \qquad \vdots \qquad \Pi_{p_n} \quad \Phi_{s_n}}{(\Gamma_u \setminus\!\setminus \mathtt{vars}(p_1)\cdots \setminus\!\setminus \mathtt{vars}(p_n)) +_{i\in\{1,\ldots,n\}} \Gamma_{s_i} \vdash u[p_1\backslash s_1]\cdots[p_n\backslash s_n] : A}\ \mathtt{Match}$$

  We can conclude since $\mathtt{vars}(\mathtt{c}(p_i)_n) = \mathtt{vars}(p_1) \cup \cdots \cup (p_n)$, and

$$\begin{aligned} \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_t) &= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u) +_{i\in\{1,\ldots,n\}} \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{p_i}) + \\ &\quad 1\cdot\mathtt{DPat} +_{i\in\{1,\ldots,n\}} \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{s_i}) \\ &= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{t'}) + 1\cdot\mathtt{DPat}. \end{aligned}$$

  – Let $\mathtt{L}_0 = \mathtt{L}_1[q\backslash r]$. Moreover, consider the following type derivation $\Phi_u \rhd_{\mathcal{PM}} \Gamma_u \vdash u : A$. Then, Φ_t must be of the following form:

$$\dfrac{\Phi_u \qquad \dfrac{(\Pi_{p_i} \rhd_{\mathcal{PM}} \Gamma_u|_{p_i} \Vdash p_i : M_i)_{i\in\{1,\ldots,n\}}}{\Gamma_u|_{\mathtt{c}(p_i)_n} \Vdash \mathtt{c}(p_i)_n : \{\!\{\mathtt{c}(M_i)_n\}\!\}}\ \mathtt{DPat} \qquad \dfrac{\begin{array}{c}\Phi_{\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)} \rhd_{\mathcal{PM}} \Gamma_{\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)} \vdash \mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!) : \{\!\{\mathtt{c}(M_i)_n\}\!\} \\ \Pi_q \rhd_{\mathcal{PM}} \Gamma_{\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)}|_q \Vdash q : M_q \\ \Phi_r \rhd_{\mathcal{PM}} \Gamma_r \vdash r : M_q\end{array}}{(\Gamma_{\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)} \setminus\!\setminus \mathtt{vars}(q)) + \Gamma_r \vdash \mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)[q\backslash r] : \{\!\{\mathtt{c}(M_i)_n\}\!\}}\ \mathtt{Match}}{(\Gamma_u \setminus\!\setminus \mathtt{vars}(\mathtt{c}(p_i)_n)) + (\Gamma_{\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)} \setminus\!\setminus \mathtt{vars}(q)) + \Gamma_r \vdash u[\mathtt{c}(p_i)_n\backslash\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)[q\backslash r]] : A}\ \mathtt{Match}$$

  where $\Gamma = (\Gamma_u \setminus\!\setminus \mathtt{vars}(\mathtt{c}(p_i)_n)) + (\Gamma_{\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)} \setminus\!\setminus \mathtt{vars}(q)) + \Gamma_r$, and we also have that $\Gamma_u|_{\mathtt{c}(p_i)_n} = +_{i\in\{1,\ldots,n\}}(\Gamma_u|_{p_i})$. Since $\mathtt{bv}(\mathtt{L}_1) \subseteq \mathtt{bv}(\mathtt{L}_0)$, we can do the following step $u[\mathtt{c}(p_i)_n\backslash\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)] \to_{\mathtt{PM(match)}} \mathtt{L}_1(\!|u[p_1\backslash s_1]\cdots[p_n\backslash s_n]|\!)$. Moreover, we can build derivation $\Psi_{u[\mathtt{c}(p_i)_n\backslash\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)]}$ for $u[\mathtt{c}(p_i)_n\backslash\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)]$ as follows:

$$\dfrac{\Phi_u \qquad \dfrac{(\Pi_{p_i})_{i\in\{1,\ldots,n\}}}{\Gamma_u|_{\mathtt{c}(p_i)_n} \Vdash \mathtt{c}(p_i)_n : \{\!\{\mathtt{c}(M_i)_n\}\!\}}\ \mathtt{DPat} \qquad \Phi_{\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)}}{(\Gamma_u \setminus\!\setminus \mathtt{vars}(\mathtt{c}(p_i)_n)) + \Gamma_{\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)} \vdash u[\mathtt{c}(p_i)_n\backslash\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)] : A}\ \mathtt{Match}$$

Therefore, by the *i.h.* over $u[\mathtt{c}(p_i)_n \backslash \mathtt{L}_1 (\!| \mathtt{c}(s_i)_n |\!)] \to_{\mathtt{PM(match)}} \mathtt{L}_1 (\!| u[p_1 \backslash s_1] \cdots [p_n \backslash s_n] |\!)$, there is $\Psi_{\mathtt{L}_1 (\!| u[p_1 \backslash s_1] \cdots [p_n \backslash s_n] |\!)} \triangleright_{\mathcal{PM}} (\Gamma_u \setminus\!\setminus \mathtt{vars}(\mathtt{c}(p_i)_n)) + \Gamma_{\mathtt{L}_1 (\!| \mathtt{c}(s_i)_n |\!)} \vdash \mathtt{L}_1 (\!| u[p_1 \backslash s_1] \cdots [p_n \backslash s_n] |\!) : A$, such that $\#_{\mathtt{App,VPat,DPat}}(\Psi_{u[\mathtt{c}(p_i)_n \backslash \mathtt{L}_1 (\!| \mathtt{c}(s_i)_n |\!)]}) = \#_{\mathtt{App,VPat,DPat}}(\Psi_{\mathtt{L}_1 (\!| u[p_1 \backslash s_1] \cdots [p_n \backslash s_n] |\!)}) + 1 \cdot \mathtt{DPat}$. Since $\mathtt{bv}(\mathtt{L}_0) \cap \mathtt{fv}(u) = \emptyset$, we know that $\mathtt{fv}(u) \cap \mathtt{vars}(q) = \emptyset$ and thus $\mathtt{vars}(q) \cap \mathtt{dom}(\Gamma_u) = \emptyset$ (Lemma 6.30). Therefore, we know $(\Gamma_u \setminus\!\setminus \mathtt{vars}(\mathtt{c}(p_i)_n)) + \Gamma_{\mathtt{L}_1 (\!| \mathtt{c}(s_i)_n |\!)}|_q = \Gamma_{\mathtt{L}_1 (\!| \mathtt{c}(s_i)_n |\!)}|_q$, but also $((\Gamma_u \setminus\!\setminus \mathtt{vars}(\mathtt{c}(p_i)_n)) + \Gamma_{\mathtt{L}_1 (\!| \mathtt{c}(s_i)_n |\!)}) \setminus\!\setminus \mathtt{vars}(q) = (\Gamma_u \setminus\!\setminus \mathtt{vars}(\mathtt{c}(p_i)_n)) + (\Gamma_{\mathtt{L}_1 (\!| \mathtt{c}(s_i)_n |\!)} \setminus\!\setminus \mathtt{vars}(q))$, and we can build $\Phi_{t'}$ as follows:

$$\frac{\Psi_{\mathtt{L}_1 (\!| u[p_1 \backslash s_1] \cdots [p_n \backslash s_n] |\!)} \quad \Pi_q \quad \Phi_r}{((\Gamma_u \setminus\!\setminus \mathtt{vars}(\mathtt{c}(p_i)_n)) + \Gamma_{\mathtt{L}_1 (\!| \mathtt{c}(s_i)_n |\!)}) \setminus\!\setminus \mathtt{vars}(q) + \Gamma_r \vdash \mathtt{L}_1 (\!| u[p_1 \backslash s_1] \cdots [p_n \backslash s_n] |\!)[q \backslash r] : A}\ \mathtt{Match}$$

We can conclude since we know that $\Gamma = (\Gamma_u \setminus\!\setminus \mathtt{vars}(\mathtt{c}(p_i)_n)) + (\Gamma_{\mathtt{L}_1 (\!| \mathtt{c}(s_i)_n |\!)} \setminus\!\setminus \mathtt{vars}(q)) + \Gamma_r = ((\Gamma_u \setminus\!\setminus \mathtt{vars}(\mathtt{c}(p_i)_n)) + \Gamma_{\mathtt{L}_1 (\!| \mathtt{c}(s_i)_n |\!)}) \setminus\!\setminus \mathtt{vars}(q) + \Gamma_r$, and

$$\begin{aligned}
\#_{\mathtt{App,VPat,DPat}}(\Phi_t) &= \#_{\mathtt{App,VPat,DPat}}(\Phi_u) +_{\{1,\ldots,n\}} \#_{\mathtt{App,VPat,DPat}}(\Pi_{p_i}) + 1 \cdot \mathtt{DPat} \\
&+ \#_{\mathtt{App,VPat,DPat}}(\Phi_{\mathtt{L}_1 (\!| \mathtt{c}(s_i)_n |\!)}) + \#_{\mathtt{App,VPat,DPat}}(\Pi_q) \\
&+ \#_{\mathtt{App,VPat,DPat}}(\Phi_r) \\
&= \#_{\mathtt{App,VPat,DPat}}(\Psi_{u[\mathtt{c}(p_i)_n \backslash \mathtt{L}_1 (\!| \mathtt{c}(s_i)_n |\!)]}) + \#_{\mathtt{App,VPat,DPat}}(\Pi_q) \\
&+ \#_{\mathtt{App,VPat,DPat}}(\Phi_r) \\
&= \#_{\mathtt{App,VPat,DPat}}(\Psi_{\mathtt{L}_1 (\!| u[p_1 \backslash s_1] \cdots [p_n \backslash s_n] |\!)}) + 1 \cdot \mathtt{DPat} + \#_{\mathtt{App,VPat,DPat}}(\Pi_q) \\
&+ \#_{\mathtt{App,VPat,DPat}}(\Phi_r) \\
&= \#_{\mathtt{App,VPat,DPat}}(\Phi_{t'}) + 1 \cdot \mathtt{DPat}.
\end{aligned}$$

- Case $t = u[x \backslash s] \to_{\mathtt{PM(sub)}} u\{x \setminus s\} = t'$. Then, Φ_t must be of the following form:

$$\frac{\Phi_u \triangleright_{\mathcal{PM}} \Gamma_u; x : M \vdash u : A \qquad \dfrac{}{x : M \Vdash x : M}\ (\mathtt{VPat}) \qquad \Phi_s \triangleright_{\mathcal{PM}} \Gamma_s \vdash s : M}{\Gamma_u + \Gamma_s \vdash u[x \backslash s] : A}\ \mathtt{Match}$$

  where $\Gamma = \Gamma_u + \Gamma_s$. By Lemma 6.36 over Φ_u and Φ_s, we know there exists $\Phi_{u\{x \backslash s\}} \triangleright_{\mathcal{PM}} \Gamma_u + \Gamma_s \vdash u\{x \setminus s\} : A$, such that $\#_{\mathtt{App,VPat,DPat}}(\Phi_{u\{x \backslash s\}}) = \#_{\mathtt{App,VPat,DPat}}(\Phi_u) + \#_{\mathtt{App,VPat,DPat}}(\Phi_s)$. Therefore, we can pick $\Phi_{t'} = \Phi_{u\{x \backslash s\}}$ and can conclude since

$$\begin{aligned}
[small]\#_{\mathtt{App,VPat,DPat}}(\Phi_t) &= \#_{\mathtt{App,VPat,DPat}}(\Phi_u) + 1 \cdot \mathtt{VPat} + \#_{\mathtt{App,VPat,DPat}}(\Phi_s) \\
&= \#_{\mathtt{App,VPat,DPat}}(\Phi_{u\{x \backslash s\}}) + 1 \cdot \mathtt{VPat} \\
&= \#_{\mathtt{App,VPat,DPat}}(\Phi_{t'}) + 1 \cdot \mathtt{VPat}.
\end{aligned}$$

- Case $t = \mathtt{case}\ \mathtt{L}_0 (\!| \mathtt{c}(s_i)_n |\!)\ \mathtt{of}\ (\ldots, \mathtt{c}(p_i)_n.u_i, \ldots) \to_{\mathtt{PM(branch)}} \mathtt{L}_0 (\!| u_i[p_1 \backslash s_1] \cdots [p_n \backslash s_n] |\!) = t'$, such that $\mathtt{bv}(\mathtt{L}_0) \cap \mathtt{fv}(u_i) = \emptyset$. We proceed by induction on the list matching context M_0:

– Let $\mathrm{L}_0 = \Box$. Moreover, consider the following type derivation $\Phi_{u_i} \rhd_{\mathcal{PM}} \Gamma_{u_i} \vdash u_i : A$. Then, Φ_t must be of the following form:

$$\dfrac{\dfrac{\dfrac{\left(\Phi_{s_i} \rhd_{\mathcal{PM}} \Gamma_{s_i} \vdash s_i : M_i\right)_{i\in\{1,\dots,n\}}}{+_{i\in\{1,\dots,n\}}\Gamma_{s_i} \vdash \mathtt{c}(s_i)_n : \mathtt{c}(M_i)_n}\,\mathtt{Data}\ \mathtt{Many}}{+_{i\in\{1,\dots,n\}}\Gamma_{s_i} \vdash \mathtt{c}(s_i)_n : \{\!\{\mathtt{c}(M_i)_n\}\!\}} \qquad \dfrac{(\Pi_{p_i} \rhd_{\mathcal{PM}} \Gamma_{u_i}|_{p_i} \Vdash p_i : M_i)_{i\in\{1,\dots,n\}}}{\Gamma_{u_i}|_{\mathtt{c}(p_i)_n} \Vdash \mathtt{c}(p_i)_n : \{\!\{\mathtt{c}(M_i)_n\}\!\}}\,\mathtt{DPat} \qquad \Phi_{u_i}}{(\Gamma_{u_i} \setminus\!\setminus \mathtt{vars}(\mathtt{c}(p_i)_n)) +_{i\in\{1,\dots,n\}} \Gamma_{s_i} \vdash \mathtt{case}\ \mathtt{c}(s_i)_n\ \mathtt{of}\ (\dots,\mathtt{c}(p_i)_n.u_i,\dots) : A}\,\mathtt{Case}$$

where $\Gamma = (\Gamma_{u_i} \setminus\!\setminus \mathtt{vars}(\mathtt{c}(p_i)_n)) +_{i\in\{1,\dots,n\}} \Gamma_{s_i}$, and $\Gamma_{u_i}|_{\mathtt{c}(p_i)_n} = +_{i\in\{1,\dots,n\}}(\Gamma_{u_i}|_{p_i})$. By α-conversion, we can assume $\mathtt{fv}(\mathtt{c}(s_i)_n) \cap \mathtt{vars}(\mathtt{c}(p_i)_n) = \emptyset$ and thus $\mathtt{dom}(+_{\{1,\dots,n\}})\Gamma_{s_i} \cap \mathtt{vars}(\mathtt{c}(p_i)_n) = \emptyset$ (Lemma 6.30). Therefore, $(+_{\{1,\dots,n\}}\Gamma_{s_i}) \setminus\!\setminus \mathtt{vars}(\mathtt{c}(p_i)_n) = \emptyset$, but also $(+_{\{1,\dots,n\}}\Gamma_{s_i})|_{\mathtt{c}(p_i)_n} = \emptyset$, and we can build $\Phi_{t'}$ as follows:

$$\dfrac{\dfrac{\Phi_{u_i} \quad \Pi_{p_1} \quad \Phi_{s_1}}{\Gamma_{u_i} \setminus\!\setminus \mathtt{vars}(p_1) + \Gamma_{s_1} \vdash u[p_1 \backslash s_1] : A}\,\mathtt{Match} \quad \vdots \quad \Pi_{p_n} \quad \Phi_{s_n}}{(\Gamma_{u_i} \setminus\!\setminus \mathtt{vars}(p_1) \cdots \setminus\!\setminus \mathtt{vars}(p_n)) +_{i\in\{1,\dots,n\}} \Gamma_{s_i} \vdash u_i[p_1\backslash s_1]\cdots[p_n\backslash s_n] : A}\,\mathtt{Match}$$

We can conclude since $\mathtt{vars}(\mathtt{c}(p_i)_n) = \mathtt{vars}(p_1) \cup \cdots \cup \mathtt{vars}(p_n)$, and

$$\begin{aligned} \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_t) &= +_{i\in\{1,\dots,n\}}\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{s_i}) +_{i\in\{1,\dots,n\}} \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi_{p_i}) \\ &+ 1\cdot\mathtt{DPat} + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{u_i}) \\ &= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{t'}) + 1\cdot\mathtt{DPat}. \end{aligned}$$

– Let $\mathrm{L}_0 = \mathrm{L}_1[q\backslash r]$. Moreover, consider the following type derivation $\Phi_{u_i} \rhd_{\mathcal{PM}} \Gamma_{u_i} \vdash u_i : A$. Then, Φ_t must be of the following form:

$$\dfrac{\begin{array}{c}\Phi_{\mathrm{L}_1(\!|\mathtt{c}(s_i)_n|\!)} \rhd_{\mathcal{PM}} \Gamma_{\mathrm{L}_1(\!|\mathtt{c}(s_i)_n|\!)} \vdash \mathrm{L}_1(\!|\mathtt{c}(s_i)_n|\!) : \{\!\{\mathtt{c}(M_i)_n\}\!\} \\ \Pi_q \rhd_{\mathcal{PM}} \Gamma_{\mathrm{L}_1(\!|\mathtt{c}(s_i)_n|\!)}|_q \Vdash q : M_q \\ \Phi_r \rhd_{\mathcal{PM}} \Gamma_r \vdash r : M_q\end{array}}{(\Gamma_{\mathrm{L}_1(\!|\mathtt{c}(s_i)_n|\!)} \setminus\!\setminus \mathtt{vars}(q)) + \Gamma_r \vdash \mathrm{L}_1(\!|\mathtt{c}(s_i)_n|\!)[q\backslash r] : \{\!\{\mathtt{c}(M_i)_n\}\!\}}\,\mathtt{Match} \qquad \dfrac{(\Pi_{p_i} \rhd_{\mathcal{PM}} \Gamma_{u_i}|_{p_i} \Vdash p_i : M_i)_{i\in\{1,\dots,n\}}}{\Gamma_{u_i}|_{\mathtt{c}(p_i)_n} \Vdash \mathtt{c}(p_i)_n : \{\!\{\mathtt{c}(M_i)_n\}\!\}}\,\mathtt{DPat} \qquad \Phi_{u_i}}{(\Gamma_{u_i} \setminus\!\setminus \mathtt{vars}(\mathtt{c}(p_i)_n)) + (\Gamma_{\mathrm{L}_1(\!|\mathtt{c}(s_i)_n|\!)} \setminus\!\setminus \mathtt{vars}(q)) + \Gamma_r \vdash \mathtt{case}\ \mathrm{L}_1(\!|\mathtt{c}(s_i)_n|\!)[q\backslash r]\ \mathtt{of}\ (\dots,\mathtt{c}(p_i)_n.u_i,\dots) : A}\,\mathtt{Case}$$

where $\Gamma = (\Gamma_{u_i} \setminus\!\setminus \mathtt{vars}(\mathtt{c}(p_i)_n)) + (\Gamma_{\mathrm{L}_1(\!|\mathtt{c}(s_i)_n|\!)} \setminus\!\setminus \mathtt{vars}(q)) + \Gamma_r$, and we also have that $\Gamma_{u_i}|_{\mathtt{c}(p_i)_n} = +_{i\in\{1,\dots,n\}}(\Gamma_{u_i}|_{p_i})$. Since $\mathtt{bv}(\mathrm{L}_1) \subseteq \mathtt{bv}(\mathrm{L}_0)$, we can do the following step $\mathtt{case}\ \mathrm{L}_1(\!|\mathtt{c}(s_i)_n|\!)\ \mathtt{of}\ (\dots,\mathtt{c}(p_i)_n.u_i,\dots) \to_{\mathtt{PM}(\mathtt{branch})} \mathrm{L}_1(\!|u_i[p_1\backslash s_1]\cdots[p_n\backslash s_n]|\!)$. Moreover, we can build type derivation $\Psi_{\mathtt{case}\ \mathrm{L}_1(\!|\mathtt{c}(s_i)_n|\!)\ \mathtt{of}\ (\dots,\mathtt{c}(p_i)_n.u_i,\dots)}$ for term $\mathtt{case}\ \mathrm{L}_1(\!|\mathtt{c}(s_i)_n|\!)\ \mathtt{of}\ (\dots,\mathtt{c}(p_i)_n.u_i,\dots)$ as follows:

$$\dfrac{\Phi_{\mathrm{L}_1(\!|\mathtt{c}(s_i)_n|\!)} \qquad \dfrac{(\Pi_{p_i})_{i\in\{1,\dots,n\}}}{\Gamma_{u_i}|_{\mathtt{c}(p_i)_n} \Vdash \mathtt{c}(p_i)_n : \{\!\{\mathtt{c}(M_i)_n\}\!\}}\,\mathtt{DPat} \qquad \Phi_{u_i}}{(\Gamma_{u_i} \setminus\!\setminus \mathtt{vars}(\mathtt{c}(p_i)_n)) + \Gamma_{\mathrm{L}_1(\!|\mathtt{c}(s_i)_n|\!)} \vdash \mathtt{case}\ \mathrm{L}_1(\!|\mathtt{c}(s_i)_n|\!)\ \mathtt{of}\ (\dots,\mathtt{c}(p_i)_n.u_i,\dots) : A}\,\mathtt{Case}$$

Therefore, by the *i.h.* over

$$\texttt{case}\ \mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)\ \texttt{of}\ (\ldots,\mathtt{c}(p_i)_n.u_i,\ldots) \to_{\mathtt{PM(branch)}} \mathtt{L}_1(\!|u_i[p_1\backslash s_1]\cdots[p_n\backslash s_n]|\!),$$

there is

$$\Psi_{\mathtt{L}_1(\!|u_i[p_1\backslash s_1]\cdots[p_n\backslash s_n]|\!)} \rhd_{\mathcal{PM}} (\Gamma_{u_i} \setminus\!\setminus \mathtt{vars}(\mathtt{c}(p_i)_n)) + \Gamma_{\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)} \vdash \mathtt{L}_1(\!|u_i[p_1\backslash s_1]\cdots[p_n\backslash s_n]|\!) : A,$$

such that

$$\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Psi_{\texttt{case}\ \mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)\ \texttt{of}\ (\ldots,\mathtt{c}(p_i)_n.u_i,\ldots)}) = \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Psi_{\mathtt{L}_1(\!|u_i[p_1\backslash s_1]\cdots[p_n\backslash s_n]|\!)}) + 1\cdot\mathtt{DPat}.$$

Since $\mathtt{bv}(\mathtt{L}_0)\cap\mathtt{fv}(u_i)=\emptyset$, we know that $\mathtt{fv}(u_i)\cap\mathtt{vars}(q)=\emptyset$ and thus $\mathtt{vars}(q)\cap\mathtt{dom}(\Gamma_{u_i})=\emptyset$ (Lemma 6.30). Therefore, we know $(\Gamma_{u_i}\setminus\!\setminus\mathtt{vars}(\mathtt{c}(p_i)_n)) + \Gamma_{\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)}|_q = \Gamma_{\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)}|_q$, also $((\Gamma_{u_i}\setminus\!\setminus\mathtt{vars}(\mathtt{c}(p_i)_n)) + \Gamma_{\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)})\setminus\!\setminus\mathtt{vars}(q) = (\Gamma_{u_i}\setminus\!\setminus\mathtt{vars}(\mathtt{c}(p_i)_n)) + (\Gamma_{\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)}\setminus\!\setminus\mathtt{vars}(q))$, and we can build $\Phi_{t'}$ as follows:

$$\frac{\Psi_{\mathtt{L}_1(\!|u_i[p_1\backslash s_1]\cdots[p_n\backslash s_n]|\!)} \quad \Pi_q \quad \Phi_r}{((\Gamma_{u_i}\setminus\!\setminus\mathtt{vars}(\mathtt{c}(p_i)_n)) + \Gamma_{\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)})\setminus\!\setminus\mathtt{vars}(q) + \Gamma_r \vdash \mathtt{L}_1(\!|u_i[p_1\backslash s_1]\cdots[p_n\backslash s_n]|\!)[q\backslash r] : A}\ \mathtt{Match}$$

We can conclude since

$$\begin{aligned}\Gamma &= (\Gamma_{u_i}\setminus\!\setminus\mathtt{vars}(\mathtt{c}(p_i)_n)) + (\Gamma_{\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)}\setminus\!\setminus\mathtt{vars}(q)) + \Gamma_r\\ &= ((\Gamma_{u_i}\setminus\!\setminus\mathtt{vars}(\mathtt{c}(p_i)_n)) + \Gamma_{\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)})\setminus\!\setminus\mathtt{vars}(q) + \Gamma_r\end{aligned}$$

and

$$\begin{aligned}\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_t) &= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)}) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi_q)\\ &+ \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_r) +_{\{1,\ldots,n\}} \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi_{p_i})\\ &+ 1\cdot\mathtt{DPat} + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{u_i})\\ &= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Psi_{\texttt{case}\ \mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)\ \texttt{of}\ (\ldots,\mathtt{c}(p_i)_n.u_i,\ldots)})\\ &+ \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi_q) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_r)\\ &= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Psi_{\mathtt{L}_1(\!|u[p_1\backslash s_1]\cdots[p_n\backslash s_n]|\!)}) + 1\cdot\mathtt{DPat}\\ &+ \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi_q) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_r)\\ &= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{t'}) + 1\cdot\mathtt{DPat}.\end{aligned}$$

- The remaining cases correspond to contextual propagation of the reduction and follow immediately from the *i.h.*.

□

Theorem 6.38 (Soundness). *Let $t\in\Lambda^{PM}_{\circ}$. If there is $\Phi\rhd_{\mathcal{PM}}\emptyset\vdash t:A$ tight, such that $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi) = n\cdot\mathtt{App}+m\cdot\mathtt{VPat}+l\cdot\mathtt{DPat}$, then $t \overset{n\cdot\mathtt{dB}+m\cdot\mathtt{sub}+l_1\cdot\mathtt{branch}+l_2\cdot\mathtt{match}}{\twoheadrightarrow_{\mathtt{PM}}} t'$, such that $l_1+l_2=l$ and $t'\in\mathtt{NO}^{\mathtt{CF}}$.*

Proof. Let $\Phi \rhd_{\mathcal{PM}} \emptyset \vdash t : A$ be tight, such that $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi) = n \cdot \mathtt{App} + m \cdot \mathtt{VPat} + l \cdot \mathtt{DPat}$. We proceed by induction on $n + m + l$:

- If $n + m + l = 0$, then $n = m = l = 0$. By Lemma 6.31, $t \in \mathtt{NO}^{\mathtt{CF}}$. Therefore, we can conclude with $t' = t$.
- If $n + m + l > 0$, then $t \notin \mathtt{NO}^{\mathtt{CF}}$ by Lemma 6.31. By Lemma 6.20, there is $u \in \Lambda^{PM}_{\circ}$, such that $t \to_{\mathtt{PM}(S)} u$, for some $S \in \{\mathtt{dB}, \mathtt{sub}, \mathtt{branch}, \mathtt{match}\}$. By Lemma 6.37, there is $\Phi' \rhd_{\mathcal{PM}} \emptyset \vdash u : A$ tight, such that $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi) = \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi') + 1 \cdot R$, where
  - $R = \mathtt{App}$, if $S = \mathtt{dB}$;
  - $R = \mathtt{VPat}$, if $S = \mathtt{sub}$;
  - and $R = \mathtt{DPat}$, if $S \in \{\mathtt{branch}, \mathtt{match}\}$.

  Let us assume, without loss of generality, that $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi') = n' \cdot \mathtt{App} + m' \cdot \mathtt{VPat} + l' \cdot \mathtt{DPat}$. Then, $n' + m' + l' + 1 = n + m + l$. Now, by applying the *i.h.*, we know

  $$u \overset{n' \cdot \mathtt{dB} + m' \cdot \mathtt{sub} + l'_1 \cdot \mathtt{branch} + l'_2 \cdot \mathtt{match}}{\twoheadrightarrow_{\mathtt{PM}}} u',$$

  such that $l'_1 + l'_2 = l'$ and $u' \in \mathtt{NO}^{\mathtt{CF}}$. Therefore, we also have that the following holds

  $$t \to_{\mathtt{PM}(S)} u \overset{n' \cdot \mathtt{dB} + m' \cdot \mathtt{sub} + l'_1 \cdot \mathtt{branch} + l'_2 \cdot \mathtt{match}}{\twoheadrightarrow_{\mathtt{PM}}} u'.$$

  Thus, we can take $t' = u'$, since $t \overset{n \cdot \mathtt{dB} + m \cdot \mathtt{sub} + l_1 \cdot \mathtt{branch} + l_2 \cdot \mathtt{match}}{\twoheadrightarrow_{\mathtt{PM}}} t'$.

□

D.2.2 Quantitative Completeness

Lemma 6.39 (Weighted Anti-substitution). *Let $t \in \Lambda^{PM}$ and $u \in \Lambda^{PM}_{\circ}$ and assume $\Phi_{t\{x \setminus u\}} \rhd_{\mathcal{PM}} \Gamma \vdash t\{x \setminus u\} : A$. Then, there exist $\Phi_t \rhd \Sigma; x : M \vdash t : A$ and $\Phi_u \rhd \Delta \vdash u : M$, such that $\Gamma = \Sigma + \Delta$ and $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{t\{x \setminus u\}}) = \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_t) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u)$.*

Proof. We generalize the original statement by allowing $\Phi_{t\{x \setminus u\}}$ to conclude with either a term type A or a multiset type M.

If $\Phi_{t\{x \setminus u\}} \rhd \Gamma \vdash t\{x \setminus u\} : T$, where $T \in \{A, M\}$. Then, there exist derivations $\Phi_t \rhd \Sigma; x : M \vdash t : T$ and $\Phi_u \rhd \Delta \vdash u : M$, such that $\Gamma = \Sigma + \Delta$ and $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{t\{x \setminus u\}}) = \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_t) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u)$.

We start by splitting into two cases, according to the form of t:

- Case $t = x$. Then, $t\{x \setminus u\} = u$ and either $T = A$ or $T = M_0$. Let $T = A$. Then, we can build Φ_t and Φ_u respectively as follows:

$$\frac{}{x : \{\!\{A\}\!\} \vdash x : A}\ \texttt{Ax} \qquad \frac{\Phi_{t\{x\setminus u\}}}{\Delta \vdash u : \{\!\{A\}\!\}}\ \texttt{Many}$$

  where $\Sigma = \emptyset$ and $M = \{\!\{A\}\!\}$. We can conclude since $\Gamma = \Delta = \Sigma + \Delta$ and

$$\begin{aligned} \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_{t\{x\setminus u\}}) &= \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u) \\ &= \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u) + \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_t). \end{aligned}$$

  Now, let $T = M_0 = \{\!\{A_i\}\!\}_{i\in \mathtt{I}}$. Then, we can build Φ_t as follows:

$$\frac{\left(\dfrac{}{x : \{\!\{A_i\}\!\} \vdash x : A_i}\ \texttt{Ax}\right)_{i\in\mathtt{I}}}{x : \{\!\{A_i\}\!\}_{i\in\mathtt{I}} \vdash x : \{\!\{A_i\}\!\}_{i\in\mathtt{I}}}\ \texttt{Many}$$

  where $\Sigma = \emptyset$ and $M = M_0$, and pick $\Phi_u = \Phi_{t\{x\setminus u\}}$, where $\Delta = \Gamma$. We can conclude since $\Gamma = \Delta = \Sigma + \Delta$ and

$$\begin{aligned} \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_{t\{x\setminus u\}}) &= \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u) \\ &= \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u) + \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_t). \end{aligned}$$

- Case $t \neq x$. The proof follows by induction on $\Phi_{t\{x\setminus u\}}$. We reason according to the last rule:

  - Rule `Ax`. Then, $t = y$ and $t\{x \setminus u\} = y$. Therefore, we can pick $\Phi_t = \Phi_{t\{x\setminus u\}}$, where $\Sigma; x : M = \Gamma$ and thus $\Sigma = \Gamma$ and $M = \{\!\{\,\}\!\}$, and build Φ_u as follows:

$$\frac{}{\vdash u : \{\!\{\,\}\!\}}\ \texttt{Many}$$

    where $\Delta = \emptyset$. We can conclude since $\Gamma = \Sigma = \Sigma + \Delta$ and $\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_{t\{x\setminus u\}}) = \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_t) = \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_t) + \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u)$.

  - Rule `Many`. Then, $\Phi_{t\{x\setminus u\}}$ must be of the following form:

$$\frac{(\ \Phi_i \triangleright \Gamma_i \vdash t\{x \setminus u\} : A_i)_{i\in\mathtt{I}}}{+_{i\in\mathtt{I}}\Gamma_i \vdash t\{x \setminus u\} : \{\!\{A_i\}\!\}_{i\in\mathtt{I}}}\ \texttt{Many}$$

    where $\Gamma = +_{i\in\mathtt{I}}\Gamma_i$ and $T = \{\!\{A_i\}\!\}_{i\in\mathtt{I}}$. By the induction hypothesis over each Φ_i, there exist $\Phi_i' \triangleright \Sigma_i; x : M_i \vdash t : A_i$ and $\Phi_u^i \triangleright \Delta_i \vdash u : M_i$, such that $\Gamma_i = \Sigma_i + \Delta_i$ and $\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_i) = \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_i') + \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u^i)$. By Lemma 6.29 over the Φ_u^i, we know there exists $\Phi_u \triangleright \Delta \vdash u : M$, such that $\Delta = +_{i\in\mathtt{I}}\Delta_i$, $M = \sqcup_{i\in\mathtt{I}} M_i$, and $\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u) = +_{i\in\mathtt{I}}\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u^i)$. We can build Φ_t as follows:

$$\frac{(\ \Phi_t^i)_{i\in\mathtt{I}}}{+_{i\in\mathtt{I}}(\Sigma_i; x : M_i) \vdash t : \{\!\{A_i\}\!\}_{i\in\mathtt{I}}}\ \texttt{Many}$$

where $\Sigma = +_{i\in \mathtt{I}}\Sigma_i$ and $M = \sqcup_{i\in \mathtt{I}} M_i$. We can conclude since $\Gamma = +_{i\in \mathtt{I}}\Gamma_i = +_{i\in \mathtt{I}}(\Sigma_i + \Delta_i) = (+_{i\in \mathtt{I}}\Sigma_i) + (+_{i\in \mathtt{I}}\Delta_i) = \Sigma + \Delta$ and

$$\begin{array}{rcl} \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{t\{x\setminus u\}}) & = & +_{i\in \mathtt{I}}\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_i) \\ & = & +_{i\in \mathtt{I}}(\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi'_i) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi^i_u)) \\ & = & +_{i\in \mathtt{I}}\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi'_i) +_{i\in \mathtt{I}} \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi^i_u) \\ & = & \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_t) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u). \end{array}$$

– Rule `Abs`. Then, $t\{x \setminus u\} = \lambda p.s$, such that $s = s'\{x \setminus u\}$ and $t = \lambda p.s'$. Therefore, $\mathtt{vars}(p) \cap \mathtt{fv}(u) = \emptyset$ and $\Phi_{t\{x\setminus u\}}$ must be of the following form:

$$\frac{\Phi_s \triangleright \Gamma_s \vdash s : A \qquad \Pi \triangleright \Gamma_s|_p \Vdash p : M_0}{\Gamma_s \setminus\!\setminus \mathtt{vars}(p) \vdash \lambda p.s : M_0 \to A}\ \mathtt{Abs}$$

where $\Gamma = \Gamma_s \setminus\!\setminus \mathtt{vars}(p)$, and $T = M_0 \to A$. By the induction hypothesis over Φ_s, there exist $\Phi_{s'} \triangleright \Sigma_{s'}; x : M \vdash s' : A$ and $\Phi_u \triangleright \Delta \vdash u : M$, such that $\Gamma_s = \Sigma_{s'} + \Delta$ and $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_s) = \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{s'}) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u)$. Since $\mathtt{vars}(p) \cap \mathtt{fv}(u) = \emptyset$, we know $\mathtt{vars}(p) \cap \mathtt{dom}(\Delta) = \emptyset$ (by Lemma 6.30). Therefore, $\Gamma_s|_p = (\Sigma_{s'} + \Delta)|_p = \Sigma_{s'}|_p$, $\Gamma \setminus\!\setminus \mathtt{vars}(p) = (\Sigma_{s'} + \Delta) \setminus\!\setminus \mathtt{vars}(p) = (\Sigma_{s'} \setminus\!\setminus \mathtt{vars}(p)) + \Delta$, and we can build Φ_t as follows:

$$\frac{\Phi_{s'} \qquad \Pi}{(\Sigma_{s'}; x : M) \setminus\!\setminus \mathtt{vars}(p) \vdash \lambda p.s' : M_0 \to A}\ \mathtt{Abs}$$

where $(\Sigma_{s'}; x : M) \setminus\!\setminus \mathtt{vars}(p) = (\Sigma_{s'} \setminus\!\setminus \mathtt{vars}(p)); x : M$ (since we may assume $x \notin \mathtt{vars}(p)$ by α-conversion). Thus, $\Sigma = \Sigma_{s'} \setminus\!\setminus \mathtt{vars}(p)$, and we can conclude since $\Gamma = \Gamma_s \setminus\!\setminus \mathtt{vars}(p) = (\Sigma_{s'} + \Delta) \setminus\!\setminus \mathtt{vars}(p) = (\Sigma_{s'} \setminus\!\setminus \mathtt{vars}(p)) + \Delta = \Sigma + \Delta$ and

$$\begin{array}{rcl} \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{t\{x\setminus u\}}) & = & \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_s) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi) \\ & = & \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{s'}) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi) \\ & = & \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_t) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u). \end{array}$$

– Rule `Ans`. Then, $t\{x \setminus u\} = \lambda p.s$, such that $s = s'\{x \setminus u\}$ and $t = \lambda p.s'$. Therefore, $\mathtt{vars}(p) \cap \mathtt{fv}(u) = \emptyset$ and $\Phi_{t\{x\setminus u\}}$ must be of the following form:

$$\frac{}{\vdash \lambda p.s : \star}\ \mathtt{Ans}$$

where $\Gamma = \emptyset$ and $T = \star$. Therefore, we can build Φ_t as follows:

$$\frac{}{\vdash \lambda p.s' : \star}\ \mathtt{Ans}$$

where $\Sigma = \emptyset$ and $M = \{\!\{\ \}\!\}$, and Φ_u as follows:

$$\frac{}{\vdash u : \{\!\{\ \}\!\}}\ \texttt{Many}$$

where $\Delta = \emptyset$. We can conclude since $\Gamma = \emptyset = \Sigma + \Delta$ and

$$\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_{t\{x\setminus u\}}) = \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_t) + \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u).$$

– Rule `App`. Then, $t\{x \setminus u\} = sr$, such that $s = s'\{x \setminus u\}$, $r = r'\{x \setminus u\}$, and $t = s'r'$. Therefore, $\Phi_{t\{x\setminus u\}}$ must be of the following form:

$$\frac{\Phi_s \triangleright \Gamma_s \vdash s : M_0 \to A \qquad \Phi_r \triangleright \Gamma_r \vdash r : M_0}{\Gamma_s + \Gamma_r \vdash sr : A}\ \texttt{App}$$

where $\Gamma = \Gamma_s + \Gamma_r$ and $T = A$. By the induction hypothesis over Φ_s (resp. Φ_r), we know there exist derivations $\Phi_{s'} \triangleright \Sigma_{s'}; x : M_1 \vdash s' : M_0 \to A$ (resp. $\Phi_{r'} \triangleright \Sigma_{r'}; x : M_2 \vdash r' : A$) and $\Phi_u^1 \triangleright \Delta_1 \vdash u : M_1$ (resp. $\Phi_u^2 \triangleright \Delta_2 \vdash u : M_2$), such that $\Gamma_s = \Sigma_{s'} + \Delta_1$ (resp. $\Gamma_r = \Sigma_{r'} + \Delta_2$) and $\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_s) = \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_{s'}) + \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u^1)$ (resp. $\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_r) = \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_{r'}) + \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u^2)$). By Lemma 6.29 over Φ_u^1 and Φ_u^2, we know there exists a derivation $\Phi_u \triangleright \Delta \vdash u : M$, such that $\Delta = \Delta_1 + \Delta_2$, $M = M_1 \sqcup M_2$ and $\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u) = \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u^1) + \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u^2)$. We can build Φ_t as follows:

$$\frac{\Phi_{s'} \qquad \Phi_{r'}}{(\Gamma_{s'}; x : M_1) + (\Gamma_{r'}; x : M_2) \vdash s'r' : A}\ \texttt{App}$$

where $(\Gamma_{s'}; x : M_1) + (\Gamma_{r'}; x : M_2) = \Gamma_{s'} + \Gamma_{r'}; x : M$. Thus, $\Sigma = \Gamma_{s'} + \Gamma_{r'}$. We can conclude since $\Gamma = \Gamma_s + \Gamma_r = \Sigma_{s'} + \Delta_1 + \Sigma_{r'} + \Delta_2 = \Sigma + \Delta$ and

$$\begin{aligned}
\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_{t\{x\setminus u\}}) &= \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_s) + \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_r) + 1 \\
&= \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_{s'}) + \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u^1) \\
&+ \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_{r'}) + \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u^2) + 1 \\
&= \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_t) + \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u).
\end{aligned}$$

– Rule `Data`. Then, $t\{x \setminus u\} = \mathtt{c}(s_i)_n$, such that each $s_i = s_i'\{x \setminus u\}$ and $t = \mathtt{c}(s_i')_n$. Therefore, $\Phi_{t\{x\setminus u\}}$ must be of the following form:

$$\frac{\left(\Phi_{s_i} \triangleright \Gamma_i \vdash s_i : M_i^0\right)_{i\in\{1,\dots,n\}}}{+_{i\in\{1,\dots,n\}}\Gamma_{s_i} \vdash \mathtt{c}(s_i)_n : \mathtt{c}(M_i^0)_n}\ \texttt{Data}$$

where $\Gamma = +_{i\in\{1,\dots,n\}}\Gamma_{s_i}$, and $T = \mathtt{c}(M_i^0)_n$. By the induction hypothesis over each Φ_{s_i}, we know there exist $\Phi_{s_i'} \triangleright \Sigma_{s_i'}; x : M_i \vdash s_i' : M_i^0$ and $\Phi_u^i \triangleright \Delta_i \vdash u : M_i$, such that $\Gamma_{s_i} = \Sigma_{s_i'} + \Delta_i$, and $\#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_{s_i}) = \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_{s_i'}) + \#_{\texttt{App},\texttt{VPat},\texttt{DPat}}(\Phi_u^i)$. By Lemma 6.29 over Φ_u^i, we know there exists $\Phi_u \triangleright \Delta \vdash u : M$, such that $\Delta = +_{\{1,\dots,n\}}\Delta_i$, $M = \sqcup_{i\in\{1,\dots,n\}}M_i$ and

$\#_{\texttt{App,VPat,DPat}}(\Phi_u) = +_{i\in\{1,\ldots,n\}}\#_{\texttt{App,VPat,DPat}}(\Phi^i_u)$. We can build Φ_t as follows:

$$\frac{\left(\Phi_{s'_i}\right)_{i\in\{1,\ldots,n\}}}{+_{i\in\{1,\ldots,n\}}(\Sigma_{s'_i}; x : M_i) \vdash s'_i : \mathtt{c}(M^0_i)_n}\ \texttt{Data}$$

where $+_{i\in\{1,\ldots,n\}}(\Sigma_{s'_i}; x : M_i) = (+_{i\in\{1,\ldots,n\}}\Sigma_{s'_i}); x : M$. Thus, $\Sigma = +_{i\in\{1,\ldots,n\}}\Sigma_i$, and we can conclude since $\Gamma = +_{i\in\{1,\ldots,n\}}\Gamma_{s_i} = +_{i\in\{1,\ldots,n\}}(\Sigma_{s'_i} + \Delta_i) = (+_{i\in\{1,\ldots,n\}}\Sigma_{s'_i}) + (+_{i\in\{1,\ldots,n\}}\Delta_i) = \Sigma + \Delta$, and

$$\begin{aligned}
\#_{\texttt{App,VPat,DPat}}(\Phi_{t\{x\setminus u\}}) &= +_{i\in\{1,\ldots,n\}}\#_{\texttt{App,VPat,DPat}}(\Phi_{s_i}) \\
&= +_{i\in\{1,\ldots,n\}}(\#_{\texttt{App,VPat,DPat}}(\Phi_{s'_i}) + \#_{\texttt{App,VPat,DPat}}(\Phi^i_u)) \\
&= +_{i\in\{1,\ldots,n\}}\#_{\texttt{App,VPat,DPat}}(\Phi_{s'_i}) +_{i\in\{1,\ldots,n\}} \#_{\texttt{App,VPat,DPat}}(\Phi^i_u) \\
&= \#_{\texttt{App,VPat,DPat}}(\Phi_t) + \#_{\texttt{App,VPat,DPat}}(\Phi_u).
\end{aligned}$$

– Rule `Match`. Then, $t\{x \setminus u\} = s[p\backslash r]$, such that $s = s'\{x \setminus u\}$, $r = r'\{x \setminus u\}$ and $t = s'[p\backslash r']$. Therefore, $\texttt{vars}(p) \cap \texttt{fv}(u) = \emptyset$ and $\Phi_{t\{x\setminus u\}}$ must be of the following form:

$$\frac{\Phi_s \triangleright \Gamma_s \vdash s : A \qquad \Pi \triangleright \Gamma_s|_p \vdash p : M_0 \qquad \Phi_r \triangleright \Gamma_r \vdash r : M_0}{(\Gamma_s \setminus\!\setminus \texttt{vars}(p)) + \Gamma_r \vdash s[p\backslash r] : A}\ \texttt{Match}$$

where $\Gamma = (\Gamma_s \setminus\!\setminus \texttt{vars}(p)) + \Gamma_r$, and $T = A$. By the induction hypothesis over Φ_s (resp. Φ_r), we know there exist $\Phi_{s'} \triangleright \Sigma_{s'}; x : M_1 \vdash s' : A$ (resp. $\Phi_{r'} \triangleright \Sigma_{r'}; x : M_2 \vdash r' : M_0$) and $\Phi^1_u \triangleright \Delta_1 \vdash u : M_1$ (resp. $\Phi^2_u \triangleright \Delta_2 \vdash u : M_2$), such that $\Gamma_s = \Sigma_{s'} + \Delta_1$ (resp. $\Gamma_r = \Sigma_{r'} + \Delta_2$), and $\#_{\texttt{App,VPat,DPat}}(\Phi_s) = \#_{\texttt{App,VPat,DPat}}(\Phi_{s'}) + \#_{\texttt{App,VPat,DPat}}(\Phi^1_u)$ (resp. $\#_{\texttt{App,VPat,DPat}}(\Phi_r) = \#_{\texttt{App,VPat,DPat}}(\Phi_{r'}) + \#_{\texttt{App,VPat,DPat}}(\Phi^2_u)$). By Lemma 6.29 over Φ^1_u and Φ^2_u, we know there exists $\Phi_u \triangleright \Delta \vdash u : M$, such that $\Delta = \Delta_1 + \Delta_2$, $M = M_1 \sqcup M_2$, and $\#_{\texttt{App,VPat,DPat}}(\Phi_u) = \#_{\texttt{App,VPat,DPat}}(\Phi^1_u) + \#_{\texttt{App,VPat,DPat}}(\Phi^2_u)$. Since $\texttt{vars}(p) \cap \texttt{fv}(u) = \emptyset$, we know $\texttt{vars}(p) \cap \texttt{dom}(\Delta) = \emptyset$ (by Lemma 6.30). Therefore, $\Gamma_s|_p = (\Sigma_{s'} + \Delta_1)|_p = \Sigma_{s'}|_p$, $\Gamma_s \setminus\!\setminus \texttt{vars}(p) = (\Sigma_{s'} + \Delta) \setminus\!\setminus \texttt{vars}(p) = (\Sigma_{s'} \setminus\!\setminus \texttt{vars}(p)) + \Delta$, and we can build Φ_t as follows:

$$\frac{\Phi_{s'} \qquad \Pi \qquad \Phi_{r'}}{((\Sigma_{s'}; x : M_1) \setminus\!\setminus \texttt{vars}(p)) + \Sigma_{r'}; x : M_2 \vdash s'[p\backslash r'] : A}\ \texttt{Match}$$

where $(\Sigma_{s'}; x : M_1) \setminus\!\setminus \texttt{vars}(p) = \Sigma_{s'} \setminus\!\setminus \texttt{vars}(p); x : M_1$ (since we may assume $x \notin \texttt{vars}(p)$ by α-conversion). Thus, we know $((\Sigma_{s'}; x : M_1) \setminus\!\setminus \texttt{vars}(p)) + \Sigma_{r'}; x : M_2 = (\Sigma_{s'} \setminus\!\setminus \texttt{vars}(p)) + \Sigma_{r'}; x : M$, $\Sigma = (\Sigma_{s'} \setminus\!\setminus \texttt{vars}(p)) + \Sigma_{r'}$, and we can conclude since

$\Gamma = (\Gamma_s \setminus\setminus \mathtt{vars}(p)) + \Gamma_r = ((\Sigma_{s'} + \Delta_1) \setminus\setminus \mathtt{vars}(p)) + \Sigma_{r'} + \Delta_2 = \Sigma + \Delta$, and

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{t\{x\setminus u\}}) &= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_s) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_r) \\
&= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{s'}) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u^1) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi) \\
&+ \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{r'}) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u^2) \\
&= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_t) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u).
\end{aligned}$$

– Rule Case. Then, $t\{x \setminus u\} = \mathtt{case}\ r\ \mathtt{of}\ (\hat{p}_i.s_i)_n$, such that $r = r'\{x \setminus u\}$, $s_i = s_i'\{x \setminus u\}$ for each $i \in \mathtt{I}$, and $t = \mathtt{case}\ r'\ \mathtt{of}\ (\hat{p}_i.s_i')_n$. Therefore, $\mathtt{vars}(\hat{p}_i) \cap \mathtt{fv}(u) = \emptyset$ for all $i \in \mathtt{I}$, and $\Phi_{t\{x\setminus u\}}$ must be of the following form:

$$\frac{\Phi_r \triangleright \Gamma_r \vdash r : M_0 \qquad \Pi \triangleright \Gamma_{s_k}|_{\hat{p}_k} \vdash \hat{p}_k : M_0 \qquad \Phi_{s_k} \triangleright \Gamma_{s_k} \vdash s_k : A}{(\Gamma_{s_k} \setminus\setminus \mathtt{vars}(\hat{p}_k)) + \Gamma_r \vdash \mathtt{case}\ r\ \mathtt{of}\ (\hat{p}_i.s_i)_n : A}\ \mathtt{Case}$$

where $\Gamma = (\Gamma_{s_k} \setminus\setminus \mathtt{vars}(\hat{p}_k)) + \Gamma_r$, and $T = A$. By the induction hypothesis over Φ_{s_k} (resp. Φ_r), we know there exist $\Phi_{s_k'} \triangleright \Sigma_{s_k'}; x : M_1 \vdash s_k' : A$ (resp. $\Phi_{r'} \triangleright \Sigma_{r'}; x : M_2 \vdash r' : M_0$) and $\Phi_u^1 \triangleright \Delta_1 \vdash u : M_1$ (resp. $\Phi_u^2 \triangleright \Delta_2 \vdash u : M_2$), such that $\Gamma_{s_k} = \Sigma_{s_k'} + \Delta_1$ (resp. $\Gamma_r = \Sigma_{r'} + \Delta_2$) and $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{s_k}) = \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{s_k'}) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u^1)$ (resp. $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_r) = \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{r'}) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u^2)$). By Lemma 6.29 over Φ_u^1 and Φ_u^2, we know there exists $\Phi_u \triangleright \Delta \vdash u : M$, such that $\Delta = \Delta_1 + \Delta_2$, $M = M_1 \sqcup M_2$, and $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u) = \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u^1) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u^2)$. Since $\mathtt{vars}(\hat{p}_k) \cap \mathtt{fv}(u) = \emptyset$, we know $\mathtt{vars}(\hat{p}_k) \cap \mathtt{dom}(\Delta) = \emptyset$ (by Lemma 6.30). Therefore, $\Gamma_{s_k}|_{\hat{p}_k} = (\Sigma_{s_k'} + \Delta_1)|_{\hat{p}_k} = \Sigma_{s_k'}|_{\hat{p}_k}$, $\Gamma_{s_k} \setminus\setminus \mathtt{vars}(\hat{p}_k) = (\Sigma_{s_k'} + \Delta) \setminus\setminus \mathtt{vars}(\hat{p}_k) = (\Sigma_{s_k'} \setminus\setminus \mathtt{vars}(\hat{p}_k)) + \Delta$, and we can build Φ_t as follows:

$$\frac{\Phi_{r'} \qquad \Pi \qquad \Phi_{s_k'}}{((\Sigma_{s_k'}; x : M_1) \setminus\setminus \mathtt{vars}(\hat{p}_k)) + \Sigma_{r'}; x : M_2 \vdash \mathtt{case}\ r'\ \mathtt{of}\ (\hat{p}_i.s_i)_n : A}\ \mathtt{Case}$$

where $(\Sigma_{s_k'}; x : M_1) \setminus\setminus \mathtt{vars}(\hat{p}_k) = \Sigma_{s_k'} \setminus\setminus \mathtt{vars}(\hat{p}_k); x : M_1$ (since we may assume $x \notin \mathtt{vars}(\hat{p}_k)$ by α-conversion). Thus, we know that $((\Sigma_{s_k'}; x : M_1) \setminus\setminus \mathtt{vars}(\hat{p}_k)) + \Sigma_{r'}; x : M_2 = (\Sigma_{s_k'} \setminus\setminus \mathtt{vars}(\hat{p}_k)) + \Sigma_{r'}; x : M$, $\Sigma = (\Sigma_{s_k'} \setminus\setminus \mathtt{vars}(\hat{p}_k)) + \Sigma_{r'}$, and we can conclude since $\Gamma = (\Gamma_{s_k} \setminus\setminus \mathtt{vars}(\hat{p}_k)) + \Gamma_r = ((\Sigma_{s_k'} + \Delta_1) \setminus\setminus \mathtt{vars}(p)) + \Sigma_{r'} + \Delta_2 = \Sigma + \Delta$, and

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{t\{x\setminus u\}}) &= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{s_k}) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_r) \\
&= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{s_k'}) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u^1) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi) \\
&+ \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{r'}) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u^2) \\
&= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_t) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u).
\end{aligned}$$

□

Lemma 6.40 (Exact Subject Expansion). *Let $t \in \Lambda^{PM}$, and $S \in \{\mathtt{dB}, \mathtt{branch}, \mathtt{match}, \mathtt{sub}\}$. If $t \to_{\mathtt{PM}(S)} t'$ and $\Phi_{t'} \rhd_{\mathcal{PM}} \Gamma \vdash t' : A$, then there is $\Phi_t \rhd_{\mathcal{PM}} \Gamma \vdash t : A$, such that $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{t'}) + 1 \cdot R = \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_t)$, where*

- $R = \mathtt{App}$, *if* $S = \mathtt{dB}$;
- $R = \mathtt{VPat}$, *if* $S = \mathtt{sub}$;
- $R = \mathtt{DPat}$ *if* $S \in \{\mathtt{branch}, \mathtt{match}\}$.

Proof. The proof follows by induction on $t \to_{\mathtt{PM}} t'$:

- Case $t = \mathtt{L}_0(\!|\lambda p.u|\!)s \to_{\mathtt{PM}(\mathtt{dB})} \mathtt{L}_0(\!|u[p\backslash s]|\!) = t'$, such that $\mathtt{bv}(\mathtt{L}_0) \cap \mathtt{fv}(s) = \emptyset$. We proceed by induction on the list matching context $\mathtt{L}_0$:
  - Let $\mathtt{L}_0 = \Box$. Then, $\Phi_{t'}$ must be of the following form:

$$\frac{\Phi_u \rhd \Gamma_u \vdash u : A \qquad \Pi \rhd \Gamma_u|_p \Vdash p : M \qquad \Phi_s \rhd \Gamma_s \vdash s : M}{(\Gamma_u \setminus\!\setminus \mathtt{vars}(p)) + \Gamma_s \vdash u[p\backslash s] : A}\ \mathtt{Match}$$

    where $\Gamma = (\Gamma_u \setminus\!\setminus \mathtt{vars}(p)) + \Gamma_s$. Therefore, we can build Φ_t as follows:

$$\dfrac{\dfrac{\Phi_u \qquad \Pi}{\Gamma_u \setminus\!\setminus \mathtt{vars}(p) \vdash \lambda p.u : M \to A}\ (\mathtt{Abs}) \qquad \Phi_s}{(\Gamma_u \setminus\!\setminus \mathtt{vars}(p)) + \Gamma_s \vdash (\lambda p.u)s : A}\ (\mathtt{App})$$

    We can conclude with

$$\begin{aligned} \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{t'}) + 1 \cdot \mathtt{App} &= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi) \\ &+ \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_s) + 1 \cdot \mathtt{App} \\ &= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}([]\Phi_t]. \end{aligned}$$

  - Let $\mathtt{L}_0 = \mathtt{L}_1[q\backslash r]$. Then, $\Phi_{t'}$ must be of the following form:

$$\frac{\Phi_{\mathtt{L}_1(\!|u[p\backslash s]|\!)} \rhd \Gamma_{\mathtt{L}_1(\!|u[p\backslash s]|\!)} \vdash \mathtt{L}_1(\!|u[p\backslash s]|\!) : A \qquad \Pi \rhd \Gamma_{\mathtt{L}_1(\!|u[p\backslash s]|\!)}|_q \Vdash q : M_q \qquad \Phi_r \rhd \Gamma_r \vdash r : M_q}{(\Gamma_{\mathtt{L}_1(\!|u[p\backslash s]|\!)} \setminus\!\setminus \mathtt{vars}(q)) + \Gamma_r \vdash \mathtt{L}_1(\!|u[p\backslash s]|\!)[q\backslash r] : A}\ \mathtt{Match}$$

    where $\Gamma = (\Gamma_{\mathtt{L}_1(\!|u[p\backslash s]|\!)} \setminus\!\setminus \mathtt{vars}(q)) + \Gamma_r$. Since $\mathtt{bv}(\mathtt{L}_1) \subseteq \mathtt{bv}(\mathtt{L}_0)$, we can do the step $\mathtt{L}_1(\!|\lambda p.u|\!)s \to_{\mathtt{PM}(\mathtt{dB})} \mathtt{L}_1(\!|u[p\backslash s]|\!)$. By the *i.h.* over $\mathtt{L}_1(\!|\lambda p.u|\!)s \to_{\mathtt{PM}} \mathtt{L}_1(\!|u[p\backslash s]|\!)$, we know there exists a derivation $\Psi_{\mathtt{L}_1(\!|\lambda p.u|\!)s}$ for $\mathtt{L}_1(\!|\lambda p.u|\!)s$, which must be of the following form:

$$\frac{\Phi_{\mathtt{L}_1(\!|\lambda p.u|\!)} \rhd \Gamma_{\mathtt{L}_1(\!|\lambda p.u|\!)} \vdash \mathtt{L}_1(\!|\lambda p.u|\!) : M_0 \to A \qquad \Phi_s \rhd \Gamma_s \vdash s : M_0}{\Gamma_{\mathtt{L}_1(\!|u[p\backslash s]|\!)} \vdash \mathtt{L}_1(\!|\lambda p.u|\!)s : A}\ (\mathtt{App})$$

    where $\Gamma_{\mathtt{L}_1(\!|u[p\backslash s]|\!)} = \Gamma_{\mathtt{L}_1(\!|\lambda p.u|\!)} + \Gamma_s$ and $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Psi_{\mathtt{L}_1(\!|\lambda p.u|\!)s}) = \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{\mathtt{L}_1(\!|u[p\backslash s]|\!)}) + 1 \cdot \mathtt{App}$. Since $\mathtt{bv}(\mathtt{L}_0) \cap \mathtt{fv}(s) = \emptyset$, we know that $\mathtt{fv}(s) \cap \mathtt{vars}(q) = \emptyset$ and thus $\mathtt{vars}(q) \cap$

$\mathtt{dom}(\Gamma_s) = \emptyset$ (Lemma 6.30). Therefore, we can build Φ_t as follows:

$$\dfrac{\dfrac{\Phi_{\mathtt{L}_1\langle\!|\lambda p.u|\!\rangle} \quad \Pi \quad \Phi_r}{(\Gamma_{\mathtt{L}_1\langle\!|\lambda p.u|\!\rangle} \setminus\!\setminus \mathtt{vars}(q)) + \Gamma_r \vdash \mathtt{L}_1\langle\!|\lambda p.u|\!\rangle[q\backslash r] : M_0 \to A}\,\mathtt{Match} \quad \Phi_s}{(\Gamma_{\mathtt{L}_1\langle\!|\lambda p.u|\!\rangle} \setminus\!\setminus \mathtt{vars}(q)) + \Gamma_r + \Gamma_s \vdash \mathtt{L}_1\langle\!|\lambda p.u|\!\rangle[q\backslash r]s : A}\,(\mathtt{App})$$

We can conclude since $(\Gamma_{\mathtt{L}_1\langle\!|\lambda p.u|\!\rangle} \setminus\!\setminus \mathtt{vars}(q)) + \Gamma_r + \Gamma_s = (\Gamma_{\mathtt{L}_1\langle\!|u[p\backslash s]|\!\rangle} \setminus\!\setminus \mathtt{vars}(q)) + \Gamma_r$, and

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{t'}) + 1 \cdot \mathtt{App} &= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{\mathtt{L}_1\langle\!|u[p\backslash s]|\!\rangle}) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi) \\
&+ \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_r) + 1 \cdot \mathtt{App} \\
&= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Psi_{\mathtt{L}_1\langle\!|\lambda p.u|\!\rangle s}) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi) \\
&+ \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_r) \\
&= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{\mathtt{L}_1\langle\!|\lambda p.u|\!\rangle}) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_s) + 1 \cdot \mathtt{App} \\
&+ \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_r) \\
&= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_t).
\end{aligned}$$

- Case $t = u[\mathtt{c}(p_i)_n \backslash \mathtt{L}_0\langle\!|\mathtt{c}(s_i)_n|\!\rangle] \to_{\mathtt{PM}(\mathtt{match})} \mathtt{L}_0\langle\!|u[p_1\backslash s_1]\cdots[p_n\backslash s_n]|\!\rangle = t'$, such that $\mathtt{bv}(\mathtt{L}_0) \cap \mathtt{fv}(u) = \emptyset$. We proceed by induction on the list matching context $\mathtt{L}_0$:

  - Case $\mathtt{L}_0 = \Box$. Then, $\Phi_{t'}$ must be of the following form:

$$\dfrac{\dfrac{\begin{array}{c}\Phi_u \triangleright \Gamma_u \vdash u : A \\ \Pi_{p_1} \triangleright \Gamma_u|_{p_1} \Vdash p_1 : M_1 \\ \Phi_{s_1} \triangleright \Gamma_{s_1} \vdash s_1 : M_1\end{array}}{\Gamma_u \setminus\!\setminus \mathtt{vars}(p_1) + \Gamma_{s_1} \vdash u[p_1\backslash s_1] : A}\,\mathtt{Match} \quad \begin{array}{c}\vdots \\ \end{array} \quad \dfrac{\Pi_{p_n} \triangleright \Gamma_u|_{p_n} \Vdash p_1 : M_n \quad \Phi_{s_n} \triangleright \Gamma_{s_n} \vdash s_n : M_n}{((((\Gamma_u \setminus\!\setminus \mathtt{vars}(p_1)) + \Gamma_{s_1})\cdots) \setminus\!\setminus \mathtt{vars}(p_n)) + \Gamma_{s_n} \vdash u[p_1\backslash s_1]\cdots[p_n\backslash s_n] : A}\,\mathtt{Match}}{}\,\mathtt{Match}$$

    where $\Gamma = ((((\Gamma_u \setminus\!\setminus \mathtt{vars}(p_1)) + \Gamma_{s_1})\cdots) \setminus\!\setminus \mathtt{vars}(p_n)) + \Gamma_{s_n}$. By α-conversion, we can assume that $\mathtt{fv}(\mathtt{c}(s_i)_n) \cap \mathtt{vars}(\mathtt{c}(p_i)_n) = \emptyset$ and thus $\mathtt{dom}(+_{\{1,\ldots,n\}}\Gamma_{s_i}) \cap \mathtt{vars}(\mathtt{c}(p_i)_n) = \emptyset$ (Lemma 6.30). Thus, $(+_{\{1,\ldots,n\}}\Gamma_{s_i}) \setminus\!\setminus \mathtt{vars}(\mathtt{c}(p_i)_n) = +_{\{1,\ldots,n\}}\Gamma_{s_i}$, $(+_{\{1,\ldots,n\}}\Gamma_{s_i})|_{\mathtt{c}(p_i)_n} = \emptyset$, and we can build Φ_t as follows:

$$\dfrac{\Phi_u \quad \dfrac{(\,\Pi_{p_i})_{i\in\{1,\ldots,n\}}}{\Gamma_u|_{\mathtt{c}(p_i)_n} \Vdash \mathtt{c}(p_i)_n : \{\!\{\mathtt{c}(M_i)_n\}\!\}}\,\mathtt{DPat} \quad \dfrac{\dfrac{(\,\Phi_{s_i})_{i\in\{1,\ldots,n\}}}{+_{i\in\{1,\ldots,n\}}\Gamma_{s_i} \vdash \mathtt{c}(s_i)_n : \mathtt{c}(M_i)_n}\,\mathtt{Data}}{+_{i\in\{1,\ldots,n\}}\Gamma_{s_i} \vdash \mathtt{c}(s_i)_n : \{\!\{\mathtt{c}(M_i)_n\}\!\}}\,\mathtt{Many}}{(\Gamma_u \setminus\!\setminus \mathtt{vars}(\mathtt{c}(p_i)_n)) +_{i\in\{1,\ldots,n\}} \Gamma_{s_i} \vdash u[\mathtt{c}(p_i)_n \backslash \mathtt{c}(s_i)_n] : A}\,\mathtt{Match}$$

We can conclude since $\Gamma = ((((\Gamma_u \setminus\setminus \mathtt{vars}(p_1)) + \Gamma_{s_1}) \cdots) \setminus\setminus \mathtt{vars}(p_n)) + \Gamma_{s_n}$ $= (\Gamma_u \setminus\setminus \mathtt{vars}(\mathtt{c}(p_i)_n)) +_{i\in\{1,\ldots,n\}} \Gamma_{s_i}$ and

$$\begin{array}{rcl} \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{t'}) + 1 \cdot \mathtt{DPat} & = & \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u) +_{i\in\{1,\ldots,n\}} \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi_{p_i}) \\ & + & {}_{i\in\{1,\ldots,n\}}\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{s_i}) + 1 \cdot \mathtt{DPat} \\ & = & \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_t). \end{array}$$

– Case $\mathtt{L}_0 = \mathtt{L}_1[q\backslash r]$. Then, $\Phi_{t'}$ must be of the following form:

$$\frac{\begin{array}{c}\Phi_{\mathtt{L}_1(\!|u[p_1\backslash s_1]\cdots[p_n\backslash s_n]|\!)} \triangleright \Gamma_{\mathtt{L}_1(\!|u[p_1\backslash s_1]\cdots[p_n\backslash s_n]|\!)} \vdash \mathtt{L}_1(\!|u[p_1\backslash s_1]\cdots[p_n\backslash s_n]|\!) : A \\ \Pi_q \triangleright \Gamma_{\mathtt{L}_1(\!|u[p_1\backslash s_1]\cdots[p_n\backslash s_n]|\!)|_q} \vdash q : M_q \\ \Phi_r \triangleright \Gamma_r \vdash r : M_q\end{array}}{(\Gamma_{\mathtt{L}_1(\!|u[p_1\backslash s_1]\cdots[p_n\backslash s_n]|\!)} \setminus\setminus \mathtt{vars}(q)) + \Gamma_r \vdash \mathtt{L}_1(\!|u[p_1\backslash s_1]\cdots[p_n\backslash s_n]|\!)[q\backslash r] : A}\ \mathtt{Match}$$

where $\Gamma = (\Gamma_{\mathtt{L}_1(\!|u[p_1\backslash s_1]\cdots[p_n\backslash s_n]|\!)} \setminus\setminus \mathtt{vars}(q)) + \Gamma_r$. Since $\mathtt{bv}(\mathtt{L}_0) \cap \mathtt{fv}(u) = \emptyset$, we can do the step $u[\mathtt{c}(p_i)_n\backslash \mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)] \to_{\mathtt{PM}(\mathtt{match})} \mathtt{L}_1(\!|u[p_1\backslash s_1]\cdots[p_n\backslash s_n]|\!)$. By the *i.h.* over $u[\mathtt{c}(p_i)_n\backslash \mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)] \to_{\mathtt{PM}(\mathtt{match})} \mathtt{L}_1(\!|u[p_1\backslash s_1]\cdots[p_n\backslash s_n]|\!)$, we know there exists a derivation $\Psi_{u[\mathtt{c}(p_i)_n\backslash \mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)]}$ for $u[\mathtt{c}(p_i)_n\backslash \mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)]$, which must be of the following form:

$$\frac{\Phi_u \triangleright \Gamma_u \vdash u : A \quad \Pi \triangleright \Gamma_u|_{\mathtt{c}(p_i)_n} \Vdash \mathtt{c}(p_i)_n : M \quad \Phi_{\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)} \triangleright \Gamma_{\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)} \vdash \mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!) : M}{\Gamma_{\mathtt{L}_1(\!|u[p_1\backslash s_1]\cdots[p_n\backslash s_n]|\!)} \vdash u[\mathtt{c}(p_i)_n\backslash \mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)] : A}\ \mathtt{Match}$$

where $\Gamma_{\mathtt{L}_1(\!|u[p_1\backslash s_1]\cdots[p_n\backslash s_n]|\!)} = (\Gamma_u \setminus\setminus \mathtt{vars}(\mathtt{c}(p_i)_n)) + \Gamma_{\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)}$ and we also have that the following holds $\Psi_{u[\mathtt{c}(p_i)_n\backslash \mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)]} = \Phi_{\mathtt{L}_1(\!|u[p_1\backslash s_1]\cdots[p_n\backslash s_n]|\!)} + 1 \cdot \mathtt{DPat}$. Since $\mathtt{bv}(\mathtt{L}_0) \cap \mathtt{fv}(u)$, we know that $\mathtt{fv}(u) \cap \mathtt{vars}(q) = \emptyset$ and thus $\mathtt{vars}(q) \cap \mathtt{dom}(\Gamma_u) = \emptyset$ (Lemma 6.30). Therefore, we can build Φ_t as follows:

$$\frac{\Phi_u \quad \Pi \quad \dfrac{\Phi_{\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)} \quad \Pi_q \quad \Phi_r}{(\Gamma_{\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)} \setminus\setminus \mathtt{vars}(q)) + \Gamma_r \vdash \mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)[q\backslash r] : M}\ \mathtt{Match}}{(\Gamma_u \setminus\setminus \mathtt{vars}(\mathtt{c}(p_i)_n)) + (\Gamma_{\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)} \setminus\setminus \mathtt{vars}(q)) + \Gamma_r \vdash u[\mathtt{c}(p_i)_n\backslash \mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)[q\backslash r]] : A}\ \mathtt{Match}$$

We can conclude since we know that $\mathtt{vars}(\mathtt{c}(p_i)_n) = \mathtt{vars}(p_1) \cup \cdots \cup \mathtt{vars}(p_n)$, but also $\Gamma = (\Gamma_{\mathtt{L}_1(\!|u[p_1\backslash s_1]\cdots[p_n\backslash s_n]|\!)} \setminus\setminus \mathtt{vars}(q)) + \Gamma_r = (\Gamma_u \setminus\setminus \mathtt{vars}(\mathtt{c}(p_i)_n)) + (\Gamma_{\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)} \setminus\setminus \mathtt{vars}(q)) +$

Γ_r, and

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{t'}) + 1 \cdot \mathtt{DPat} &= \Phi_{\mathtt{L}_1(\!|u[p_1\backslash s_1]\cdots[p_n\backslash s_n]|\!)} + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi_q) \\
&+ \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_r) + 1 \cdot \mathtt{DPat} \\
&= \Psi_{u[\mathtt{c}(p_i)_n\backslash \mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)]} + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi_q) \\
&+ \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_r) \\
&= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi) \\
&+ \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{\mathtt{L}(\!|\mathtt{c}(s_i)_n|\!)}) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi_q) \\
&+ \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_r) \\
&= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_t).
\end{aligned}$$

- Case $t = u[x\backslash s] \to_{\mathtt{PM(sub)}} u\{x \setminus s\} = t'$. By Lemma 6.39 over $\Phi_{t'}$, we know there exist $\Phi_u \triangleright \Gamma_u; x : M \vdash u : A$ and $\Phi_s \triangleright \Gamma_s \vdash s : M$, such that $\Gamma = \Gamma_u + \Gamma_s$ and $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{t'}) = \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_s)$. Therefore, we can build Φ_t as follows:

$$\dfrac{\Phi_u \qquad \dfrac{}{x : M \Vdash x : M}\,(\mathtt{VPat}) \qquad \Phi_s}{\Gamma_u + \Gamma_s \vdash u[x\backslash s] : A}\,\mathtt{Match}$$

And we can conclude with

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{t'}) + 1 \cdot \mathtt{VPat} &= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_u) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_s) + 1 \cdot \mathtt{VPat} \\
&= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_t).
\end{aligned}$$

- Case $t = \mathtt{case}\ \mathtt{L}_0(\!|\mathtt{c}(s_i)_n|\!)\ \mathtt{of}\ (\ldots, \mathtt{c}(p_i)_n.u_i, \ldots) \to_{\mathtt{PM(branch)}} \mathtt{L}_0(\!|u_i[p_1\backslash s_1]\cdots[p_n\backslash s_n]|\!) = t'$, such that $\mathtt{bv}(\mathtt{L}_0) \cap \mathtt{fv}(s_i) = \emptyset$. We proceed by induction on the list matching context $\mathtt{L}_0$:

  - Case $\mathtt{L}_0 = \Box$. Then, $\Phi_{t'}$ must be of the following form:

$$\dfrac{\dfrac{\begin{array}{c}\Phi_{u_i} \triangleright \Gamma_{u_i} \vdash u_i : A \\ \Pi_{p_1} \triangleright \Gamma_{u_i}|_{p_1} \Vdash p_1 : M_1 \\ \Phi_{s_1} \triangleright \Gamma_{s_1} \vdash s_1 : M_1\end{array}}{\Gamma_{u_i} \setminus\!\setminus \mathtt{vars}(p_1) + \Gamma_{s_1} \vdash u_i[p_1\backslash s_1] : A}\,\mathtt{Match} \qquad \vdots \qquad \Pi_{p_n} \triangleright \Gamma_{u_i}|_{p_n} \Vdash p_n : M_n \quad \Phi_{s_n} \triangleright \Gamma_{s_n} \vdash s_n : M_n}{((((\Gamma_{u_i} \setminus\!\setminus \mathtt{vars}(p_1)) + \Gamma_{s_1}) \cdots) \setminus\!\setminus \mathtt{vars}(p_n)) + \Gamma_{s_n} \vdash u_i[p_1\backslash s_1] \cdots [p_n\backslash s_n] : A}\,\mathtt{Match}$$

    where $\Gamma = ((((\Gamma_{u_i} \setminus\!\setminus \mathtt{vars}(p_1)) + \Gamma_{s_1}) \cdots) \setminus\!\setminus \mathtt{vars}(p_n)) + \Gamma_{s_n}$. By α-conversion, we can assume that $\mathtt{fv}(\mathtt{c}(s_i)_n) \cap \mathtt{vars}(\mathtt{c}(p_i)_n) = \emptyset$ and thus $\mathtt{dom}(+_{\{1,\ldots,n\}}\Gamma_{s_i}) \cap \mathtt{vars}(\mathtt{c}(p_i)_n) = \emptyset$ (Lemma 6.30). Thus, $(+_{\{1,\ldots,n\}}\Gamma_{s_i}) \setminus\!\setminus \mathtt{vars}(\mathtt{c}(p_i)_n) = +_{\{1,\ldots,n\}}\Gamma_{s_i}$, $(+_{\{1,\ldots,n\}}\Gamma_{s_i})|_{\mathtt{c}(p_i)_n} = \emptyset$, and we can build Φ_t as follows:

$$\dfrac{\dfrac{\dfrac{(\,\Phi_{s_i})_{i\in\{1,\ldots,n\}}}{+_{i\in\{1,\ldots,n\}}\Gamma_{s_i} \vdash \mathtt{c}(s_i)_n : \mathtt{c}(M_i)_n}\,\mathtt{Data}}{+_{i\in\{1,\ldots,n\}}\Gamma_{s_i} \vdash \mathtt{c}(s_i)_n : \{\!\{\mathtt{c}(M_i)_n\}\!\}}\,\mathtt{Many} \qquad \dfrac{(\,\Pi_{p_i})_{i\in\{1,\ldots,n\}}}{\Gamma_{u_i}|_{\mathtt{c}(p_i)_n} \Vdash \mathtt{c}(p_i)_n : \{\!\{\mathtt{c}(M_i)_n\}\!\}}\,\mathtt{DPat} \qquad \Phi_{u_i}}{(\Gamma_{u_i} \setminus\!\setminus \mathtt{vars}(\mathtt{c}(p_i)_n)) + _{i\in\{1,\ldots,n\}}\,\Gamma_{s_i} \vdash \mathtt{case}\ \mathtt{c}(s_i)_n\ \mathtt{of}\ (\ldots, \mathtt{c}(p_i)_n.u_i, \ldots) : A}\,\mathtt{Case}$$

We can conclude since we have that $\mathtt{vars}(\mathtt{c}(p_i)_n) = \mathtt{vars}(p_1) \cup \cdots \cup \mathtt{vars}(p_n)$, $\Gamma = ((((\Gamma_{u_i} \setminus\setminus \mathtt{vars}(p_1)) + \Gamma_{s_1}) \cdots) \setminus\setminus \mathtt{vars}(p_n)) + \Gamma_{s_n} = (\Gamma_{u_i} \setminus\setminus \mathtt{vars}(\mathtt{c}(p_i)_n)) +_{i \in \{1,\ldots,n\}} \Gamma_{s_i}$, and

$$\begin{array}{rcl} \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{t'}) + 1 \cdot \mathtt{DPat} & = & \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{u_i}) +_{i \in \{1,\ldots,n\}} \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi_{p_i}) \\ & + & {}_{i \in \{1,\ldots,n\}} \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{s_i}) + 1 \cdot \mathtt{DPat} \\ & = & \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_t). \end{array}$$

– Case $\mathtt{L}_0 = \mathtt{L}_1[q \backslash r]$. Then, $\Phi_{t'}$ must be of the following form:

$$\frac{\begin{array}{c} \Phi_{\mathtt{L}_1(\!|u_i[p_1 \backslash s_1] \cdots [p_n \backslash s_n]|\!)} \triangleright \Gamma_{\mathtt{L}_1(\!|u_i[p_1 \backslash s_1] \cdots [p_n \backslash s_n]|\!)} \vdash \mathtt{L}_1(\!|u_i[p_1 \backslash s_1] \cdots [p_n \backslash s_n]|\!) : A \\ \Pi_q \triangleright \Gamma_{\mathtt{L}_1(\!|u_i[p_1 \backslash s_1] \cdots [p_n \backslash s_n]|\!)} |_q \vdash q : M_q \\ \Phi_r \triangleright \Gamma_r \vdash r : M_q \end{array}}{(\Gamma_{\mathtt{L}_1(\!|u_i[p_1 \backslash s_1] \cdots [p_n \backslash s_n]|\!)} \setminus\setminus \mathtt{vars}(q)) + \Gamma_r \vdash \mathtt{L}_1(\!|u_i[p_1 \backslash s_1] \cdots [p_n \backslash s_n]|\!)[q \backslash r] : A}\ \mathtt{Match}$$

where $\Gamma = (\Gamma_{\mathtt{L}_1(\!|u_i[p_1 \backslash s_1] \cdots [p_n \backslash s_n]|\!)} \setminus\setminus \mathtt{vars}(q)) + \Gamma_r$. Since $\mathtt{bv}(\mathtt{L}_0) \cap \mathtt{fv}(u_i) = \emptyset$, we can do the step $\mathtt{case}\ \mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)\ \mathtt{of}\ (\ldots, \mathtt{c}(p_i)_n.u_i, \ldots) \to_{\mathtt{PM(branch)}} \mathtt{L}_1(\!|u_i[p_1 \backslash s_1] \cdots [p_n \backslash s_n]|\!)$. By the *i.h.* over $\mathtt{case}\ \mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)\ \mathtt{of}\ (\ldots, \mathtt{c}(p_i)_n.u_i, \ldots) \to_{\mathtt{PM(branch)}} \mathtt{L}_1(\!|u_i[p_1 \backslash s_1] \cdots [p_n \backslash s_n]|\!)$, we know there exists a type derivation for the form $\Psi_{\mathtt{case}\ \mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)\ \mathtt{of}\ (\ldots,\mathtt{c}(p_i)_n.u_i,\ldots)}$ for $\mathtt{case}\ \mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)\ \mathtt{of}\ (\ldots, \mathtt{c}(p_i)_n.u_i, \ldots)$, which must be of the following form:

$$\frac{\Phi_{\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)} \triangleright \Gamma_{\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)} \vdash \mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!) : M \quad \Pi \triangleright \Gamma_{u_i}|_{\mathtt{c}(p_i)_n} \Vdash \mathtt{c}(p_i)_n : M \quad \Phi_{u_i} \triangleright \Gamma_{u_i} \vdash u_i : A}{\Gamma_{\mathtt{L}_1(\!|u_i[p_1 \backslash s_1] \cdots [p_n \backslash s_n]|\!)} \vdash \mathtt{case}\ \mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)\ \mathtt{of}\ (\ldots, \mathtt{c}(p_i)_n.u_i, \ldots) : A}\ \mathtt{Case}$$

where $\Gamma_{\mathtt{L}_1(\!|u_i[p_1 \backslash s_1] \cdots [p_n \backslash s_n]|\!)} = (\Gamma_{u_i} \setminus\setminus \mathtt{vars}(\mathtt{c}(p_i)_n)) + \Gamma_{\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)}$ and it is also the case that $\Psi_{\mathtt{case}\ \mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)\ \mathtt{of}\ (\ldots,\mathtt{c}(p_i)_n.u_i,\ldots)} = \Phi_{\mathtt{L}_1(\!|u_i[p_1 \backslash s_1] \cdots [p_n \backslash s_n]|\!)} + 1 \cdot \mathtt{DPat}$. Since $\mathtt{bv}(\mathtt{L}_0) \cap \mathtt{fv}(u_i)$, we know that $\mathtt{fv}(u_i) \cap \mathtt{vars}(q) = \emptyset$ and thus $\mathtt{vars}(q) \cap \mathtt{dom}(\Gamma_{u_i}) = \emptyset$ (Lemma 6.30). Therefore, we can build Φ_t as follows:

$$\frac{\dfrac{\Phi_{\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)} \quad \Pi_q \quad \Phi_r}{(\Gamma_{\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)} \setminus\setminus \mathtt{vars}(q)) + \Gamma_r \vdash \mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)[q \backslash r] : M}\ \mathtt{Match} \quad \Pi \quad \Phi_{u_i}}{(\Gamma_{u_i} \setminus\setminus \mathtt{vars}(\mathtt{c}(p_i)_n)) + (\Gamma_{\mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)} \setminus\setminus \mathtt{vars}(q)) + \Gamma_r \vdash \mathtt{case}\ \mathtt{L}_1(\!|\mathtt{c}(s_i)_n|\!)[q \backslash r]\ \mathtt{of}\ (\ldots, \mathtt{c}(p_i)_n.u_i, \ldots) : A}\ \mathtt{Match}$$

We can conclude since $\Gamma = (\Gamma_{\mathtt{L}_1\langle\!|u_i[p_1\setminus s_1]\cdots[p_n\setminus s_n]|\!\rangle} \setminus\!\setminus \mathtt{vars}(q)) + \Gamma_r = (\Gamma_{u_i} \setminus\!\setminus \mathtt{vars}(\mathtt{c}(p_i)_n)) + (\Gamma_{\mathtt{L}_1\langle\!|\mathtt{c}(s_i)_n|\!\rangle} \setminus\!\setminus \mathtt{vars}(q)) + \Gamma_r$, and

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{t'}) + 1\cdot\mathtt{DPat} &= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{\mathtt{L}_1\langle\!|u_i[p_1\setminus s_1]\cdots[p_n\setminus s_n]|\!\rangle}) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi_q)\\
&+ \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_r) + 1\cdot\mathtt{DPat}\\
&= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Psi_{\mathtt{case}\ \mathtt{L}_1\langle\!|\mathtt{c}(s_i)_n|\!\rangle\ \mathtt{of}\ (\ldots,\mathtt{c}(p_i)_n.u_i,\ldots)})\\
&+ \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi_q) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_r)\\
&= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{\mathtt{L}_1\langle\!|\mathtt{c}(s_i)_n|\!\rangle}) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi)\\
&+ \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_{u_i}) + \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Pi_q)\\
&+ \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_r)\\
&= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi_t).
\end{aligned}$$

- The remaining cases follow easily from the *i.h.*.

□

Theorem 6.41 (Completeness). *Let $t \in \Lambda^{PM}_{\circ}$. If $t \overset{n\cdot\mathtt{dB}+m\cdot\mathtt{sub}+l_1\cdot\mathtt{branch}+l_2\cdot\mathtt{match}}{\twoheadrightarrow_{\mathtt{PM}}} t'$, such that $t' \in \mathtt{NO}^{\mathtt{CF}}$, then there is $\Phi \rhd_{\mathcal{PM}} \emptyset \vdash t : A$ tight, such that $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi) = n\cdot\mathtt{App} + m\cdot\mathtt{VPat} + l\cdot\mathtt{DPat}$, where $l_1 + l_2 = l$.*

Proof. Let $t \overset{n\cdot\mathtt{dB}+m\cdot\mathtt{sub}+l_1\cdot\mathtt{branch}+l_2\cdot\mathtt{match}}{\twoheadrightarrow_{\mathtt{PM}}} t'$, such that $t' \in \mathtt{NO}^{\mathtt{CF}}$. We proceed by induction on $n+m+l = n+m+l_1+l_2$:

- If $n+m+l_1+l_2 = 0$, then $n = m = l_1 = l_2 = 0$. Therefore, $t' = t$, and we can conclude by Lemma 6.35.
- If $n+m+l_1+l_2 > 0$, then $t \notin \mathtt{NO}^{\mathtt{CF}} \subseteq \mathtt{NO}$ by Lemma 6.15. Therefore, there is $u \in \Lambda^{PM}_{\circ}$, such that $t \to_{\mathtt{PM}(S)} u \overset{n'\cdot\mathtt{dB}+m'\cdot\mathtt{sub}+l'_1\cdot\mathtt{branch}+l'_2\cdot\mathtt{match}}{\twoheadrightarrow_{\mathtt{PM}}} t'$, and $n'+m'+l'_1+l'_2+1 = n+m+l_1+l_2$. Now, by the *i.h.*, there is $\Psi \rhd_{\mathcal{PM}} \emptyset \vdash u : A$ tight, such that $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Psi) = n'\cdot\mathtt{App} + m'\cdot\mathtt{VPat} + (l'_1+l'_2)\cdot\mathtt{DPat}$. By Lemma 6.40, there is $\Phi' \rhd_{\mathcal{PM}} \emptyset \vdash t : A$, such that $\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Psi) + 1\cdot R = \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi')$, where
  - $R = \mathtt{App}$, if $S = \mathtt{dB}$;
  - $R = \mathtt{VPat}$, if $S = \mathtt{sub}$;
  - and $R = \mathtt{DPat}$, if $S \in \{\mathtt{branch}, \mathtt{match}\}$.

Moreover, Φ' is tight since weighted subject expansion preserves tightness. Therefore, we can take $\Phi = \Phi'$, since

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Phi) &= \#_{\mathtt{App},\mathtt{VPat},\mathtt{DPat}}(\Psi) + 1 \cdot R \\
&= n' \cdot \mathtt{App} + m' \cdot \mathtt{VPat} + (l'_1 + l'_2) \cdot \mathtt{DPat} \\
&= n \cdot \mathtt{App} + m \cdot \mathtt{VPat} + (l_1 + l_2) \cdot \mathtt{DPat} \\
&= n \cdot \mathtt{App} + m \cdot \mathtt{VPat} + l \cdot \mathtt{DPat}.
\end{aligned}$$

□

Appendix E

A Quantitative Study of Global State

E.1 The λ-Calculus with Global State

E.1.1 Syntax and Operational Semantics

Lemma 7.18 (Characterization of CBN-Normal Forms and CBV-Normal Forms)**.** *Let* $(t, s) \in \Lambda^{\mathcal{G}} \times \mathcal{S}_{\circ}$. *Then,* $(t, s) \not\to_{\mathtt{CBN}\ (\textit{resp.}\ \mathtt{CBV})}$ *if and only if* $t \in \mathtt{NO}_{\mathtt{CBN}}$ *(resp.* $\mathtt{NO}_{\mathtt{CBV}}$*).*

Proof. By induction on the structure of t. The proof is identical to that of Proposition 2.17, since the state component s can neither prevent nor create CBPV-reduction steps, which are entirely determined by the term t. □

E.2 The $\lambda!$-Calculus with Global State

E.2.1 Syntax and Operational Semantics

Proposition 7.25 (Characterization of CBPV-Normal Forms)**.** *Let* $(t, s) \in !\Lambda^{\mathcal{G}} \times \mathcal{S}_{\circ}$. *Then,* $(t, s) \not\to_{\mathtt{CBPV}}$ *if and only if* $t \in \mathtt{NO}_{\mathtt{CBPV}}$.

Proof. By induction on the structure of t. The proof is identical to that of Proposition 3.7, since the state component s can neither prevent nor create CBPV-reduction steps, which are entirely determined by the term t. □

Proposition 7.29 (Characterization of Clash-Free CBPV-Normal Forms)**.** *Let* $(t, s) \in !\Lambda^{\mathcal{G}}$. *Then,* $(t, s) \not\to_{\mathtt{CBPV}}$ *and* t *is clash-free if and only if* $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$.

Proof. By following the same structure as the proof of Proposition 3.11, since the store component s can neither prevent nor create CBPV-reduction steps or clashes, which are entirely determined by the term t. □

E.2.2 Subsuming Call-By-Name and Call-By-Value

Lemma 7.31 (Preservation of Values). *Let $v \in \mathrm{V}^{\mathcal{G}}$, then $(v)^{\mathtt{CBV}} \in !\mathrm{V}^{\mathcal{G}}$.*

Proof. By a simple case analysis on the definition of $\mathrm{V}^{\mathcal{G}}$, observing that $(\cdot)^{\mathtt{CBV}}$ maps each value in $\mathrm{V}^{\mathcal{G}}$ to a corresponding value in $!\mathrm{V}^{\mathcal{G}}$. □

E.2.3 Correctness of Full Simulations

Lemma 7.37 (Stability of Administrative Steps). *Let $(t,s) \in \Lambda_{\circ}^{\mathcal{G}} \times \mathcal{S}_{\circ}$ and $(t,s) \Rightarrow_{\mathtt{ADMIN}} (u,s)$. Then, u has the same top level constructor (application, abstraction, dereliction, bang-term, or operation) than t.*

Proof. We reason by cases on the outermost constructor of t:

- Case t is an application or an abstraction. Administrative steps only consist of $\beta_!$-steps, whereas applications and abstractions can only be reduced by $\beta_{\mathtt{v}}$-steps at the top level. Hence, no administrative step can change the outermost constructor, and u has the same constructor as t.
- Case t is a dereliction. Administrative steps only fire $\beta_!$-steps inside bangs or abstractions, and thus cannot eliminate or create a top-level dereliction. Therefore, u is also a dereliction.
- Case t is a bang-term. A bang can only be consumed by a $\beta_!$-step when it occurs under a dereliction. Since t is a top-level bang-term, no administrative step applies at the root, and u is also a bang-term.
- Case t is a $\mathtt{get}$ operation. $\mathtt{get}$ operations can only be reduced by $\mathtt{get}$-steps, which are not administrative. Hence, administrative steps cannot affect the outermost constructor, and u is also a $\mathtt{get}$ operation.
- Case $t = \mathtt{set}((l,r),e)$. Similarly, $\mathtt{set}$ operations can only be reduced by $\mathtt{set}$-steps, which are not administrative. Therefore, u is also a $\mathtt{set}$ operation.

□

Lemma 7.38 (Non-Interference of Single Administrative Steps). *Let $S \in \{\beta_{\mathtt{v}}, \mathtt{get}, \mathtt{set}\}$, and $(t,s) \in \Lambda_{\circ}^{\mathcal{G}} \times \mathcal{S}_{\circ}$. If $((t,s))^{\mathtt{CBN}} \not\Rightarrow_{\mathtt{CBPV}(S)}$ and $((t,s))^{\mathtt{CBN}} \Rightarrow_{\mathtt{ADMIN}} (u,(s)^{\mathtt{CBN}})$, then $(u,(s)^{\mathtt{CBN}}) \not\Rightarrow_{\mathtt{CBPV}(S)}$.*

Proof. We proceed by cases on the structure of $t \in \Lambda_{\circ}^{\mathcal{G}}$. Let $S \in \{\beta_{\mathtt{v}}, \mathtt{get}, \mathtt{set}\}$ be fixed:

- If $t = x$, then $t \notin \Lambda_{\circ}^{\mathcal{G}}$. Thus, this case does not apply.

- If $t = \lambda x.r$, then $((t,s))^{\mathtt{CBN}} = ((t)^{\mathtt{CBN}},(s)^{\mathtt{CBN}}) = ((\lambda x.r)^{\mathtt{CBN}},(s)^{\mathtt{CBN}}) = (\lambda x.(r)^{\mathtt{CBN}},(s)^{\mathtt{CBN}})$. By definition, abstractions are never $\mathtt{CBPV}(S)$-redexes, hence $(\lambda x.(r)^{\mathtt{CBN}},(s)^{\mathtt{CBN}}) \not\Rightarrow_{\mathtt{CBPV}(S)}$. Let $(\lambda x.(r)^{\mathtt{CBN}},(s)^{\mathtt{CBN}}) \Rightarrow_{\mathtt{ADMIN}} (u,(s)^{\mathtt{CBN}})$. By Lemma 7.37, we must have $u = \lambda x.r_0$. Therefore, $(\lambda x.r_0,(s)^{\mathtt{CBN}}) \not\Rightarrow_{\mathtt{CBPV}(S)}$, by definition.

- If $t = r\,e$, then $((t,s))^{\mathtt{CBN}} = ((t)^{\mathtt{CBN}},(s)^{\mathtt{CBN}}) = ((r\,e)^{\mathtt{CBN}},(s)^{\mathtt{CBN}}) = ((r)^{\mathtt{CBN}}\,!(e)^{\mathtt{CBN}},(s)^{\mathtt{CBN}})$. Assume that $((r)^{\mathtt{CBN}}\,!(e)^{\mathtt{CBN}},(s)^{\mathtt{CBN}}) \not\Rightarrow_{\mathtt{CBPV}(S)}$, and let $((r)^{\mathtt{CBN}}\,!(e)^{\mathtt{CBN}},(s)^{\mathtt{CBN}}) \Rightarrow_{\mathtt{ADMIN}} (u,(s)^{\mathtt{CBN}})$. By Lemma 7.37, $u = r_0\,e_0$. Moreover, one of the following holds:

  (i) $((r)^{\mathtt{CBN}},(s)^{\mathtt{CBN}}) \Rightarrow_{\mathtt{ADMIN}} (r_0,(s)^{\mathtt{CBN}})$ and $e_0 = !(e)^{\mathtt{CBN}}$;

  (ii) $(!(e)^{\mathtt{CBN}},(s)^{\mathtt{CBN}}) \Rightarrow_{\mathtt{ADMIN}} (e_0,(s)^{\mathtt{CBN}})$ and $r_0 = (r)^{\mathtt{CBN}}$.

  – Case (i). By Lemma 7.37, r_0 cannot be an abstraction. Hence, if $S = \beta_{\mathsf{v}}$, no β_{v}-step is available at the top level. For $S \in \{\mathtt{get},\mathtt{set}\}$, applications are never $\mathtt{CBPV}(S)$-redexes. Therefore, $(r_0\,e_0,(s)^{\mathtt{CBN}}) \not\Rightarrow_{\mathtt{CBPV}(S)}$.

  – Case (ii). By Lemma 7.37, e_0 is a bang-term. Moreover, $r_0 = (r)^{\mathtt{CBN}}$. Since

  $$((r)^{\mathtt{CBN}}\,!(e)^{\mathtt{CBN}},(s)^{\mathtt{CBN}}) \not\Rightarrow_{\mathtt{CBPV}(\beta_{\mathsf{v}})},$$

  $(r)^{\mathtt{CBN}}$ is not an abstraction. Hence, no β_{v}-step is available, and applications are never $\mathtt{CBPV}(\mathtt{get})$- or $\mathtt{CBPV}(\mathtt{set})$-redexes. Thus, $(r_0\,e_0,(s)^{\mathtt{CBN}}) \not\Rightarrow_{\mathtt{CBPV}(S)}$.

- If $t = \mathtt{get}(l,\lambda x.r)$, then we know $((t,s))^{\mathtt{CBN}} = ((t)^{\mathtt{CBN}},(s)^{\mathtt{CBN}}) = ((\mathtt{get}(l,\lambda x.r))^{\mathtt{CBN}},(s)^{\mathtt{CBN}}) = (\mathtt{get}(l,\lambda x.(r)^{\mathtt{CBN}}),(s)^{\mathtt{CBN}})$. In this case, a $\mathtt{CBPV}(\mathtt{get})$-step is available, so $((t,s))^{\mathtt{CBN}} \not\Rightarrow_{\mathtt{CBPV}(S)}$ can only hold if $S \in \{\beta_{\mathsf{v}},\mathtt{set}\}$. Let $(\mathtt{get}(l,\lambda x.(r)^{\mathtt{CBN}}),(s)^{\mathtt{CBN}}) \Rightarrow_{\mathtt{ADMIN}} (u,(s)^{\mathtt{CBN}})$. By Lemma 7.37, $u = \mathtt{get}(l,\lambda x.r_0)$. Since $\mathtt{get}$-terms are never $\mathtt{CBPV}(\beta_{\mathsf{v}})$- or $\mathtt{CBPV}(\mathtt{set})$-redexes, we conclude $(u,(s)^{\mathtt{CBN}}) \not\Rightarrow_{\mathtt{CBPV}(S)}$.

- If $t = \mathtt{set}((l,e),r)$, then we know $((t,s))^{\mathtt{CBN}} = ((t)^{\mathtt{CBN}},(s)^{\mathtt{CBN}}) = ((\mathtt{set}((l,e),r))^{\mathtt{CBN}},(s)^{\mathtt{CBN}}) = (\mathtt{set}((l,!(e)^{\mathtt{CBN}}),(r)^{\mathtt{CBN}}),(s)^{\mathtt{CBN}})$. In this case, a $\mathtt{CBPV}(\mathtt{set})$-step is available, so the premise $((t,s))^{\mathtt{CBN}} \not\Rightarrow_{\mathtt{CBPV}(S)}$ can only hold if $S \in \{\beta_{\mathsf{v}},\mathtt{get}\}$. Let $(\mathtt{set}((l,!(e)^{\mathtt{CBN}}),(r)^{\mathtt{CBN}}),(s)^{\mathtt{CBN}}) \Rightarrow_{\mathtt{ADMIN}} (u,(s)^{\mathtt{CBN}})$. By Lemma 7.37, $u = \mathtt{set}((l,e_0),r_0)$. Since $\mathtt{set}$-terms are never $\mathtt{CBPV}(\beta_{\mathsf{v}})$- or $\mathtt{CBPV}(\mathtt{get})$-redexes, we conclude $(u,(s)^{\mathtt{CBN}}) \not\Rightarrow_{\mathtt{CBPV}(S)}$.

□

Lemma 7.40 (Preservation under Substitution). *Let $s \in \mathcal{S}$, $t \in \Lambda^{\mathcal{G}}$, $u \in \Lambda^{\mathcal{G}}_{\circ}$, and $v \in \mathrm{V}^{\mathcal{G}}_{\circ}$:*

1. $((t)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\},s) \Rightarrow^{=}_{\beta_!} \twoheadrightarrow_{\mathtt{ADMIN}} ((t\{x \setminus u\})^{\mathtt{CBN}},s)$.
2. $((t)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\},s) = ((t\{x \setminus v\})^{\mathtt{CBV}},s)$.

Proof.

1. The proof follows by induction on $t \in \Lambda^{\mathcal{G}}$, just like for Lemma B.4. We implicitly use the fact that any $\beta_!$-step produced by the *i.h.* is simulated by a (possibly empty) sequence of administrative steps in the global semantics. We only show the cases where t is an application or an operation:

   - If $t = r\,e$, then
   $$\begin{aligned}
   &((t)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}, s)\\
   &= ((r\,e)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}, s)\\
   &= (((r)^{\mathtt{CBN}}\,!(e)^{\mathtt{CBN}})\{x \setminus !(u)^{\mathtt{CBN}}\}, s)\\
   &= ((r)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}\,(!(e)^{\mathtt{CBN}})\{x \setminus !(u)^{\mathtt{CBN}}\}, s)\\
   &= ((r)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}\,!(e)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}, s).
   \end{aligned}$$
   By applying the *i.h.*, we have
   $$((r)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}, s) \Rightarrow_{\beta_!}^{=} \twoheadrightarrow_{\mathtt{ADMIN}} ((r\{x \setminus u\})^{\mathtt{CBN}}, s).$$
   By context closure of administrative steps, we obtain
   $$\begin{aligned}
   &((r)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}\,!((e)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}), s)\\
   &\Rightarrow_{\beta_!}^{=} \twoheadrightarrow_{\mathtt{ADMIN}}\\
   &((r\{x \setminus u\})^{\mathtt{CBN}}\,!((e)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}), s).
   \end{aligned}$$
   By applying the *i.h.*, we have
   $$((e)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}, s) \Rightarrow_{\beta_!}^{=} \twoheadrightarrow_{\mathtt{ADMIN}} ((e\{x \setminus u\})^{\mathtt{CBN}}, s).$$
   By context closure of administrative steps, we obtain
   $$\begin{aligned}
   &((r\{x \setminus u\})^{\mathtt{CBN}}\,!((e)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}), s)\\
   &\Rightarrow_{\beta_!}^{=} \twoheadrightarrow_{\mathtt{ADMIN}} ((r\{x \setminus u\})^{\mathtt{CBN}}\,!(e\{x \setminus u\})^{\mathtt{CBN}}, s)\\
   &= ((r\{x \setminus u\}\,e\{x \setminus u\})^{\mathtt{CBN}}, s)\\
   &= (((r\,e)\{x \setminus u\})^{\mathtt{CBN}}, s)\\
   &= ((t\{x \setminus u\})^{\mathtt{CBN}}, s).
   \end{aligned}$$

   - If $t = \mathtt{get}(l, \lambda y.r)$, then
   $$\begin{aligned}
   &((t)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}, s)\\
   &= ((\mathtt{get}(l, \lambda y.r))^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}, s)\\
   &= (\mathtt{get}(l, \lambda y.(r)^{\mathtt{CBN}})\{x \setminus !(u)^{\mathtt{CBN}}\}, s)\\
   &= (\mathtt{get}(l, \lambda y.(r)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}), s).
   \end{aligned}$$

By applying the *i.h.*, we have

$$((r)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}, s) \Rightarrow_{\beta_!}^{=} \twoheadrightarrow_{\mathtt{ADMIN}} (r\{x \setminus u\}, s)^{\mathtt{CBN}}.$$

By context closure of administrative steps, we obtain

$$\begin{aligned}&(\mathtt{get}(l, \lambda y.(r)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}), s)\\ &\twoheadrightarrow_{\mathtt{ADMIN}} (\mathtt{get}(l, \lambda y.(r\{x \setminus u\})^{\mathtt{CBN}}), s)\\ &= ((\mathtt{get}(l, \lambda y.r)\{x \setminus u\})^{\mathtt{CBN}}, s).\end{aligned}$$

- If $t = \mathtt{set}((l, e), r)$, then

$$\begin{aligned}&((t)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}, s)\\ &= ((\mathtt{set}((l, e), r))^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}, s)\\ &= (\mathtt{set}((l, !(e)^{\mathtt{CBN}}), (r)^{\mathtt{CBN}})\{x \setminus !(u)^{\mathtt{CBN}}\}, s)\\ &= (\mathtt{set}((l, !(e)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}), (r)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}), s).\end{aligned}$$

By the *i.h.*, we have

$$((e)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}, s) \Rightarrow_{\beta_!}^{=} \twoheadrightarrow_{\mathtt{ADMIN}} ((e\{x \setminus u\})^{\mathtt{CBN}}, s),$$

and

$$((r)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}, s) \Rightarrow_{\beta_!}^{=} \twoheadrightarrow_{\mathtt{ADMIN}} ((r\{x \setminus u\})^{\mathtt{CBN}}, s).$$

By context closure of administrative steps, we obtain

$$\begin{aligned}&(\mathtt{set}((l, !(e)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}), (r)^{\mathtt{CBN}}\{x \setminus !(u)^{\mathtt{CBN}}\}), s)\\ &\twoheadrightarrow_{\mathtt{ADMIN}} (\mathtt{set}((l, !(e\{x \setminus u\})^{\mathtt{CBN}}), (r\{x \setminus u\})^{\mathtt{CBN}}), s)\\ &= ((\mathtt{set}((l, e), r)\{x \setminus u\})^{\mathtt{CBN}}, s).\end{aligned}$$

The remaining cases follow easily from the *i.h.*:

2. The proof follows by induction on the structure of $t \in \Lambda^{\mathcal{G}}$ and is similar to that of Lemma B.4. We only show the cases when t is an application or an operation:

   - If $t = r\, e$, then

$$((r\, e)^{\mathtt{CBV}}, s) = (\mathtt{der}((r)^{\mathtt{CBV}})\, (e)^{\mathtt{CBV}}, s).$$

   Therefore,

$$((r\, e)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\}, s) = (\mathtt{der}((r)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\})\, (e)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\}, s).$$

By applying the *i.h.* to r, we have

$$((r)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\}, s) = ((r\{x \setminus v\})^{\mathtt{CBV}}, s).$$

By applying the *i.h.* to e, we have

$$((e)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\}, s) = ((e\{x \setminus v\})^{\mathtt{CBV}}, s).$$

Therefore, we have that

$$\begin{aligned}
&(\mathtt{der}((r)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\})\,(e)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\}, s)\\
&= (\mathtt{der}((r\{x \setminus v\})^{\mathtt{CBV}})\,(e\{x \setminus v\})^{\mathtt{CBV}}, s)\\
&= ((r\{x \setminus v\}\; e\{x \setminus v\})^{\mathtt{CBV}}, s)\\
&= (((r\,e)\{x \setminus v\})^{\mathtt{CBV}}, s)\\
&= ((t\{x \setminus v\})^{\mathtt{CBV}}, s).
\end{aligned}$$

- If $t = \mathtt{get}(l, \lambda x.u)$, then

$$\begin{aligned}
&((t)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\}, (s)^{\mathtt{CBV}})\\
&= ((\mathtt{get}(l, \lambda x.u))^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\}, s)\\
&= (\mathtt{get}(l, \lambda x.(u)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\}), s)\\
&\overset{\text{by } i.h.}{=} (\mathtt{get}(l, \lambda x.(u\{x \setminus v\})^{\mathtt{CBV}}), s)\\
&= ((\mathtt{get}(l, \lambda x.u)\{x \setminus v\})^{\mathtt{CBV}}, s).
\end{aligned}$$

- If $t = \mathtt{set}((l, w), u)$, then

$$\begin{aligned}
&((t)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\}, s)\\
&= ((\mathtt{set}((l, w), u))^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\}, s)\\
&= (\mathtt{set}((l, (w)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\}), (u)^{\mathtt{CBV}}(v)^{\mathtt{CBV}}), s)\\
&\overset{\text{by } i.h.}{=} (\mathtt{set}((l, (w\{x \setminus v\})^{\mathtt{CBV}}), (u\{x \setminus v\})^{\mathtt{CBV}}), s)\\
&= ((\mathtt{set}((l, w), u)\{x \setminus v\})^{\mathtt{CBV}}, s).
\end{aligned}$$

The remaining cases follow easily from the *i.h.*.

□

Lemma 7.41 (Correctness of One-Step Simulations). *Let $s \in \mathcal{S}_\circ$ and $t \in \Lambda^{\mathcal{G}}_\circ$.*

1. *Let $S \in \{\beta, \mathtt{get}, \mathtt{set}\}$. If $(t, s) \Rightarrow_{\mathtt{CBN}(S)} (u, q)$, then $((t, s))^{\mathtt{CBN}} \overset{1\cdot S' + m\cdot\beta_!}{\Rightarrow\!\!\!\!\Rightarrow}_{\mathtt{CBPV}\cup\mathtt{ADMIN}} ((u, q))^{\mathtt{CBN}}$, for some $m \geq 0$, such that $S' = \beta_{\mathtt{v}}$ if $S = \beta$, and $S' = S$, otherwise.*
2. *Let $S \in \{\beta_{\mathtt{v}}, \mathtt{get}, \mathtt{set}\}$. If $(t, s) \Rightarrow_{\mathtt{CBV}(S)} (u, q)$, then $((t, s))^{\mathtt{CBV}} \overset{1\cdot S + m\cdot\beta_!}{\Rightarrow\!\!\!\!\Rightarrow}_{\mathtt{CBPV}} ((u, q))^{\mathtt{CBV}}$, for some $m \geq 0$.*

Proof.

1. Let $(t,s) \Rightarrow_{\mathtt{CBN}(S)} (u,q)$. The proof follow by induction on the definition of $\mathtt{N}$:

   - Case $\mathtt{N} = \Box$:

     – If $S = \beta$, then

     $$(t,s) = ((\lambda x.r)\, e, s) \mapsto_{\beta} (r\{x \setminus e\}, s) = (u,s).$$

     Note that

     $$\begin{aligned}((t,s))^{\mathtt{CBN}} &= (((\lambda x.r)\, e)^{\mathtt{CBN}}, (s)^{\mathtt{CBN}}) \\ &= (\lambda x.(r)^{\mathtt{CBN}}\, !(e)^{\mathtt{CBN}}, (s)^{\mathtt{CBN}}) \\ &\mapsto_{\beta_{\mathtt{v}}} ((r)^{\mathtt{CBN}}\{x \setminus !(e)^{\mathtt{CBN}}\}, (s)^{\mathtt{CBN}}).\end{aligned}$$

     Therefore,

     $$((r)^{\mathtt{CBN}}\{x \setminus !(e)^{\mathtt{CBN}}\}, (s)^{\mathtt{CBN}}) \overset{\text{by Lemma 7.40}}{\Rightarrow}^{=}_{\beta_!} \twoheadrightarrow_{\mathtt{ADMIN}} ((r\{x \setminus e\})^{\mathtt{CBN}}, (s)^{\mathtt{CBN}}) = ((u,s))^{\mathtt{CBN}}.$$

     Thus,

     $$((t,s))^{\mathtt{CBN}} = (((\lambda x.r)\, e)^{\mathtt{CBN}}, (s)^{\mathtt{CBN}}) \twoheadrightarrow^{1\cdot\beta_{\mathtt{v}} + m\cdot\beta_!}_{\mathtt{CBPV} \cup \mathtt{ADMIN}} ((r\{x \setminus e\})^{\mathtt{CBN}}, (s)^{\mathtt{CBN}}) = ((u,s))^{\mathtt{CBN}},$$

     for some $m \geq 0$.

     – If $S = \mathtt{get}$, then

     $$(t,s) = (\mathtt{get}(l, \lambda x.r), s) \mapsto_{\mathtt{get}} (r\{x \setminus s\,(l)\}, s) = (u,q).$$

     Notice that

     $$((\mathtt{get}(l, \lambda x.r), s))^{\mathtt{CBN}} = (\mathtt{get}(l, \lambda x.(r)^{\mathtt{CBN}}), (s)^{\mathtt{CBN}}),$$

     and

     $$((r\{x \setminus s\,(l)\}, s))^{\mathtt{CBN}} = ((r\{x \setminus s\,(l)\})^{\mathtt{CBN}}, (s)^{\mathtt{CBN}}).$$

     Moreover, by Lemma 7.40, we know

     $$((r)^{\mathtt{CBN}}\{x \setminus (s\,(l))^{\mathtt{CBN}}\}, (s)^{\mathtt{CBN}}) \Rightarrow^{=}_{\beta_!} \twoheadrightarrow_{\mathtt{ADMIN}} ((r\{x \setminus s\,(l)\})^{\mathtt{CBN}}, (s)^{\mathtt{CBN}}),$$

and it is easy to see that $(s\,(l))^{\mathtt{CBN}} = (s)^{\mathtt{CBN}}\,(l)$. By contextual closure of $\twoheadrightarrow$ and the *i.h.*, we have

$$\begin{array}{rcl}
((t,s))^{\mathtt{CBN}} & = & ((\mathtt{get}(l,\lambda x.r),s))^{\mathtt{CBN}} \\
& = & (\mathtt{get}(l,\lambda x.(r)^{\mathtt{CBN}}),(s)^{\mathtt{CBN}}) \\
& \mapsto_{\mathtt{get}} & (r\{x \setminus (s)^{\mathtt{CBN}}\,(l)\},(s)^{\mathtt{CBN}}) \\
& = & ((r)^{\mathtt{CBN}}\{x \setminus (s\,(l))^{\mathtt{CBN}}\},(s)^{\mathtt{CBN}}) \\
\Rightarrow^{=}_{\beta_!} \twoheadrightarrow_{\mathtt{ADMIN}} & & ((r\{x \setminus s\,(l)\})^{\mathtt{CBN}},(s)^{\mathtt{CBN}}) \\
& = & ((r\{x \setminus s\,(l)\},s))^{\mathtt{CBN}} \\
& = & ((u,q))^{\mathtt{CBN}}.
\end{array}$$

– If $S = \mathtt{set}$, then we have

$$(t,s) = (\mathtt{set}((l,e),r),s) \mapsto_{\mathtt{set}} (r,s\{l := e\}) = (u,q).$$

Notice that

$$((\mathtt{set}((l,e),r),s))^{\mathtt{CBN}} = (\mathtt{set}((l,!(e)^{\mathtt{CBN}}),(r)^{\mathtt{CBN}}),(s)^{\mathtt{CBN}}),$$

and

$$((r,s\{l := e\}))^{\mathtt{CBN}} = ((r)^{\mathtt{CBN}},(s\{l := e\})^{\mathtt{CBN}}).$$

Moreover, it is easy to see that $(s\{l := e\})^{\mathtt{CBN}} = (s)^{\mathtt{CBN}}\{l := !(e)^{\mathtt{CBN}}\}$. By contextual closure of $\twoheadrightarrow$ and the *i.h.*, we have

$$\begin{array}{rcl}
((t,s))^{\mathtt{CBN}} & = & ((\mathtt{set}((l,e),r),s))^{\mathtt{CBN}} \\
& = & (\mathtt{set}((l,!(e)^{\mathtt{CBN}}),(r)^{\mathtt{CBN}}),(s)^{\mathtt{CBN}}) \\
& \mapsto_{\mathtt{set}} & ((r)^{\mathtt{CBN}},(s)^{\mathtt{CBN}}\{l := !(e)^{\mathtt{CBN}}\}) \\
& = & ((r)^{\mathtt{CBN}},(s\{l := e\})^{\mathtt{CBN}}) \\
& = & ((r,s\{l := e\}))^{\mathtt{CBN}} \\
& = & ((u,q))^{\mathtt{CBN}}.
\end{array}$$

- Case $\mathtt{N} = \mathtt{N}'\,r$, then

$$\begin{array}{rl}
(t,s) & = (\mathtt{N}'[e]\,r,s) \\
& \Rightarrow_{\mathtt{CBN}(S)} (\mathtt{N}'[e']\,r,q) \\
& = (u,s),
\end{array}$$

such that $(e,s) \mapsto_S (e',s)$. Note that

$$((\mathtt{N}'[e]\,r)^{\mathtt{CBN}},(s)^{\mathtt{CBN}}) = ((\mathtt{N}'[e])^{\mathtt{CBN}}\,!(r)^{\mathtt{CBN}},(s)^{\mathtt{CBN}}).$$

By applying the *i.h.* to $\mathtt{N}'$, we have

$$((\mathtt{N}'[e])^{\mathtt{CBN}}, (s)^{\mathtt{CBN}}) \overset{1\cdot S' + m\cdot\beta_!}{\twoheadrightarrow}_{\mathtt{CBPV}\cup\mathtt{ADMIN}} ((\mathtt{N}'[e'])^{\mathtt{CBN}}, (s)^{\mathtt{CBN}}), \text{for some } m \geq 0,$$

such that $S' = \beta_{\mathtt{v}}$ if $S = \beta$, and $S' = S$, otherwise. By contextual closure of $\twoheadrightarrow$ and the *i.h.*, we have,

$$\begin{aligned}((\mathtt{N}'[e])^{\mathtt{CBN}}\, !(r)^{\mathtt{CBN}}, (s)^{\mathtt{CBN}}) &\overset{1\cdot S' + m\cdot\beta_!}{\twoheadrightarrow}_{\mathtt{CBPV}\cup\mathtt{ADMIN}} ((\mathtt{N}'[e'])^{\mathtt{CBN}}\, !(r)^{\mathtt{CBN}}, (s)^{\mathtt{CBN}}) \\ &= ((\mathtt{N}'[e']\, r)^{\mathtt{CBN}}, (s)^{\mathtt{CBN}}) \\ &= ((u,s))^{\mathtt{CBN}}.\end{aligned}$$

Thus,

$$\begin{aligned}((t,s))^{\mathtt{CBN}} &= ((\mathtt{N}'[e]\, r)^{\mathtt{CBN}}, (s)^{\mathtt{CBN}}) \\ &\overset{1\cdot\beta_{\mathtt{v}} + m\cdot\beta_!}{\twoheadrightarrow}_{\mathtt{CBPV}\cup\mathtt{ADMIN}} ((\mathtt{N}'[e']\, r)^{\mathtt{CBN}}, (s)^{\mathtt{CBN}}) \\ &= ((u,s))^{\mathtt{CBN}}.\end{aligned}$$

2. Let $(t,s) \Rightarrow_{\mathtt{CBV}(S)} (u,q)$. The proof follow by induction on the definition of $\mathtt{V}$:

- Case $\mathtt{V} = \Box$:
  - If $S = \beta_{\mathtt{v}}$, then

$$(t,s) = ((\lambda x.r)\, v, s) = (r\{x \setminus v\}, s) = (u,s).$$

Note that

$$\begin{aligned}(((\lambda x.r)\, v, s))^{\mathtt{CBV}} &= (\mathtt{der}(!(\lambda x.(r)^{\mathtt{CBV}}))\, (v)^{\mathtt{CBV}}, (s)^{\mathtt{CBV}}) \\ &\Rightarrow_{\mathtt{CBPV}(\beta_!)} ((\lambda x.(r)^{\mathtt{CBV}})\, (v)^{\mathtt{CBV}}, (s)^{\mathtt{CBV}}) \\ &\overset{\text{by Lemma 7.31}}{\Rightarrow}_{\mathtt{CBPV}(\beta_{\mathtt{v}})} ((r)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\}, (s)^{\mathtt{CBV}}).\end{aligned}$$

Therefore,

$$((r)^{\mathtt{CBV}}\{x \setminus (v)^{\mathtt{CBV}}\}, (s)^{\mathtt{CBV}}) \overset{\text{by Lemma 7.40}}{=} ((r\{x \setminus v\})^{\mathtt{CBV}}, (s)^{\mathtt{CBV}}) = ((u)^{\mathtt{CBV}}, (s)^{\mathtt{CBV}}).$$

Thus,

$$((t,s))^{\mathtt{CBV}} = (((\lambda x.r)\, v)^{\mathtt{CBV}}, (s)^{\mathtt{CBV}}) \Rightarrow_{\mathtt{CBPV}} ((r\{x \setminus v\})^{\mathtt{CBV}}, (s)^{\mathtt{CBV}}) = ((u,s))^{\mathtt{CBV}}.$$

  - If $S = \mathtt{get}$, then

$$(t,s) = (\mathtt{get}(l, \lambda x.r), s) \mapsto_{\mathtt{get}} (r\{x \setminus s\,(l)\}, s) = (u,q).$$

Notice that

$$((\mathtt{get}(l, \lambda x.r), s))^{\mathtt{CBV}} = (\mathtt{get}(l, \lambda x.(r)^{\mathtt{CBV}}), (s)^{\mathtt{CBV}}),$$

and

$$((r\{x \setminus v\}, s))^{\mathtt{CBV}} = ((r\{x \setminus s\,(l)\})^{\mathtt{CBV}}, (s)^{\mathtt{CBV}}).$$

Moreover, by Lemma 7.40, we know

$$((r)^{\mathtt{CBV}}\{x \setminus (s\,(l))^{\mathtt{CBV}}\}, (s)^{\mathtt{CBV}}) = ((r\{x \setminus s\,(l)\})^{\mathtt{CBV}}, (s)^{\mathtt{CBV}}),$$

and it is easy to see that $(s\,(l))^{\mathtt{CBV}} = (s)^{\mathtt{CBV}}\,(l)$. By contextual closure of $\twoheadrightarrow$ and the *i.h.*,

$$\begin{array}{rcl}
((t,s))^{\mathtt{CBV}} & = & ((\mathtt{get}(l, \lambda x.r), s))^{\mathtt{CBV}} \\
& = & (\mathtt{get}(l, \lambda x.(r)^{\mathtt{CBV}}), (s)^{\mathtt{CBV}}) \\
& \mapsto_{\mathtt{get}} & (r\{x \setminus (s)^{\mathtt{CBV}}\,(l)\}, (s)^{\mathtt{CBV}}) \\
& = & ((r)^{\mathtt{CBV}}\{x \setminus (s\,(l))^{\mathtt{CBV}}\}, (s)^{\mathtt{CBV}}) \\
& = & ((r\{x \setminus s\,(l)\})^{\mathtt{CBV}}, (s)^{\mathtt{CBV}}) \\
& = & ((r\{x \setminus s\,(l)\}, s))^{\mathtt{CBV}} \\
& = & ((u,q))^{\mathtt{CBV}}.
\end{array}$$

– If $S = \mathtt{set}$, then

$$(t,s) = (\mathtt{set}((l,v),r)) \mapsto_{\mathtt{set}} (r, s\{l := v\}) = (u,q).$$

Notice that

$$((\mathtt{set}((l,v),r),s))^{\mathtt{CBV}} = (\mathtt{set}((l,(v)^{\mathtt{CBV}}),(r)^{\mathtt{CBV}}),(s)^{\mathtt{CBV}}),$$

and

$$((r, s\{l := v\}))^{\mathtt{CBV}} = ((r)^{\mathtt{CBV}}, (s\{l := v\})^{\mathtt{CBV}}).$$

Moreover, it is easy to see that $(s\{l := v\})^{\mathtt{CBV}} = (s)^{\mathtt{CBV}}\{l := (v)^{\mathtt{CBV}}\}$. By contextual closure of $\twoheadrightarrow$ and the *i.h.*,

$$\begin{array}{rcl}
((t,s))^{\mathtt{CBV}} & = & ((\mathtt{set}((l,v),r),s))^{\mathtt{CBV}} \\
& = & (\mathtt{set}((l,(v)^{\mathtt{CBV}}),(r)^{\mathtt{CBV}}),(s)^{\mathtt{CBV}}) \\
& \mapsto_{\mathtt{set}} & ((r)^{\mathtt{CBV}}, (s)^{\mathtt{CBV}}\{l := (v)^{\mathtt{CBV}}\}) \\
& = & ((r)^{\mathtt{CBV}}, (s\{l := v\})^{\mathtt{CBV}}) \\
& = & ((r, s\{l := v\}))^{\mathtt{CBV}} \\
& = & ((u,q))^{\mathtt{CBV}}.
\end{array}$$

- Case $\mathtt{V} = \mathtt{V}'\, r$, then

$$\begin{array}{rcl}(t,s) & = & (\mathtt{V}'[e]\, r, s)\\ & \Rightarrow_{\mathtt{CBV}(S)} & (\mathtt{V}'[e']\, r, s)\\ & = & (u,q),\end{array}$$

such that

$$(e,s) \mapsto\!\!\!\Rightarrow_S (e',s).$$

By applying the *i.h.* to $\mathtt{V}'$, we know that

$$((\mathtt{V}'[e],s))^{\mathtt{CBV}} \overset{1\cdot S + m_0\cdot\beta_!}{\twoheadrightarrow}_{\mathtt{CBPV}} ((\mathtt{V}'[e'],s))^{\mathtt{CBV}}.$$

By contextual closure of $\twoheadrightarrow$ and the *i.h.*,

$$\begin{array}{rcl}((t,s))^{\mathtt{CBV}} & = & ((\mathtt{V}'[e]\, r, s))^{\mathtt{CBV}}\\ & = & ((\mathtt{V}'[e])^{\mathtt{CBV}}\, (r)^{\mathtt{CBV}}, (s)^{\mathtt{CBV}})\\ & = & (\mathtt{der}((\mathtt{V}'[e])^{\mathtt{CBV}})\, (r)^{\mathtt{CBV}}, (s)^{\mathtt{CBV}})\\ & \overset{1\cdot S + m_0\cdot\beta_!}{\twoheadrightarrow}_{\mathtt{CBPV}} & (\mathtt{der}((\mathtt{V}'[e'])^{\mathtt{CBV}})\, (r)^{\mathtt{CBV}}, (s)^{\mathtt{CBV}})\\ & = & ((\mathtt{V}'[e']\, r)^{\mathtt{CBV}}, (s)^{\mathtt{CBV}})\\ & = & ((u,s))^{\mathtt{CBV}}.\end{array}$$

- Case $\mathtt{V} = v\, \mathtt{V}'$, then

$$(t,s) = (v\, \mathtt{V}'[r], s) \Rightarrow_{\mathtt{CBV}(S)} (v\, \mathtt{V}'[r'], s) = (u,s),$$

such that $(\mathtt{V}'[r],s) \Rightarrow_S (\mathtt{V}'[r'],s)$. Notice that v must be an abstraction. Therefore, without loss of generality, let us assume $v = \lambda x.e$. Then, $(v\, \mathtt{V}'[r])^{\mathtt{CBV}} = (\lambda x.(e)^{\mathtt{CBV}})\, (\mathtt{V}'[r])^{\mathtt{CBV}}$. By applying the *i.h.* to $\mathtt{V}'$, we know that

$$((\mathtt{V}'[r], (s)^{\mathtt{CBV}}))^{\mathtt{CBV}} \overset{1\cdot S + m_0\cdot\beta_!}{\twoheadrightarrow}_{\mathtt{CBPV}} ((\mathtt{V}'[r'], (s)^{\mathtt{CBV}}))^{\mathtt{CBV}}.$$

By contextual closure of $\twoheadrightarrow$ and the *i.h.*,

$$\begin{array}{rcl}((t,s))^{\mathtt{CBV}} & = & ((v\, \mathtt{V}'[r], s))^{\mathtt{CBV}}\\ & = & ((v\, \mathtt{V}'[r])^{\mathtt{CBV}}, (s)^{\mathtt{CBV}})\\ & = & ((\lambda x.(e)^{\mathtt{CBV}})\, (\mathtt{V}'[r])^{\mathtt{CBV}}, (s)^{\mathtt{CBV}})\\ & \overset{1\cdot S + m_0\cdot\beta_!}{\twoheadrightarrow}_{\mathtt{CBPV}} & ((\lambda x.(e)^{\mathtt{CBV}})\, (\mathtt{V}'[r'])^{\mathtt{CBV}}, (s)^{\mathtt{CBV}})\\ & = & ((v\, \mathtt{V}'[r'])^{\mathtt{CBV}}, (s)^{\mathtt{CBV}})\\ & = & ((u)^{\mathtt{CBV}}, (s)^{\mathtt{CBV}}).\end{array}$$

□

Theorem 7.42 (Correctness of Full Simulations). *Let $s \in \mathcal{S}_\circ$, and $t \in \Lambda^{\mathcal{G}}_\circ$.*

1. *If $(t,s) \overset{n\cdot\beta + g\cdot\mathtt{get}+s\cdot\mathtt{set}}{\Rightarrow\!\!\!\!\!\!\Rightarrow}_{\mathtt{CBN}} (u,q)$, then $((t,s))^{\mathtt{CBN}} \overset{n\cdot\beta_{\mathtt{v}}+m\cdot\beta_{!}+g\cdot\mathtt{get}+s\cdot\mathtt{set}}{\Rightarrow\!\!\!\!\!\!\Rightarrow}_{\mathtt{CBPV}\cup\mathtt{ADMIN}} ((u,q))^{\mathtt{CBN}}$, for some $m \geq 0$.*
2. *If $(t,s) \overset{n\cdot\beta_{\mathtt{v}} + g\cdot\mathtt{get}+s\cdot\mathtt{set}}{\Rightarrow\!\!\!\!\!\!\Rightarrow}_{\mathtt{CBV}} (u,q)$, then $((t,s))^{\mathtt{CBV}} \overset{n\cdot\beta_{\mathtt{v}}+m\cdot\beta_{!}+g\cdot\mathtt{get}+s\cdot\mathtt{set}}{\Rightarrow\!\!\!\!\!\!\Rightarrow}_{\mathtt{CBPV}} ((u,q))^{\mathtt{CBV}}$, for some $m \geq 0$.*

Proof. The proof follows the same induction on the length of the reduction sequence as that of Lemma 3.19. Each individual reduction step is simulated using Lemma 7.41, and the overall result follows by concatenation of the simulated sequences and additivity of step counts. □

E.3 State-Passing Non-Idempotent Intersection Types

E.3.1 Call-By-Name

Lemma 7.54 (CBN Multiset-Typings Split and Merge). *Let $t \in \Lambda^{\mathcal{G}}$. Then, there is $\Phi \rhd_{\mathcal{N}_{\mathcal{G}}} \Gamma \vdash t : M$, such that $M = \sqcup_{i\in\mathtt{I}} M_i$ if and only if there are $\Phi_i \rhd_{\mathcal{N}_{\mathcal{G}}} \Gamma_i \vdash t : M_i$, such that $\Gamma = +_{i\in\mathtt{I}}\Gamma_i$, and $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = +_{i\in\mathtt{I}}\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_i)$.*

Proof. The proof is identical to the split-and-merge argument for multiset typings in the pure λ-calculus.

The only typing rules contributing to the weight are `App`, `Get`, and `Set`, all of which are additive with respect to multiset types and typing environments. Moreover, the typing rules for global state operations do not introduce any interaction between distinct components of a multiset type.

Therefore, given a derivation typing t with a multiset type $M = \sqcup_{i\in\mathtt{I}} M_i$, the derivation necessarily ends with a multiset introduction rule and can be decomposed into independent derivations typing t at each M_i, with environments summing pointwise. Conversely, independent derivations for each M_i can be merged into a single derivation for M.

In both directions, the size measure is preserved by additivity of the rules, yielding

$$\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = +_{i\in\mathtt{I}}\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_i).$$

□

Lemma 7.55 (Contexts and Free Variables for CBN). *If $\Phi \rhd_{\mathcal{N}_{\mathcal{G}}} \Gamma \vdash t : \mathbb{A}$ (resp. M), then $\mathtt{dom}(\Gamma) \subseteq \mathtt{fv}(t)$.*

Proof. The proof follows by induction on the structure of the typing derivation Φ. Note that all typing rules of $\mathcal{N}_{\mathcal{G}}$ either: (i) introduce assumptions only for variables occurring syntactically in the term being typed, or (ii) combine typing environments by pointwise addition. In particular:

- The variable rule introduces a singleton environment whose domain is exactly the variable occurring in the term.
- The abstraction rule removes the bound variable from the environment, preserving the invariant.
- Application combines environments of the subterms, and free variables of an application are exactly the union of free variables of its components.
- The `Get` and `Set` rules do not introduce new variable bindings; they only propagate the environments of their subterms, and thus preserve the inclusion.

Since no rule introduces assumptions for variables not occurring free in the term, we conclude that $\mathtt{dom}(\Gamma) \subseteq \mathtt{fv}(t)$. □

Lemma 7.56 (Zero Weight Tight Type Derivations Characterize CBN-Normal Forms)**.** *Let $t \in \Lambda_\circ^{\mathcal{G}}$ and $s \in \mathcal{S}_\circ$. If $\Phi \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash (t,s) : P$ is tight, then $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = 0 \cdot \mathtt{App} + 0 \cdot \mathtt{Get} + 0 \cdot \mathtt{Set}$ if and only if $t \in \mathtt{NO}_{\mathtt{CBN}}$.*

Proof. If $\Phi \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash (t,s) : P$ is tight, then it must be of the form

$$\frac{\Phi_1 \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash t : S \Rightarrow (A \times \{l \mapsto [\,]\}_{l \in \mathcal{L}}) \qquad \Phi_2 \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash s : S}{\emptyset \vdash (t,s) : A \times \{l \mapsto [\,]\}_{l \in \mathcal{L}}}\ \mathtt{Conf}$$

such that $P = \star \times \{l \mapsto [\,]\}_{l \in \mathcal{L}}$.

($\Rightarrow$) Let $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = 0 \cdot \mathtt{App} + 0 \cdot \mathtt{Get} + 0 \cdot \mathtt{Set}$. We proceed by cases, according to the last rule R of Φ_1:

- Case $R = \mathtt{Ax}$. Then, t is a variable. This leads to a contradiction with $t \in \Lambda_\circ^{\mathcal{G}}$. Therefore, this case does not apply.
- Case $R = \mathtt{Abs}$. Then, A is an arrow type. This leads to a contradiction with $A = \star$. Therefore, this case does not apply.
- Case $R = \mathtt{Ans}$. Then, t is a closed abstraction. Thus, $t \in \mathtt{NO}_{\mathtt{CBN}}$.
- Case $R = \mathtt{App}$. This leads to a contradiction with $\#_{\mathtt{App}}(\Phi) = 0 \cdot \mathtt{App}$. Therefore, this case does not apply.
- Case $R = \mathtt{Many}$. Then, A is a multi-type. This leads to a contradiction with $A = \star$. Therefore, this case does not apply.
- Case $R = \mathtt{Get}$. This leads to a contradiction with $\#_{\mathtt{Get}}(\Phi) = 0 \cdot \mathtt{Get}$. Therefore, this case does not apply.

– Case $R = \mathtt{Set}$. This leads to a contradiction with $\#_{\mathtt{Set}}(\Phi) = 0 \cdot \mathtt{Set}$. Therefore, this case does not apply.

Therefore, Φ_1 must end with rule `Ans` and t must be a closed abstraction. Thus, we can conclude that $t \in \mathtt{NO}_{\mathtt{CBN}}$.

($\Leftarrow$) Let $t \in \mathtt{NO}_{\mathtt{CBN}}$. Then, $t = \lambda x.u$, $S = \{l \mapsto [\,]\}_{l \in \mathcal{L}}$, and Φ must be of the form

$$\dfrac{\dfrac{}{\emptyset \vdash \lambda x.u : \{l \mapsto [\,]\}_{l\in\mathcal{L}} \Rightarrow (\star \times S)}\ \mathtt{Ans} \qquad \dfrac{\left(\dfrac{}{\emptyset \vdash s\,(l) : [\,]}\ \mathtt{EList}\right)_{l\in\mathcal{L}}}{\emptyset \vdash \{l \mapsto s\,(l)\}_{l\in\mathcal{L}} : \{l \mapsto [\,]\}_{l\in\mathcal{L}}}\ \mathtt{State}}{\emptyset \vdash (\lambda x.u, s) : \star \times S}\ \mathtt{Conf}$$

Therefore, we can conclude that $\#_{\mathtt{App,Get,Set}}(\Phi) = 0 \cdot \mathtt{App} + 0 \cdot \mathtt{Get} + 0 \cdot \mathtt{Set}$.

□

Lemma 7.57 (Tight Typability of CBN-Normal Forms)**.** *Let $s \in \mathcal{S}_\circ$. If $t \in \mathtt{NO}_{\mathtt{CBN}}$, then there is a derivation $\Phi \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash (t, s) : P$ that is tight.*

Proof. If $t \in \mathtt{NO}_{\mathtt{CBN}}$, then t is a closed abstraction, *i.e.* $t = \lambda x.u$. Therefore, we can take $P = \star \times S$, where $S = \{l \mapsto [\,]\}_{l\in\mathcal{L}}$, and Φ to be

$$\dfrac{\dfrac{}{\emptyset \vdash \lambda x.u : \{l \mapsto [\,]\}_{l\in\mathcal{L}} \Rightarrow (\star \times S)}\ \mathtt{Ans} \qquad \dfrac{\left(\dfrac{}{\emptyset \vdash s\,(l) : [\,]}\ \mathtt{EList}\right)_{l\in\mathcal{L}}}{\emptyset \vdash \{l \mapsto s\,(l)\}_{l\in\mathcal{L}} : \{l \mapsto [\,]\}_{l\in\mathcal{L}}}\ \mathtt{State}}{\emptyset \vdash (\lambda x.u, s) : \star \times S}\ \mathtt{Conf}$$

We can conclude since the above type derivation is tight. □

Lemma 7.59. *Let $l \in \mathcal{L}$ and $s \in \mathcal{S}$. There is $\Phi \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash s : S$, such that $S\,(l) \neq [\,]$, if and only if there are $\Psi \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash s\,(l) : \mathtt{head}(S\,(l))$ and $\Pi \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash s : S\{l := \mathtt{tail}(S\,(l))\}$. Moreover, $\#_{\mathtt{App,Get,Set}}(\Phi) = \#_{\mathtt{App,Get,Set}}(\Psi) + \#_{\mathtt{App,Get,Set}}(\Pi)$.*

Proof. ($\Rightarrow$) Notice that one of the premises of Φ must be derivation Φ_l:

$$\dfrac{\Phi_1 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash s\,(l) : \mathtt{head}(S\,(l)) \qquad \Phi_2 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash s\,(l) : \mathtt{tail}(S\,(l))}{\emptyset \vdash s\,(l) : \mathtt{head}(S\,(l)) \cdot \mathtt{tail}(S\,(l))}\ \mathtt{List}$$

such that $\mathtt{head}(S\,(l)) \cdot \mathtt{tail}(S\,(l)) = S\,(l)$. Since state typings are built pointwise by rule `State`, we can remove subderivation Φ_1 from Φ while keeping all other locations unchanged, to get a derivation $\Phi' \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash s : S\{l := \mathtt{tail}(S\,(l))\}$, such that $\#_{\mathtt{App,Get,Set}}(\Phi') = \#_{\mathtt{App,Get,Set}}(\Phi) - \#_{\mathtt{App,Get,Set}}(\Phi_1)$. Therefore, we can take $\Psi = \Phi_1$ and $\Pi = \Phi'$, since

$$\begin{aligned} \#_{\mathtt{App,Get,Set}}(\Phi) &= \#_{\mathtt{App,Get,Set}}(\Phi_1) + \#_{\mathtt{App,Get,Set}}(\Phi') \\ &= \#_{\mathtt{App,Get,Set}}(\Psi) + \#_{\mathtt{App,Get,Set}}(\Pi). \end{aligned}$$

($\Leftarrow$) Notice that one of the premises of Π must be derivation $\Pi_l \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash s\,(l) : \texttt{tail}(S\,(l))$. Therefore, we can build the following derivation Π'_l:

$$\frac{\Psi \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash s\,(l) : \texttt{head}(S\,(l)) \qquad \Pi_l \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash s\,(l) : \texttt{tail}(S\,(l))}{\emptyset \vdash s\,(l) : \texttt{head}(S\,(l)) \cdot \texttt{tail}(S\,(l))}\ \texttt{List}$$

such that $\texttt{head}(S\,(l)) \cdot \texttt{tail}(S\,(l)) = S\,(l)$. Since state typings are built pointwise by rule `State`, we can remove replace derivation Π_l with Π'_l in Π while keeping all other locations unchanged, to get a derivation $\Phi' \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash s : S$, such that $\#_{\texttt{App,Get,Set}}(\Phi') = \#_{\texttt{App,Get,Set}}(\Psi) + \#_{\texttt{App,Get,Set}}(\Pi)$. Therefore, we can take $\Phi = \Phi'$, since $\#_{\texttt{App,Get,Set}}(\Phi) = \#_{\texttt{App,Get,Set}}(\Psi) + \#_{\texttt{App,Get,Set}}(\Pi)$.

□

Lemma 7.60. *Let $l \in \mathcal{L}$ and $s \in \mathcal{S}$. There are $\Phi \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash s : S$, such that $S\,(l) = [\,]$, and $\Psi \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash t : L$ if and only if there is $\Pi \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash s\{l := t\} : S\{l := L\}$. Moreover, $\#_{\texttt{App,Get,Set}}(\Phi) + \#_{\texttt{App,Get,Set}}(\Psi) = \#_{\texttt{App,Get,Set}}(\Pi)$.*

Proof. Recall that store typings are built pointwise by rule `State`, with one premise per location.

($\Rightarrow$) One of the premises of Φ must be derivation $\Phi_l \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash s\,(l) : [\,]$, such that $\#_{\texttt{App,Get,Set}}(\Phi_l) = 0 \cdot \texttt{App} + 0 \cdot \texttt{Get} + 0 \cdot \texttt{Set}$. Now, let us assume that s is updated by replacing $s\,(l)$ with t. Then, $(s\{l := t\})\,(l) = t$. By replacing derivation Φ_l with Ψ in Φ, we obtain a derivation $\Pi' \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash s\{l := t\} : S\{l := L\}$, such that $(S\{l := L\}\,)\,(l) = L$. Therefore, we can take $\Pi = \Pi'$, since

$$\begin{aligned} \#_{\texttt{App,Get,Set}}(\Phi) + \#_{\texttt{App,Get,Set}}(\Psi) &= \#_{\texttt{App,Get,Set}}(\Phi) - \#_{\texttt{App,Get,Set}}(\Phi_l) + \#_{\texttt{App,Get,Set}}(\Psi) \\ &= \#_{\texttt{App,Get,Set}}(\Pi). \end{aligned}$$

($\Leftarrow$) One of the premises of Π must be derivation $\Phi_l \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash (s\{l := t\})\,(l) : (S\{l := L\})\,(l)$, such that $(s\{l := t\})\,(l) = t$ and $(S\{l := L\})\,(l) = L$. Moreover, we can build the following type derivation Φ'_l for $s\,(l)$:

$$\frac{}{\emptyset \vdash s\,(l) : [\,]}\ \texttt{EList}$$

such that $\#_{\texttt{App,Get,Set}}(\Phi'_l) = 0 \cdot \texttt{App} + 0 \cdot \texttt{Get} + 0 \cdot \texttt{Set}$. Now, we can build a type derivation $\Phi' \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash s : S$ for s, such that $S\,(l) = [\,]$, by replacing derivation Φ_l with Φ'_l in Π. Therefore,

we can take $\Phi = \Phi'$ and Ψ to be the premise $\Phi_l \triangleright_{\mathcal{N}_\mathcal{G}} \emptyset \vdash t : L$ extracted from $\Pi.$, since

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi') + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_l) \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi) - \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_l) \\
&+ \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi'_l) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_l) \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi'_l) \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi).
\end{aligned}$$

□

E.3.1.1 Quantitative Soundness

Lemma 7.61 (Weighted Substitution for CBN)**.** *Let $t, u \in \Lambda^{\mathcal{G}}$. If we have $\Phi \triangleright_{\mathcal{N}_\mathcal{G}} \Gamma; x : M \vdash t : \mathbb{A}$ (resp. N or L) and $\Psi \triangleright_{\mathcal{N}_\mathcal{G}} \emptyset \vdash u : M$, then $\Pi \triangleright_{\mathcal{N}_\mathcal{G}} \Gamma \vdash t\{x \setminus u\} : \mathbb{A}$ (resp. N or L), such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi)$.*

Proof. The proof follows by induction on the structure of Φ. We proceed by cases, according to the last rule R of Φ. We only show rules `Ax`, `Get`, and `Set`. The remaining cases follow easily from the *i.h.*:

- Case $R = \mathtt{Ax}$. Then, t is a variable, and we must consider the two following possibilities:
  - If $t = y \neq x$, then Φ must be of the form

    $$\frac{}{y : \{\!\{\mathbb{A}\}\!\}; x : \{\!\{\,\}\!\} \vdash y : \mathbb{A}}\ \mathtt{Ax}$$

    $\Gamma = y : \{\!\{\mathbb{A}\}\!\}$, and $M = \{\!\{\,\}\!\}$. Moreover, $t\{x \setminus u\} = y\{x \setminus u\} = y$, and Ψ must be of the form

    $$\frac{}{\emptyset \vdash u : \{\!\{\,\}\!\}}\ \mathtt{Many}$$

    Therefore, we can take $\Pi = \Phi$, since $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) = 0 \cdot \mathtt{App} + 0 \cdot \mathtt{Get} + 0 \cdot \mathtt{Set}$, and thus $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi)$.
  - If $t = x$, then Φ must be of the form

    $$\frac{}{x : \{\!\{\mathbb{A}\}\!\} \vdash x : \mathbb{A}}\ \mathtt{Ax}$$

    $\Gamma = \emptyset$, and $M = \{\!\{\mathbb{A}\}\!\}$. Moreover, $t\{x \setminus u\} = x\{x \setminus u\} = u$. Therefore, we can take $\Pi = \Psi$, since $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = 0 \cdot \mathtt{App} + 0 \cdot \mathtt{Get} + 0 \cdot \mathtt{Set}$, and thus $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi)$.

- Case $R = \texttt{Get}$. By α-conversion, we may assume $y \neq x$. Then, $t = \texttt{get}(l, \lambda y.r)$, and Φ must be of the form

$$\frac{\Phi' \triangleright_{\mathcal{N}_\mathcal{G}} \Gamma; y : \texttt{head}(S\,(l)); x : M \vdash r : S\{l := \texttt{tail}(S\,(l))\} \Rightarrow P}{\Gamma \vdash \texttt{get}(l, \lambda y.r) : S \Rightarrow P}\ \texttt{Get}$$

  and $\mathbb{A} = S \Rightarrow P$. By applying the *i.h.* to Φ', there is $\Pi' \triangleright_{\mathcal{N}_\mathcal{G}} \Gamma; y : \texttt{head}(S\,(l)) \vdash r\{x \setminus u\} : S\{l := \texttt{tail}(S\,(l))\} \Rightarrow P$, such that $\#_{\texttt{App,Get,Set}}(\Pi') = \#_{\texttt{App,Get,Set}}(\Phi') + \#_{\texttt{App,Get,Set}}(\Psi)$. Therefore, we can take Π to be

$$\frac{\Pi' \triangleright_{\mathcal{N}_\mathcal{G}} \Gamma; y : \texttt{head}(S\,(l)) \vdash r\{x \setminus u\} : S\{l := \texttt{tail}(S\,(l))\} \Rightarrow P}{\Gamma \vdash \texttt{get}(l, \lambda y.r\{x \setminus u\}) : S \Rightarrow P}\ \texttt{Get}$$

  since $\texttt{get}(l, \lambda y.r\{x \setminus u\}) = \texttt{get}(l, \lambda y.r)\{x \setminus u\}$, and

$$\begin{aligned}
\#_{\texttt{App,Get,Set}}(\Pi) &= \#_{\texttt{App,Get,Set}}(\Pi') + 1 \cdot \texttt{Get} \\
&= \#_{\texttt{App,Get,Set}}(\Phi') + \#_{\texttt{App,Get,Set}}(\Psi) + 1 \cdot \texttt{Get} \\
&= \#_{\texttt{App,Get,Set}}(\Phi) + \#_{\texttt{App,Get,Set}}(\Psi).
\end{aligned}$$

- Case $R = \texttt{Set}$. Then, $t = \texttt{set}((l, e), r)$, and Φ must be of the form

$$\frac{\Phi_1 \triangleright_{\mathcal{N}_\mathcal{G}} \Gamma_1; x : M_1 \vdash e : L \qquad \Phi_2 \triangleright_{\mathcal{N}_\mathcal{G}} \Gamma_2; x : M_2 \vdash r : S\{l := L\} \Rightarrow P}{(\Gamma_1 + \Gamma_2); x : M_1 \sqcup M_2 \vdash \texttt{set}((l, e), r) : S \Rightarrow P}\ \texttt{Set}$$

  $\Gamma = \Gamma_1 + \Gamma_2$, $M = M_1 \sqcup M_2$, and $\mathbb{A} = S \Rightarrow P$. By Lemma 7.54, there are $\Psi_1 \triangleright_{\mathcal{N}_\mathcal{G}} \emptyset \vdash u : M_1$ and $\Psi_2 \triangleright_{\mathcal{N}_\mathcal{G}} \emptyset \vdash u : M_2$, such that $\#_{\texttt{App,Get,Set}}(\Psi) = \#_{\texttt{App,Get,Set}}(\Psi_1) + \#_{\texttt{App,Get,Set}}(\Psi_2)$. By applying the *i.h.* to Φ_1, we know there exists $\Pi_1 \triangleright_{\mathcal{N}_\mathcal{G}} \Gamma_1 \vdash e\{x \setminus u\} : L$, such that $\#_{\texttt{App,Get,Set}}(\Pi_1) = \#_{\texttt{App,Get,Set}}(\Phi_1) + \#_{\texttt{App,Get,Set}}(\Psi_1)$. By applying the *i.h.* to Φ_2, we know there exists $\Pi_2 \triangleright_{\mathcal{N}_\mathcal{G}} \Gamma_2 \vdash r\{x \setminus u\} : S\{l := L\} \Rightarrow P$, such that $\#_{\texttt{App,Get,Set}}(\Pi_2) = \#_{\texttt{App,Get,Set}}(\Phi_2) + \#_{\texttt{App,Get,Set}}(\Psi_2)$. Therefore, we can take Π to be

$$\frac{\Pi_1 \triangleright_{\mathcal{N}_\mathcal{G}} \Gamma_1 \vdash e\{x \setminus u\} : L \qquad \Pi_2 \triangleright_{\mathcal{N}_\mathcal{G}} \Gamma_2 \vdash r\{x \setminus u\} : S\{l := L\} \Rightarrow P}{\Gamma_1 + \Gamma_2 \vdash \texttt{set}((l, e\{x \setminus u\}), r\{x \setminus u\}) : S \Rightarrow P}\ \texttt{Set}$$

  since we have that $\texttt{set}((l, e\{x \setminus u\}), r\{x \setminus u\}) = \texttt{set}((l, e), r)\{x \setminus u\}$, and

$$\begin{aligned}
\#_{\texttt{App,Get,Set}}(\Pi) &= \#_{\texttt{App,Get,Set}}(\Pi_1) + \#_{\texttt{App,Get,Set}}(\Pi_2) + 1 \cdot \texttt{Set} \\
&= \#_{\texttt{App,Get,Set}}(\Phi_1) + \#_{\texttt{App,Get,Set}}(\Psi_1) + \#_{\texttt{App,Get,Set}}(\Phi_2) \\
&+ \#_{\texttt{App,Get,Set}}(\Psi_2) + 1 \cdot \texttt{Set} \\
&= \#_{\texttt{App,Get,Set}}(\Phi) + \#_{\texttt{App,Get,Set}}(\Psi).
\end{aligned}$$

□

Lemma 7.62 (Exact Subject Reduction for CBN). *Let* $t \in \Lambda^{\mathcal{G}}_{\circ}$, $s \in \mathcal{S}_{\circ}$, *and* $S \in \{\beta, \mathtt{get}, \mathtt{set}\}$. *If* $(t,s) \Rightarrow_{\mathtt{CBN}(S)} (u,q)$ *and* $\Phi \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash (t,s) : P$, *then there is* $\Psi \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash (u,q) : P$, *such that* $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) + 1 \cdot R$, *where*

- $R = \mathtt{App}$, *if* $S = \beta$;
- $R = \mathtt{Get}$, *if* $S = \mathtt{get}$;
- *and* $R = \mathtt{Set}$, *if* $S = \mathtt{set}$.

Proof. Let $(t,s) = (\mathtt{N}[t_0], s) \Rightarrow_{\mathtt{CBN}(S)} (\mathtt{N}[u_0], q) = (u,q)$, such that $(t_0,s) \Mapsto_{\mathtt{CBN}(S)} (u_0,q)$. By α-conversion, we may assume all bound variables are fresh. The proof follows by induction on the definition of $\mathtt{N}$:

- Case $\mathtt{N} = \Box$:
  - If $S = \beta$, then $(t_0,s) = ((\lambda x.r)\, e, s) \Mapsto_{\beta} (r\{x \setminus e\}, s) = (u_0,q)$, and Φ must be of the form

$$\dfrac{\dfrac{\Phi_1 \rhd_{\mathcal{N}_{\mathcal{G}}} x : N \vdash r : S \Rightarrow P}{\emptyset \vdash \lambda x.r : S \Rightarrow (N \to (S \Rightarrow P) \times S)}\,\mathtt{Abs} \quad \Phi_2 \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash e : N}{\emptyset \vdash (\lambda x.r)\, e : S \Rightarrow P}\,\mathtt{App} \quad \Phi_3 \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash s : S}{\emptyset \vdash ((\lambda x.r)\, e, s) : P}\,\mathtt{Conf}$$

    Given the forms of Φ_1 and Φ_2, we can apply Lemma 7.61 to obtain a derivation $\Pi \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash e\{x \setminus e\} : S \Rightarrow P$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2)$. Therefore, we can take Ψ to be

$$\dfrac{\Pi \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash e\{x \setminus e\} : S \Rightarrow P \qquad \Phi_3 \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash s : S}{\emptyset \vdash (r\{x \setminus e\}, s) : P}\,\mathtt{Conf}$$

    since

$$\begin{aligned} \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{App} + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_3) \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi) + 1 \cdot \mathtt{App} + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_3) \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) + 1 \cdot \mathtt{App}. \end{aligned}$$

  - If $S = \mathtt{get}$, then $(t_0,s) = (\mathtt{get}(l, \lambda x.r), s) \Rightarrow_{\mathtt{CBN}(\mathtt{get})} (r\{x \setminus s\,(l)\}, s) = (u_0,q)$, and Φ must be of the form

$$\dfrac{\dfrac{\Phi_1 \rhd_{\mathcal{N}_{\mathcal{G}}} x : \mathtt{head}(S\,(l)) \vdash r : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P}{\emptyset \vdash \mathtt{get}(l, \lambda x.r) : S \Rightarrow P}\,\mathtt{Get} \quad \Phi_2 \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash s : S}{\emptyset \vdash (\mathtt{get}(l, \lambda x.r), s) : P}\,\mathtt{Conf}$$

    By applying Lemma 7.59 to Φ_2, there are derivations $\Pi_1 \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash s\,(l) : \mathtt{head}(S\,(l))$ and $\Psi_2 \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash s : S\{l := \mathtt{tail}(S\,(l))\}$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi_1) +$

$\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2)$. By applying Lemma 7.61 to Φ_1 and Π_1, there is $\Psi_1 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash r\{x \setminus s\,(l)\} : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi_1)$. Therefore, we can take Ψ to be

$$\frac{\Psi_1 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash r\{x \setminus s\,(l)\} : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P \quad \Psi_2 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash s : S\{l := \mathtt{tail}(S\,(l))\}}{\emptyset \vdash (r\{x \setminus s\,(l)\}, s) : P}\ \mathtt{Conf}$$

since

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{Get} \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2) + 1 \cdot \mathtt{Get} \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2) + 1 \cdot \mathtt{Get} \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) + 1 \cdot \mathtt{Get},
\end{aligned}$$

and $R = \mathtt{Get}$.

– If $S = \mathtt{set}$, then $(t_0, s) = (\mathtt{set}((l,e),r),s) \Rightarrow_{\mathtt{CBN}(\mathtt{set})} (r, s\{l := e\}) = (u_0, q)$, and Φ must be of the form

$$\frac{\dfrac{\Phi_1 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash e : L \quad \Phi_2 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash r : S\{l := L\} \Rightarrow P \quad S\,(l) = [\,]}{\emptyset \vdash \mathtt{set}((l,e),r) : S \Rightarrow P}\ \mathtt{Set} \qquad \Phi_3 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash s : S}{\emptyset \vdash (\mathtt{set}((l,e),r),s) : P}\ \mathtt{Conf}$$

By applying Lemma 7.60 to Φ_3 and Φ_1, there is $\Psi_3 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash s\{l := e\} : S\{l := L\}$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_3) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_3)$. Therefore, we can take Ψ to be

$$\frac{\Phi_2 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash r : S\{l := L\} \Rightarrow P \qquad \Psi_3 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash s\{l := e\} : S\{l := L\}}{\emptyset \vdash (r, s\{l := e\}) : P}\ \mathtt{Conf}$$

since

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{Set} + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_3) \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_3) + 1 \cdot \mathtt{Set} \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) + 1 \cdot \mathtt{Set},
\end{aligned}$$

and $R = \mathtt{Set}$.

- Case $\mathtt{N} = \mathtt{N}'\,r$. Then, we have that $(t,s) = (\mathtt{N}'[t_0]\,r, s) \Rightarrow_{\mathtt{CBN}(S)} (\mathtt{N}'[u_0]\,r, q) = (u, q)$, such that $(\mathtt{N}'[t_0], s) \Rightarrow_{\mathtt{CBN}(S)} (\mathtt{N}'[u_0], q)$, and Φ must be of the form

$$\frac{\dfrac{\Phi_1 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash \mathtt{N}'[t_0] : S \Rightarrow ((N \to (S \Rightarrow P)) \times S) \quad \Phi_2 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash r : N}{\emptyset \vdash \mathtt{N}'[t_0]\,r : S \Rightarrow P}\ \mathtt{App} \qquad \Phi_3 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash s : S}{\emptyset \vdash (\mathtt{N}'[t_0]\,r, s) : P}\ \mathtt{Conf}$$

Notice that we can build the following type derivation for $(\mathtt{N}'[t_0], s)$:

$$\dfrac{\Phi_1 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash \mathtt{N}'[t_0] : S \Rightarrow ((N \to (S \Rightarrow P)) \times S) \qquad \Phi_3 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash s : S}{\emptyset \vdash (\mathtt{N}'[t_0], s) : (N \to (S \Rightarrow P)) \times S}\ \mathtt{Conf}$$

Therefore, by applying the *i.h.* to $(\mathtt{N}'[t_0], s)$, we know there is a type derivation for $(\mathtt{N}'[u_0], q)$ whose premises must be of the form $\Psi_1 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash \mathtt{N}'[u_0] : Q \Rightarrow ((N \to (S \Rightarrow P)) \times S)$ and $\Psi_3 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash q : Q$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_3) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_3) + 1 \cdot R$, where

- $R = \mathtt{App}$, if $S = \beta$;
- $R = \mathtt{Get}$, if $S = \mathtt{get}$;
- and $R = \mathtt{Set}$, if $S = \mathtt{set}$.

Therefore, we can take Ψ to be

$$\dfrac{\dfrac{\Psi_1 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash \mathtt{N}'[u_0] : Q \Rightarrow ((N \to (S \Rightarrow P)) \times S) \qquad \Phi_2 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash r : N}{\emptyset \vdash \mathtt{N}'[u_0]\, r : Q \Rightarrow P}\ \mathtt{App} \qquad \Psi_3 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash q : Q}{\emptyset \vdash (\mathtt{N}'[u_0]\, r, q) : P}\ \mathtt{Conf}$$

since

$$\begin{aligned} \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{App} + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_3) \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + 1 \cdot R + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{App} + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_3) \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) + 1 \cdot R. \end{aligned}$$

□

Theorem 7.63 (Exact Soundness for CBN). *Let $t \in \Lambda^{\mathcal{G}}_\circ$ and $s \in \mathcal{S}_\circ$. If there is $\Phi \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash (t, s) : P$ tight, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = n \cdot \mathtt{App} + g \cdot \mathtt{Get} + s \cdot \mathtt{Set}$, then $(t, s) \overset{n\cdot\beta + g\cdot\mathtt{get} + s\cdot\mathtt{set}}{\Rightarrow}_{\mathtt{CBN}} (u, q)$, such that $u \in \mathtt{NO}_{\mathtt{CBN}}$.*

Proof. Let $\Phi \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash (t, s) : P$ be tight, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = n \cdot \mathtt{App} + g \cdot \mathtt{Get} + s \cdot \mathtt{Set}$. We proceed by induction on $n + g + s$:

- If $n + g + s = 0$, then $n = g = s = 0$. By Lemma 7.56, $t \in \mathtt{NO}_{\mathtt{CBN}}$. Therefore, we can conclude with $u = t$ and $q = s$.
- If $n + g + s > 0$, then $t \notin \mathtt{NO}_{\mathtt{CBN}}$ by Lemma 7.56. Since the store never blocks CBN reductions, by Lemma 7.18 there are $r \in \Lambda^{\mathcal{G}}_\circ$ and $s' \in \mathcal{S}_\circ$, such that $(t, s) \Rightarrow_{\mathtt{CBN}(S)} (r, s')$, for some $S \in \{\beta, \mathtt{get}, \mathtt{set}\}$. By Lemma 7.62, there is $\Phi' \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash (r, s') : P$ tight, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi') + 1 \cdot R$, where

- $R = \mathtt{App}$, if $S = \beta$;
- $R = \mathtt{Get}$, if $S = \mathtt{get}$;
- and $R = \mathtt{Set}$, if $S = \mathtt{set}$.

Let us assume, without loss of generality, that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi') = n' \cdot \mathtt{App} + g' \cdot \mathtt{Get} + s' \cdot \mathtt{Set}$. Then, $n' + g' + s' + 1 = n + g + s$. Now, by applying the *i.h.*, $(r, s')^{n' \cdot \beta + g' \cdot \mathtt{get} + s' \cdot \mathtt{set}}_{\mathtt{CBN}} (u', q')$, such that $u' \in \mathtt{NO}_{\mathtt{CBN}}$. Therefore, $(t, s) \Rightarrow_{\mathtt{CBN}(S)} (r, s')^{n' \cdot \beta + g' \cdot \mathtt{get} + s' \cdot \mathtt{set}}_{\mathtt{CBN}} (u', q')$. Thus, letting $(u, q) = (u', q')$, we can conclude $(t, s)^{n \cdot \beta + g \cdot \mathtt{get} + s \cdot \mathtt{set}}_{\mathtt{CBN}} (u, q)$.

□

E.3.1.2 Quantitative Completeness

Lemma 7.64 (Weighted Anti-Substitution for CBN). *Let $t \in \Lambda^{\mathcal{G}}$ and $u \in \Lambda^{\mathcal{G}}_{\circ}$. If $\Phi \rhd_{\mathcal{N}_{\mathcal{G}}} \Gamma \vdash t\{x \setminus u\} : \mathbb{A}$ (resp. N or L), then there exist $\Psi \rhd_{\mathcal{N}_{\mathcal{G}}} \Gamma; x : M \vdash t : \mathbb{A}$ (resp. N or L) and $\Pi \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash u : M$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi)$.*

Proof. The proof follows by induction on the structure of t. We only show the cases where t is a variable or an operation:

- Case $t = y \neq x$. Then, $t\{x \setminus u\} = y\{x \setminus u\} = y$. Therefore, we can take $M = \{\!\{\,\}\!\}$, $\Psi = \Phi$, and Π as the following derivation

$$\frac{}{\emptyset \vdash u : \{\!\{\,\}\!\}}\ \mathtt{Many}$$

since $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi)$.

- Case $t = x$. Then, $t\{x \setminus u\} = x\{x \setminus u\} = u$, and, since $\mathtt{fv}(u) = \emptyset$, we know that $\Gamma = \emptyset$ by Lemma 7.55. Now, we must consider two cases, according to the type assigned to u in Φ:
  - Let $\Phi \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash u : \mathbb{A}$. Then, we can take $M = \{\!\{\mathbb{A}\}\!\}$, Π to be

$$\frac{\Phi \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash u : \mathbb{A}}{\emptyset \vdash u : \{\!\{\mathbb{A}\}\!\}}\ \mathtt{Many}$$

    and Ψ to be

$$\frac{}{x : \{\!\{\mathbb{A}\}\!\} \vdash x : \mathbb{A}}\ \mathtt{Ax}$$

    since $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi)$.
  - Let $\Phi \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash u : N$. Then, we can take $M = N$, $\Pi = \Phi$, assume (without loss of generality) that $M = \{\!\{\mathbb{A}_i\}\!\}_{i \in \mathtt{I}}$, and Ψ to be

$$\frac{\left(\dfrac{}{x : \{\!\{\mathbb{A}_i\}\!\} \vdash x : \mathbb{A}_i}\ \mathtt{Ax}\right)_{i \in \mathtt{I}}}{x : \{\!\{\mathbb{A}_i\}\!\}_{i \in \mathtt{I}} \vdash x : \{\!\{A_i\}\!\}_{i \in \mathtt{I}}}\ \mathtt{Many}$$

$$\text{since } \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi).$$

- Case $t = \mathtt{get}(l, \lambda y.r)$. Assume $y \neq x$ by α-conversion. Then, $t\{x \setminus u\} = \mathtt{get}(l, \lambda y.r\{x \setminus u\})$, and Φ must be of the form:

$$\frac{\Phi' \rhd_{\mathcal{N}_\mathcal{G}} \Gamma; y : \mathtt{head}(S\,(l)) \vdash r\{x \setminus u\} : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P}{\Gamma \vdash \mathtt{get}(l, \lambda y.r\{x \setminus u\}) : S \Rightarrow P}\ \mathtt{Get}$$

  and $\mathbb{A} = S \Rightarrow P$. By applying the *i.h.* to r, we know there exist $\Psi' \rhd_{\mathcal{N}_\mathcal{G}} \Gamma; y : \mathtt{head}(S\,(l)); x : M \vdash r : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P$ and $\Pi' \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash u : M$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi') = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi') + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi')$. Therefore, we can conclude by taking $\Pi = \Pi'$ and Ψ to be

$$\frac{\Psi' \rhd_{\mathcal{N}_\mathcal{G}} \Gamma; y : \mathtt{head}(S\,(l)); x : M \vdash r : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P}{\Gamma; x : M \vdash \mathtt{get}(l, \lambda y.r) : S \Rightarrow P}\ \mathtt{Get}$$

  since

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi') + 1 \cdot \mathtt{Get} \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi') + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi') + 1 \cdot \mathtt{Get} \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi).
\end{aligned}$$

- Case $t = \mathtt{set}((l, e), r)$. Then, $t\{x \setminus u\} = \mathtt{set}((l, e\{x \setminus u\}), r\{x \setminus u\})$, and Φ must be of the form

$$\frac{\Phi_1 \rhd_{\mathcal{N}_\mathcal{G}} \Gamma_1 \vdash e\{x \setminus u\} : L \qquad \Phi_2 \rhd_{\mathcal{N}_\mathcal{G}} \Gamma_2 \vdash r\{x \setminus u\} : S\{l := L\} \Rightarrow P}{\Gamma_1 + \Gamma_2 \vdash \mathtt{set}((l, e\{x \setminus u\}), r\{x \setminus u\}) : S \Rightarrow P}\ \mathtt{Set}$$

  $\Gamma = \Gamma_1 + \Gamma_2$, and $\mathbb{A} = S \Rightarrow P$. By applying the *i.h.* to e, we know there exist $\Psi_1 \rhd_{\mathcal{N}_\mathcal{G}} \Gamma_1; x : M_1 \vdash e : L$ and $\Pi_1 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash u : M_1$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi_1)$. By applying the *i.h.* to r, we know there exist $\Psi_2 \rhd_{\mathcal{N}_\mathcal{G}} \Gamma_2; x : M_2 \vdash r : S\{l := L\} \Rightarrow P$ and $\Pi_2 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash u : M_2$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi_2)$. By Lemma 7.54, we know there exists $\Pi' \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash u : M_1 \sqcup M_2$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi') = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi_2)$. Therefore, we can conclude by taking $M = M_1 \sqcup M_2$, $\Pi = \Pi'$ and Ψ to be

$$\frac{\Psi_1 \rhd_{\mathcal{N}_\mathcal{G}} \Gamma_1; x : M_1 \vdash e : L \qquad \Psi_2 \rhd_{\mathcal{N}_\mathcal{G}} \Gamma_2; x : M_2 \vdash r : S\{l := L\} \Rightarrow P}{(\Gamma_1 + \Gamma_2); x : M_1 \sqcup M_2 \vdash \mathtt{set}((l, e), r) : S \Rightarrow P}\ \mathtt{Set}$$

  since

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{Set} \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2) \\
&+ \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi_2) + 1 \cdot \mathtt{Set} \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi).
\end{aligned}$$

The remaining cases follow easily from the *i.h.*. □

Lemma 7.65 (Exact Subject Expansion for CBN). *Let* $t \in \Lambda^{\mathcal{G}}_{\circ}$, $s \in \mathcal{S}_{\circ}$, *and* $S \in \{\beta, \mathtt{get}, \mathtt{set}\}$. *If* $(t,s) \Rightarrow_{\mathtt{CBN}(S)} (u,q)$ *and* $\Phi \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash (u,q) : P$, *then there is* $\Psi \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash (t,s) : P$, *such that* $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) + 1 \cdot R = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi)$, *where*

- $R = \mathtt{App}$, *if* $S = \beta$;
- $R = \mathtt{Get}$, *if* $S = \mathtt{get}$;
- *and* $R = \mathtt{Set}$, *if* $S = \mathtt{set}$.

Proof. Let $(t,s) = (\mathtt{N}[t_0], s) \Rightarrow_{\mathtt{CBN}(S)} (\mathtt{N}[u_0], q) = (u,q)$, such that $(t_0, s) \mapsto\!\!\!\Rightarrow_{\mathtt{CBN}(S)} (u_0, q)$. The proof follows by induction on the definition of $\mathtt{N}$:

- Let $\mathtt{N} = \Box$:
  - If $S = \beta$, $(t_0, s) = ((\lambda x.r)\, e, s) \mapsto\!\!\!\Rightarrow_{\beta} (r\{x \setminus e\}, s) = (u_0, q)$, and Φ is of the form
    $$\dfrac{\Phi_1 \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash r\{x \setminus e\} : S \Rightarrow P \qquad \Phi_2 \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash s : S}{\emptyset \vdash (r\{x \setminus e\}, s) : P}\ \mathtt{Conf}$$
    Given the form of Φ_1, we can apply Lemma 7.64 to obtain derivations $\Psi_1 \rhd_{\mathcal{N}_{\mathcal{G}}} x : M \vdash r : S \Rightarrow P$ and $\Psi_2 \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash e : M$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2)$. Therefore, we can take Ψ to be
    $$\dfrac{\dfrac{\dfrac{\Psi_1 \rhd_{\mathcal{N}_{\mathcal{G}}} x : M \vdash r : S \Rightarrow P}{\emptyset \vdash \lambda x.r : S \Rightarrow ((M \to (S \Rightarrow P)) \times S)}\ \mathtt{Abs} \qquad \Psi_2 \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash e : M}{\emptyset \vdash (\lambda x.r)\, e : S \Rightarrow P}\ \mathtt{App} \qquad \Phi_2 \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash s : S}{\emptyset \vdash ((\lambda x.r)\, e, s) : P}\ \mathtt{Conf}$$
    since
    $$\begin{aligned} \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) + 1 \cdot \mathtt{App} &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{App} \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2) \\ &+ \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{App} \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi). \end{aligned}$$
  - If $S = \mathtt{get}$, then $(t_0, s) = (\mathtt{get}(l, \lambda x.r), s) \Rightarrow_{\mathtt{CBN}(\mathtt{get})} (r\{x \setminus s\,(l)\}, s) = (u_0, q)$, and Φ is of the form
    $$\dfrac{\Phi_1 \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash r\{x \setminus s\,(l)\} : S \Rightarrow P \qquad \Phi_2 \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash s : S}{\emptyset \vdash (r\{x \setminus s\,(l)\}, s) : P}\ \mathtt{Conf}$$
    By applying Lemma 7.64 to Φ_1, there exist $\Psi_1 \rhd_{\mathcal{N}_{\mathcal{G}}} x : M \vdash r : S \Rightarrow P$ and $\Pi_2 \rhd_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash s\,(l) : M$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi_2)$. Let $R = M \cdot S\,(l)$. Then, $\mathtt{head}(R\,(l)) = M$, $\mathtt{tail}(R\,(l)) = S\,(l) = L$, and $R\{l := \mathtt{tail}(R\,(l))\} = S$. Thus,

by applying Lemma 7.59 to Π_2 and Φ_2, there is a type derivation $\Psi_2 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash s : R$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi_2)$. Therefore, we can take Ψ to be

$$\dfrac{\dfrac{\Psi_1 \rhd_{\mathcal{N}_\mathcal{G}} x : \mathtt{head}(R\,(l)) \vdash r : R\{l := \mathtt{tail}(R\,(l))\} \Rightarrow P}{\emptyset \vdash \mathtt{get}(l, \lambda x.r) : R \Rightarrow P}\,\mathtt{Get} \qquad \Psi_2 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash s : R}{\emptyset \vdash (\mathtt{get}(l, \lambda x.r), s) : P}\,\mathtt{Conf}$$

since

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) + 1 \cdot \mathtt{Get} &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{Get} \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi_2) \\
&+ \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{Get} \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2) + 1 \cdot \mathtt{Get} \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi),
\end{aligned}$$

and $R = \mathtt{Get}$.

– If $S = \mathtt{set}$, then $(t_0, s) = (\mathtt{set}((l, e), r), s) \Rightarrow_{\mathtt{CBN}(\mathtt{set})} (r, s\{l := e\}) = (u_0, q)$, and Φ is of the form

$$\dfrac{\Phi_1 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash r : S \Rightarrow P \qquad \Phi_2 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash s\{l := e\} : S}{\emptyset \vdash (r, s\{l := e\}) : P}\,\mathtt{Conf}$$

Let $R = S\{l := [\,]\}$, and $R\{l := S\,(l)\} = R\{l := L\} = S$. Then, by applying Lemma 7.60 to Φ_2, there are $\Psi_2 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash s : R$ and $\Psi_1 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash e : L$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2)$. Then, we can take Ψ to be

$$\dfrac{\dfrac{\begin{array}{c}\Psi_1 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash e : L \\ \Phi_1 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash r : R\{l := L\} \Rightarrow P \qquad R\,(l) = [\,]\end{array}}{\emptyset \vdash \mathtt{set}((l, e), r) : R \Rightarrow P}\,\mathtt{Set} \qquad \Psi_2 \rhd_{\mathcal{N}_\mathcal{G}} \emptyset \vdash s : R}{\emptyset \vdash (\mathtt{set}((l, e), r), s) : P}\,\mathtt{Conf}$$

since

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) + 1 \cdot \mathtt{Set} &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{Set} \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2) \\
&+ \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + 1 \cdot \mathtt{Set} \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi),
\end{aligned}$$

and $R = \mathtt{Set}$.

- Case $\mathtt{N} = \mathtt{N}'\, r$. Then, we have that $(t, s) = (\mathtt{N}'[t_0]\, r, s) \Rightarrow_{\mathtt{CBN}(S)} (\mathtt{N}'[u_0]\, r, q) = (u, q)$, such that $(\mathtt{N}'[t_0], s) \Rightarrow_{\mathtt{CBN}(S)} (\mathtt{N}'[u_0], q)$, and Φ must be of the form

$$\dfrac{\dfrac{\Phi_1 \rhd_{\mathcal{N}_\mathcal{G}} \varnothing \vdash \mathtt{N}'[u_0] : Q \Rightarrow ((N \to (R \Rightarrow P)) \times R) \quad \Phi_2 \rhd_{\mathcal{N}_\mathcal{G}} \varnothing \vdash r : N}{\varnothing \vdash \mathtt{N}'[u_0]\, r : Q \Rightarrow P}\ \mathtt{App} \quad \Phi_3 \rhd_{\mathcal{N}_\mathcal{G}} \varnothing \vdash q : Q}{\varnothing \vdash (\mathtt{N}'[u_0]\, r, q) : P}\ \mathtt{Conf}$$

  Notice that we can build the following type derivation for $(\mathtt{N}'[u_0], q)$:

$$\dfrac{\Phi_1 \rhd_{\mathcal{N}_\mathcal{G}} \varnothing \vdash \mathtt{N}'[u_0] : Q \Rightarrow ((N \to (R \Rightarrow P)) \times R) \quad \Phi_3 \rhd_{\mathcal{N}_\mathcal{G}} \varnothing \vdash q : Q}{\varnothing \vdash (\mathtt{N}'[u_0], q) : (N \to (R \Rightarrow P)) \times R}\ \mathtt{Conf}$$

  Therefore, by applying the *i.h.* to $(\mathtt{N}'[u_0], q)$, we know there is a type derivation for $(\mathtt{N}'[t_0], s)$ whose premises must be of the form $\Psi_1 \rhd_{\mathcal{N}_\mathcal{G}} \varnothing \vdash \mathtt{N}'[t_0] : S \Rightarrow ((N \to (R \Rightarrow P)) \times R)$ and $\Psi_3 \rhd_{\mathcal{N}_\mathcal{G}} \varnothing \vdash s : S$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_3) + 1 \cdot R = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_3)$, where

  - $R = \mathtt{App}$, if $S = \beta$;
  - $R = \mathtt{Get}$, if $S = \mathtt{get}$;
  - and $R = \mathtt{Set}$, if $S = \mathtt{set}$.

  Therefore, we can take Ψ to be

$$\dfrac{\dfrac{\Psi_1 \rhd_{\mathcal{N}_\mathcal{G}} \varnothing \vdash \mathtt{N}'[t_0] : S \Rightarrow ((N \to (R \Rightarrow P)) \times R) \quad \Phi_2 \rhd_{\mathcal{N}_\mathcal{G}} \varnothing \vdash r : N}{\varnothing \vdash \mathtt{N}'[t_0]\, r : S \Rightarrow P}\ \mathtt{App} \quad \Psi_3 \rhd_{\mathcal{N}_\mathcal{G}} \varnothing \vdash s : S}{\varnothing \vdash (\mathtt{N}'[t_0]\, r, s) : P}\ \mathtt{Conf}$$

  since

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) + 1 \cdot R &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{App} \\
&+ \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_3) + 1 \cdot R \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{App} + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_3) \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi).
\end{aligned}$$

□

Theorem 7.66 (Exact Completeness for CBN). *Let $t \in \Lambda^{\mathcal{G}}_{\circ}$ and $s \in \mathcal{S}_{\circ}$. If $(t, s) \overset{n\cdot\beta + g\cdot\mathtt{get} + s\cdot\mathtt{set}}{\Rightarrow}_{\mathtt{CBN}} (u, q)$, such that $u \in \mathtt{NO}_{\mathtt{CBN}}$, then there is $\Phi \rhd_{\mathcal{N}_\mathcal{G}} \varnothing \vdash (t, s) : P$ tight, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = n \cdot \mathtt{App} + g \cdot \mathtt{Get} + s \cdot \mathtt{Set}$.*

Proof. Let $(t, s) \overset{n\cdot\beta + g\cdot\mathtt{get} + s\cdot\mathtt{set}}{\Rightarrow}_{\mathtt{CBN}} (u, q)$, such that $u \in \mathtt{NO}_{\mathtt{CBN}}$. We proceed by induction on $n + s + g$:

- If $n+g+s=0$, then $n=g=s=0$. Therefore, $t \in \mathtt{NO}_{\mathtt{CBN}}$ by Lemma 7.18. Thus, we can conclude by Lemma 7.57.

- If $n+g+s>0$, then $t \notin \mathtt{NO}_{\mathtt{CBN}}$ by Lemma 7.56. Since $t \notin \mathtt{NO}_{\mathtt{CBN}}$, by Lemma 7.18 there exist r, s' and $S \in \{\beta, \mathtt{get}, \mathtt{set}\}$ such that $(t,s) \Rightarrow_{\mathtt{CBN}(S)} (r,s') \overset{n'\cdot\beta+g'\cdot\mathtt{get}+s'\cdot\mathtt{set}}{\Rightarrow\!\!\!\!\!\!\Rightarrow}_{\mathtt{CBN}} (u,q)$, and $n'+g'+s'+1 = n+g+s$. Now, by applying the *i.h.*, there is $\Phi' \triangleright_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash (r,s') : A \times Q$ tight, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi') = n' \cdot \mathtt{App} + g' \cdot \mathtt{Get} + s' \cdot \mathtt{Set}$. By Lemma 7.65, there is $\Phi' \triangleright_{\mathcal{N}_{\mathcal{G}}} \emptyset \vdash (t,s) : A \times Q$ tight, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi') + 1 \cdot R = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi)$, where

  - $R = \mathtt{App}$, if $S = \beta$;
  - $R = \mathtt{Get}$, if $S = \mathtt{get}$;
  - and $R = \mathtt{Set}$, if $S = \mathtt{set}$.

  Therefore, we can take $\Phi = \Phi'$, since

$$\begin{aligned} \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi') + 1 \cdot R \\ &= n' \cdot \mathtt{App} + g' \cdot \mathtt{Get} + s' \cdot \mathtt{Set} + 1 \cdot R \\ &= n \cdot \mathtt{App} + g \cdot \mathtt{Get} + s \cdot \mathtt{Set}. \end{aligned}$$

□

E.3.2 Call-By-Value

Lemma 7.73 (CBV Multiset-Typings Split and Merge). *Let $v \in \mathrm{V}^{\mathcal{G}}$. Then, there is $\Phi \triangleright_{\mathcal{V}_{\mathcal{G}}} \Gamma \vdash v : M$, such that $M = \sqcup_{i\in\mathrm{I}} M_i$, if and only if there are $\Phi_i \triangleright_{\mathcal{V}_{\mathcal{G}}} \Gamma_i \vdash v : M_i$, such that $\Gamma = +_{i\in\mathrm{I}}\Gamma_i$, and $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = +_{i\in\mathrm{I}}\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_i)$.*

Proof. The proof is identical to that of Lemma 7.54. Indeed, the multiset splitting and merging for values is identical to that for terms in CBN case. □

Lemma 7.74 (Contexts and Free Variables for CBV). *If $\Phi \triangleright_{\mathcal{V}_{\mathcal{G}}} \Gamma \vdash t : \mathbb{M}$ (resp. M), then $\mathtt{dom}(\Gamma) \subseteq \mathtt{fv}(t)$.*

Proof. The proof is identical to that of Lemma 7.55. Indeed, the CBV typing system does not differ from the CBN one in variable occurrences and binding structure. Therefore, every variable that is assigned a non-empty multi-type in the typing context must occur free in the typed term. □

Lemma 7.75 (CBV Values Are Not Effectful). *Let $t \in \Lambda^{\mathcal{G}}$ and $\Phi \triangleright_{\mathcal{V}_{\mathcal{G}}} \Gamma \vdash t : S \Rightarrow (M \times Q)$. If $t \in \mathrm{V}^{\mathcal{G}}$, then $Q = S$ and Φ ends with rule $\mathtt{Lift}$.*

Proof. Since $t \in \mathrm{V}^{\mathcal{G}}$, t is either a variable or an abstraction. We analyze the possible last rule of Φ:

- Rules App, Get, and Set cannot be the last rule of Φ, since they type applications or effectful operations, which are not values.
- The only remaining possibility is that Φ ends with rule Lift. By definition of Lift, this rule preserves the state component, hence $Q = S$.

Therefore, if t is a value and $\Phi \rhd_{\mathcal{V}_\mathcal{G}} \Gamma \vdash t : S \Rightarrow (M \times Q)$, then necessarily $Q = S$ and Φ ends with rule Lift. □

Lemma 7.76 (Zero Weight Tight Type Derivations Characterize CBV-Normal Forms)**.** *Let $t \in \Lambda^{\mathcal{G}}_{\circ}$ and $s \in \mathcal{S}_{\circ}$. If $\Phi \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash (t, s) : P$ is tight, then $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = 0 \cdot \mathtt{App} + 0 \cdot \mathtt{Get} + 0 \cdot \mathtt{Set}$ if and only if $t \in \mathtt{NO}_{\mathtt{CBV}}$.*

Proof. If $\Phi \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash (t, s) : M \times Q$ is tight, then it must be of the form

$$\frac{\Phi_1 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash t : S \Rightarrow (M \times \{l \mapsto [\,]\}_{l \in \mathcal{L}}) \qquad \Phi_2 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash s : S}{\emptyset \vdash (t, s) : M \times \{l \mapsto [\,]\}_{l \in \mathcal{L}}}\ \mathtt{Conf}$$

such that $P = \{\!\{\ \}\!\} \times \{l \mapsto [\,]\}_{l \in \mathcal{L}}$.

($\Rightarrow$) Let $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = 0 \cdot \mathtt{App} + 0 \cdot \mathtt{Get} + 0 \cdot \mathtt{Set}$. Observe that in the CBV type system, the only rule that assigns a monadic type to a value without introducing any App, Get, or Set steps is Lift. We proceed by cases, according to the last rule R of Φ_1:

- Case $R = \mathtt{Ax}$. Then, t is a variable. This leads to a contradiction with $t \in \Lambda^{\mathcal{G}}_{\circ}$. Therefore, this case does not apply.
- Case $R = \mathtt{Abs}$. Then, A is not a monadic type. This leads to a contradiction with $A = \{\!\{\ \}\!\}$. Therefore, this case does not apply.
- Case $R = \mathtt{App}$. This leads to a contradiction with $\#_{\mathtt{App}}(\Phi) = 0 \cdot \mathtt{App}$. Therefore, this case does not apply.
- Case $R = \mathtt{Lift}$. Then, t is a closed abstraction. Thus, $t \in \mathtt{NO}_{\mathtt{CBV}}$.
- Case $R = \mathtt{Get}$. This leads to a contradiction with $\#_{\mathtt{Get}}(\Phi) = 0 \cdot \mathtt{Get}$. Therefore, this case does not apply.
- Case $R = \mathtt{Set}$. This leads to a contradiction with $\#_{\mathtt{Set}}(\Phi) = 0 \cdot \mathtt{Set}$. Therefore, this case does not apply.

Therefore, Φ_1 must end with rule Lift and t must be a closed abstraction. Thus, we can conclude that $t \in \mathtt{NO}_{\mathtt{CBV}}$.

($\Leftarrow$) Let $t \in \mathtt{NO}_{\mathtt{CBV}}$. Then, $t = \lambda x.u$ is a closed abstraction, $S = \{l \mapsto [\,]\}_{l \in \mathcal{L}}$, and Φ must be of the form

$$\dfrac{\dfrac{}{\emptyset \vdash \lambda x.u : \{\!\{\,\}\!\}}\,\mathtt{Abs}}{\emptyset \vdash \lambda x.u : \{l \mapsto [\,]\}_{l \in \mathcal{L}} \Rightarrow (\{\!\{\,\}\!\} \times S)}\,\mathtt{Lift} \qquad \dfrac{\left(\dfrac{}{\emptyset \vdash s\,(l) : [\,]}\,\mathtt{EList}\right)_{l \in \mathcal{L}}}{\emptyset \vdash \{l \mapsto s\,(l)\}_{l \in \mathcal{L}} : \{l \mapsto [\,]\}_{l \in \mathcal{L}}}\,\mathtt{State}}{\emptyset \vdash (\lambda x.u, s) : \{\!\{\,\}\!\} \times S}\,\mathtt{Conf}$$

Therefore, we can conclude that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = 0 \cdot \mathtt{App} + 0 \cdot \mathtt{Get} + 0 \cdot \mathtt{Set}$.

□

Lemma 7.77 (Tight Typability of CBV-Normal Forms)**.** *Let $s \in \mathcal{S}_\circ$. If $t \in \mathtt{NO}_{\mathtt{CBV}}$, then there is a derivation $\Phi \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash (t, s) : P$ that is tight.*

Proof. If $t \in \mathtt{NO}_{\mathtt{CBV}}$, then $t \in \mathrm{V}^{\mathcal{G}}_\circ$. Therefore, t is a closed abstraction, *i.e.* $t = \lambda x.u$, and we can take $P = \{\!\{\,\}\!\} \times \{l \mapsto [\,]\}_{l \in \mathcal{L}}$, and Φ to be

$$\dfrac{\dfrac{}{\emptyset \vdash \lambda x.u : \{\!\{\,\}\!\}}\,\mathtt{Abs}}{\emptyset \vdash \lambda x.u : \{l \mapsto [\,]\}_{l \in \mathcal{L}} \Rightarrow P}\,\mathtt{Lift} \qquad \dfrac{\left(\dfrac{}{\emptyset \vdash s\,(l) : [\,]}\,\mathtt{EList}\right)_{l \in \mathcal{L}}}{\emptyset \vdash \{l \mapsto s\,(l)\}_{l \in \mathcal{L}} : \{l \mapsto [\,]\}_{l \in \mathcal{L}}}\,\mathtt{State}}{\emptyset \vdash (\lambda x.u, s) : P}\,\mathtt{Conf}$$

We can conclude since the above type derivation is tight. □

Lemma 7.79. *Let $l \in \mathcal{L}$ and $s \in \mathcal{S}$. There is $\Phi \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash s : S$, such that $S\,(l) \neq [\,]$, if and only if there are $\Psi \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash s\,(l) : \mathtt{head}(S\,(l))$ and $\Pi \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash s : S\{l := \mathtt{tail}(S\,(l))\}$. Moreover, $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi)$.*

Proof. We omit the proof since it is identical to that of the CBN $\lambda^{\mathcal{G}}$-calculus (see Lemma 7.59), the typing of states and the associated weight bookkeeping being independent of the evaluation strategy. □

Lemma 7.80. *Let $l \in \mathcal{L}$ and $s \in \mathcal{S}$. There are $\Phi \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash s : S$, such that $S\,(l) = [\,]$, and $\Psi \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash v : L$ if and only if there is $\Pi \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash s\{l := v\} : S\{l := L\}$. Moreover, $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi)$.*

Proof. e omit the proof since it is identical to that of the CBN $\lambda^{\mathcal{G}}$-calculus (see Lemma 7.60), as state typing and its weight bookkeeping are independent of the evaluation strategy. □

E.3.2.1 Quantitative Soundness

Lemma 7.81 (Weighted Substitution for CBV)**.** *Let $t \in \Lambda^{\mathcal{G}}_\circ$, and $v \in \mathrm{V}^{\mathcal{G}}$. If $\Phi \rhd_{\mathcal{V}_\mathcal{G}} \Gamma; x : M \vdash t : \mathbb{M}$ (resp. N or L) and $\Psi \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash v : M$, then $\Pi \rhd_{\mathcal{V}_\mathcal{G}} \Gamma \vdash t\{x \setminus v\} : \mathbb{M}$ (resp. N or L), such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi)$.*

Proof. The proof follows by induction on the structure of Φ. We proceed by cases, according to the last rule R of Φ. We only show rules `Ax`, `Get` and `Set`. The remaining cases follow easily from the *i.h.* and Lemma 7.75:

- Case $R = \texttt{Ax}$. Then, t is a variable, and we must consider the two following possibilities:
  - If $t = y \neq x$, then Φ must be of the form
  $$\frac{}{y : N; x : \{\!\{\ \}\!\} \vdash y : N}\ \texttt{Ax}$$
  $\Gamma = y : N$, and $M = \{\!\{\ \}\!\}$. Moreover, $t\{x \setminus v\} = y\{x \setminus v\} = y$, and Ψ must be of the form
  $$\frac{}{\emptyset \vdash v : \{\!\{\ \}\!\}}\ \texttt{Ax}$$
  Therefore, we can take $\Pi = \Phi$, since $\#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Psi) = 0 \cdot \texttt{App} + 0 \cdot \texttt{Get} + 0 \cdot \texttt{Set}$, and thus $\#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Pi) = \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Phi) + \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Psi)$.
  - If $t = x$, then Φ must be of the form
  $$\frac{}{x : N \vdash x : N}\ \texttt{Ax}$$
  and $\Gamma = \emptyset$, and $M = N$. Moreover, $t\{x \setminus v\} = x\{x \setminus v\} = v$. Therefore, we can take $\Pi = \Psi$, since $\#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Phi) = 0 \cdot \texttt{App} + 0 \cdot \texttt{Get} + 0 \cdot \texttt{Set}$, and thus $\#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Pi) = \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Phi) + \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Psi)$.
- Case $R = \texttt{Get}$. Then, $t = \texttt{get}(l, \lambda y.u)$, and Φ must be of the form
$$\frac{\Phi' \rhd_{\mathcal{V}_\mathcal{G}} \Gamma; y : \texttt{head}(S\,(l)); x : M \vdash u : S\{l := \texttt{tail}(S\,(l))\} \Rightarrow P}{\Gamma \vdash \texttt{get}(l, \lambda y.u) : S \Rightarrow P}\ \texttt{Get}$$
and $\mathbb{M} = S \Rightarrow P$. By applying the *i.h.* to Φ', there is $\Pi' \rhd_{\mathcal{V}_\mathcal{G}} \Gamma; y : \texttt{head}(S\,(l)) \vdash u\{x \setminus v\} : S\{l := \texttt{tail}(S\,(l))\} \Rightarrow P$, such that $\#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Pi') = \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Phi') + \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Psi)$. Therefore, we can take Π to be
$$\frac{\Pi' \rhd_{\mathcal{V}_\mathcal{G}} \Gamma; y : \texttt{head}(S\,(l)) \vdash u\{x \setminus v\} : S\{l := \texttt{tail}(S\,(l))\} \Rightarrow P}{\Gamma \vdash \texttt{get}(l, \lambda y.u\{x \setminus v\}) : S \Rightarrow P}\ \texttt{Get}$$
since $\texttt{get}(l, \lambda y.u\{x \setminus v\}) = \texttt{get}(l, \lambda y.u)\{x \setminus v\}$, and
$$\begin{aligned}
\#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Pi) &= \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Pi') + 1 \cdot \texttt{Get} \\
&= \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Phi') + \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Psi) + 1 \cdot \texttt{Get} \\
&= \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Phi) + \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Psi).
\end{aligned}$$

- Case $R = \mathtt{Set}$. Then, $t = \mathtt{set}((l,w),r)$, and Φ must be of the form

$$\frac{\Phi_1 \rhd_{\mathcal{V}_\mathcal{G}} \Gamma_1; x : M_1 \vdash w : L \qquad \Phi_2 \rhd_{\mathcal{V}_\mathcal{G}} \Gamma_2; x : M_2 \vdash r : S\{l := L\} \Rightarrow P}{(\Gamma_1 + \Gamma_2); x : M_1 \sqcup M_2 \vdash \mathtt{set}((l,w),r) : S \Rightarrow P}\ \mathtt{Set}$$

  $\Gamma = \Gamma_1 + \Gamma_2$, $M = M_1 \sqcup M_2$, and $\mathbb{M} = S \Rightarrow P$. By Lemma 7.73, there exist $\Psi_1 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash v : M_1$ and $\Psi_2 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash v : M_2$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2)$. By applying the *i.h.* to Φ_1, we know there exists $\Pi_1 \rhd_{\mathcal{V}_\mathcal{G}} \Gamma_1 \vdash w\{x \setminus v\} : L$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi_1) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1)$. By applying the *i.h.* to Φ_2, we know there exists $\Pi_2 \rhd_{\mathcal{V}_\mathcal{G}} \Gamma_2 \vdash r\{x \setminus v\} : S\{l := L\} \Rightarrow P$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi_2) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2)$. Therefore, we can take Π to be

$$\frac{\Pi_1 \rhd_{\mathcal{V}_\mathcal{G}} \Gamma_1 \vdash w\{x \setminus v\} : L \qquad \Pi_2 \rhd_{\mathcal{V}_\mathcal{G}} \Gamma_2 \vdash r\{x \setminus v\} : S\{l := L\} \Rightarrow P}{\Gamma_1 + \Gamma_2 \vdash \mathtt{set}((l,w\{x \setminus v\}),r\{x \setminus v\}) : S \Rightarrow P}\ \mathtt{Set}$$

  since we have that $\mathtt{set}((l,w\{x \setminus v\}),r\{x \setminus v\}) = \mathtt{set}((l,w),r)\{x \setminus v\}$, and

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi) &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi_2) + 1 \cdot \mathtt{Set} \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) \\
&+ \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2) + 1 \cdot \mathtt{Set} \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) + 1 \cdot \mathtt{Set} \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi).
\end{aligned}$$

□

Lemma 7.82 (Exact Subject Reduction for CBV)**.** *Let $t \in \Lambda^{\mathcal{G}}_{\circ}$, $s \in \mathcal{S}_{\circ}$, and $S \in \{\beta_{\mathtt{v}}, \mathtt{get}, \mathtt{set}\}$. If $(t,s) \Rightarrow_{\mathtt{CBV}(S)} (u,q)$ and $\Phi \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash (t,s) : P$, then there is $\Psi \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash (u,q) : P$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) + 1 \cdot R$, where*

- *$R = \mathtt{App}$, if $S = \beta_{\mathtt{v}}$;*
- *$R = \mathtt{Get}$, if $S = \mathtt{get}$;*
- *and $R = \mathtt{Set}$, if $S = \mathtt{set}$.*

Proof. Let $(t,s) = (\mathtt{V}[t_0],s) \Rightarrow_{\mathtt{CBV}(S)} (\mathtt{V}[u_0],q) = (u,q)$, such that $(t_0,s) \mapsto_{\mathtt{CBV}(S)} (u_0,q)$. The proof follows by induction on the definition of $\mathtt{V}$:

- Case $\mathtt{V} = \Box$:

– If $S = \beta_{\mathsf{v}}$, then $(t_0, s) = ((\lambda x.r)\, v, s) \mapsto_{\beta_{\mathsf{v}}} (r\{x \setminus v\}, s) = (u_0, q)$, and Φ must be of the form

$$\dfrac{\dfrac{\dfrac{\Phi_1 \rhd_{\mathcal{V}_\mathcal{G}} x : N \vdash r : S \Rightarrow P}{\emptyset \vdash \lambda x.r : \{\!\{N \to (S \Rightarrow P)\}\!\}}\,\mathtt{Abs}}{\emptyset \vdash \lambda x.r : S \Rightarrow (\{\!\{N \to (S \Rightarrow P)\}\!\} \times S)}\,\mathtt{Lift} \quad \dfrac{\Phi_2 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash v : N}{\emptyset \vdash v : S \Rightarrow (N \times S)}\,\mathtt{Lift}}{\emptyset \vdash (\lambda x.r)\, v : S \Rightarrow P}\,\mathtt{App} \quad \Phi_3 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash s : S}{\emptyset \vdash ((\lambda x.r)\, v, s) : P}\,\mathtt{Conf}$$

according to Lemma 7.75. Given the forms of Φ_1 and Φ_2, we can apply Lemma 7.81 to obtain a derivation $\Pi \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash r\{x \setminus v\} : S \Rightarrow P$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2)$. Therefore, we can take Ψ to be

$$\dfrac{\Pi \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash r\{x \setminus v\} : S \Rightarrow P \qquad \Phi_3 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash s : S}{\emptyset \vdash (r\{x \setminus v\}, s) : P}\,\mathtt{Conf}$$

since

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{App} + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_3) \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi) + 1 \cdot \mathtt{App} + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_3) \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) + 1 \cdot \mathtt{App}.
\end{aligned}$$

– If $S = \mathtt{get}$, then $(t_0, s) = (\mathtt{get}(l, \lambda x.r), s) \Rightarrow_{\mathtt{CBV}(\mathtt{get})} (r\{x \setminus s\,(l)\}, s) = (u_0, q)$, and Φ must be of the form

$$\dfrac{\dfrac{\Phi_1 \rhd_{\mathcal{V}_\mathcal{G}} x : \mathtt{head}(S\,(l)) \vdash r : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P}{\emptyset \vdash \mathtt{get}(l, \lambda x.r) : S \Rightarrow P}\,\mathtt{Get} \quad \Phi_2 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash s : S}{\emptyset \vdash (\mathtt{get}(l, \lambda x.r), s) : P}\,\mathtt{Conf}$$

By applying Lemma 7.79 to Φ_2, there are derivations $\Pi_1 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash s\,(l) : \mathtt{head}(S\,(l))$ and $\Psi_2 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash s : S\{l := \mathtt{tail}(S\,(l))\}$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2)$. By applying Lemma 7.81 to Φ_1 and Π_1, there is $\Psi_1 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash r\{x \setminus s\,(l)\} : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi_1)$. Therefore, we can take Ψ to be

$$\dfrac{\Psi_1 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash r\{x \setminus s\,(l)\} : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P \qquad \Psi_2 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash s : S\{l := \mathtt{tail}(S\,(l))\}}{\emptyset \vdash (r\{x \setminus s\,(l)\}, s) : P}\,\mathtt{Conf}$$

since

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{Get} \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2) + 1 \cdot \mathtt{Get} \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2) + 1 \cdot \mathtt{Get} \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) + 1 \cdot \mathtt{Get}.
\end{aligned}$$

– If $S = \mathtt{set}$, then $(t_0, s) = (\mathtt{set}((l, v), r), s) \Rightarrow_{\mathtt{CBV}(\mathtt{set})} (r, s\{l := v\}) = (u_0, q)$, and Φ must be of the form

$$\dfrac{\dfrac{\Phi_1 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash v : L \;\; \Phi_2 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash r : S\{l := L\} \Rightarrow P \;\; S\,(l) = [\,]}{\emptyset \vdash \mathtt{set}((l, v), r) : S \Rightarrow P}\,\mathtt{Set} \qquad \Phi_3 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash s : S}{\emptyset \vdash (\mathtt{set}((l, v), r), s) : P}\,\mathtt{Conf}$$

By applying Lemma 7.60 to Φ_1 and Φ_3, there is $\Psi_3 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash s\{l := v\} : S\{l := L\}$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_3) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_3)$. Therefore, we can take Ψ to be

$$\dfrac{\Phi_2 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash r : S\{l := L\} \Rightarrow P \qquad \Psi_3 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash s\{l := v\} : S\{l := L\}}{\emptyset \vdash (r, s\{l := v\}) : P}\,\mathtt{Conf}$$

since

$$\begin{aligned} \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{Set} + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_3) \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_3) + 1 \cdot \mathtt{Set} \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) + 1 \cdot \mathtt{Set}. \end{aligned}$$

- Case $\mathtt{V} = \mathtt{V}'\, r$. Then, we have that $(t, s) = (\mathtt{V}'[t_0]\, r, s) \Rightarrow_{\mathtt{CBV}(S)} (\mathtt{V}'[u_0]\, r, q) = (u, q)$, such that $(\mathtt{V}'[t_0], s) \Rightarrow_{\mathtt{CBV}(S)} (\mathtt{V}'[u_0], q)$, and Φ must be of the form

$$\dfrac{\dfrac{\begin{array}{c}\Phi_1 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash \mathtt{V}'[t_0] : S \Rightarrow (\{\!\{N \to (T \Rightarrow P)\}\!\} \times R) \\ \Phi_2 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash r : R \Rightarrow (N \times T)\end{array}}{\emptyset \vdash \mathtt{V}'[t_0]\, r : S \Rightarrow P}\,\mathtt{App} \qquad \Phi_3 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash s : S}{\emptyset \vdash (\mathtt{V}'[t_0]\, r, s) : P}\,\mathtt{Conf}$$

Notice that we can build the following type derivation for $(\mathtt{V}'[t_0], s)$:

$$\dfrac{\Phi_1 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash \mathtt{V}'[t_0] : S \Rightarrow (\{\!\{N \to (T \Rightarrow P)\}\!\} \times R) \qquad \Phi_3 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash s : S}{\emptyset \vdash (\mathtt{V}'[t_0], s) : \{\!\{N \to (T \times P)\}\!\} \Rightarrow R}\,\mathtt{Conf}$$

Therefore, by applying the *i.h.* to $(\mathtt{V}'[t_0], s)$, we know there is a type derivation for $(\mathtt{V}'[u_0], q)$ whose premises are of the form $\Psi_1 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash \mathtt{V}'[u_0] : Q \Rightarrow ((N \to (T \Rightarrow P)) \times R)$ and $\Psi_3 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash q : Q$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_3) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_3) + 1 \cdot R$, where

  – $R = \mathtt{App}$, if $S = \beta_\mathtt{v}$;

  – $R = \mathtt{Get}$, if $S = \mathtt{get}$;

  – and $R = \mathtt{Set}$, if $S = \mathtt{set}$.

Therefore, we can take Ψ to be

$$\dfrac{\dfrac{\Psi_1 \rhd_{\mathcal{V}_\mathcal{G}} \varnothing \vdash \mathtt{V}'[u_0] : Q \Rightarrow (\{\!\{N \to (T \Rightarrow P)\}\!\} \times R) \quad \Phi_2 \rhd_{\mathcal{V}_\mathcal{G}} \varnothing \vdash r : R \Rightarrow (N \times T)}{\varnothing \vdash \mathtt{V}'[u_0]\, r : Q \Rightarrow P}\,\mathtt{App} \quad \Psi_3 \rhd_{\mathcal{V}_\mathcal{G}} \varnothing \vdash q : Q}{\varnothing \vdash (\mathtt{V}'[u_0]\, r, q) : P}\,\mathtt{Conf}$$

since

$$\begin{aligned} \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{App} + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_3) \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + 1 \cdot R + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{App} + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_3) \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) + 1 \cdot R. \end{aligned}$$

- Case $\mathtt{V} = w\,\mathtt{V}'$. Then, $(t, s) = (w\,\mathtt{V}'[t_0], s) \Rightarrow_{\mathtt{CBV}} (w\,\mathtt{V}'[u_0], q) = (u, q)$, such that $(\mathtt{V}[t_0], s) \Rightarrow_{\mathtt{CBV}} (\mathtt{V}[u_0], q)$, and Φ must be of the form

$$\dfrac{\dfrac{\dfrac{\Phi_1 \rhd_{\mathcal{V}_\mathcal{G}} \varnothing \vdash w : \{\!\{N \to (R \Rightarrow P)\}\!\}}{\varnothing \vdash w : S \Rightarrow (\{\!\{N \to (R \Rightarrow P)\}\!\} \times S)}\,\mathtt{Lift} \quad \Phi_2 \rhd_{\mathcal{V}_\mathcal{G}} \varnothing \vdash \mathtt{V}'[t_0] : S \Rightarrow (N \times R)}{\varnothing \vdash w\,\mathtt{V}'[t_0] : S \Rightarrow P}\,\mathtt{App} \quad \Phi_3 \rhd_{\mathcal{V}_\mathcal{G}} \varnothing \vdash s : S}{\varnothing \vdash (w\,\mathtt{V}'[t_0], s) : P}\,\mathtt{Conf}$$

according to Lemma 7.75. Notice that we can build the following type derivation for $(\mathtt{V}[t_0], s)$:

$$\dfrac{\Phi_2 \rhd_{\mathcal{V}_\mathcal{G}} \varnothing \vdash \mathtt{V}'[t_0] : S \Rightarrow (N \times R) \quad \Phi_3 \rhd_{\mathcal{V}_\mathcal{G}} \varnothing \vdash s : S}{\varnothing \vdash (\mathtt{V}[t_0], s) : N \times R}\,\mathtt{Conf}$$

Therefore, by applying the *i.h.* to $(\mathtt{V}'[t_0], s)$, we know there is a type derivation for $(\mathtt{V}[u_0], q)$ whose premises are of the form $\Psi_2 \rhd_{\mathcal{V}_\mathcal{G}} \varnothing \vdash \mathtt{V}'[u_0] : Q \Rightarrow (N \times R)$ and $\Psi_3 \rhd_{\mathcal{V}_\mathcal{G}} \varnothing \vdash q : Q$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_3) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_3) + 1 \cdot R$,

  - $R = \mathtt{App}$, if $S = \beta_\mathsf{v}$;
  - $R = \mathtt{Get}$, if $S = \mathtt{get}$;
  - and $R = \mathtt{Set}$, if $S = \mathtt{set}$.

Therefore, we can take Ψ to be

$$\dfrac{\dfrac{\dfrac{\Phi_1 \rhd_{\mathcal{V}_\mathcal{G}} \varnothing \vdash w : \{\!\{N \to (R \Rightarrow P)\}\!\}}{\varnothing \vdash w : Q \Rightarrow (\{\!\{N \to (R \Rightarrow P)\}\!\} \times Q)}\,\mathtt{Lift} \quad \Psi_2 \rhd_{\mathcal{V}_\mathcal{G}} \varnothing \vdash \mathtt{V}'[u_0] : Q \Rightarrow (N \times R)}{\varnothing \vdash w\,\mathtt{V}'[u_0] : Q \Rightarrow P}\,\mathtt{App} \quad \Psi_3 \rhd_{\mathcal{V}_\mathcal{G}} \varnothing \vdash q : Q}{\varnothing \vdash (w\,\mathtt{V}'[u_0], q) : P}\,\mathtt{Conf}$$

since

$$\begin{aligned}\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1\cdot\mathtt{App} + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_3)\\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + 1\cdot R + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2) + 1\cdot\mathtt{App} + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_3)\\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) + 1\cdot R.\end{aligned}$$

□

Theorem 7.83 (Exact Soundness for CBV). *Let $t \in \Lambda^{\mathcal{G}}_{\circ}$. If there is $\Phi \rhd_{\mathcal{V}_{\mathcal{G}}} \emptyset \vdash (t,s) : P$ tight, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = n\cdot\mathtt{App} + g\cdot\mathtt{Get} + s\cdot\mathtt{Set}$, then $(t,s)^{n\cdot\beta_{\mathtt{v}}+g\cdot\mathtt{get}+s\cdot\mathtt{set}}_{\mathtt{CBV}}\!\!\Rightarrow (u,q)$, such that $u \in \mathtt{NO}_{\mathtt{CBV}}$.*

Proof. Let $\Phi \rhd_{\mathcal{V}_{\mathcal{G}}} \emptyset \vdash (t,s) : P$ be tight, such that $\#_{\mathtt{App},\mathtt{Op}_i}(\Phi) = n\cdot\mathtt{App} + g\cdot\mathtt{Get} + s\cdot\mathtt{Set}$. We proceed by induction on $n+g+s$:

- If $n+g+s = 0$, then $n = g = s = 0$. By Lemma 7.76, $t \in \mathtt{NO}_{\mathtt{CBV}}$. Therefore, we can conclude with $u = t$ and $q = s$.
- If $n+g+s > 0$, then $t \notin \mathtt{NO}_{\mathtt{CBV}}$ by Lemma 7.76. By Lemma 7.18, there are $r \in \Lambda^{\mathcal{G}}_{\circ}$ and $s' \in \mathcal{S}_{\circ}$, such that $(t,s) \Rightarrow_{\mathtt{CBV}(S)} (r,s')$, for some $S \in \{\beta_{\mathtt{v}}, \mathtt{get}, \mathtt{set}\}$. By Lemma 7.82, there is $\Phi' \rhd_{\mathcal{V}_{\mathcal{G}}} \emptyset \vdash (r,s') : P$ tight, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi') + 1\cdot R$, where
  - $R = \mathtt{App}$, if $S = \beta_{\mathtt{v}}$;
  - $R = \mathtt{Get}$, if $S = \mathtt{get}$;
  - and $R = \mathtt{Set}$, if $S = \mathtt{set}$.

  Let us assume, without loss of generality, that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi') = n'\cdot\mathtt{App} + g'\cdot\mathtt{Get} + s'\cdot\mathtt{Set}$. Then, $n'+g'+s'+1 = n+g+s$. Now, by applying the *i.h.*, $(r,s')^{n'\cdot\beta_{\mathtt{v}}+g'\cdot\mathtt{get}+s'\cdot\mathtt{set}}_{\mathtt{CBV}}\!\!\Rightarrow (u',q')$, such that $u' \in \mathtt{NO}_{\mathtt{CBV}}$. Therefore, $(t,s) \Rightarrow_{\mathtt{CBV}(S)} (r,s')^{n'\cdot\beta_{\mathtt{v}}+g'\cdot\mathtt{get}+s'\cdot\mathtt{set}}_{\mathtt{CBV}}\!\!\Rightarrow (u',q')$. Thus, we can take $(u,q) = (u',q')$, since $(t,s)^{n\cdot\beta_{\mathtt{v}}+g\cdot\mathtt{get}+s\cdot\mathtt{set}}_{\mathtt{CBV}}\!\!\Rightarrow (u,q)$.

The argument is identical in structure to the CBN case, relying on exact subject reduction and the characterization of CBV normal forms. □

E.3.2.2 Quantitative Completeness

Lemma 7.84 (Weighted Anti-Substitution for CBV). *Let $t \in \Lambda^{\mathcal{G}}$ and $v \in \mathrm{V}^{\mathcal{G}}_{\circ}$. If $\Phi \rhd_{\mathcal{V}_{\mathcal{G}}} \Gamma \vdash t\{x \setminus v\} : \mathbb{M}$ (resp. N or L), then $\Psi \rhd_{\mathcal{V}_{\mathcal{G}}} \Gamma; x : M \vdash t : \mathbb{M}$ (resp. N or L) and $\Pi \rhd_{\mathcal{V}_{\mathcal{G}}} \emptyset \vdash v : M$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi)$.*

Proof. The proof follows by induction on the structure of t. We only show the cases where t is a variable or an operation. The remaining cases following easily from the *i.h.* and Lemma 7.75:

- Case $t = y \neq x$. Then, $t\{x \setminus v\} = y\{x \setminus v\} = y$. Therefore, we can take $M = \{\!\{\ \}\!\}$, $\Psi = \Phi$, and Π as the following derivation

$$\frac{}{\emptyset \vdash v : \{\!\{\ \}\!\}}\ \texttt{Many}$$

since $\#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Phi) = \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Psi) = \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Psi) + \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Pi)$.

- Case $t = x$. Then, $t\{x \setminus v\} = x\{x \setminus v\} = v$, and, since $\texttt{fv}(v) = \emptyset$, we know that $\Gamma = \emptyset$ by Lemma 7.74. Now, we must consider two cases, according to the type assigned to v in Φ:

  - Let Φ be of the form

$$\frac{\Phi' \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash v : N}{\emptyset \vdash v : S \Rightarrow (N \times S)}\ \texttt{Lift}$$

    Then, we can take $M = N$, $\Pi = \Phi'$ and Ψ to be

$$\frac{\dfrac{}{x : N \vdash x : N}\ \texttt{Ax}}{x : N \vdash x : S \Rightarrow (N \times S)}\ \texttt{Lift}$$

    since $\#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Phi) = \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Pi) = \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Psi) + \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Pi)$.

  - Let $\Phi \rhd_{\mathcal{V}} \emptyset \vdash v : N$. Then, we can take $M = N$, $\Pi = \Phi$, and Ψ to be

$$\frac{}{x : N \vdash x : N}\ \texttt{Ax}$$

    since $\#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Phi) = \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Pi) = \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Psi) + \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Pi)$.

- Case $t = \texttt{get}(l, \lambda y.u)$. Then, $t\{x \setminus v\} = \texttt{get}(l, \lambda y.u\{x \setminus v\})$, and Φ must be of the form:

$$\frac{\Phi' \rhd_{\mathcal{V}_\mathcal{G}} \Gamma; y : \texttt{head}(S\,(l)) \vdash u\{x \setminus v\} : S\{l := \texttt{tail}(S\,(l))\} \Rightarrow P}{\Gamma \vdash \texttt{get}(l, \lambda y.u\{x \setminus v\}) : S \Rightarrow P}\ \texttt{Get}$$

  and $\mathbb{M} = S \Rightarrow P$. By applying the *i.h.* to u, we know $\Psi' \rhd_{\mathcal{V}_\mathcal{G}} \Gamma; y : \texttt{head}(S\,(l)); x : M \vdash u : S\{l := \texttt{tail}(S\,(l))\} \Rightarrow P$ exists and $\Pi' \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash v : M$, such that $\#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Phi') = \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Psi') + \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Pi')$. Therefore, we can conclude by taking $\Pi = \Pi'$ and Ψ to be

$$\frac{\Psi' \rhd_{\mathcal{V}_\mathcal{G}} \Gamma; y : \texttt{head}(S\,(l)); x : M \vdash u : S\{l := \texttt{tail}(S\,(l))\} \Rightarrow P}{\Gamma; x : M \vdash \texttt{get}(l, \lambda y.u) : S \Rightarrow P}\ \texttt{Get}$$

  since

$$\begin{aligned}\#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Phi) &= \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Phi') + 1 \cdot \texttt{Get} \\ &= \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Psi') + \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Pi') + 1 \cdot \texttt{Get} \\ &= \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Psi) + \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Pi).\end{aligned}$$

- Case $t = \texttt{set}((l, w), u)$. Then, $t\{x \setminus v\} = \texttt{set}((l, w\{x \setminus v\}), u\{x \setminus v\})$, and Φ must be of the form

$$\frac{\Phi_1 \rhd_{\mathcal{V}_\mathcal{G}} \Gamma_1 \vdash w\{x \setminus v\} : L \qquad \Phi_2 \rhd_{\mathcal{V}_\mathcal{G}} \Gamma_2 \vdash u\{x \setminus v\} : S\{l := L\} \Rightarrow P}{\Gamma_1 + \Gamma_2 \vdash \texttt{set}((l, w\{x \setminus v\}), u\{x \setminus v\}) : S \Rightarrow P}\ \texttt{Set}$$

$\Gamma = \Gamma_1 + \Gamma_2$, and $\mathbb{M} = S \Rightarrow P$. By applying the *i.h.* to w, we know there exist $\Psi_1 \triangleright_{\mathcal{V}_\mathcal{G}} \Gamma_1; x : M_1 \vdash w : L$ and $\Pi_1 \triangleright_{\mathcal{V}_\mathcal{G}} \varnothing \vdash v : M_1$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi_1)$. By applying the *i.h.* to u, we know there exist $\Psi_2 \triangleright_{\mathcal{V}_\mathcal{G}} \Gamma_2; x : M_2 \vdash u : S\{l := L\} \Rightarrow P$ and $\Pi_2 \triangleright_{\mathcal{V}_\mathcal{G}} \varnothing \vdash v : M_2$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi_2)$. By Lemma 7.73, there exists a type derivation $\Pi' \triangleright_{\mathcal{V}_\mathcal{G}} \varnothing \vdash v : M_1 \sqcup M_2$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi') = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi_2)$. Therefore, we can conclude by taking $M = M_1 \sqcup M_2$, $\Pi = \Pi'$ and Ψ to be

$$\frac{\Psi_1 \triangleright_{\mathcal{V}_\mathcal{G}} \Gamma_1; x : M_1 \vdash w : L \qquad \Psi_2 \triangleright_{\mathcal{V}_\mathcal{G}} \Gamma_2; x : M_2 \vdash u : S\{l := L\} \Rightarrow P}{(\Gamma_1 + \Gamma_2); x : M_1 \sqcup M_2 \vdash \mathtt{set}((l, w), u) : S \Rightarrow P}\ \mathtt{Set}$$

since

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{Set} \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi_1) \\
&+ \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi_2) + 1 \cdot \mathtt{Set} \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi') + 1 \cdot \mathtt{Set} \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi).
\end{aligned}$$

□

Lemma 7.85 (Exact Subject Expansion for CBV). *Let $t, u \in \Lambda^{\mathcal{G}}_{\circ}$, $s \in \mathcal{S}_{\circ}$, and $S \in \{\beta_{\mathtt{v}}, \mathtt{get}, \mathtt{set}\}$. If $(t, s) \Rightarrow_{\mathtt{CBV}(S)} (u, q)$ and $\Phi \triangleright_{\mathcal{V}_\mathcal{G}} \varnothing \vdash (u, q) : P$, then there is $\Psi \triangleright_{\mathcal{V}_\mathcal{G}} \varnothing \vdash (t, s) : P$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) + 1 \cdot R = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi)$, where*

- $R = \mathtt{App}$, *if* $S = \beta_{\mathtt{v}}$;
- $R = \mathtt{Get}$, *if* $S = \mathtt{get}$;
- *and* $R = \mathtt{Set}$, *if* $S = \mathtt{set}$.

Proof. Let $(t, s) = (\mathtt{V}[t_0], s) \Rightarrow_{\mathtt{CBV}(S)} (\mathtt{V}[u_0], q) = (u, q)$, such that $(t_0, s) \mapsto_{\mathtt{CBV}(S)} (u_0, q)$. The proof follows by induction on the definition of $\mathtt{V}$:

- Let $\mathtt{V} = \Box$:
  - If $S = \beta_{\mathtt{v}}$, then $(t_0, s) = ((\lambda x.r)\, v, s) \mapsto_{\beta_{\mathtt{v}}} (r\{x \setminus v\}, s) = (u_0, q)$, and Φ is of the form

$$\frac{\Phi_1 \triangleright_{\mathcal{V}_\mathcal{G}} \varnothing \vdash r\{x \setminus v\} : S \Rightarrow P \qquad \Phi_2 \triangleright_{\mathcal{V}_\mathcal{G}} \varnothing \vdash s : S}{\varnothing \vdash (r\{x \setminus v\}, s) : P}\ \mathtt{Conf}$$

    Given the form of Φ_1, we can apply Lemma 7.84 to obtain derivations $\Psi_1 \triangleright_{\mathcal{V}_\mathcal{G}} x : M \vdash r : S \Rightarrow P$ and $\Psi_2 \triangleright_{\mathcal{V}_\mathcal{G}} \varnothing \vdash v : M$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) +$

$\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2)$. Therefore, we can take Ψ to be

$$\dfrac{\dfrac{\dfrac{\Psi_1 \rhd_{\mathcal{V}_\mathcal{G}} x : M \vdash r : S \Rightarrow P}{\varnothing \vdash \lambda x.r : \{\!\{M \to (S \Rightarrow P)\}\!\}}\,\mathtt{Abs}}{\varnothing \vdash \lambda x.r : S \Rightarrow (\{\!\{M \to (S \Rightarrow P)\}\!\} \times S)}\,\mathtt{Lift} \quad \dfrac{\Psi_2 \rhd_{\mathcal{V}_\mathcal{G}} \varnothing \vdash v : M}{\varnothing \vdash v : S \Rightarrow (M \times S)}\,\mathtt{Lift}}{\dfrac{\varnothing \vdash (\lambda x.r)\, v : S \Rightarrow P \qquad \Phi_2 \rhd_{\mathcal{V}_\mathcal{G}} \varnothing \vdash s : S}{\varnothing \vdash ((\lambda x.r)\, v, s) : P}\,\mathtt{Conf}}\,\mathtt{App}$$

since

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) + 1 \cdot \mathtt{App} &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{App} \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2) \\
&+ \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{App} \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi).
\end{aligned}$$

– If $S = \mathtt{get}$, then $(t_0, s) = (\mathtt{get}(l, \lambda x.r), s) \Rightarrow_{\mathtt{CBV}(\mathtt{get})} (r\{x \setminus s\,(l)\}, s) = (u_0, q)$, and Φ is of the form

$$\dfrac{\Phi_1 \rhd_{\mathcal{V}_\mathcal{G}} \varnothing \vdash r\{x \setminus s\,(l)\} : S \Rightarrow P \qquad \Phi_2 \rhd_{\mathcal{V}_\mathcal{G}} \varnothing \vdash s : S}{\varnothing \vdash (r\{x \setminus s\,(l)\}, s) : P}\,\mathtt{Conf}$$

By applying Lemma 7.84 to Φ_1, there exist $\Psi_1 \rhd_{\mathcal{V}_\mathcal{G}} x : M \vdash r : S \Rightarrow P$ and $\Pi \rhd_{\mathcal{V}_\mathcal{G}} \varnothing \vdash s\,(l) : M$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi)$. Let $R = M \cdot S\,(l)$. Then, $\mathtt{head}(R\,(l)) = M$, $\mathtt{tail}(R\,(l)) = S\,(l) = L$, and $R\{l := \mathtt{tail}(R\,(l))\} = S$. Thus, by applying Lemma 7.79 to Π and Φ_2, there is a type derivation $\Psi_2 \rhd_{\mathcal{V}_\mathcal{G}} \varnothing \vdash s : R$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi)$. Therefore, we can take Ψ to be

$$\dfrac{\dfrac{\Psi_1 \rhd_{\mathcal{V}_\mathcal{G}} x : \mathtt{head}(R\,(l)) \vdash r : R\{l := \mathtt{tail}(R\,(l))\} \Rightarrow P}{\varnothing \vdash \mathtt{get}(l, \lambda x.r) : R \Rightarrow P}\,\mathtt{Get} \qquad \Psi_2 \rhd_{\mathcal{V}_\mathcal{G}} \varnothing \vdash s : R}{\varnothing \vdash (\mathtt{get}(l, \lambda x.r), s) : P}\,\mathtt{Conf}$$

since

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) + 1 \cdot \mathtt{Get} &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{Get} \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Pi) \\
&+ \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{Get} \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2) + 1 \cdot \mathtt{Get} \\
&= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi).
\end{aligned}$$

– If $S = \mathtt{set}$, then $(t_0, s) = (\mathtt{set}((l, v), r), s) \Rightarrow_{\mathtt{CBV}(\mathtt{set})} (r, s\{l := v\}) = (u_0, q)$, and Φ is of the form

$$\dfrac{\Phi_1 \rhd_{\mathcal{V}_\mathcal{G}} \varnothing \vdash r : S \Rightarrow P \qquad \Phi_2 \rhd_{\mathcal{V}_\mathcal{G}} \varnothing \vdash s\{l := v\} : S}{\varnothing \vdash (r, s\{l := v\}) : P}\,\mathtt{Conf}$$

Let $R = S\{l := [\,]\}$, and $R\{l := S\,(l)\} = R\{l := L\} = S$. Then, by applying Lemma 7.80

to Φ_2, there are $\Psi_2 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash s : R$ and $\Psi_1 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash v : L$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2)$. Then, we can take Ψ to be

$$\dfrac{\dfrac{\Psi_1 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash v : L}{\Phi_1 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash r : R\{l := L\} \Rightarrow P \qquad R\,(l) = [\,]}\,\mathtt{Set}}{\emptyset \vdash \mathtt{set}((l,v),r) : R \Rightarrow P}\qquad \Psi_2 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash s : R}{\emptyset \vdash (\mathtt{set}((l,v),r),s) : P}\,\mathtt{Conf}$$

since

$$\begin{aligned}\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) + 1 \cdot \mathtt{Set} &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{Set}\\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1)\\ &\quad\ \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2) + 1 \cdot \mathtt{Set}\\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi).\end{aligned}$$

- Case $\mathtt{V} = \mathtt{V}'\,r$. Then, we have that $(t,s) = (\mathtt{V}'[t_0]\,r, s) \Rightarrow_{\mathtt{CBV}(S)} (\mathtt{V}'[u_0]\,r, q) = (u,q)$, such that $(\mathtt{V}'[t_0], s) \Rightarrow_{\mathtt{CBV}(S)} (\mathtt{V}'[u_0], q)$, and Φ must be of the form

$$\dfrac{\dfrac{\Phi_1 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash \mathtt{V}'[u_0] : Q \Rightarrow (\{\!\{N \to (T \Rightarrow P)\}\!\} \times R)}{\emptyset \vdash \mathtt{V}'[u_0]\,r : Q \Rightarrow P}\,\mathtt{App}\qquad \Phi_2 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash r : R \Rightarrow (N \times T)}{\emptyset \vdash (\mathtt{V}'[u_0]\,r, q) : P}\,\mathtt{Conf}\qquad \Phi_3 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash q : Q$$

Notice that we can build the following type derivation for $(\mathtt{V}'[u_0], q)$:

$$\dfrac{\Phi_1 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash \mathtt{V}'[u_0] : Q \Rightarrow (\{\!\{N \to (T \Rightarrow P)\}\!\} \times R) \qquad \Phi_3 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash q : Q}{\emptyset \vdash (\mathtt{V}'[u_0], q) : \{\!\{N \to (T \Rightarrow P)\}\!\} \times R}\,\mathtt{Conf}$$

Therefore, by applying the *i.h.* to $(\mathtt{V}'[u_0], q)$, we know there is a type derivation for $(\mathtt{V}'[t_0], s)$ whose premises are of the form $\Psi_1 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash \mathtt{V}'[t_0] : S \Rightarrow (\{\!\{N \to (T \Rightarrow P)\}\!\} \times R)$ and $\Psi_3 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash s : S$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_3) + 1 \cdot R = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_3)$, where

  - $R = \mathtt{App}$, if $S = \beta_\mathtt{v}$;
  - $R = \mathtt{Get}$, if $S = \mathtt{get}$;
  - and $R = \mathtt{Set}$, if $S = \mathtt{set}$.

Therefore, we can take Ψ to be

$$\dfrac{\dfrac{\Psi_1 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash \mathtt{V}'[t_0] : S \Rightarrow (\{\!\{N \to (T \Rightarrow P)\}\!\} \times R)}{\emptyset \vdash \mathtt{V}'[t_0]\,r : S \Rightarrow P}\,\mathtt{App}\qquad \Phi_2 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash r : R \Rightarrow (N \times T)}{\emptyset \vdash (\mathtt{V}'[t_0]\,r, s) : P}\,\mathtt{Conf}\qquad \Psi_3 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash s : S$$

since

$$\begin{aligned}\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) + 1 \cdot R &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{App} \\ &+ \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_3) + 1 \cdot R \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{App} + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_3) \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi).\end{aligned}$$

- Case $\mathtt{V} = w\,\mathtt{V}'$. Then, we have that $(t, s) = (w\,\mathtt{V}'[t_0], s) \Rightarrow_{\mathtt{CBV}(S)} (w\,\mathtt{V}'[u_0], q) = (u, q)$, such that $(\mathtt{V}[t_0], s) \Rightarrow_{\mathtt{CBV}(S)} (\mathtt{V}[u_0], q)$, and Φ must be of the form

$$\dfrac{\dfrac{\dfrac{\Phi_1 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash w : \{\!\{N \to (T \Rightarrow P)\}\!\}}{\emptyset \vdash w : Q \Rightarrow (\{\!\{N \to (T \Rightarrow P)\}\!\} \times Q)}\,\mathtt{Lift} \qquad \Phi_2 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash \mathtt{V}'[u_0] : Q \Rightarrow (N \times T)}{\emptyset \vdash w\,\mathtt{V}'[u_0] : Q \Rightarrow P}\,\mathtt{App} \qquad \Phi_3 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash q : Q}{\emptyset \vdash (w\,\mathtt{V}'[u_0], q) : P}\,\mathtt{Conf}$$

Notice that we can build the following type derivation for $(\mathtt{V}[u_0], q)$:

$$\dfrac{\Phi_2 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash \mathtt{V}'[u_0] : Q \Rightarrow (N \times T) \qquad \Phi_3 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash q : Q}{\emptyset \vdash (\mathtt{V}[u_0], q) : N \times T}\,\mathtt{Conf}$$

Therefore, by applying the *i.h.* to $\mathtt{V}'[u_0]$, we know there is a type derivation for $(\mathtt{V}[t_0], s)$ whose premises are of the form $\Psi_2 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash \mathtt{V}'[t_0] : S \Rightarrow (N \times T)$ and $\Psi_3 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash s : S$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_3) + 1 \cdot R = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_3)$, where

  - $R = \mathtt{App}$, if $S = \beta_\mathtt{v}$;
  - $R = \mathtt{Get}$, if $S = \mathtt{get}$;
  - and $R = \mathtt{Set}$, if $S = \mathtt{set}$.

Therefore, we can take Ψ to be

$$\dfrac{\dfrac{\dfrac{\Phi_1 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash w : \{\!\{N \to (T \Rightarrow P)\}\!\}}{\emptyset \vdash w : S \Rightarrow (\{\!\{N \to (T \Rightarrow P)\}\!\} \times S)}\,\mathtt{Lift} \qquad \Psi_2 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash \mathtt{V}'[t_0] : S \Rightarrow (N \times T)}{\emptyset \vdash w\,\mathtt{V}'[t_0] : S \Rightarrow P}\,\mathtt{App} \qquad \Psi_3 \rhd_{\mathcal{V}_\mathcal{G}} \emptyset \vdash s : S}{\emptyset \vdash (w\,\mathtt{V}'[t_0], s) : P}\,\mathtt{Conf}$$

since

$$\begin{aligned}\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) + 1 \cdot R &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{App} \\ &+ \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_3) + 1 \cdot R \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2) + 1 \cdot \mathtt{App} + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_3) \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi).\end{aligned}$$

□

Theorem 7.86 (Exact Completeness for CBV). *Let $t \in \Lambda^{\mathcal{G}}_{\circ}$. If $(t,s) \overset{n\cdot\beta_{\mathtt{v}}+g\cdot\mathtt{get}+s\cdot\mathtt{set}}{\Rightarrow}_{\mathtt{CBV}} (u,q)$, such that $u \in \mathtt{NO}_{\mathtt{CBV}}$, then there is $\Phi \rhd_{\mathcal{V}_{\mathcal{G}}} \emptyset \vdash (t,s) : P$ tight, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = n \cdot \mathtt{App} + g \cdot \mathtt{Get} + s \cdot \mathtt{Set}$.*

Proof. Let $(t,s) \overset{n\cdot\beta_{\mathtt{v}}+g\cdot\mathtt{get}+s\cdot\mathtt{set}}{\Rightarrow}_{\mathtt{CBV}} (u,q)$, such that $u \in \mathtt{NO}_{\mathtt{CBV}}$. We proceed by induction on $n+s+g$:

- If $n+g+s=0$, then $n=g=s=0$. Therefore, $t \in \mathtt{NO}_{\mathtt{CBV}}$, by Lemma 7.18. Thus, we can conclude by Lemma 7.77.
- If $n+g+s>0$, then $t \notin \mathtt{NO}_{\mathtt{CBV}}$ by Lemma 7.18. Therefore, there are $r \in \Lambda^{\mathcal{G}}_{\circ}$ and $s' \in \mathcal{S}_{\circ}$, such that $(t,s) \Rightarrow_{\mathtt{CBV}(S)} (r,s') \overset{n'\cdot\beta_{\mathtt{v}}+g'\cdot\mathtt{get}+s'\cdot\mathtt{set}}{\Rightarrow}_{\mathtt{CBV}} (u,q)$, and $n'+g'+s'+1 = n+g+s$. Now, by applying the *i.h.*, there is $\Phi' \rhd_{\mathcal{V}_{\mathcal{G}}} \emptyset \vdash (r,s') : P$ tight, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi') = n' \cdot \mathtt{App} + g' \cdot \mathtt{Get} + s' \cdot \mathtt{Set}$. By Lemma 7.85, there is $\Phi' \rhd_{\mathcal{V}_{\mathcal{G}}} \emptyset \vdash (t,s) : P$ tight, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi') + 1 \cdot R = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi)$, where
  - $R = \mathtt{App}$, if $S = \beta_{\mathtt{v}}$;
  - $R = \mathtt{Get}$, if $S = \mathtt{get}$;
  - and $R = \mathtt{Set}$, if $S = \mathtt{set}$.

  Therefore, we can take $\Phi = \Phi'$, since

$$\begin{aligned}\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi') + 1 \cdot R \\ &= n' \cdot \mathtt{App} + g' \cdot \mathtt{Get} + s' \cdot \mathtt{Set} + 1 \cdot R \\ &= n \cdot \mathtt{App} + g \cdot \mathtt{Get} + s \cdot \mathtt{Set}.\end{aligned}$$

□

E.3.3 Subsuming Call-By-Name and Call-By-Value

Lemma 7.93 (CBPV Multiset-Typings Split and Merge). *Let $t \in !\Lambda^{\mathcal{G}}$. Then, there is $\Phi \rhd_{\mathcal{P}_{\mathcal{G}}} \Gamma \vdash t : M$, such that $M = \sqcup_{i\in I} M_i$, if and only if there are $\Phi_i \rhd_{\mathcal{P}_{\mathcal{G}}} \Gamma_i \vdash t : M_i$, such that $\Gamma = +_{i\in I}\Gamma_i$, and $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) = +_{i\in I}\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi_i)$.*

Proof. The proof is by a straightforward case analysis on the structure of CBPV terms. For value terms, the statement follows directly from the CBV multiset split and merge lemma; for computation terms, it follows from the corresponding CBN lemma. Since CBPV typing rules combine contexts, multisets, and weights additively and componentwise, the result follows immediately. □

Lemma 7.94 (Contexts and Free Variables for CBPV). *If $\Phi \rhd_{\mathcal{P}_{\mathcal{G}}} \Gamma \vdash t : A$ (resp. M), then $\mathtt{dom}(\Gamma) \subseteq \mathtt{fv}(t)$.*

Proof. The proof follows by a straightforward induction on the structure of the typing derivation. Since the CBPV type system does not admit weakening and all typing rules propagate contexts only through variable occurrences, every variable in the typing context must occur free in the term. □

Lemma 7.95 (CBPV Values Are Not Effectful)**.** *Let* $t \in !\Lambda^{\mathcal{G}}$ *and* $\Phi \rhd_{\mathcal{P}_{\mathcal{G}}} \varnothing \vdash t : S \Rightarrow (A \times Q)$*. If* $t \in !\mathrm{V}^{\mathcal{G}}$*, then* $Q = S$ *and* Φ *ends with rule* `Lift`*.*

Proof. Let $\Phi \rhd_{\mathcal{P}_{\mathcal{G}}} \varnothing \vdash t : S \Rightarrow (A \times Q)$ and assume $t \in !\mathrm{V}^{\mathcal{G}}$. We proceed by inspection of the last rule R of Φ. Since t is a value, the last rule of the derivation cannot be `App`, `Der`, `Get`, or `Set`, as these rules type applications, derelictions, or effectful operation. Therefore, the only possible last rule is `Lift`. By the definition of `Lift`, the input and output state types coincide, hence $Q = S$. □

Lemma 7.96. *Let* $t \in !\Lambda^{\mathcal{G}}_{\circ}$ *and* $s \in \mathcal{S}_{\circ}$*. If* $\Phi \rhd_{\mathcal{P}_{\mathcal{G}}} \varnothing \vdash (t,s) : P$ *is tight, then* $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) = 0 \cdot \mathtt{App} + 0 \cdot \mathtt{Der} + 0 \cdot \mathtt{Get} + 0 \cdot \mathtt{Set}$ *if and only if* $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$*.*

Proof. We start by noticing that, if $\Phi \rhd_{\mathcal{P}_{\mathcal{G}}} \varnothing \vdash (t,s) : P$ is tight, then Φ must be of the form

$$\dfrac{\Phi' \rhd_{\mathcal{P}_{\mathcal{G}}} \varnothing \vdash t : S \Rightarrow (A \times \{l \mapsto [\,]\}_{l \in \mathcal{L}}) \qquad \Phi'' \rhd_{\mathcal{P}_{\mathcal{G}}} \varnothing \vdash s : S}{\varnothing \vdash (t,s) : A \times \{l \mapsto [\,]\}_{l \in \mathcal{L}}}\ \mathtt{Conf}$$

$P = A \times \{l \mapsto [\,]\}_{l \in \mathcal{L}}$, and $A = \{\!\{\,\}\!\}$ or $A = \star$.

($\Rightarrow$) Let $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) = 0 \cdot \mathtt{App} + 0 \cdot \mathtt{Der} + 0 \cdot \mathtt{Get} + 0 \cdot \mathtt{Set}$. We proceed by cases, according to the last rule R of Φ':

- Case $R = \mathtt{Ax}$. Then, Φ' does not conclude with a monadic type. Therefore, this case does not apply.
- Case $R = \mathtt{Abs}$. Then, A is an arrow type. Thus, this lead to a contradiction with $A = \{\!\{\,\}\!\}$ or $A = \star$. Therefore, this case does not apply.
- Case $R = \mathtt{Ans}$. Then, t is an closed abstraction. Therefore, we can immediately conclude that $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$.
- Case $R = \mathtt{App}$. This lead to a contradiction with $\#_{\mathtt{App}}(\Phi') = 0 \cdot \mathtt{App}$. Therefore, this case does not apply.
- Case $R = \mathtt{Bg}$. Then, Φ' does not conclude with a monadic type. Therefore, this case does not apply.
- Case $R = \mathtt{Lift}$. Then, t is a closed value, and Φ must be of the form

$$\dfrac{\dfrac{\dfrac{}{\varnothing \vdash t : \{\!\{\,\}\!\}}\,\mathtt{Bg}}{\varnothing \vdash t : S \Rightarrow (\{\!\{\,\}\!\} \times S)}\,\mathtt{Lift} \qquad \dfrac{\left(\dfrac{}{\varnothing \vdash s\,(l) : [\,]}\,\mathtt{EList}\right)_{l \in \mathcal{L}}}{\varnothing \vdash \{l \mapsto s\,(l)\}_{l \in \mathcal{L}} : \{l \mapsto [\,]\}_{l \in \mathcal{L}}}\,\mathtt{State}}{\varnothing \vdash (t,s) : \{\!\{\,\}\!\} \times \{l \mapsto [\,]\}_{l \in \mathcal{L}}}\,\mathtt{Conf}$$

and $S = \{l \mapsto [\,]\}_{l \in \mathcal{L}}$. Therefore, we can immediately conclude that $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$.

- Case $R = \mathtt{Der}$. This leads to a contradiction with $\#_{\mathtt{Der}}(\Phi') = 0 \cdot \mathtt{Der}$. Therefore, this case does not apply.
- Case $R = \mathtt{Get}$. This leads to a contradiction with $\#_{\mathtt{Get}}(\Phi') = 0 \cdot \mathtt{Get}$. Therefore, this case does not apply.
- Case $R = \mathtt{Set}$. This leads to a contradiction with $\#_{\mathtt{Set}}(\Phi') = 0 \cdot \mathtt{Set}$. Therefore, this case does not apply.

Therefore, Φ' must end either with rule `Ans`, or with rule `Lift` whose premise ends with `Bg`. In the first case, t is a closed abstraction; in the second case, t is a closed bang-term. In both cases, we conclude that $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$.

($\Leftarrow$) Let $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$. Then, $t = \lambda x.u$ or $t = !u$. Therefore, Φ' must be of the form

$$\frac{}{\varnothing \vdash \lambda x.u : S \Rightarrow (\star \times S)}\ \mathtt{Ans} \quad \text{or} \quad \frac{\dfrac{}{\varnothing \vdash !u : \{\!\{\,\}\!\}}\ \mathtt{Bg}}{\varnothing \vdash !u : S \Rightarrow (\{\!\{\,\}\!\} \times S)}\ \mathtt{Lift}$$

$S = \{l \mapsto [\,]\}_{l \in \mathcal{L}}$, and Φ'' must be of the form

$$\frac{\left(\dfrac{}{\varnothing \vdash s\,(l) : [\,]}\ \mathtt{EList}\right)_{l \in \mathcal{L}}}{\varnothing \vdash \{l \mapsto s\,(l)\}_{l \in \mathcal{L}} : \{l \mapsto [\,]\}_{l \in \mathcal{L}}}\ \mathtt{State}$$

Thus, we can conclude that

$$\begin{aligned} \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) &= \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi') + \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi'') \\ &= 0 \cdot \mathtt{App} + 0 \cdot \mathtt{Der} + 0 \cdot \mathtt{Get} + 0 \cdot \mathtt{Set}. \end{aligned}$$

□

Lemma 7.97 (Tight Typability of Clash-Free CBPV-Normal Forms)**.** *Let $t \in !\Lambda^{\mathcal{G}}_{\circ}$. If $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$, then there is a derivation $\Phi \rhd_{\mathcal{P}_{\mathcal{G}}} \varnothing \vdash (t,s) : P$ that is tight.*

Proof. If $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$, then $t = \lambda x.u$ or $t = !u$. Let $P = A \times S$ and $S = \{l \mapsto [\,]\}_{l \in \mathcal{L}}$. Moreover, let Φ' be

$$\frac{}{\varnothing \vdash \lambda x.u : S \Rightarrow (\star \times S)}\ \mathtt{Ans} \quad \text{or} \quad \frac{\dfrac{}{\varnothing \vdash !u : \{\!\{\,\}\!\}}\ \mathtt{Bg}}{\varnothing \vdash !u : S \Rightarrow (\{\!\{\,\}\!\} \times S)}\ \mathtt{Lift}$$

and Φ'' be

$$\frac{\left(\dfrac{}{\varnothing \vdash s\,(l) : [\,]}\ \mathtt{EList}\right)_{l \in \mathcal{L}}}{\varnothing \vdash \{l \mapsto s\,(l)\}_{l \in \mathcal{L}} : \{l \mapsto [\,]\}_{l \in \mathcal{L}}}\ \mathtt{State}$$

Then, we can take Φ to be

$$\frac{\Phi' \rhd_{\mathcal{P}_{\mathcal{G}}} \varnothing \vdash t : S \Rightarrow (A \times S) \qquad \Phi'' \rhd_{\mathcal{P}_{\mathcal{G}}} \varnothing \vdash s : \{l \mapsto [\,]\}_{l \in \mathcal{L}}}{\varnothing \vdash (t,s) : A \times \{l \mapsto [\,]\}_{l \in \mathcal{L}}}\ \mathtt{Conf}$$

where $A = \star$ or $A = \{\!\{\ \}\!\}$, according to t. Then, we can conclude since Φ is tight. □

Lemma 7.98. *Let $l \in \mathcal{L}$ and $s \in \mathcal{S}$. There is $\Phi \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash s : S$, such that $S(l) \neq [\,]$, if and only if there are $\Psi \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash s(l) : \mathtt{head}(S(l))$ and $\Pi \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash s : S\{l := \mathtt{tail}(S(l))\}$. Moreover, $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Pi)$.*

Proof. The proof is identical to the CBV case (Lemma 7.79), since CBPV state typings are built using the same list-typing and state rules, and none of the CBPV-specific rules (`Der`, `Bg`, `Ans`) apply to states. □

Lemma 7.99. *Let $l \in \mathcal{L}$ and $s \in \mathcal{S}$. There are $\Phi \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash s : S$, such that $S(l) = [\,]$, and $\Psi \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash v : L$ if and only if there is $\Pi \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash s\{l := v\} : S\{l := L\}$. Moreover, $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi) = \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Pi)$.*

Proof. The proof is identical to the CBV case (Lemma 7.80), since CBPV state typings are built using the same list and state rules, and CBPV-specific rules (`Der`, `Bg`, `Ans`) do not apply to states. □

E.3.3.1 Quantitative Soundness

Lemma 7.100 (Weighted Substitution for CBPV)**.** *Let $t \in\ !\Lambda^{\mathcal{G}}$ and $v \in\ !\mathrm{V}^{\mathcal{G}}$. If $\Phi \rhd_{\mathcal{P}_\mathcal{G}} \Gamma; x : M \vdash t : \mathbb{A}$ (resp. N or L) and $\Psi \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash v : M$, then $\Pi \rhd_{\mathcal{P}_\mathcal{G}} \Gamma \vdash t\{x \setminus v\} : \mathbb{A}$ (resp. N or L), such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Pi) = \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi)$.*

Proof. The proof follows by induction on the structure of Φ. We proceed by cases, according to the last rule R in Φ. We omit `Abs`, `Ans`, `Get` and `Set`, since these cases are very similar to the corresponding ones for the CBN $\lambda^{\mathcal{G}}$-calculus (see Lemma 7.61), and `Ax`, `App`, and `Lift`, since these cases are very similar to the corresponding ones for the CBV $\lambda^{\mathcal{G}}$-calculus (see Lemma 7.81):

- Case $R = \mathtt{Bg}$. Then, $t = !u$, $t\{x \setminus v\} = !u\{x \setminus v\} = !(u\{x \setminus v\})$, and Φ must be of the form

$$\frac{(\Phi_i \rhd_{\mathcal{P}_\mathcal{G}} \Gamma_i; x : M_i \vdash u : \mathbb{A}_i)_{i \in \mathtt{I}}}{+_{i \in \mathtt{I}} \Gamma_i; x : \sqcup_{i \in \mathtt{I}} M_i \vdash\ !u : \{\!\{\mathbb{A}_i\}\!\}_{i \in \mathtt{I}}}\ \mathtt{Bg}$$

  $\Gamma = +_{i \in \mathtt{I}} \Gamma_i$, and $N = \sqcup_{i \in \mathtt{I}} M_i$. By Lemma 7.93, since $M = \sqcup_{i \in \mathtt{I}} M_i$, we know there are $\Psi_i \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash v : M_i$, for each $i \in \mathtt{I}$, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi) = +_{i \in \mathtt{I}} \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi_i)$. By applying the *i.h.* to each Φ_i, we know there are $\Pi_i \rhd_{\mathcal{P}_\mathcal{G}} \Gamma_i \vdash u\{x \setminus v\} : \mathbb{A}_i$, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Pi_i) = \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi_i) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi_i)$, for each $i \in \mathtt{I}$. Notice that $u\{x \setminus v\}$ is a value. Thus, we can take Π to be

$$\frac{(\Pi_i \rhd_{\mathcal{P}_\mathcal{G}} \Gamma_i \vdash u\{x \setminus v\} : \mathbb{A}_i)_{i \in \mathtt{I}}}{+_{i \in \mathtt{I}} \Gamma_i \vdash\ !(u\{x \setminus v\}) : \{\!\{\mathbb{A}_i\}\!\}_{i \in \mathtt{I}}}\ \mathtt{Bg}$$

since we have

$$\begin{aligned}\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Pi) &= +_{i\in \mathtt{I}}\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Pi_i)\\ &= +_{i\in \mathtt{I}}\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi_i) + +_{i\in \mathtt{I}}\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi_i)\\ &= \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi).\end{aligned}$$

- Case $R = \mathtt{Der}$. Then, $t = \mathtt{der}(u)$, $t\{x \setminus v\} = \mathtt{der}(u)\{x \setminus v\} = \mathtt{der}(u\{x \setminus v\})$, and Φ must be of the form
$$\frac{\Phi' \rhd_{\mathcal{P}_{\mathcal{G}}} \Gamma; x : M \vdash u : S \Rightarrow (\{\!\{R \Rightarrow P\}\!\} \times R)}{\Gamma; x : M \vdash \mathtt{der}(u) : S \Rightarrow P}\ \mathtt{Der}$$
and $\mathbb{A} = S \Rightarrow P$. By applying the *i.h.* to Φ', we know there is $\Pi' \rhd_{\mathcal{P}_{\mathcal{G}}} \Gamma \vdash u\{x \setminus v\} : S \Rightarrow (\{\!\{R \Rightarrow P\}\!\} \times R)$, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Pi') = \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi') + \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi)$. Thus, we can take Π to be
$$\frac{\Pi' \rhd_{\mathcal{P}_{\mathcal{G}}} \Gamma \vdash u\{x \setminus v\} : S \Rightarrow (\{\!\{R \Rightarrow P\}\!\} \times R)}{\Gamma \vdash \mathtt{der}(u\{x \setminus v\}) : S \Rightarrow P}\ \mathtt{Der}$$
since
$$\begin{aligned}\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Pi) &= \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Pi')\\ &= \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi') + \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi)\\ &= \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi).\end{aligned}$$

□

Lemma 7.101 (Exact Subject Reduction for CBPV). *Let $t \in !\Lambda^{\mathcal{G}}_{\circ}$, $s \in \mathcal{S}_{\circ}$, and $S \in \{\beta_{\mathtt{v}}, \beta_{!}, \mathtt{get}, \mathtt{set}\}$. If $(t,s) \Rightarrow_{\mathtt{CBPV}(S)} (u,q)$ and $\Phi \rhd_{\mathcal{P}_{\mathcal{G}}} \emptyset \vdash (t,s) : P$, then there is $\Psi \rhd_{\mathcal{P}_{\mathcal{G}}} \emptyset \vdash (u,q) : P$, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi) + 1 \cdot R$, where*

- $R = \mathtt{App}$, *if* $S = \beta_{\mathtt{v}}$;
- $R = \mathtt{Der}$, *if* $S = \beta_{!}$;
- $R = \mathtt{Get}$, *if* $S = \mathtt{get}$;
- *and* $R = \mathtt{Set}$, *if* $S = \mathtt{set}$.

Proof. Let $(t,s) = (\mathtt{B}[t_0], s) \Rightarrow_{\mathtt{CBPV}(S)} (\mathtt{B}[u_0], q) = (u,q)$, such that $S \in \{\beta_{\mathtt{v}}, \beta_{!}, \mathtt{get}, \mathtt{set}\}$. The proof follow by induction on the definition of $\mathtt{B}$. We omit the cases where $\mathtt{B} = \mathtt{B}'\, r$ and $\mathtt{B} = (\lambda x.r)\,\mathtt{B}'$, since they are very similar to the corresponding ones for the CBV $\lambda^{\mathcal{G}}$-calculus (see Lemma 7.82):

- Case $\mathtt{B} = \Box$:

– If $S = \beta_{\mathtt{v}}$, then $(t_0, s) = ((\lambda x.r)\, v, s) \mapsto_{\beta_{\mathtt{v}}} (r\{x \setminus v\}, s) = (u_0, q)$, and Φ must be of the form

$$\dfrac{\dfrac{\Phi_1 \rhd_{\mathcal{P}_{\mathcal{G}}} x : M \vdash r : S \Rightarrow P}{\varnothing \vdash \lambda x.r : S \Rightarrow ((M \to (S \Rightarrow P)) \times S)}\,\mathtt{Abs} \quad \dfrac{\Phi_2 \rhd_{\mathcal{P}_{\mathcal{G}}} \varnothing \vdash v : M}{\varnothing \vdash v : S \Rightarrow (M \times S)}\,\mathtt{Lift}}{\varnothing \vdash (\lambda x.r)\, v : S \Rightarrow P}\,\mathtt{App} \quad \Phi_3 \rhd_{\mathcal{P}_{\mathcal{G}}} \varnothing \vdash s : S \over \varnothing \vdash ((\lambda x.r)\, v, s) : P\,\mathtt{Conf}$$

according to Lemma 7.95. By applying Lemma 7.100 to Φ_1 and Φ_2, we know there is $\Psi_1 \rhd_{\mathcal{P}_{\mathcal{G}}} \varnothing \vdash r\{x \setminus v\} : S \Rightarrow P$, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi_1) = \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi_2)$. Thus, we can take Ψ to be

$$\dfrac{\Psi_1 \rhd_{\mathcal{P}_{\mathcal{G}}} \varnothing \vdash r\{x \setminus v\} : S \Rightarrow P \qquad \Phi_3 \rhd_{\mathcal{P}_{\mathcal{G}}} \varnothing \vdash s : S}{\varnothing \vdash (r\{x \setminus v\}, s) : P}\,\mathtt{Conf}$$

since

$$\begin{aligned} \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) &= \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{App} \\ &= \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi_1) + 1 \cdot \mathtt{App} = \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi) + 1 \cdot \mathtt{App}. \end{aligned}$$

– If $S = \beta_!$, then $(t_0, s) = (\mathtt{der}(!r), s) \mapsto_{\beta_!} (r, s) = (u_0, q)$, then Φ must be of the form

$$\dfrac{\dfrac{\dfrac{\Phi_1 \rhd_{\mathcal{P}_{\mathcal{G}}} \varnothing \vdash r : S \Rightarrow P}{\varnothing \vdash\, !r : \{\!\{S \Rightarrow P\}\!\}}\,\mathtt{Bg}}{\varnothing \vdash\, !r : S \Rightarrow (\{\!\{S \Rightarrow P\}\!\} \times S)}\,\mathtt{Lift}}{\varnothing \vdash \mathtt{der}(!r) : S \Rightarrow P}\,\mathtt{Der} \quad \Phi_2 \rhd_{\mathcal{P}_{\mathcal{G}}} \varnothing \vdash s : S \over \varnothing \vdash (\mathtt{der}(!r), s) : P\,\mathtt{Conf}$$

Thus, we can take Ψ to be

$$\dfrac{\Phi_1 \rhd_{\mathcal{P}_{\mathcal{T}}} \varnothing \vdash r : S \Rightarrow P \qquad \Phi_2 \rhd_{\mathcal{P}_{\mathcal{G}}} \varnothing \vdash s : S}{\varnothing \vdash (r, s) : P}\,\mathtt{Conf}$$

since we have

$$\begin{aligned} \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) &= \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi_1) + 1 \cdot \mathtt{Der} \\ &= \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi) + 1 \cdot \mathtt{Der}. \end{aligned}$$

– If $S \in \{\mathtt{get}, \mathtt{set}\}$, we proceed similarly to the corresponding cases for the CBV $\lambda^{\mathcal{G}}$-calculus (see Lemma 7.82).

- Case $\mathtt{B} = \mathtt{der}(\mathtt{B}')$. Then, $(t, s) = (\mathtt{der}(\mathtt{B}'[t_0]), s) \Rightarrow_{\mathtt{CBPV}(S)} (\mathtt{der}(\mathtt{B}'[u_0]), q) = (u, q)$, such that $(\mathtt{B}'[t_0], s) \Rightarrow_{\mathtt{CBPV}(S)} (\mathtt{B}'[u_0], q)$, and Φ must be of the form

$$\dfrac{\dfrac{\Phi_1 \rhd_{\mathcal{P}_{\mathcal{G}}} \varnothing \vdash \mathtt{B}'[t_0] : S \Rightarrow (\{\!\{R \Rightarrow P\}\!\} \times R)}{\varnothing \vdash \mathtt{der}(\mathtt{B}'[t_0]) : S \Rightarrow P}\,\mathtt{Der} \quad \Phi_2 \rhd_{\mathcal{P}_{\mathcal{G}}} \varnothing \vdash s : S}{\varnothing \vdash (\mathtt{der}(\mathtt{B}'[t_0]), s) : P}\,\mathtt{Conf}$$

Notice that we can build the following type derivation for $(\mathtt{B}'[t_0], s)$:

$$\dfrac{\Phi_1 \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash \mathtt{B}'[t_0] : S \Rightarrow (\{\!\{R \Rightarrow P\}\!\} \times R) \qquad \Phi_2 \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash s : S}{\emptyset \vdash (\mathtt{B}'[t_0], s) : \{\!\{R \Rightarrow P\}\!\} \times R}\ \mathtt{Conf}$$

Therefore, by applying the *i.h.* to $(\mathtt{B}'[t_0], s)$, there is a type derivation for $(\mathtt{B}'[u_0], q)$ whose premises are of the form $\Psi_1 \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash \mathtt{B}'[u_0] : Q \Rightarrow (\{\!\{R \Rightarrow P\}\!\} \times R)$ and $\Psi_2 \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash q : Q$, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi_2) = \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi_2) + 1 \cdot R$, where

- $R = \mathtt{App}$, if $S = \beta_\mathsf{v}$;
- $R = \mathtt{Der}$, if $S = \beta_!$;
- $R = \mathtt{Get}$, if $S = \mathtt{get}$;
- and $R = \mathtt{Set}$, if $S = \mathtt{set}$.

Therefore, we can take Ψ to be

$$\dfrac{\dfrac{\Psi_1 \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash \mathtt{B}'[u_0] : Q \Rightarrow (\{\!\{R \Rightarrow P\}\!\} \times R)}{\emptyset \vdash \mathtt{der}(\mathtt{B}'[u_0]) : Q \Rightarrow P}\ \mathtt{Der} \qquad \Psi_2 \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash q : Q}{\emptyset \vdash (\mathtt{der}(\mathtt{B}'[u_0]), q) : P}\ \mathtt{Conf}$$

since we have

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) &= \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{Der} \\
&= \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi_2) + 1 \cdot R + 1 \cdot \mathtt{Der} \\
&= \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi) + 1 \cdot R.
\end{aligned}$$

□

Theorem 7.102 (Exact Soundness for CBPV). *Let $t \in !\Lambda^{\mathcal{G}}_\circ$. If there is $\Phi \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash (t, s) : P$ tight, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) = n \cdot \mathtt{App} + m \cdot \mathtt{Der} + g \cdot \mathtt{Get} + s \cdot \mathtt{Set}$, then $(t, s) \overset{n\cdot\beta_\mathsf{v} + m\cdot\beta_! + g\cdot\mathtt{get} + s\cdot\mathtt{set}}{\Rightarrow\!\!\!\!\!\rightarrow}_{\mathtt{CBPV}} (u, q)$, such that $u \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$.*

Proof. Let $\Phi \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash (t, s) : P$ be tight, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) = n \cdot \mathtt{App} + m \cdot \mathtt{Der} + g \cdot \mathtt{Get} + s \cdot \mathtt{Set}$. We proceed by induction on $n + m + g + s$:

- If $n + m + g + s = 0$, then $n = m = g = s = 0$. By Lemma 7.96, $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$. Therefore, we can conclude with $u = t$ and $q = s$.
- If $n + m + g + s > 0$, then $t \notin \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$ by Lemma 7.96. By Proposition 7.25, there are $r \in !\Lambda^{\mathcal{G}}_\circ$ and $s' \in \mathcal{S}_\circ$, such that $(t, s) \Rightarrow_{\mathtt{CBPV}(S)} (r, s')$, for some $S \in \{\beta_\mathsf{v}, \beta_!, \mathtt{get}, \mathtt{set}\}$. By Lemma 7.101, there is $\Phi' \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash (r, s') : P$, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi') + 1 \cdot R$, where

- $R = \mathtt{App}$, if $S = \beta_{\mathsf{v}}$;
- $R = \mathtt{Der}$, if $S = \beta_{!}$;
- $R = \mathtt{Get}$, if $S = \mathtt{get}$;
- and $R = \mathtt{Set}$, if $S = \mathtt{set}$.

Let us assume, without loss of generality, that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi') = n' \cdot \mathtt{App} + m' \cdot \mathtt{Der} + g' \cdot \mathtt{Get} + s' \cdot \mathtt{Set}$. Then, $n' + m' + g' + s' + 1 = n + m + g + s$. Now, by the *i.h.*, we know that $(r, s')^{n' \cdot \beta_{\mathsf{v}} + m' \cdot \beta_{!} + g' \cdot \mathtt{get} + s' \cdot \mathtt{set}} \Rightarrow\!\!\!\!\!\!\Rightarrow_{\mathtt{CBPV}} (u', q')$, such that $u' \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$. Therefore, we have that $(t, s) \Rightarrow_{\mathtt{CBPV}(S)} (r, s')^{n' \cdot \beta_{\mathsf{v}} + m' \cdot \beta_{!} + g' \cdot \mathtt{get} + s' \cdot \mathtt{set}} \Rightarrow\!\!\!\!\!\!\Rightarrow_{\mathtt{CBPV}} (u', q')$. Thus, we can take $(u, q) = (u', q')$, since $(t, s)^{n \cdot \beta_{\mathsf{v}} + m \cdot \beta_{!} + g \cdot \mathtt{get} + s \cdot \mathtt{set}} \Rightarrow\!\!\!\!\!\!\Rightarrow_{\mathtt{CBPV}} (u, q)$.

□

E.3.3.2 Quantitative Completeness

Lemma 7.103 (Typability Implies Clash-Freeness)**.** *Let $t \in !\Lambda^{\mathcal{G}}_{\circ}$ and $s \in \mathcal{S}_{\circ}$. If $\Phi \rhd_{\mathcal{P}_{\mathcal{G}}} \emptyset \vdash (t, s) : P$, then t is clash-free.*

Proof. Let us assume, towards a contradiction, that (t, s) is not clash-free. Then, there is a CBPV context $\mathtt{B}$, a clash u, and a state $q \in \mathcal{S}_{\circ}$, such that $(t, s) \Rightarrow_{\mathtt{CBPV}} \mathtt{B}[(u, q)] = (\mathtt{B}[u], q)$. Moreover, by Lemma 7.101, there must be a type derivation $\Psi \rhd_{\mathcal{P}_{\mathcal{G}}} \emptyset \vdash (\mathtt{B}[u], q) : P$. Therefore, if we show that $(\mathtt{B}[u], q)$ is *not* $\mathcal{P}_{\mathcal{G}}$-typable, then we are done. This follows by a simple induction on the definition of $\mathtt{B}$. We only show the case where $\mathtt{B} = \Box$. The remaining cases follow easily from the *i.h.*. We proceed by inspecting the possible forms of the clash u:

- If $u = v\, r$, then u must be typed by rule `App`, and thus v must be assigned an arrow type. However, it is easy to see that v can only be typed by rules `Ax` or `Bg`, and thus can only be assigned a multi-monadic type. This contradicts our assumption. Therefore, this case does not apply.
- If $u = \mathtt{der}(\lambda x.r)$, then u must be typed by rule `Der`, and thus $\lambda x.r$ must be assigned a multi-monadic type. However, it is easy to see that $\lambda x.r$ can only be typed by rules `Abs` or `Ans`, and thus can only be assigned type an arrow type or $\star$. This contradicts our assumption. Therefore, this case does not apply.
- If $u = r\,(\lambda x.e)$, then u must be typed by rule `App`, and thus $\lambda x.e$ must be assigned a multi-monadic type. However, it is easy to see that $\lambda x.e$ can only be typed by rules `Abs` or `Ans`, and thus can only be assigned type an arrow type or $\star$. This contradicts our assumption. Therefore, this case does not apply.

Finally, since none of the above cases apply, we can conclude that $(\mathtt{B}[u], q)$ is not $\mathcal{P}_{\mathcal{G}}$-typable, as we wanted. □

Lemma 7.104 (Typability Characterizes CBPV Clash-Free Normal Forms). *Let $t \in !\Lambda^{\mathcal{G}}_{\circ}$ and $s \in \mathcal{S}_{\circ}$. Then, $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$ if and only if $t \in \mathtt{NO}_{\mathtt{CBPV}}$ and (t, s) is $\mathcal{P}_{\mathcal{G}}$-typable.*

Proof. ($\Rightarrow$) This direction follows smoothly from Proposition 7.29 and Lemma 7.97.

($\Leftarrow$) Since $t \in \mathtt{NO}_{\mathtt{CBPV}}$, then (t, s) is a CBPV-normal form. Moreover, since (t, s) is $\mathcal{P}_{\mathcal{G}}$-typable, then (t, s) is also clash-free by Lemma 7.103. Therefore, t is a clash-free CBPV-normal form, and thus $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$ by Proposition 7.29.

□

Lemma 7.105 (Weighted Anti-Substitution for CBPV). *Let $t \in !\Lambda^{\mathcal{G}}$, and $v \in !\mathrm{V}^{\mathcal{G}}_{\circ}$. If $\Phi \rhd_{\mathcal{P}_{\mathcal{G}}} \Gamma \vdash t\{x \setminus v\} : \mathbb{A}$ (resp. N or L), then $\Psi \rhd_{\mathcal{P}_{\mathcal{G}}} \Gamma; x : M \vdash t : \mathbb{A}$ (resp. N or L) and $\Pi \rhd_{\mathcal{P}_{\mathcal{G}}} \emptyset \vdash v : M$, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Pi)$.*

Proof. The proof follows by induction on the structure of t. We omit the cases where t is an abstraction, since these are very similar to the corresponding ones for the CBN $\lambda^{\mathcal{G}}$-calculus (see Lemma 7.64), and the cases where t is a variable, an application, or an operation, since these are very similar to the corresponding one for the CBV $\lambda^{\mathcal{G}}$-calculus (see Lemma 7.84):

- Case $t = !u$. Then, $t\{x \setminus v\} = !u\{x \setminus v\} = !(u\{x \setminus v\})$. Therefore, Φ must be of the form
$$\frac{(\Phi_i \rhd_{\mathcal{P}_{\mathcal{G}}} \Gamma_i \vdash u\{x \setminus v\} : \mathbb{A}_i)_{i \in \mathtt{I}}}{+_{i \in \mathtt{I}} \Gamma_i \vdash !(u\{x \setminus v\}) : \{\!\{\mathbb{A}_i\}\!\}_{i \in \mathtt{I}}}\ \mathtt{Bg}$$
$\Gamma = +_{i \in \mathtt{I}} \Gamma_i$, and $N = \{\!\{\mathbb{A}_i\}\!\}_{i \in \mathtt{I}}$. By applying the *i.h.* to each Φ_i, there are $\Psi_i \rhd_{\mathcal{P}_{\mathcal{G}}} \Gamma_i; x : M_i \vdash u : \mathbb{A}_i$ and $\Pi_i \rhd_{\mathcal{P}_{\mathcal{G}}} \emptyset \vdash v : M_i$, for each $i \in \mathtt{I}$, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi_i) = \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi_i) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Pi_i)$. Let us take $M = \sqcup_{i \in \mathtt{I}} M_i$. Then, by Lemma 7.93, we know there is $\Pi' \rhd_{\mathcal{P}_{\mathcal{G}}} \emptyset \vdash v : M$, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Pi') = +_{i \in \mathtt{I}} \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Pi_i)$. Thus, we can take $\Pi = \Pi'$, and Ψ to be
$$\frac{(\Psi_i \rhd_{\mathcal{P}_{\mathcal{G}}} \Gamma_i; x : M_i \vdash u : \mathbb{A}_i)_{i \in \mathtt{I}}}{+_{i \in \mathtt{I}} \Gamma_i; x : \sqcup_{i \in \mathtt{I}} M_i \vdash !u : \{\!\{\mathbb{A}_i\}\!\}_{i \in \mathtt{I}}}\ \mathtt{Bg}$$
since we have
$$\begin{aligned} \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) &= +_{i \in \mathtt{I}} \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi_i) \\ &= +_{i \in \mathtt{I}} \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi_i) + +_{i \in \mathtt{I}} \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Pi_i) \\ &= \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Pi). \end{aligned}$$

- Case $t = \mathtt{der}(u)$. Then, $t\{x \setminus v\} = \mathtt{der}(u)\{x \setminus v\} = \mathtt{der}(u\{x \setminus v\})$. Therefore, Φ must be of the form
$$\frac{\Phi' \rhd_{\mathcal{P}_\mathcal{G}} \Gamma \vdash u\{x \setminus v\} : S \Rightarrow (\{\!\{R \Rightarrow P\}\!\} \times R)}{\Gamma \vdash \mathtt{der}(u\{x \setminus v\}) : S \Rightarrow P}\ \mathtt{Der}$$
and $\mathbb{A} = S \Rightarrow P$. By applying the *i.h.* to Φ', there are $\Psi' \rhd_{\mathcal{P}_\mathcal{G}} \Gamma; x : M \vdash u : S \Rightarrow (\{\!\{R \Rightarrow (A \times Q)\}\!\} \times R)$ and $\Pi' \rhd_{\mathcal{P}_\mathcal{G}} \varnothing \vdash v : M$, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi') = \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi') + \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Pi')$. Thus, we can take $\Pi = \Pi'$, and Ψ to be
$$\frac{\Psi' \rhd_{\mathcal{P}_\mathcal{G}} \Gamma; x : M \vdash u : S \Rightarrow (\{\!\{R \Rightarrow P\}\!\} \times R)}{\Gamma; x : M \vdash \mathtt{der}(u) : S \Rightarrow P}\ \mathtt{Der}$$
since we have that
$$\begin{aligned}\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) &= \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi') \\ &= \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi') + \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Pi') \\ &= \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Pi).\end{aligned}$$

□

Lemma 7.106 (Exact Subject Expansion for CBPV). *Let $t \in !\Lambda^{\mathcal{G}}_{\circ}$, $s \in \mathcal{S}_{\circ}$, and $S \in \{\beta_{\mathtt{v}}, \beta_{!}, \mathtt{get}, \mathtt{set}\}$. If $(t,s) \Rightarrow_{\mathtt{CBPV}(S)} (u,q)$ and $\Phi \rhd_{\mathcal{P}_\mathcal{G}} \varnothing \vdash (u,q) : P$, then there is $\Psi \rhd_{\mathcal{P}_\mathcal{G}} \varnothing \vdash (t,s) : P$, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) + 1 \cdot R = \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi)$, where*

- $R = \mathtt{App}$, *if* $S = \beta_{\mathtt{v}}$;
- $R = \mathtt{Der}$, *if* $S = \beta_{!}$;
- $R = \mathtt{Get}$, *if* $S = \mathtt{get}$;
- *and* $R = \mathtt{Set}$, *if* $S = \mathtt{set}$.

Proof. Let $(t,s) = (\mathtt{B}[t_0], s) \Rightarrow_{\mathtt{CBPV}(S)} (\mathtt{B}[u_0], q) = (u,q)$, such that $S \in \{\beta_{\mathtt{v}}, \beta_{!}, \mathtt{get}, \mathtt{set}\}$. The proof follows by induction on the definition of $\mathtt{B}$. We proceed by cases, according to the context $\mathtt{B}$. We omit the cases where $\mathtt{B} = \mathtt{B}'\, r$ and $\mathtt{B} = (\lambda x.r)\, \mathtt{B}'$, since these cases are very similar to the corresponding ones for the CBV $\lambda^{\mathcal{G}}$-calculus (see Lemma 7.85):

- Case $\mathtt{B} = \Box$:
  - If $S = \beta_{\mathtt{v}}$, then $(t_0, s) = ((\lambda x.r)\, v, s) \mapsto_{\beta_{\mathtt{v}}} (r\{x \setminus v\}, s) = (u_0, q)$, then Φ must be of the form
$$\frac{\Phi_1 \rhd_{\mathcal{P}_\mathcal{G}} \varnothing \vdash r\{x \setminus v\} : S \Rightarrow P \qquad \Phi_2 \rhd_{\mathcal{P}_\mathcal{G}} \varnothing \vdash s : S}{\varnothing \vdash (r\{x \setminus v\}, s) : P}\ \mathtt{Conf}$$
By applying Lemma 7.105 to Φ_1, we know there are $\Psi_1 \rhd_{\mathcal{P}_\mathcal{G}} x : M \vdash r : S \Rightarrow P$ and $\Psi_2 \rhd_{\mathcal{P}_\mathcal{G}} \varnothing \vdash v : M$, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi_1) = \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi_2)$.

Thus, we can take Ψ to be

$$\dfrac{\dfrac{\Psi_1 \rhd_{\mathcal{P}_\mathcal{G}} x : M \vdash r : S \Rightarrow P}{\emptyset \vdash \lambda x.r : S \Rightarrow ((M \to (S \Rightarrow P)) \times S)}\,\texttt{Abs} \quad \dfrac{\Psi_2 \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash v : M}{\emptyset \vdash v : S \Rightarrow (M \times S)}\,\texttt{Lift}}{\dfrac{\emptyset \vdash (\lambda x.r)\, v : S \Rightarrow P \qquad \Phi_2 \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash s : S}{\emptyset \vdash ((\lambda x.r)\, v, s) : P}\,\texttt{Conf}}\,\texttt{App}$$

since

$$\begin{aligned}
\#_{\texttt{App},\texttt{Der},\texttt{Get},\texttt{Set}}(\Phi) + 1 \cdot \texttt{App} &= \#_{\texttt{App},\texttt{Der},\texttt{Get},\texttt{Set}}(\Phi_1) + \#_{\texttt{App},\texttt{Der},\texttt{Get},\texttt{Set}}(\Phi_2) + 1 \cdot \texttt{App} \\
&= \#_{\texttt{App},\texttt{Der},\texttt{Get},\texttt{Set}}(\Psi_1) + \#_{\texttt{App},\texttt{Der},\texttt{Get},\texttt{Set}}(\Psi_2) + 1 \cdot \texttt{App} \\
&= \#_{\texttt{App},\texttt{Der},\texttt{Get},\texttt{Set}}(\Psi).
\end{aligned}$$

– If $S = \beta_!$, then $(t_0, s) = (\texttt{der}(!r), s) \mapsto_{\beta_!} (r, s) = (u_0, q)$, then Φ must be of the form

$$\dfrac{\Phi_1 \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash r : S \Rightarrow P \qquad \Phi_2 \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash s : S}{\emptyset \vdash (r, q) : P}\,\texttt{Conf}$$

Thus, we can take Ψ to be

$$\dfrac{\dfrac{\dfrac{\dfrac{\Phi_1 \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash r : S \Rightarrow P}{\emptyset \vdash\, !r : \{\!\{S \Rightarrow P\}\!\}}\,\texttt{Bg}}{\emptyset \vdash\, !r : S \Rightarrow (\{\!\{S \Rightarrow P\}\!\} \times S)}\,\texttt{Lift}}{\emptyset \vdash \texttt{der}(!r) : S \Rightarrow P}\,\texttt{Der} \qquad \Phi_2 \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash s : S}{\emptyset \vdash (\texttt{der}(!r), s) : P}\,\texttt{Conf}$$

since

$$\begin{aligned}
\#_{\texttt{App},\texttt{Der},\texttt{Get},\texttt{Set}}(\Phi) + 1 \cdot \texttt{Der} &= \#_{\texttt{App},\texttt{Der},\texttt{Get},\texttt{Set}}(\Phi_1) + \#_{\texttt{App},\texttt{Der},\texttt{Get},\texttt{Set}}(\Phi_2) + 1 \cdot \texttt{Der} \\
&= \#_{\texttt{App},\texttt{Der},\texttt{Get},\texttt{Set}}(\Psi).
\end{aligned}$$

– If $S \in \{\texttt{get}, \texttt{set}\}$, we proceed similarly to the corresponding cases for the CBV $\lambda^\mathcal{G}$-calculus (see Lemma 7.82).

- $\texttt{B} = \texttt{der}(\texttt{B}')$. Then, we know that $(t, s) = (\texttt{der}(\texttt{B}'[t_0]), s) \Rightarrow_{\texttt{CBPV}(S)} (\texttt{der}(\texttt{B}'[u_0]), q) = (u, q)$, $(\texttt{B}'[t_0], s) \Rightarrow_{\texttt{CBPV}(S)} (\texttt{B}'[u_0], q)$, and Φ must be of the form

$$\dfrac{\dfrac{\Phi_1 \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash \texttt{B}'[u_0] : Q \Rightarrow (\{\!\{R \Rightarrow P\}\!\} \times R)}{\emptyset \vdash \texttt{der}(\texttt{B}'[u_0]) : Q \Rightarrow P}\,\texttt{Der} \qquad \Phi_2 \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash q : Q}{\emptyset \vdash (\texttt{der}(\texttt{B}'[u_0]), q) : P}\,\texttt{Conf}$$

Notice that we can build the following type derivation for $(\texttt{B}'[u_0], q)$:

$$\dfrac{\Phi_1 \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash \texttt{B}'[u_0] : Q \Rightarrow (\{\!\{R \Rightarrow P\}\!\} \times R) \qquad \Phi_2 \rhd_{\mathcal{P}_\mathcal{G}} \emptyset \vdash q : Q}{\emptyset \vdash (\texttt{B}'[u_0], q) : \{\!\{R \Rightarrow P\}\!\} \times R}\,\texttt{Conf}$$

Therefore, by applying the *i.h.* to $(\mathtt{B}'[u_0], q)$, we know there is a type derivation for $(\mathtt{B}'[t_0], s)$ whose premises are of the form $\Psi_1 \rhd_{\mathcal{P}_\mathcal{G}} \varnothing \vdash \mathtt{B}'[t_0] : S \Rightarrow (\{\!\{R \Rightarrow P\}\!\} \times R)$ and $\Psi_2 \rhd_{\mathcal{P}_\mathcal{G}} \varnothing \vdash s : S$, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot R = \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi_2)$, where

- $R = \mathtt{App}$, if $S = \beta_\mathsf{v}$;
- $R = \mathtt{Der}$, if $S = \beta_!$;
- $R = \mathtt{Get}$, if $S = \mathtt{get}$;
- and $R = \mathtt{Set}$, if $S = \mathtt{set}$.

Therefore, we can take Ψ to be

$$\dfrac{\dfrac{\Psi_1 \rhd_{\mathcal{P}_\mathcal{G}} \varnothing \vdash \mathtt{B}'[t_0] : S \Rightarrow (\{\!\{R \Rightarrow P\}\!\} \times R)}{\varnothing \vdash \mathtt{der}(\mathtt{B}'[t_0]) : S \Rightarrow P}\,\mathtt{Der} \qquad \Psi_2 \rhd_{\mathcal{P}_\mathcal{G}} \varnothing \vdash s : S}{\varnothing \vdash (\mathtt{der}(\mathtt{B}'[t_0]), s) : P}\,\mathtt{Conf}$$

since

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) + 1 \cdot R &= \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{Der} + 1 \cdot R \\
&= \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi_2) + 1 \cdot \mathtt{Der} \\
&= \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Psi).
\end{aligned}$$

The proof follows the same structure as exact subject expansion for CBV, with the additional cases corresponding to the CBPV constructs bg and der. □

Theorem 7.107 (Exact Completeness for CBPV)**.** *Let $t \in \Lambda^{\mathcal{G}}_\circ$. If $(t,s) \overset{n\cdot\beta_\mathsf{v} + m\cdot\beta_! + g\cdot\mathtt{get} + s\cdot\mathtt{set}}{\Rightarrow\!\!\!\!\Rightarrow}_{\mathtt{CBPV}} (u,q)$, such that $u \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$, then there is $\Phi \rhd_{\mathcal{P}_\mathcal{G}} \varnothing \vdash (t,s) : P$ tight, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) = n \cdot \mathtt{App} + m \cdot \mathtt{Der} + g \cdot \mathtt{Get} + s \cdot \mathtt{Set}$.*

Proof. Let $(t,s) \overset{n\cdot\beta_\mathsf{v} + m\cdot\beta_! + g\cdot\mathtt{get} + s\cdot\mathtt{set}}{\Rightarrow\!\!\!\!\Rightarrow}_{\mathtt{CBPV}} (u,q)$, such that $u \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$. We proceed by induction on $n + m + g + s$:

- If $n + m + g + s = 0$, then $n = m = g = s = 0$. Therefore, $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$. Thus we can conclude by Lemma 7.97.
- If $n + m + g + s > 0$, then $t \notin \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$ by Proposition 3.11. Therefore, there are $r \in\, !\Lambda^{\mathcal{G}}_\circ$ and $s' \in \mathcal{S}_\circ$, such that $(t,s) \Rightarrow_{\mathtt{CBPV}(S)} (r,s') \overset{n'\cdot\beta_\mathsf{v} + m'\cdot\beta_! + g'\cdot\mathtt{get} + s'\cdot\mathtt{set}}{\Rightarrow\!\!\!\!\Rightarrow}_{\mathtt{CBPV}} (u,q)$, and $n' + m' + g' + s' + 1 = n + m + g + s$. Now, by the *i.h.*, there is $\Phi' \rhd_{\mathcal{P}_\mathcal{G}} \varnothing \vdash (r,s') : P$ tight, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi') = n' \cdot \mathtt{App} + m' \cdot \mathtt{Der} + g' \cdot \mathtt{Get} + s' \cdot \mathtt{Set}$. By Lemma 7.106, there is $\Phi'' \rhd_{\mathcal{P}_\mathcal{G}} \varnothing \vdash (t,s) : P$, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi') + 1 \cdot R = \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi'')$, where
  - $R = \mathtt{App}$, if $S = \beta_\mathsf{v}$;

- $R = \mathtt{Der}$, if $S = \beta_!$;
- $R = \mathtt{Get}$, if $S = \mathtt{get}$;
- and $R = \mathtt{Set}$, if $S = \mathtt{set}$.

Therefore, we can take $\Phi = \Phi''$, since

$$\begin{aligned} \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi) &= \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi'') \\ &= \#_{\mathtt{App},\mathtt{Der},\mathtt{Get},\mathtt{Set}}(\Phi') + 1 \cdot R \\ &= n' \cdot \mathtt{App} + m' \cdot \mathtt{Der} + g' \cdot \mathtt{Get} + s' \cdot \mathtt{Set} + 1 \cdot R \\ &= n \cdot \mathtt{App} + m \cdot \mathtt{Der} + g \cdot \mathtt{Get} + s \cdot \mathtt{Set}. \end{aligned}$$

□

E.3.3.3 Soundness and Completeness of Translations

Lemma 7.111 (Soundness of Term Translations)**.** *Let $t \in \Lambda^{\mathcal{G}}$.*

1. *If $\Phi \rhd_{\mathcal{N}_{\mathcal{G}}} \Gamma \vdash t : \mathbb{A}$ (resp. M or L), then $\Psi \rhd_{\mathcal{P}_{\mathcal{G}}} \Gamma \vdash (t)^{\mathtt{CBN}}$ (resp. $!(t)^{\mathtt{CBN}}$ or $!(t)^{\mathtt{CBN}}$) : $\mathbb{A}$ (resp. M or L). Moreover, $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi)$.*
2. *If $\Phi \rhd_{\mathcal{V}_{\mathcal{G}}} \Gamma \vdash t : \mathbb{M}$ (resp. M or L), then $\Psi \rhd_{\mathcal{P}_{\mathcal{G}}} (\Gamma)^{\mathtt{CBV}} \vdash (t)^{\mathtt{CBV}} : (\mathbb{M})^{\mathtt{CBV}}$ (resp. $(M)^{\mathtt{CBV}}$ or $(L)^{\mathtt{CBV}}$). Moreover, $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi)$.*

Proof. 1. The proof follows by induction on the structure of Φ. We proceed by cases, according to the last rule R of Φ:

- Case $R = \mathtt{Ax}$. Then, $t = x$, and $(t)^{\mathtt{CBN}} = (x)^{\mathtt{CBN}} = \mathtt{der}(x)$. Moreover, Φ must be of the form
$$\frac{}{x : \{\!\{\mathbb{A}\}\!\} \vdash x : \mathbb{A}}\ \mathtt{Ax}$$
and $\Gamma = x : \{\!\{\mathbb{A}\}\!\}$. Without loss of generality, let us assume $\mathbb{A} = S \Rightarrow P$. Then, we can take Ψ to be
$$\dfrac{\dfrac{\dfrac{}{x : \{\!\{\mathbb{A}\}\!\} \vdash x : \mathbb{A}}\ \mathtt{Ax}}{x : \{\!\{\mathbb{A}\}\!\} \vdash x : S \Rightarrow (\{\!\{\mathbb{A}\}\!\} \times S)}\ \mathtt{Lift}}{x : \{\!\{\mathbb{A}\}\!\} \vdash \mathtt{der}(x) : \mathbb{A}}\ \mathtt{Der}$$
since
$$\begin{aligned} \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) &= 0 \cdot \mathtt{App} + 0 \cdot \mathtt{Get} + 0 \cdot \mathtt{Set} \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi). \end{aligned}$$
- Case $R = \mathtt{Abs}$. Then, $t = \lambda x.u$, and $(t)^{\mathtt{CBN}} = (\lambda x.u)^{\mathtt{CBN}} = \lambda x.(u)^{\mathtt{CBN}}$. Moreover, Φ must be of the form
$$\frac{\Phi_1 \rhd_{\mathcal{N}_{\mathcal{G}}} \Gamma_1 \vdash u : \mathbb{B}}{\Gamma_1 \setminus\!\setminus x \vdash \lambda x.u : S \Rightarrow ((\Gamma_1(x) \to \mathbb{B}) \times S)}\ \mathtt{Abs}$$

$\Gamma = \Gamma_1 \setminus\!\setminus x$, and $\mathbb{A} = S \Rightarrow ((\Gamma_1(x) \to \mathbb{B}) \times S)$. By applying the *i.h.* to Φ_1, we know there exists $\Psi_1 \triangleright_{\mathcal{P}_\mathcal{G}} \Gamma_1 \vdash (u)^{\mathtt{CBN}} : \mathbb{B}$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1)$. Therefore, we can take Ψ to be

$$\frac{\Psi_1 \triangleright_{\mathcal{P}_\mathcal{G}} \Gamma_1 \vdash (u)^{\mathtt{CBN}} : \mathbb{B}}{\Gamma_1 \setminus\!\setminus x \vdash \lambda x.(u)^{\mathtt{CBN}} : S \Rightarrow ((\Gamma_1(x) \to \mathbb{B}) \times S)}\ \mathtt{Abs}$$

since

$$\begin{aligned} \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi). \end{aligned}$$

- Case $R = \mathtt{Ans}$. Then, $t = \lambda x.u$, and $(t)^{\mathtt{CBN}} = (\lambda x.u)^{\mathtt{CBN}} = \lambda x.(u)^{\mathtt{CBN}}$. Moreover, Φ must be of the form

$$\frac{}{\emptyset \vdash \lambda x.u : S \Rightarrow (\star \times S)}\ \mathtt{Ans}$$

$\Gamma = \emptyset$, and $\mathbb{A} = S \Rightarrow (\star \times S)$. Therefore, we can take Ψ to be

$$\frac{}{\emptyset \vdash \lambda x.(u)^{\mathtt{CBN}} : S \Rightarrow (\star \times S)}\ \mathtt{Ans}$$

since

$$\begin{aligned} \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) &= 0 \cdot \mathtt{App} + 0 \cdot \mathtt{Get} + 0 \cdot \mathtt{Set} \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi). \end{aligned}$$

- Case $R = \mathtt{App}$. Then, $t = u\,r$, and $(t)^{\mathtt{CBN}} = (u\,r)^{\mathtt{CBN}} = (u)^{\mathtt{CBN}}\,!(r)^{\mathtt{CBN}}$. Moreover, Φ must be of the form

$$\frac{\Phi_1 \triangleright_{\mathcal{N}_\mathcal{G}} \Gamma_1 \vdash u : S \Rightarrow ((M \to (Q \Rightarrow P)) \times Q) \qquad \Phi_2 \triangleright_{\mathcal{N}_\mathcal{G}} \Gamma_2 \vdash r : M}{\Gamma_1 + \Gamma_2 \vdash u\,r : S \Rightarrow P}\ \mathtt{App}$$

$\Gamma = \Gamma_1 + \Gamma_2$, and $\mathbb{A} = S \Rightarrow P$. By applying the *i.h.* to Φ_1, we know there exists $\Psi_1 \triangleright_{\mathcal{P}_\mathcal{G}} \Gamma_1 \vdash (u)^{\mathtt{CBN}} : S \Rightarrow ((M \to (Q \Rightarrow P)) \times Q)$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1)$. By applying the *i.h.* to Φ_2, we know there exists $\Psi_2 \triangleright_{\mathcal{P}_\mathcal{G}} \Gamma_2 \vdash\ !(r)^{\mathtt{CBN}} : M$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2)$. Therefore, we can take Ψ to be

$$\frac{\Psi_1 \triangleright_{\mathcal{P}_\mathcal{G}} \Gamma_1 \vdash (u)^{\mathtt{CBN}} : S \Rightarrow ((M \to (Q \Rightarrow P)) \times Q) \qquad \Psi_2 \triangleright_{\mathcal{P}_\mathcal{G}} \Gamma_2 \vdash\ !(r)^{\mathtt{CBN}} : M}{\Gamma_1 + \Gamma_2 \vdash (u)^{\mathtt{CBN}}\,!(r)^{\mathtt{CBN}} : S \Rightarrow P}\ \mathtt{App}$$

since

$$\begin{aligned} \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2) \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi). \end{aligned}$$

- Case $R = \mathtt{Many}$. Then, Φ must be of the form

$$\frac{\left(\Phi_i \rhd_{\mathcal{N}_\mathcal{G}} \Gamma_i \vdash t : \mathbb{A}_i\right)_{i\in \mathtt{I}}}{+_{i\in \mathtt{I}}\Gamma_i \vdash t : \{\!\{\mathbb{A}_i\}\!\}_{i\in \mathtt{I}}}\ \mathtt{Many}$$

  $\Gamma = +_{i\in\mathtt{I}}\Gamma_i$, $M = \{\!\{\mathbb{A}_i\}\!\}_{i\in\mathtt{I}}$. By applying the *i.h.* to each Φ_i, we know there exist $\Psi_i \rhd_{\mathcal{P}_\mathcal{G}} \Gamma_i \vdash t : \mathbb{A}_i$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_i) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_i)$, for each $i \in \mathtt{I}$. Therefore, we can take Ψ to be

$$\frac{\left(\Psi_i \rhd_{\mathcal{P}_\mathcal{G}} \Gamma_i \vdash t : \mathbb{A}_i\right)_{i\in \mathtt{I}}}{+_{i\in \mathtt{I}}\Gamma_i \vdash !t : \{\!\{\mathbb{A}_i\}\!\}_{i\in \mathtt{I}}}\ \mathtt{Bg}$$

  since

$$\begin{aligned}\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) &= +_{i\in\mathtt{I}}\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_i)\\ &= +_{i\in\mathtt{I}}\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_i)\\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi).\end{aligned}$$

- Case $R = \mathtt{Get}$. Then, $t = \mathtt{get}(l, \lambda x.u)$, and $(\mathtt{get}(l,\lambda x.u))^{\mathtt{CBN}} = \mathtt{get}(l, \lambda x.(u)^{\mathtt{CBN}})$. Moreover, Φ must be of the form

$$\frac{\Phi_1 \rhd_{\mathcal{N}_\mathcal{G}} \Gamma; x : \mathtt{head}(S\,(l)) \vdash u : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P}{\Gamma \vdash \mathtt{get}(l, \lambda x.u) : S \Rightarrow P}\ \mathtt{Get}$$

  and $\mathbb{A} = S \Rightarrow P$. By applying the *i.h.* to Φ_1, we know there is $\Psi_1 \rhd_{\mathcal{P}_\mathcal{G}} \Gamma; x : \mathtt{head}(S\,(l)) \vdash (u)^{\mathtt{CBN}} : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1)$. Therefore, we can take Ψ to be

$$\frac{\Psi_1 \rhd_{\mathcal{P}_\mathcal{G}} \Gamma; x : \mathtt{head}(S\,(l)) \vdash (u)^{\mathtt{CBN}} : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P}{\Gamma \vdash \mathtt{get}(l, \lambda x.(u)^{\mathtt{CBN}}) : S \Rightarrow P}\ \mathtt{Get}$$

  since

$$\begin{aligned}\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + 1\cdot\mathtt{Get}\\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + 1\cdot\mathtt{Get}\\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi).\end{aligned}$$

- Case $R = \mathtt{Set}$. Then, $t = \mathtt{set}((l,r),u)$, and $(\mathtt{set}((l,r),u))^{\mathtt{CBN}} = \mathtt{set}((l,!(r)^{\mathtt{CBN}}),(u)^{\mathtt{CBN}})$. Moreover, Φ must be of the form

$$\frac{\Phi_1 \rhd_{\mathcal{N}_\mathcal{G}} \Gamma_1 \vdash r : L \qquad \Phi_2 \rhd_{\mathcal{N}_\mathcal{G}} \Gamma_2 \vdash u : S\{l := L\} \Rightarrow P}{\Gamma_1 + \Gamma_2 \vdash \mathtt{set}((l,r),u) : S \Rightarrow P}\ \mathtt{Set}$$

  $\Gamma = \Gamma_1 + \Gamma_2$, and $\mathbb{A} = S \Rightarrow P$. By applying the *i.h.* to Φ_1, we know there is $\Psi_1 \rhd_{\mathcal{P}_\mathcal{G}} \Gamma_1 \vdash !(r)^{\mathtt{CBN}} : L$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1)$. By applying the *i.h.* to Φ_2, we know there is $\Psi_2 \rhd_{\mathcal{P}_\mathcal{G}} \Gamma_2 \vdash (u)^{\mathtt{CBN}} : S\{l := L\} \Rightarrow P$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2)$. Thus, we can take Ψ to be

$$\frac{\Psi_1 \rhd_{\mathcal{P}_\mathcal{G}} \Gamma_1 \vdash !(r)^{\mathtt{CBN}} : L \qquad \Psi_2 \rhd_{\mathcal{P}_\mathcal{G}} \Gamma_2 \vdash (u)^{\mathtt{CBN}} : S\{l := L\} \Rightarrow P}{\Gamma_1 + \Gamma_2 \vdash \mathtt{set}((l,!(r)^{\mathtt{CBN}}),(u)^{\mathtt{CBN}}) : S \Rightarrow P}\ \mathtt{Set}$$

since

$$\begin{aligned}\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{Set} \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2) + 1 \cdot \mathtt{Set} \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi).\end{aligned}$$

- Case $R = \mathtt{EList}$. Then, Φ must be of the form

$$\frac{}{\emptyset \vdash t : [\,]}\ \mathtt{EList}$$

$\Gamma = \emptyset$, and $L = [\,]$. Therefore, we can take Ψ to be

$$\frac{}{\emptyset \vdash !t : [\,]}\ \mathtt{EList}$$

since

$$\begin{aligned}\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) &= 0 \cdot \mathtt{App} + 0 \cdot \mathtt{Get} + 0 \cdot \mathtt{Set} \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi).\end{aligned}$$

- Case $R = \mathtt{List}$. Then, Φ must be of the form

$$\frac{\Phi_1 \rhd_{\mathcal{N}_\mathcal{G}} \Gamma_1 \vdash t : M \qquad \Phi_2 \rhd_{\mathcal{N}_\mathcal{G}} \Gamma_2 \vdash t : L'}{\Gamma_1 + tye_2 \vdash t : M \cdot L'}\ \mathtt{List}$$

$\Gamma = \Gamma_1 + \Gamma_2$, and $L = M \cdot L'$. By applying the *i.h.* to Φ_1, there exists $\Psi_1 \rhd_{\mathcal{P}_\mathcal{G}} \Gamma_1 \vdash !t : M$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1)$. By applying the *i.h.* to Φ_2, there exists $\Psi_2 \rhd_{\mathcal{P}_\mathcal{G}} \Gamma_2 \vdash !t : L'$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2)$. Therefore, we can take Ψ to be

$$\frac{\Psi_1 \rhd_{\mathcal{P}_\mathcal{G}} \Gamma_1 \vdash !t : M \qquad \Psi_2 \rhd_{\mathcal{P}_\mathcal{G}} \Gamma_2 \vdash !t : L'}{\Gamma_1 + \Gamma_2 \vdash !t : L'}\ \mathtt{List}$$

since

$$\begin{aligned}\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2) \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi).\end{aligned}$$

2. The proof follow by induction on the structure of Φ. We proceed by cases, according to the last rule R of Φ:

- Case $R = \mathtt{Ax}$. Then, $t = x$, $(x)^{\mathtt{CBV}} = x$. Moreover, Φ must be of the form

$$\frac{}{x : M \vdash x : M}\ \mathtt{Ax}$$

and $\Gamma = x : M$. Therefore, we can take Ψ to be

$$\frac{}{x : (M)^{\mathtt{CBV}} \vdash x : (M)^{\mathtt{CBV}}}\ \mathtt{Ax}$$

since

$$\begin{aligned}\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) &= 0\cdot\mathtt{App}+0\cdot\mathtt{Get}+0\cdot\mathtt{Set}\\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi).\end{aligned}$$

- Case $R=\mathtt{Abs}$. Then, $t=\lambda x.u$, and $(\lambda x.u)^{\mathtt{CBV}}=!\lambda x.(u)^{\mathtt{CBV}}$. Moreover, Φ must be of the form

$$\frac{(\Phi_i \rhd_{\mathcal{V}_\mathcal{G}} \Gamma_i; x:M_i \vdash u:\mathbb{N}_i)_{i\in\mathtt{I}}}{+_{i\in\mathtt{I}}\Gamma_i \vdash \lambda x.u : \{\!\{M_i\to\mathbb{N}_i\}\!\}_{i\in\mathtt{I}}}\ \mathtt{Abs}$$

$\Gamma=+_{i\in\mathtt{I}}\Gamma_i$, and $M=\{\!\{M_i\to\mathbb{N}_i\}\!\}_{i\in\mathtt{I}}$. By applying the *i.h.* to each Φ_i, there are $\Psi_i \rhd_{\mathcal{P}_\mathcal{G}} (\Gamma_i)^{\mathtt{CBV}}; x:(M_i)^{\mathtt{CBV}} \vdash (u)^{\mathtt{CBV}} : (\mathbb{N}_i)^{\mathtt{CBV}}$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_i)=\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_i)$, for each $i\in\mathtt{I}$. Therefore, we can take Ψ to be

$$\frac{\left(\dfrac{\Psi_i \rhd_{\mathcal{P}_\mathcal{G}} (\Gamma_i)^{\mathtt{CBV}}; x:(M_i)^{\mathtt{CBV}} \vdash (u)^{\mathtt{CBV}} : (\mathbb{N}_i)^{\mathtt{CBV}}}{(\Gamma_i)^{\mathtt{CBV}} \vdash \lambda x.(u)^{\mathtt{CBV}} : Q_i \Rightarrow (((M_i)^{\mathtt{CBV}}\to(\mathbb{N}_i)^{\mathtt{CBV}})\times Q_i)}\ \mathtt{Abs}\right)_{i\in\mathtt{I}}}{+_{i\in\mathtt{I}}(\Gamma_i)^{\mathtt{CBV}} \vdash\ !\lambda x.(u)^{\mathtt{CBV}} : \{\!\{Q_i \Rightarrow (((M_i)^{\mathtt{CBV}}\to(\mathbb{N}_i)^{\mathtt{CBV}})\times Q_i)\}\!\}_{i\in\mathtt{I}}}\ \mathtt{Bg}$$

since we know that $+_{i\in\mathtt{I}}(\Gamma_i)^{\mathtt{CBV}}=(+_{i\in\mathtt{I}}\Gamma_i)^{\mathtt{CBV}}$, $(M)^{\mathtt{CBV}}=(\{\!\{M_i\to\mathbb{N}_i\}\!\}_{i\in\mathtt{I}})^{\mathtt{CBV}}=\{\!\{Q_i\Rightarrow(((M_i)^{\mathtt{CBV}}\to(\mathbb{N}_i)^{\mathtt{CBV}})\times Q_i)\}\!\}_{i\in\mathtt{I}}$, and

$$\begin{aligned}\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) &= +_{i\in\mathtt{I}}\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_i)\\ &= +_{i\in\mathtt{I}}\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_i)\\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi).\end{aligned}$$

- Case $R=\mathtt{App}$. Then, $t=u\,r$, and $(u\,r)^{\mathtt{CBV}}=\mathtt{der}((u)^{\mathtt{CBV}})\,(r)^{\mathtt{CBV}}$. Moreover, Φ must be of the form

$$\frac{\Phi_1 \rhd_{\mathcal{V}_\mathcal{G}} \Gamma_1 \vdash u : S\Rightarrow(\{\!\{N\to(R\Rightarrow P)\}\!\}\times Q)\qquad \Phi_2 \rhd_{\mathcal{V}_\mathcal{G}} \Gamma_2 \vdash r : Q\Rightarrow(N\times R)}{\Gamma_1+\Gamma_2 \vdash u\,r : S\Rightarrow P}\ \mathtt{App}$$

$\Gamma=\Gamma_1+\Gamma_2$, and $\mathbb{M}=S\Rightarrow P$. By applying the *i.h.* to u, we know there exists a derivation $\Psi_1 \rhd_{\mathcal{P}_\mathcal{G}} (\Gamma_1)^{\mathtt{CBV}} \vdash (u)^{\mathtt{CBV}} : (S)^{\mathtt{CBV}} \Rightarrow (\{\!\{(Q)^{\mathtt{CBV}}\Rightarrow(((N)^{\mathtt{CBV}}\to((R)^{\mathtt{CBV}}\Rightarrow(P)^{\mathtt{CBV}}))\times(Q)^{\mathtt{CBV}})\}\!\}\times(Q)^{\mathtt{CBV}})$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1)=\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1)$. By applying the *i.h.* to r, we know there is $\Psi_2 \rhd_{\mathcal{P}_\mathcal{G}} (\Gamma_2)^{\mathtt{CBV}} \vdash (r)^{\mathtt{CBV}} : (Q)^{\mathtt{CBV}} \Rightarrow ((N)^{\mathtt{CBV}}\times(R)^{\mathtt{CBV}})$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2)=\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2)$. Therefore, we can take Ψ to be

$$\frac{\dfrac{\Psi_1 \rhd_{\mathcal{P}_\mathcal{G}} (\Gamma_1)^{\mathtt{CBV}} \vdash (u)^{\mathtt{CBV}} : (S)^{\mathtt{CBV}} \Rightarrow (\{\!\{(Q)^{\mathtt{CBV}}\Rightarrow(((N)^{\mathtt{CBV}}\to((R)^{\mathtt{CBV}}\Rightarrow(P)^{\mathtt{CBV}}))\times(Q)^{\mathtt{CBV}})\}\!\}\times(Q)^{\mathtt{CBV}})}{(\Gamma_1)^{\mathtt{CBV}} \vdash \mathtt{der}((u)^{\mathtt{CBV}}) : (S)^{\mathtt{CBV}} \Rightarrow (((N)^{\mathtt{CBV}}\to((R)^{\mathtt{CBV}}\Rightarrow(P)^{\mathtt{CBV}}))\times(Q)^{\mathtt{CBV}})}\ \mathtt{Der} \qquad \Psi_2 \rhd_{\mathcal{P}_\mathcal{G}} (\Gamma_2)^{\mathtt{CBV}} \vdash (r)^{\mathtt{CBV}} : (Q)^{\mathtt{CBV}} \Rightarrow ((N)^{\mathtt{CBV}}\times(R)^{\mathtt{CBV}})}{(\Gamma_1)^{\mathtt{CBV}}+(\Gamma_2)^{\mathtt{CBV}} \vdash \mathtt{der}((u)^{\mathtt{CBV}})\,(r)^{\mathtt{CBV}} : (S)^{\mathtt{CBV}} \Rightarrow (P)^{\mathtt{CBV}}}\ \mathtt{App}$$

since $(\Gamma_1+\Gamma_2)^{\mathtt{CBV}}=(\Gamma_1)^{\mathtt{CBV}}+(\Gamma_2)^{\mathtt{CBV}}$, $(\mathbb{M})^{\mathtt{CBV}}=(S\Rightarrow P)^{\mathtt{CBV}}=(S)^{\mathtt{CBV}}\Rightarrow(P)^{\mathtt{CBV}}$, and

$$\begin{aligned}\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1)+\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2)+1\\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1)+\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2)+1\\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi).\end{aligned}$$

- Case $R = \texttt{Lift}$. Then, Φ must be of the form

$$\frac{\Phi_1 \rhd_{\mathcal{V}_\mathcal{G}} \Gamma \vdash v : M}{\Gamma \vdash v : S \Rightarrow (M \times S)}\ \texttt{Lift}$$

and $\mathbb{M} = S \Rightarrow (M \times S)$. By applying the *i.h.* to Φ_1, we know there exists $\Psi_1 \rhd_{\mathcal{P}_\mathcal{G}} (\Gamma)^{\texttt{CBV}} \vdash (v)^{\texttt{CBV}} : (M)^{\texttt{CBV}}$, such that $\#_{\texttt{App,Get,Set}}(\Phi_1) = \#_{\texttt{App,Get,Set}}(\Psi_1)$. Notice that v is either a variable or an abstraction. If v is a variable, then $(v)^{\texttt{CBV}}$ is a variable. If v is an abstraction, then $(v)^{\texttt{CBV}}$ is a bang-term. Therefore, we can take Ψ to be

$$\frac{\Psi_1 \rhd_{\mathcal{P}_\mathcal{G}} (\Gamma)^{\texttt{CBV}} \vdash (v)^{\texttt{CBV}} : (M)^{\texttt{CBV}}}{(\Gamma)^{\texttt{CBV}} \vdash (v)^{\texttt{CBV}} : (S)^{\texttt{CBV}} \Rightarrow ((M)^{\texttt{CBV}} \times (S)^{\texttt{CBV}})}\ \texttt{Lift}$$

since $(\mathbb{M})^{\texttt{CBV}} = (S \Rightarrow (M \times S))^{\texttt{CBV}} = (S)^{\texttt{CBV}} \Rightarrow ((M)^{\texttt{CBV}} \times (S)^{\texttt{CBV}})$, and

$$\begin{aligned} \#_{\texttt{App,Get,Set}}(\Phi) &= \#_{\texttt{App,Get,Set}}(\Phi_1) \\ &= \#_{\texttt{App,Get,Set}}(\Psi_1) \\ &= \#_{\texttt{App,Get,Set}}(\Psi). \end{aligned}$$

- Case $R = \texttt{Get}$. Then, $t = \texttt{get}(l, \lambda x.u)$, and $(\texttt{get}(l, \lambda x.u))^{\texttt{CBV}} = \texttt{get}(l, \lambda x.(u)^{\texttt{CBV}})$. Moreover, Φ must be of the form

$$\frac{\Phi_1 \rhd_{\mathcal{V}_\mathcal{G}} \Gamma; x : \texttt{head}(S\,(l)) \vdash u : S\{l := \texttt{tail}(S\,(l))\} \Rightarrow P}{\Gamma \vdash \texttt{get}(l, \lambda x.u) : S \Rightarrow P}\ \texttt{Get}$$

and $\mathbb{M} = S \Rightarrow P$. By applying the *i.h.* to Φ_1, there is $\Psi_1 \rhd_{\mathcal{P}_\mathcal{G}} (\Gamma)^{\texttt{CBV}}; x : (\texttt{head}(S\,(l)))^{\texttt{CBV}} \vdash (u)^{\texttt{CBV}} : (S\{l := \texttt{tail}(S\,(l))\})^{\texttt{CBV}} \Rightarrow (P)^{\texttt{CBV}}$, such that $\#_{\texttt{App,Get,Set}}(\Phi_1) = \#_{\texttt{App,Get,Set}}(\Psi_1)$. It is easy to see that $(\texttt{head}(S\,(l)))^{\texttt{CBV}} := \texttt{head}((S)^{\texttt{CBV}}\,(l))$ and $(S\{l := \texttt{tail}(S\,(l))\})^{\texttt{CBV}} := (S)^{\texttt{CBV}}\{l := \texttt{tail}((S)^{\texttt{CBV}}\,(l))\}$. Therefore, we can take Ψ to be

$$\frac{\Psi_1 \rhd_{\mathcal{P}_\mathcal{G}} (\Gamma)^{\texttt{CBV}}; x : \texttt{head}((S)^{\texttt{CBV}}\,(l)) \vdash (u)^{\texttt{CBV}} : (S)^{\texttt{CBV}}\{l := \texttt{tail}((S)^{\texttt{CBV}}\,(l))\} \Rightarrow (P)^{\texttt{CBV}}}{(\Gamma)^{\texttt{CBV}} \vdash \texttt{get}(l, \lambda x.(u)^{\texttt{CBV}}) : (S)^{\texttt{CBV}} \Rightarrow (P)^{\texttt{CBV}}}\ \texttt{Get}$$

since $(\mathbb{M})^{\texttt{CBV}} = (S \Rightarrow P)^{\texttt{CBV}} = (S)^{\texttt{CBV}} \Rightarrow (P)^{\texttt{CBV}}$, and

$$\begin{aligned} \#_{\texttt{App,Get,Set}}(\Phi) &= \#_{\texttt{App,Get,Set}}(\Phi_1) + 1 \cdot \texttt{Get} = \#_{\texttt{App,Get,Set}}(\Psi_1) + 1 \cdot \texttt{Get} \\ &= \#_{\texttt{App,Get,Set}}(\Psi). \end{aligned}$$

- Case $R = \texttt{Set}$. Then, $t = \texttt{set}((l, v), u)$, and $(\texttt{set}((l, v), u))^{\texttt{CBV}} = \texttt{set}((l, (v)^{\texttt{CBV}}), (u)^{\texttt{CBV}})$. Moreover, Φ must be of the form

$$\frac{\Phi_1 \rhd_{\mathcal{V}_\mathcal{G}} \Gamma_1 \vdash v : L \qquad \Phi_2 \rhd_{\mathcal{V}_\mathcal{G}} \Gamma_2 \vdash u : S\{l := L\} \Rightarrow P}{\Gamma_1 + \Gamma_2 \vdash \texttt{set}((l, v), u) : S \Rightarrow P}\ \texttt{Set}$$

$\Gamma = \Gamma_1 + \Gamma_2$, and $\mathbb{M} = S \Rightarrow P$. By applying the *i.h.* to Φ_1, we know there is $\Psi_1 \rhd_{\mathcal{P}_\mathcal{G}} (\Gamma_1)^{\texttt{CBV}} \vdash (r)^{\texttt{CBV}} : (L)^{\texttt{CBV}}$, such that $\#_{\texttt{App,Get,Set}}(\Phi_1) = \#_{\texttt{App,Get,Set}}(\Psi_1)$. By applying the

i.h. to Φ_2, we know there is $\Psi_2 \rhd_{\mathcal{P}_\mathcal{G}} (\Gamma_2)^{\mathtt{CBV}} \vdash (u)^{\mathtt{CBV}} : (S\{l := L\})^{\mathtt{CBV}} \Rightarrow (P)^{\mathtt{CBV}}$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2)$. It is easy to see that $(S\{l := L\})^{\mathtt{CBV}} := (S)^{\mathtt{CBV}}\{l := (L)^{\mathtt{CBV}}\}$. Thus, we can take Ψ to be

$$\frac{\Psi_1 \rhd_{\mathcal{P}_\mathcal{G}} (\Gamma_1)^{\mathtt{CBV}} \vdash (r)^{\mathtt{CBV}} : (L)^{\mathtt{CBV}} \quad \Psi_2 \rhd_{\mathcal{P}_\mathcal{G}} (\Gamma_2)^{\mathtt{CBV}} \vdash (u)^{\mathtt{CBV}} : (S)^{\mathtt{CBV}}\{l := (L)^{\mathtt{CBV}}\} \Rightarrow (P)^{\mathtt{CBV}}}{(\Gamma_1 + \Gamma_2)^{\mathtt{CBV}} \vdash \mathtt{set}((l, (v)^{\mathtt{CBV}}), (u)^{\mathtt{CBV}}) : (S)^{\mathtt{CBV}} \Rightarrow (P)^{\mathtt{CBV}}}\ \mathtt{Set}$$

since $(\Gamma_1 + \Gamma_2)^{\mathtt{CBV}} = (\Gamma_1)^{\mathtt{CBV}} + (\Gamma_2)^{\mathtt{CBV}}$, $(\mathbb{M})^{\mathtt{CBV}} = (S \Rightarrow P)^{\mathtt{CBV}} = (S)^{\mathtt{CBV}} \Rightarrow (P)^{\mathtt{CBV}}$, and

$$\begin{aligned} \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) + 1 \cdot \mathtt{Set} \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2) + 1 \cdot \mathtt{Set} \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi). \end{aligned}$$

- Case $R = \mathtt{EList}$. Then, Φ must be of the form

$$\frac{}{\varnothing \vdash v : [\,]}\ \mathtt{EList}$$

$\Gamma = \varnothing$, and $L = [\,]$. Notice that v is either a variable or an abstraction. If v is a variable, then $(v)^{\mathtt{CBV}}$ is a variable. If v is an abstraction, then $(v)^{\mathtt{CBV}}$ is a bang-term. Therefore, we can take Ψ to be

$$\frac{}{\varnothing \vdash (v)^{\mathtt{CBV}} : [\,]}\ \mathtt{EList}$$

since $(\varnothing)^{\mathtt{CBV}} = \varnothing$, $([\,])^{\mathtt{CBV}} = [\,]$, and

$$\begin{aligned} \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) &= 0 \cdot \mathtt{App} + 0 \cdot \mathtt{Get} + 0 \cdot \mathtt{Set} \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi). \end{aligned}$$

- Case $R = \mathtt{List}$. Then, Φ must be of the form

$$\frac{\Phi_1 \rhd_{\mathcal{V}_\mathcal{G}} \Gamma_1 \vdash v : M \quad \Phi_2 \rhd_{\mathcal{V}_\mathcal{G}} \Gamma_2 \vdash v : L'}{\Gamma_1 + \Gamma_2 \vdash t : M \cdot L'}\ \mathtt{List}$$

$\Gamma = \Gamma_1 + \Gamma_2$, and $L = M \cdot L'$. By applying the *i.h.* to Φ_1, there exists $\Psi_1 \rhd_{\mathcal{P}_\mathcal{G}} (\Gamma_1)^{\mathtt{CBV}} \vdash (v)^{\mathtt{CBV}} : (M)^{\mathtt{CBV}}$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1)$. By applying the *i.h.* to Φ_2, there exists $\Psi_2 \rhd_{\mathcal{P}_\mathcal{G}} (\Gamma_2)^{\mathtt{CBV}} \vdash (v)^{\mathtt{CBV}} : (L')^{\mathtt{CBV}}$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2)$. Notice that v is either a variable or an abstraction. If v is a variable, then $(v)^{\mathtt{CBV}}$ is a variable. If v is an abstraction, then $(v)^{\mathtt{CBV}}$ is a bang-term. Therefore, we can take Ψ to be

$$\frac{\Psi_1 \rhd_{\mathcal{P}_\mathcal{G}} (\Gamma_1)^{\mathtt{CBV}} \vdash (v)^{\mathtt{CBV}} : M \quad \Psi_2 \rhd_{\mathcal{P}_\mathcal{G}} \Gamma_2 \vdash (v)^{\mathtt{CBV}} : L'}{(\Gamma_1)^{\mathtt{CBV}} + (\Gamma_2)^{\mathtt{CBV}} \vdash (v)^{\mathtt{CBV}} : (L')^{\mathtt{CBV}}}\ \mathtt{List}$$

since $(\Gamma_1 + \Gamma_2)^{\mathtt{CBV}} = (\Gamma_1)^{\mathtt{CBV}} + (\Gamma_2)^{\mathtt{CBV}}$, and

$$\begin{aligned} \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2) \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2) \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi). \end{aligned}$$

□

Lemma 7.112 (Completeness of CBN Term Translations)**.** *Let $t \in \Lambda^{\mathcal{G}}$.*
If $\Psi \triangleright_{\mathcal{P}_{\mathcal{G}}} \Gamma \vdash (t)^{\mathtt{CBN}}$ (resp. $!(t)^{\mathtt{CBN}}$or $!(t)^{\mathtt{CBN}}$) $: \mathbb{A}$ (resp. M or L), then $\Phi \triangleright_{\mathcal{N}_{\mathcal{G}}} \Gamma \vdash t : \mathbb{A}$ (resp. M or L). Moreover, $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi)$.

Proof. The proof follows by induction on the structure of Ψ. We proceed by cases, according to the last rule R of Φ:

- Case $R = \mathtt{Ax}$. Then, $(t)^{\mathtt{CBN}} = x$. However, there is no t such that $(t)^{\mathtt{CBN}} = x$, Since $(\cdot)^{\mathtt{CBN}}$ maps variables to $\mathtt{der}(x)$. Therefore, this case does not apply.

- Case $R = \mathtt{Abs}$. Then, $(t)^{\mathtt{CBN}} = \lambda x.(u)^{\mathtt{CBN}}$, and $t = \lambda x.u$. Moreover, Ψ must be of the form

$$\frac{\Psi_1 \triangleright_{\mathcal{P}_{\mathcal{G}}} \Gamma_1 \vdash (u)^{\mathtt{CBN}} : \mathbb{B}}{\Gamma_1 \setminus\!\setminus x \vdash \lambda x.(u)^{\mathtt{CBN}} : S \Rightarrow ((\Gamma_1(x) \to \mathbb{B}) \times S)}\ \mathtt{Abs}$$

  $\Gamma = \Gamma_1 \setminus\!\setminus x$, and $\mathbb{A} = S \Rightarrow ((\Gamma_1(x) \to \mathbb{B}) \times S)$. By applying the *i.h.* to Ψ_1, we know that $\Phi_1 \triangleright_{\mathcal{N}_{\mathcal{G}}} \Gamma_1 \vdash u : \mathbb{B}$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1)$. Therefore, we can take Φ to be

$$\frac{\Phi_1 \triangleright_{\mathcal{N}_{\mathcal{G}}} \Gamma_1 \vdash u : \mathbb{B}}{\Gamma_1 \setminus\!\setminus x \vdash \lambda x.u : S \Rightarrow ((\Gamma_1(x) \to \mathbb{B}) \times S)}\ \mathtt{Abs}$$

  since

$$\begin{aligned} \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1) \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1) \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi). \end{aligned}$$

- Case $R = \mathtt{Ans}$. Then, $(t)^{\mathtt{CBN}} = \lambda x.(u)^{\mathtt{CBN}}$, and $t = \lambda x.u$. Moreover, Ψ must be of the form

$$\frac{}{\varnothing \vdash \lambda x.(u)^{\mathtt{CBN}} : S \Rightarrow (\star \times S)}\ \mathtt{Ans}$$

  $\Gamma = \varnothing$, and $\mathbb{A} = S \Rightarrow (\star \times S)$. Therefore, we can take Φ to be

$$\frac{}{\varnothing \vdash \lambda x.u : S \Rightarrow (\star \times S)}\ \mathtt{Ans}$$

  since

$$\begin{aligned} \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) &= 0 \cdot \mathtt{App} + 0 \cdot \mathtt{Get} + 0 \cdot \mathtt{Set} \\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi). \end{aligned}$$

- Case $R = \texttt{App}$. Then, $(t)^{\texttt{CBN}} = (u)^{\texttt{CBN}}\,!(r)^{\texttt{CBN}}$, and $t = u\,r$. Moreover, Ψ must be of the form

$$\dfrac{\Psi_1 \rhd_{\mathcal{P}_\mathcal{G}} \Gamma_1 \vdash (u)^{\texttt{CBN}} : S \Rightarrow ((M \to (Q \Rightarrow P)) \times Q) \qquad \dfrac{\Psi_2 \rhd_{\mathcal{P}_\mathcal{G}} \Gamma_2 \vdash\, !(r)^{\texttt{CBN}} : M}{\Gamma_2 \vdash\, !(r)^{\texttt{CBN}} : Q \Rightarrow (M \times Q)}\ \texttt{Lift}}{\Gamma_1 + \Gamma_2 \vdash (u)^{\texttt{CBN}}\,!(r)^{\texttt{CBN}} : S \Rightarrow P}\ \texttt{App}$$

  $\Gamma = \Gamma_1 + \Gamma_2$, and $\mathbb{A} = S \Rightarrow P$. By applying the *i.h.* to Ψ_1, we know there exists $\Phi_1 \rhd_{\mathcal{N}_\mathcal{G}} \Gamma_1 \vdash u : S \Rightarrow ((M \to (Q \Rightarrow P)) \times Q)$, such that $\#_{\texttt{App,Get,Set}}(\Psi_1) = \#_{\texttt{App,Get,Set}}(\Phi_1)$. Notice that Ψ_2 assigns a multi-monadic type to the bang-term $!(r)^{\texttt{CBN}}$. Therefore, by applying the *i.h.* to Ψ_2, we know there exists $\Phi_2 \rhd_{\mathcal{N}_\mathcal{G}} \Gamma_2 \vdash r : M$, such that $\#_{\texttt{App,Get,Set}}(\Psi_2) = \#_{\texttt{App,Get,Set}}(\Phi_2)$. Therefore, we can take Φ to be

$$\dfrac{\Phi_1 \rhd_{\mathcal{N}_\mathcal{G}} \Gamma_1 \vdash u : S \Rightarrow ((M \to (Q \Rightarrow P)) \times Q) \qquad \Phi_2 \rhd_{\mathcal{N}_\mathcal{G}} \Gamma_2 \vdash r : M}{\Gamma_1 + \Gamma_2 \vdash u\,r : S \Rightarrow P}\ \texttt{App}$$

  since

$$\begin{aligned} \#_{\texttt{App,Get,Set}}(\Psi) &= \#_{\texttt{App,Get,Set}}(\Psi_1) + \#_{\texttt{App,Get,Set}}(\Psi_2) + 1 \cdot \texttt{App} \\ &= \#_{\texttt{App,Get,Set}}(\Phi_1) + \#_{\texttt{App,Get,Set}}(\Phi_2) + 1 \cdot \texttt{App} \\ &= \#_{\texttt{App,Get,Set}}(\Phi). \end{aligned}$$

- Case $R = \texttt{Bg}$. Then, we must be considering $!(t)^{\texttt{CBN}}$. Hence, Ψ must be of the form

$$\dfrac{\left(\Psi_i \rhd_{\mathcal{P}_\mathcal{G}} \Gamma_i \vdash (t)^{\texttt{CBN}} : \mathbb{A}_i\right)_{i \in \texttt{I}}}{+_{i \in \texttt{I}}\,\Gamma_i \vdash\, !(t)^{\texttt{CBN}} : \{\!\{\mathbb{A}_i\}\!\}_{i \in \texttt{I}}}\ \texttt{Bg}$$

  $\Gamma = +_{i \in \texttt{I}}\Gamma_i$, and $M = \{\!\{\mathbb{A}_i\}\!\}_{i \in \texttt{I}}$. By applying the *i.h.* to each Ψ_i, we know there exist $\Phi_i \rhd_{\mathcal{N}_\mathcal{G}} \Gamma_i \vdash t : \mathbb{A}_i$, such that $\#_{\texttt{App,Get,Set}}(\Psi_i) = \#_{\texttt{App,Get,Set}}(\Phi_i)$, for each $i \in \texttt{I}$. Therefore, we can take Φ to be

$$\dfrac{\left(\Phi_i \rhd_{\mathcal{N}_\mathcal{G}} \Gamma_i \vdash t : \mathbb{A}_i\right)_{i \in \texttt{I}}}{+_{i \in \texttt{I}}\,\Gamma_i \vdash t : \{\!\{\mathbb{A}_i\}\!\}_{i \in \texttt{I}}}\ \texttt{Many}$$

  since

$$\begin{aligned} \#_{\texttt{App,Get,Set}}(\Psi) &= +_{i \in \texttt{I}} \#_{\texttt{App,Get,Set}}(\Psi_i) \\ &= +_{i \in \texttt{I}} \#_{\texttt{App,Get,Set}}(\Phi_i) \\ &= \#_{\texttt{App,Get,Set}}(\Phi). \end{aligned}$$

- Case $R = \texttt{Der}$. Then, $(t)^{\texttt{CBN}} = \texttt{der}(x)$, and $t = x$. Moreover, Ψ must be of the form

$$\dfrac{\dfrac{\dfrac{}{x : \{\!\{S \Rightarrow P\}\!\} \vdash x : \{\!\{S \Rightarrow P\}\!\}}\ \texttt{Ax}}{x : \{\!\{S \Rightarrow P\}\!\} \vdash x : S \Rightarrow (\{\!\{S \Rightarrow P\}\!\} \times S)}\ \texttt{Lift}}{x : \{\!\{S \Rightarrow P\}\!\} \vdash \texttt{der}(x) : S \Rightarrow P}\ \texttt{Der}$$

  $\Gamma = x : \{\!\{S \Rightarrow P\}\!\}$, $\mathbb{A} = S \Rightarrow P$. Therefore, we can take Φ to be

$$\dfrac{}{x : \{\!\{S \Rightarrow P\}\!\} \vdash x : S \Rightarrow P}\ \texttt{Ax}$$

since

$$\begin{aligned}\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) &= 0\cdot\mathtt{App}+0\cdot\mathtt{Get}+0\cdot\mathtt{Set}\\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi).\end{aligned}$$

- Case $R=\mathtt{Lift}$. Then, we must be considering $!(t)^{\mathtt{CBN}}$. However, $\mathtt{Lift}$ assigns a monadic type to $!(t)^{\mathtt{CBN}}$. Therefore, this case does not apply.

- Case $R=\mathtt{Get}$. Then, $(t)^{\mathtt{CBN}}=\mathtt{get}(l,\lambda x.(u)^{\mathtt{CBN}})$, and $t=\mathtt{get}(l,\lambda x.u)$. Moreover, Ψ must be of the form

$$\frac{\Psi_1 \rhd_{\mathcal{P}_{\mathcal{G}}} \Gamma; x:\mathtt{head}(S\,(l)) \vdash (u)^{\mathtt{CBN}} : S\{l:=\mathtt{tail}(S\,(l))\}\Rightarrow P}{\Gamma\vdash \mathtt{get}(l,\lambda x.(u)^{\mathtt{CBN}}) : S\Rightarrow P}\ \mathtt{Get}$$

  and $\mathbb{A}=S\Rightarrow P$. By applying the *i.h.* to Ψ_1, we know there is $\Phi_1 \rhd_{\mathcal{N}_{\mathcal{G}}} \Gamma; x:\mathtt{head}(S\,(l)) \vdash u : S\{l:=\mathtt{tail}(S\,(l))\}\Rightarrow P$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1)=\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1)$.Therefore, we can take Φ to be

$$\frac{\Phi_1 \rhd_{\mathcal{N}_{\mathcal{G}}} \Gamma; x:\mathtt{head}(S\,(l)) \vdash u : S\{l:=\mathtt{tail}(S\,(l))\}\Rightarrow P}{\Gamma\vdash \mathtt{get}(l,\lambda x.u) : S\Rightarrow P}\ \mathtt{Get}$$

  since

$$\begin{aligned}\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1)+1\cdot\mathtt{Get}\\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1)+1\cdot\mathtt{Get}\\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi).\end{aligned}$$

- Case $R=\mathtt{Set}$. Then, $(t)^{\mathtt{CBN}}=\mathtt{set}((l,!(r)^{\mathtt{CBN}}),(u)^{\mathtt{CBN}})$, and $t=\mathtt{set}((l,r),u)$. Moreover, Ψ must be of the form

$$\frac{\Psi_1 \rhd_{\mathcal{P}_{\mathcal{G}}} \Gamma_1 \vdash !(r)^{\mathtt{CBN}} : L \qquad \Psi_2 \rhd_{\mathcal{P}_{\mathcal{G}}} \Gamma_2 \vdash (u)^{\mathtt{CBN}} : S\{l:=L\}\Rightarrow P}{\Gamma_1+\Gamma_2\vdash \mathtt{set}((l,!(r)^{\mathtt{CBN}}),(u)^{\mathtt{CBN}}) : S\Rightarrow P}\ \mathtt{Set}$$

  and $\mathbb{A}=S\Rightarrow P$. Notice that Ψ_1 assigns a list type to the bang-term $!(r)^{\mathtt{CBN}}$. Therefore, by applying the *i.h.* to Ψ_1, we know there is $\Phi_1 \rhd_{\mathcal{N}_{\mathcal{G}}} \Gamma_1 \vdash r : L$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1)=\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1)$. By applying the *i.h.* to Ψ_2, we know there is $\Phi_2 \rhd_{\mathcal{N}_{\mathcal{G}}} \Gamma_2 \vdash u : S\{l:=L\}\Rightarrow P$, such that $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2)=\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2)$. Thus, we can take Φ to be

$$\frac{\Phi_1 \rhd_{\mathcal{N}_{\mathcal{G}}} \Gamma_1 \vdash r : L \qquad \Phi_2 \rhd_{\mathcal{N}_{\mathcal{G}}} \Gamma_2 \vdash u : S\{l:=L\}\Rightarrow P}{\Gamma_1+\Gamma_2\vdash \mathtt{set}((l,r),u) : S\Rightarrow P}\ \mathtt{Set}$$

  since

$$\begin{aligned}\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi) &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_1)+\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi_2)+1\cdot\mathtt{Set}\\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_1)+\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi_2)+1\cdot\mathtt{Set}\\ &= \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi).\end{aligned}$$

- Case $R=\mathtt{EList}$. Then, $t=(u)^{\mathtt{CBN}}$, and Ψ must be of the form

$$\frac{}{\varnothing\vdash !(u)^{\mathtt{CBN}} : [\,]}\ \mathtt{EList}$$

$\Gamma = \emptyset$, and $L = [\,]$. Therefore, we can take Φ to be

$$\frac{}{\emptyset \vdash u : [\,]}\ \texttt{EList}$$

since

$$\begin{aligned} \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Psi) &= 0 \cdot \texttt{App} + 0 \cdot \texttt{Get} + 0 \cdot \texttt{Set} \\ &= \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Phi). \end{aligned}$$

- Case $R = \texttt{List}$. Then, $t = (u)^{\texttt{CBN}}$, and Ψ must be of the form

$$\frac{\Psi_1 \rhd_{\mathcal{P}_\mathcal{G}} \Gamma_1 \vdash !(u)^{\texttt{CBN}} : M \qquad \Psi_2 \rhd_{\mathcal{P}_\mathcal{G}} \Gamma_2 \vdash !(u)^{\texttt{CBN}} : L'}{\Gamma_1 + \Gamma_2 \vdash !(u)^{\texttt{CBN}} : M \cdot L'}\ \texttt{List}$$

$\Gamma = \Gamma_1 + \Gamma_2$, and $L = M \cdot L'$. Notice that Ψ_1 assigns a multi-monadic type to the bang-term $!(r)^{\texttt{CBN}}$. Therefore, by applying the *i.h.* to Ψ_1, we know there is $\Phi_1 \rhd_{\mathcal{N}_\mathcal{G}} \Gamma_1 \vdash u : M$, such that $\#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Psi_1) = \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Phi_1)$. Now, notice that Ψ_2 assigns a list type to the bang-term $!(r)^{\texttt{CBN}}$. Hence, by applying the *i.h.* to Ψ_2, we know there is $\Phi_2 \rhd_{\mathcal{N}_\mathcal{G}} \Gamma_2 \vdash u : L'$, such that $\#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Psi_2) = \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Phi_2)$. Therefore, we can take Φ to be

$$\frac{\Phi_1 \rhd_{\mathcal{N}_\mathcal{G}} \Gamma_1 \vdash u : M \qquad \Phi_2 \rhd_{\mathcal{N}_\mathcal{G}} \Gamma_2 \vdash u : L'}{\Gamma_1 + \Gamma_2 \vdash u : M \cdot L'}\ \texttt{List}$$

since

$$\begin{aligned} \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Psi) &= \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Psi_1) + \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Psi_2) \\ &= \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Phi_1) + \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Phi_2) \\ &= \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Phi). \end{aligned}$$

□

Lemma 7.113 (Soundness and Completeness of CBN State Translations)**.** *Let $s \in \mathcal{S}$. Then $\Phi \rhd_{\mathcal{N}_\mathcal{G}} \Gamma \vdash s : S$ if and only if $\Psi \rhd_{\mathcal{P}_\mathcal{G}} \Gamma \vdash (s)^{\texttt{CBN}} : S$. Moreover, $\#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Phi) = \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Psi)$.*

Proof. The proof follows by induction on the structure of the state $s \in \mathcal{S}$. Since state typing is defined pointwise on locations, and $(\cdot)^{\texttt{CBN}}$ preserves the structure of states, the equivalence follows directly from Lemmas 7.111 and 7.112. Size preservation is immediate, since no state typing rule contributes to $\#_{\texttt{App},\texttt{Get},\texttt{Set}}(\cdot)$. □

Lemma 7.114 (Soundness of CBV State Translations)**.** *Let $s \in \mathcal{S}$. If $\Phi \rhd_{\mathcal{V}_\mathcal{G}} \Gamma \vdash s : S$, $\Psi \rhd_{\mathcal{P}_\mathcal{G}} (\Gamma)^{\texttt{CBV}} \vdash (s)^{\texttt{CBV}} : (S)^{\texttt{CBV}}$. Moreover, $\#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Phi) = \#_{\texttt{App},\texttt{Get},\texttt{Set}}(\Psi)$.*

Proof. The proof follows by induction on the structure of the state $s \in \mathcal{S}$. Since state typing is defined pointwise on locations, and $(\cdot)^{\texttt{CBV}}$ preserves the structure of states, the equivalence follows directly from Lemma 7.111. Size preservation is immediate, since no state typing rule contributes to $\#_{\texttt{App},\texttt{Get},\texttt{Set}}(\cdot)$. □

Lemma 7.115 (Soundness and Completeness of CBN Configuration Translations)**.** *Let* $t \in \Lambda^{\mathcal{G}}$ *and* $s \in \mathcal{S}$*. Then,* $\Phi \triangleright_{\mathcal{N}_{\mathcal{G}}} \Gamma \vdash (t,s) : P$ *if and only if* $\Psi \triangleright_{\mathcal{P}_{\mathcal{G}}} \Gamma \vdash ((t,s))^{\mathtt{CBN}} : P$*. Moreover,* $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi)$.

Proof.

($\Rightarrow$) Any derivation $\Phi \triangleright_{\mathcal{N}_{\mathcal{G}}} \Gamma \vdash (t,s) : P$ must end with rule `Conf`, whose premises are typings of t and s. By Lemma 7.111, the term typing translates soundly to CBPV, and by Lemma 7.113, the state typing does as well. Reapplying rule `Conf` in $\mathcal{P}_{\mathcal{G}}$ yields a derivation $\Psi \triangleright_{\mathcal{P}_{\mathcal{G}}} \Gamma \vdash ((t,s))^{\mathtt{CBN}} : P$.

($\Leftarrow$) Conversely, any typing derivation $\Psi \triangleright_{\mathcal{P}_{\mathcal{G}}} \Gamma \vdash ((t,s))^{\mathtt{CBN}} : P$ must also end with rule `Conf`. By Lemma 7.112, the CBPV term typing yields a CBN term typing, and by Lemma 7.113, the CBPV state typing yields a CBN state typing. Reapplying rule `Conf` in $\mathcal{N}_{\mathcal{G}}$ gives a derivation $\Phi \triangleright_{\mathcal{N}_{\mathcal{G}}} \Gamma \vdash (t,s) : P$.

Finally, size preservation follows immediately, since rule `Conf` does not contribute to $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\cdot)$. □

Lemma 7.116 (Soundness of CBV Configuration Translations)**.** *Let* $t \in \Lambda^{\mathcal{G}}$ *and* $s \in \mathcal{S}$*. If* $\Phi \triangleright_{\mathcal{V}_{\mathcal{G}}} \Gamma \vdash (t,s) : P$*, then* $\Psi \triangleright_{\mathcal{P}_{\mathcal{G}}} (\Gamma)^{\mathtt{CBV}} \vdash ((t,s))^{\mathtt{CBV}} : (P)^{\mathtt{CBV}}$*. Moreover,* $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Phi) = \#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\Psi)$.

Proof. Any derivation $\Phi \triangleright_{\mathcal{V}_{\mathcal{G}}} \Gamma \vdash (t,s) : P$ must end with rule `Conf`, whose premises are a typing of t and a typing of s. By Lemma 7.111, the CBV term typing translates soundly to a CBPV typing of $(t)^{\mathtt{CBV}}$, and by Lemma 7.114, the CBV state typing translates soundly to a CBPV typing of $(s)^{\mathtt{CBV}}$. Reapplying rule `Conf` in $\mathcal{P}_{\mathcal{G}}$ yields a derivation $\Psi \triangleright_{\mathcal{P}_{\mathcal{G}}} (\Gamma)^{\mathtt{CBV}} \vdash ((t,s))^{\mathtt{CBV}} : (P)^{\mathtt{CBV}}$. Finally, size preservation follows immediately, since rule `Conf` does not contribute to $\#_{\mathtt{App},\mathtt{Get},\mathtt{Set}}(\cdot)$. □

Appendix F

A Quantitative Study of Infinite Stacks

F.1 State-Passing Non-Idempotent Intersection Types

F.1.1 Call-By-Name

Lemma 8.26 (CBN Multiset-Typings Split and Merge). *Let $t \in \Lambda^{\mathcal{S}}$. Then, there is $\Phi \triangleright_{\mathcal{N}_S} \Gamma \vdash t : M$, such that $M = \sqcup_{i\in \mathtt{I}} M_i$ if and only if there are $\Phi_i \triangleright_{\mathcal{N}_S} \Gamma_i \vdash t : M_i$, such that $\Gamma = +_{i\in \mathtt{I}}\Gamma_i$, and $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = +_{i\in \mathtt{I}}\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_i)$.*

Proof. The proof is identical to that of Lemma 7.54. Indeed, the typing rules involved in multiset formation and decomposition are the same in the global-state and infinite-stack systems; the only difference lies in the operational rules for effects, which do not intervene in this lemma. □

Lemma 8.27 (Contexts and Free Variables for CBN). *If $\Phi \triangleright_{\mathcal{N}_S} \Gamma \vdash t : \mathbb{A}$ (resp. M), then $\mathtt{dom}(\Gamma) \subseteq \mathtt{fv}(t)$.*

Proof. The proof is identical to that of Lemma 7.55. Indeed, the argument depends only on the syntactic structure of typing derivations and on the formation rules for typing environments; stack-specific rules do not introduce or remove free variables any differently from global state-specific ones. □

Lemma 8.28 (Zero Weight Tight Type Derivations Characterize $\mathtt{CBN}$-Normal Forms). *Let $t \in \Lambda^{\mathcal{S}}_{\circ}$ and $s \in \mathcal{S}_{\circ}$. If $\Phi \triangleright_{\mathcal{N}_S} \emptyset \vdash (t, s) : P$ is tight, then $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = 0 \cdot \mathtt{App} + 0 \cdot \mathtt{Pop} + 0 \cdot \mathtt{Push}$ if and only if $t \in \mathtt{NO}_{\mathtt{CBN}}$.*

Proof. The proof is identical to that of Lemma 7.56, *mutatis mutandis*, replacing global state constructs with the corresponding stack constructs. □

Lemma 8.29 (Tight Typability of $\mathtt{CBN}$-Normal Forms). *Let $s \in \mathcal{S}_{\circ}$. If $t \in \mathtt{NO}_{\mathtt{CBN}}$, then there is a derivation $\Phi \triangleright_{\mathcal{N}_S} \emptyset \vdash (t, s) : P$ that is tight.*

Proof. The proof is identical to that of Lemma 7.57, *mutatis mutandis*, replacing global state constructs with the corresponding stack constructs. □

Lemma 8.30. *Let $l \in \mathcal{L}$ and $s \in \mathcal{S}$. There are $\Phi \rhd_{\mathcal{N}_S} \emptyset \vdash s : S$ and $\Psi \rhd_{\mathcal{N}_S} \emptyset \vdash t : M$ if and only if there is $\Pi \rhd_{\mathcal{N}_S} \emptyset \vdash s\{l := t \cdot s\,(l)\} : S\{l := M \cdot S\,(l)\}$. Moreover, $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi)$.*

Proof.

($\Rightarrow$) Notice that one of the premises of Φ must be derivation $\Phi_l \rhd_{\mathcal{N}_S} \emptyset \vdash s\,(l) : S\,(l)$. Now, let us assume that s is updated by replacing $s\,(l)$ with $t \cdot s\,(l)$. Then, $(s\{l := t \cdot S\,(l)\})\,(l) = t \cdot S\,(l)$. Notice that we can obtain the following derivation Φ'_l for $t \cdot S\,(l)$:

$$\frac{\Psi \rhd_{\mathcal{N}_S} \emptyset \vdash t : M \qquad \Phi_l \rhd_{\mathcal{N}_S} \emptyset \vdash s\,(l) : S\,(l)}{\emptyset \vdash t \cdot s\,(l) : M \cdot S\,(l)}\ \mathtt{List}$$

such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi'_l) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_l)$. Therefore, by replacing derivation Φ_l with Φ'_l in Φ, we obtain a derivation $\Pi' \rhd_{\mathcal{N}_S} \emptyset \vdash s\{l := t \cdot s\,(l)\} : S\{l := M \cdot S\,(l)\}$ for $s\{l := t \cdot s\,(l)\}$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi') = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) - \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_l) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi'_l)$. Therefore, we can take $\Pi = \Pi'$, since

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi) &= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_l) - \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi'_l) \\
&+ \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi) \\
&= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_l) - \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi'_l) \\
&+ \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi'_l) - \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_l) \\
&= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi).
\end{aligned}$$

($\Leftarrow$) Notice that one of the premises of Π must be derivation $\Phi_l \rhd_{\mathcal{N}_S} \emptyset \vdash (s\{l := t \cdot s\,(l)\})\,(l) : (S\{l := M \cdot S\,(l)\})\,(l)$, such that $(s\{l := t \cdot s\,(l)\})\,(l) = t \cdot s\,(l)$ and $(S\{l := M \cdot S\,(l)\})\,(l) = M \cdot S\,(l)$. Therefore, Φ_l must be of the form:

$$\frac{\Psi' \rhd_{\mathcal{N}_S} \emptyset \vdash t : M \qquad \Phi'_l \rhd_{\mathcal{N}_S} \emptyset \vdash s\,(l) : S\,(l)}{\emptyset \vdash t \cdot s\,(l) : M \cdot S\,(l)}\ \mathtt{List}$$

such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_l) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi') + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi'_l)$. Now, notice that we can build a type derivation $\Phi' \rhd_{\mathcal{N}_S} \emptyset \vdash s : S$ for s by replacing derivation Φ_l with Φ'_l in Π. Therefore, we can take $\Phi = \Phi'$ and $\Psi = \Psi'$, since

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi) &= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi') + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi') \\
&= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi) - \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_l) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi'_l) \\
&+ \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_l) - \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi'_l) \\
&= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi).
\end{aligned}$$

□

F.1.1.1 Quantitative Soundness

Lemma 8.31 (Weighted Substitution for CBN). *Let $t, u \in \Lambda^{\mathcal{S}}$. If $\Phi \triangleright_{\mathcal{N}_{\mathcal{S}}} \Gamma; x : M \vdash t : \mathbb{A}$ (resp. N) and $\Psi \triangleright_{\mathcal{N}_{\mathcal{S}}} \varnothing \vdash u : M$, then $\Pi \triangleright_{\mathcal{N}_{\mathcal{S}}} \Gamma \vdash t\{x \setminus u\} : \mathbb{A}$ (resp. N), such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi)$.*

Proof. The proof follows by induction on the structure of Φ. We proceed by cases, according to the last rule R of Φ. We only show rules Pop and Push. The remaining cases are identical to those for the CBN $\lambda^{\mathcal{G}}$-calculus (see Lemma 7.61):

- Case $R = \mathtt{Pop}$. Then, $t = \mathtt{pop}(l, \lambda y.r)$, and Φ must be of the form

$$\frac{\Phi_1 \triangleright_{\mathcal{N}_{\mathcal{S}}} \Gamma; y : \mathtt{head}(S\,(l)); x : M \vdash r : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P}{\Gamma \vdash \mathtt{pop}(l, \lambda y.r) : S \Rightarrow P}\ \mathtt{Pop}$$

  and $\mathbb{A} = S \Rightarrow P$. By applying the *i.h.* to Φ_1, we have

$$\Pi_1 \triangleright_{\mathcal{N}_{\mathcal{S}}} \Gamma; y : \mathtt{head}(S\,(l)) \vdash r\{x \setminus u\} : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P,$$

  such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_1) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi)$. Therefore, we can take Π to be

$$\frac{\Pi_1 \triangleright_{\mathcal{N}_{\mathcal{S}}} \Gamma; y : \mathtt{head}(S\,(l)) \vdash r\{x \setminus u\} : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P}{\Gamma \vdash \mathtt{pop}(l, \lambda y.r\{x \setminus u\}) : S \Rightarrow P}\ \mathtt{Pop}$$

  since $\mathtt{pop}(l, \lambda y.r\{x \setminus u\}) = \mathtt{pop}(l, \lambda y.r)\{x \setminus u\}$, and

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi) &= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_1) + 1 \cdot \mathtt{Pop} \\
&= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi) + 1 \cdot \mathtt{Pop} \\
&= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi).
\end{aligned}$$

- Case $R = \mathtt{Push}$. Then, $t = \mathtt{push}((l, e), \lambda y.r)$, and Φ must be of the form

$$\frac{\Phi_1 \triangleright_{\mathcal{N}_{\mathcal{S}}} \Gamma_1; x : M_1 \vdash e : N \qquad \Phi_2 \triangleright_{\mathcal{N}_{\mathcal{S}}} \Gamma_2; x : M_2 \vdash r : S\{l := N \cdot S\,(l)\} \Rightarrow P}{(\Gamma_1 + \Gamma_2); x : M_1 \sqcup M_2 \vdash \mathtt{push}((l, e), r) : S \Rightarrow P}\ \mathtt{Push}$$

  $\Gamma = \Gamma_1 + \Gamma_2$, $M = M_1 \sqcup M_2$, and $\mathbb{A} = S \Rightarrow P$. By Lemma 8.26, there are $\Psi_1 \triangleright_{\mathcal{N}_{\mathcal{S}}} \varnothing \vdash u : M_1$ and $\Psi_2 \triangleright_{\mathcal{N}_{\mathcal{S}}} \varnothing \vdash u : M_2$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_2)$. By applying the *i.h.* to Φ_1, we know there exists $\Pi_1 \triangleright_{\mathcal{N}_{\mathcal{S}}} \Gamma_1 \vdash e\{x \setminus u\} : N$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_1) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_1)$. By applying the *i.h.* to Φ_2, we know there exists $\Pi_2 \triangleright_{\mathcal{N}_{\mathcal{S}}} \Gamma_2 \vdash r\{x \setminus u\} : S\{l := N \cdot S\,(l)\} \Rightarrow P$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_2) =$

$\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_2) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_2)$. Therefore, we can take Π to be

$$\dfrac{\Pi_1 \rhd_{\mathcal{N}_S} \Gamma_1 \vdash e\{x \setminus u\} : N \qquad \Pi_2 \rhd_{\mathcal{N}_S} \Gamma_2 \vdash r\{x \setminus u\} : S\{l := N \cdot S\,(l)\} \Rightarrow P}{\Gamma_1 + \Gamma_2 \vdash \mathtt{push}((l, e\{x \setminus u\}), r\{x \setminus u\}) : S \Rightarrow P}\ \mathtt{Push}$$

since we have that $\mathtt{push}((l, e\{x \setminus u\}), r\{x \setminus u\}) = \mathtt{push}((l, e), r)\{x \setminus u\}$, and

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi) &= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_2) + 1 \cdot \mathtt{Push} \\
&= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_2) \\
&+ \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_2) + 1 \cdot \mathtt{Push} \\
&= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi).
\end{aligned}$$

□

Lemma 8.32 (Exact Subject Reduction for CBN)**.** *Let* $t \in \Lambda^{\mathcal{S}}_{\circ}$, $s \in \mathcal{S}_{\circ}$, *and* $S \in \{\beta, \mathtt{pop}, \mathtt{push}\}$. *If* $(t, s) \Rightarrow_{\mathtt{CBN}(S)} (u, q)$ *and* $\Phi \rhd_{\mathcal{N}_S} \emptyset \vdash (t, s) : P$, *then there is* $\Psi \rhd_{\mathcal{N}_S} \emptyset \vdash (u, q) : P$, *such that* $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi) + 1 \cdot R$, *where*

- $R = \mathtt{App}$, *if* $S = \beta$;
- $R = \mathtt{Pop}$, *if* $S = \mathtt{pop}$;
- *and* $R = \mathtt{Push}$, *if* $S = \mathtt{push}$.

Proof. The proof follows by induction on the definition of the contexts of the reduction strategy. We only show the case where $\mathtt{N} = \Box$ and t is an operation. The remaining cases are identical to those for the CBN $\lambda^{\mathcal{G}}$-calculus (see Lemma 7.62):

- Case $\mathtt{N} = \Box$:
  - If $t = \mathtt{pop}(l, \lambda x.r)$, we have that $(\mathtt{pop}(l, \lambda x.r), s) \Rightarrow_{\mathtt{CBN}(\mathtt{pop})} (r\{x \setminus \mathtt{head}(s\,(l))\}, s\{l := \mathtt{tail}(s\,(l))\})$, and Φ must be of the form

$$\dfrac{\dfrac{\Phi_1 \rhd_{\mathcal{N}_S} x : \mathtt{head}(S\,(l)) \vdash r : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P}{\emptyset \vdash \mathtt{pop}(l, \lambda x.r) : S \Rightarrow P}\ \mathtt{Pop} \qquad \Phi_2 \rhd_{\mathcal{N}_S} \emptyset \vdash s : S}{\emptyset \vdash (\mathtt{pop}(l, \lambda x.r), s) : P}\ \mathtt{Conf}$$

    Notice that $s = s\{l := \mathtt{head}(s\,(l)) \cdot \mathtt{tail}(s\,(l))\}$ and also $S = S\{l := \mathtt{head}(S\,(l)) \cdot \mathtt{tail}(S\,(l))\}$. Thus, by applying Lemma 8.30 to Φ_2, there are $\Pi_1 \rhd_{\mathcal{N}_S} \emptyset \vdash \mathtt{head}(s\,(l)) : \mathtt{head}(S\,(l))$ and $\Psi_2 \rhd_{\mathcal{N}_S} \emptyset \vdash s\{l := \mathtt{tail}(s\,(l))\} : S\{l := \mathtt{tail}(S\,(l))\}$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_2) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_2)$. By applying Lemma 8.31 to Φ_1 and Π_1, there is $\Psi_1 \rhd_{\mathcal{N}_S} \emptyset \vdash r\{x \setminus \mathtt{head}(s\,(l))\} : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P$, such that

$\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_1) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_1)$.Therefore, we can take Ψ to be

$$\dfrac{\begin{array}{c}\Psi_1 \triangleright_{\mathcal{N}_S} \varnothing \vdash r\{x \setminus \mathtt{head}(s\,(l))\} : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P \\ \Psi_2 \triangleright_{\mathcal{N}_S} \varnothing \vdash s\{l := \mathtt{tail}(S\,(l))\} : S\{l := \mathtt{tail}(S\,(l))\}\end{array}}{\varnothing \vdash (r\{x \setminus \mathtt{head}(s\,(l))\}, s\{l := \mathtt{tail}(S\,(l))\}) : P}\ \mathtt{Conf}$$

since

$$\begin{array}{rcl}\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) & = & \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_2) + 1 \cdot \mathtt{Pop} \\ & = & \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_2) - \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_1) \\ & + & \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_1) + 1 \cdot \mathtt{Pop} \\ & = & \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_2) + 1 \cdot \mathtt{Pop} \\ & = & \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi) + 1 \cdot \mathtt{Pop}.\end{array}$$

– If $t = \mathtt{push}((l,e),r)$, then $(\mathtt{push}((l,e),r),s) \Rightarrow_{\mathtt{CBN}(\mathtt{push})} (r, s\{l := e \cdot s\,(l)\})$, and Φ must be of the form

$$\dfrac{\dfrac{\Phi_1 \triangleright_{\mathcal{N}_S} \varnothing \vdash e : M \quad \Phi_2 \triangleright_{\mathcal{N}_S} \varnothing \vdash r : S\{l := M \cdot S\,(l)\} \Rightarrow P}{\varnothing \vdash \mathtt{push}((l,e),r) : S \Rightarrow P}\ \mathtt{Push} \qquad \Phi_3 \triangleright_{\mathcal{N}_S} \varnothing \vdash s : S}{\varnothing \vdash (\mathtt{push}((l,e),r),s) : P}\ \mathtt{Conf}$$

By applying Lemma 8.30 to Φ_3 and Φ_1, there is $\Psi_3 \triangleright_{\mathcal{N}_S} \varnothing \vdash s\{l := e \cdot s\,(l)\} : S\{l := M \cdot S\,(l)\}$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_3) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_1) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_3)$. Therefore, we can take Ψ to be

$$\dfrac{\Phi_2 \triangleright_{\mathcal{N}_S} \varnothing \vdash r : S\{l := M \cdot S\,(l)\} \Rightarrow P \quad \Psi_3 \triangleright_{\mathcal{N}_S} \varnothing \vdash s\{l := e \cdot s\,(l)\} : S\{l := M \cdot S\,(l)\}}{\varnothing \vdash (r, s\{l := e \cdot s\,(l)\}) : P}\ \mathtt{Conf}$$

since

$$\begin{array}{rcl}\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) & = & \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_2) + 1 \cdot \mathtt{Push} + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_3) \\ & = & \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_2) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_3) + 1 \cdot \mathtt{Push} \\ & = & \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi) + 1 \cdot \mathtt{Push}.\end{array}$$

□

Theorem 8.33 (Exact Soundness for CBN). *Let $t \in \Lambda^{\mathcal{S}}_{\circ}$ and $s \in \mathcal{S}_{\circ}$. If there is $\Phi \triangleright_{\mathcal{N}_S} \varnothing \vdash (t,s) : P$ tight, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = n \cdot \mathtt{App} + o \cdot \mathtt{Pop} + u \cdot \mathtt{Push}$, then $(t,s) \overset{n\cdot\beta + o\cdot\mathtt{pop} + u\cdot\mathtt{push}}{\Rightarrow\!\!\!\!\!\!\Rightarrow}_{\mathtt{CBN}} (u,q)$, such that $u \in \mathtt{NO}_{\mathtt{CBN}}$.*

Proof. The proof is identical to that of Theorem 7.63, *mutatis mutandis*. The argument proceeds by induction on $n + o + u$, using exact subject reduction (Lemma 8.32) and the characterization of zero-weight derivations (Lemma 8.28). The only difference is that global state operations and typing rules are replaced by the corresponding stack operations and rules. □

F.1.1.2 Quantitative Completeness

Lemma 8.34 (Weighted Anti-Substitution for CBN). *Let $t \in \Lambda^{\mathcal{S}}$ and $u \in \Lambda^{\mathcal{S}}_{\circ}$. If $\Phi \rhd_{\mathcal{N}_S} \Gamma \vdash t\{x \setminus u\} : \mathbb{A}$ (resp. N), then there exist $\Psi \rhd_{\mathcal{N}_S} \Gamma; x : M \vdash t : \mathbb{A}$ (resp. N) and $\Pi \rhd_{\mathcal{N}_S} \emptyset \vdash u : M$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi)$.*

Proof. The proof follows by induction on the structure of t. We only show the cases where t is an operation. The remaining cases are identical to those for the CBN $\lambda^{\mathcal{G}}$-calculus (see Lemma 7.64):

- Case $t = \mathtt{pop}(l, \lambda y.r)$. Then, $t\{x \setminus u\} = \mathtt{pop}(l, \lambda y.r\{x \setminus u\})$, and Φ must be of the form:

$$\frac{\Phi' \rhd_{\mathcal{N}_S} \Gamma; y : \mathtt{head}(S\,(l)) \vdash r\{x \setminus u\} : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P}{\Gamma \vdash \mathtt{pop}(l, \lambda y.r\{x \setminus u\}) : S \Rightarrow P}\ \mathtt{Pop}$$

  and $\mathbb{A} = S \Rightarrow P$. By applying the *i.h.* to r, we know there exist $\Psi' \rhd_{\mathcal{N}_S} \Gamma; y : \mathtt{head}(S\,(l)); x : M \vdash r : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P$ and $\Pi' \rhd_{\mathcal{N}_S} \emptyset \vdash u : M$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi') = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi') + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi')$. Therefore, we can conclude by taking $\Pi = \Pi'$ and Ψ to be

$$\frac{\Psi' \rhd_{\mathcal{N}_S} \Gamma; y : \mathtt{head}(S\,(l)); x : M \vdash r : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P}{\Gamma; x : M \vdash \mathtt{pop}(l, \lambda y.r) : S \Rightarrow P}\ \mathtt{Pop}$$

  since

$$\begin{aligned} \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) &= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi') + 1 \cdot \mathtt{Pop} \\ &= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi') + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi') + 1 \cdot \mathtt{Pop} \\ &= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi). \end{aligned}$$

- Case $t = \mathtt{push}((l, e), \lambda y.r)$. Then, $t\{x \setminus u\} = \mathtt{push}((l, e\{x \setminus u\}), \lambda y.r\{x \setminus u\})$, and Φ must be of the form

$$\frac{\Phi_1 \rhd_{\mathcal{N}_S} \Gamma_1 \vdash e\{x \setminus u\} : N \qquad \Phi_2 \rhd_{\mathcal{N}_S} \Gamma_2 \vdash r\{x \setminus u\} : S\{l := N \cdot S\,(l)\} \Rightarrow P}{\Gamma_1 + \Gamma_2 \vdash \mathtt{push}((l, e\{x \setminus u\}), r\{x \setminus u\}) : S \Rightarrow P}\ \mathtt{Push}$$

  $\Gamma = \Gamma_1 + \Gamma_2$, and $\mathbb{A} = S \Rightarrow P$. By applying the *i.h.* to e, we know there exist $\Psi_1 \rhd_{\mathcal{N}_S} \Gamma_1; x : M_1 \vdash e : N$ and $\Pi_1 \rhd_{\mathcal{N}_S} \emptyset \vdash u : M_1$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_1) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_1)$. By applying the *i.h.* to r, we know there exist $\Psi_2 \rhd_{\mathcal{N}_S} \Gamma_2; x : M_2 \vdash r : S\{l := N \cdot S\,(l)\} \Rightarrow P$ and $\Pi_2 \rhd_{\mathcal{N}_S} \emptyset \vdash u : M_2$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_2) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_2) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_2)$. By Lemma 8.26, we know there exists $\Pi' \rhd_{\mathcal{N}_S} \emptyset \vdash u : M_1 \sqcup M_2$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi') = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_2)$. Therefore, we can conclude by taking $M = M_1 \sqcup M_2$, $\Pi = \Pi'$ and Ψ to be

$$\frac{\Psi_1 \rhd_{\mathcal{N}_S} \Gamma_1; x : M_1 \vdash e : N \qquad \Psi_2 \rhd_{\mathcal{N}_S} \Gamma_2; x : M_2 \vdash r : S\{l := N \cdot S\,(l)\} \Rightarrow P}{(\Gamma_1 + \Gamma_2); x : M_1 \sqcup M_2 \vdash \mathtt{push}((l, e), r) : S \Rightarrow P}\ \mathtt{Push}$$

since

$$\begin{aligned}\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) &= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_2) + 1 \cdot \mathtt{Push} \\ &= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_2) \\ &+ \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_2) + 1 \cdot \mathtt{Push} \\ &= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi).\end{aligned}$$

□

Lemma 8.35 (Exact Subject Expansion for CBN)**.** *Let* $t \in \Lambda^{\mathcal{S}}_{\circ}$, $s \in \mathcal{S}_{\circ}$, *and* $S \in \{\beta, \mathtt{pop}, \mathtt{push}\}$. *If* $(t,s) \Rightarrow_{\mathtt{CBN}(S)} (u,q)$ *and* $\Phi \rhd_{\mathcal{N}_S} \emptyset \vdash (u,q) : P$, *then there is* $\Psi \rhd_{\mathcal{N}_S} \emptyset \vdash (t,s) : P$, *such that* $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi) + 1 \cdot R = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi)$, *where*

- $R = \mathtt{App}$, *if* $S = \beta$;
- $R = \mathtt{Pop}$, *if* $S = \mathtt{pop}$;
- *and* $R = \mathtt{Push}$, *if* $S = \mathtt{push}$.

Proof. The proof follows by induction on the definition of the contexts of the reduction strategy. We only show the case where $\mathtt{N} = \Box$ and t is an operation. The remaining cases follow easily from the *i.h.*:

- Case $\mathtt{N} = \Box$:
  - If $t = \mathtt{pop}(l, \lambda x.r)$, we have that $(\mathtt{pop}(l, \lambda x.r), s) \Rightarrow_{\mathtt{CBN}(\mathtt{pop})} (r\{x \setminus \mathtt{head}(s\,(l))\}, s\{l := \mathtt{tail}(s\,(l))\})$, and Φ is of the form
    $$\dfrac{\Phi_1 \rhd_{\mathcal{N}_S} \emptyset \vdash r\{x \setminus \mathtt{head}(s\,(l))\} : S \Rightarrow P \qquad \Phi_2 \rhd_{\mathcal{N}_S} \emptyset \vdash s\{l := \mathtt{tail}(s\,(l))\} : S}{\emptyset \vdash (r\{x \setminus \mathtt{head}(s\,(l))\}, s\{l := \mathtt{tail}(s\,(l))\}) : P}\ \mathtt{Conf}$$
    By applying Lemma 8.34 to Φ_1, there exist $\Psi_1 \rhd_{\mathcal{N}_S} x : M \vdash r : S \Rightarrow P$ and $\Pi_2 \rhd_{\mathcal{N}_S} \emptyset \vdash \mathtt{head}(s\,(l)) : M$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_1) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_2)$. Let $R = S\{l := M \cdot S\,(l)\}$. Then, $\mathtt{head}(R\,(l)) = M$, $\mathtt{tail}(R\,(l)) = S\,(l)$, and $R\{l := \mathtt{tail}(R\,(l))\} = S$. Thus, by applying Lemma 8.30 to Π_2 and Φ_2, there is $\Psi_2 \rhd_{\mathcal{N}_S} \emptyset \vdash s : R$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_2) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_2) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_2)$. Therefore, we can take Ψ to be
    $$\dfrac{\dfrac{\Psi_1 \rhd_{\mathcal{N}_S} x : \mathtt{head}(R\,(l)) \vdash r : R\{l := \mathtt{tail}(R\,(l))\} \Rightarrow P}{\emptyset \vdash \mathtt{pop}(l, \lambda x.r) : R \Rightarrow P}\ \mathtt{Pop} \qquad \Psi_2 \rhd_{\mathcal{N}_S} \emptyset \vdash s : R}{\emptyset \vdash (\mathtt{pop}(l, \lambda x.r), s) : P}\ \mathtt{Conf}$$

since

$$\begin{aligned}\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) + 1 \cdot \mathtt{Pop} &= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_2) + 1 \cdot \mathtt{Pop} \\ &= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_2) \\ &+ \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_2) + 1 \cdot \mathtt{Pop} \\ &= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_2) + 1 \cdot \mathtt{Pop} \\ &= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi).\end{aligned}$$

– If $t = \mathtt{push}((l,e),r)$, then $(\mathtt{push}((l,e),r),s) \Rightarrow_{\mathtt{CBN}(\mathtt{push})} (r, s\{l := e \cdot s\,(l)\})$, and Φ is of the form

$$\frac{\Phi_1 \rhd_{\mathcal{N}_S} \varnothing \vdash r : S \Rightarrow P \qquad \Phi_2 \rhd_{\mathcal{N}_S} \varnothing \vdash s\{l := e \cdot s\,(l)\} : S}{\varnothing \vdash (r, s\{l := e \cdot s\,(l)\}) : P}\ \mathtt{Conf}$$

Let $R = S\{l := L\}$ and $R\{l := S\,(l)\} = R\{l := M \cdot L\} = R\{l := M \cdot R\,(l)\} = S$. Then, by applying Lemma 8.30 to Φ_2, there are $\Pi_1 \rhd_{\mathcal{N}_S} \varnothing \vdash e : M$ and $\Psi_2 \rhd_{\mathcal{N}_S} \varnothing \vdash s : R$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_2) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_1) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_2)$. Therefore, we can take Ψ to be

$$\frac{\dfrac{\Pi_1 \rhd_{\mathcal{N}_S} \varnothing \vdash e : M \qquad \Phi_1 \rhd_{\mathcal{N}_S} \varnothing \vdash r : R\{l := M \cdot R\,(l)\} \Rightarrow P}{\varnothing \vdash \mathtt{push}((l,e),r) : R \Rightarrow P}\ \mathtt{Push} \qquad \Psi_2 \rhd_{\mathcal{N}_S} \varnothing \vdash s : R}{\varnothing \vdash (\mathtt{push}((l,e),r),s) : P}\ \mathtt{Conf}$$

since

$$\begin{aligned}\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) + 1 \cdot \mathtt{Push} &= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_2) + 1 \cdot \mathtt{Push} \\ &= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_2) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_1) \\ &+ 1 \cdot \mathtt{Push} \\ &= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi).\end{aligned}$$

□

Theorem 8.36 (Exact Completeness for CBN). *Let $t \in \Lambda^{\mathcal{S}}_{\circ}$ and $s \in \mathcal{S}_{\circ}$. If $(t,s) \overset{n\cdot\beta+o\cdot\mathtt{pop}+u\cdot\mathtt{push}}{\Rightarrow\!\!\!\!\Rightarrow_{\mathtt{CBN}}} (u,q)$, such that $u \in \mathtt{NO}_{\mathtt{CBN}}$, then there is $\Phi \rhd_{\mathcal{N}_S} \varnothing \vdash (t,s) : P$ tight, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = n \cdot \mathtt{App} + o \cdot \mathtt{Pop} + u \cdot \mathtt{Push}$.*

Proof. The proof is identical to that of Theorem 7.66, *mutatis mutandis*. The argument proceeds by induction on $n + o + u$, using exact subject expansion (Lemma 8.35) and the tight typability of CBN-normal forms (Lemma 8.29). The only difference is that global state operations and typing rules are replaced by the corresponding stack operations and rules. □

F.1.2 Call-By-Value

Lemma 8.40 (CBV Multiset-Typings Split and Merge). *Let $t \in \Lambda^{\mathcal{S}}$. Then, there is $\Phi \triangleright_{\mathcal{V}_S} \Gamma \vdash v : M$, such that $M = \sqcup_{i\in \mathtt{I}} M_i$, if and only if there are $\Phi_i \triangleright_{\mathcal{V}_S} \Gamma_i \vdash v : M_i$, such that $\Gamma = +_{i\in \mathtt{I}}\Gamma_i$, and $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = +_{i\in \mathtt{I}}\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_i)$.*

Proof. The proof is identical to that of Lemma 7.73. Indeed, the typing rules involved in multiset formation and decomposition are the same in the global-state and infinite-stack systems; the only difference lies in the operational rules for effects, which do not intervene in this lemma. □

Lemma 8.41 (Contexts and Free Variables for CBV). *If $\Phi \triangleright_{\mathcal{V}_S} \Gamma \vdash t : \mathbb{M}$ (resp. M), then $\mathtt{dom}(\Gamma) \subseteq \mathtt{fv}(t)$.*

Proof. The proof is identical to that of Lemma 7.74. Indeed, the argument depends only on the syntactic structure of typing derivations and on the formation rules for typing environments; stack-specific rules do not introduce or remove free variables any differently from global state-specific ones. □

Lemma 8.42 (CBV Values Are Not Effectful). *Let $t \in \Lambda^{\mathcal{S}}$ and $\Phi \triangleright_{\mathcal{V}_S} \Gamma \vdash t : S \Rightarrow (M \times Q)$. If $t \in \mathrm{V}^{\mathcal{S}}$, then $Q = S$ and Φ ends with rule $\mathtt{Lift}$.*

Proof. The proof is identical to that of the corresponding lemma for the global state calculus (see Lemma 7.75), *mutatis mutandis*. The argument relies only on the syntactic form of values and on the shape of the CBV typing rules, and is independent of the particular effect operations (global state or stacks). □

Lemma 8.43 (Zero Weight Tight Type Derivations Characterize $\mathtt{CBV}$-Normal Forms). *Let $t \in \Lambda^{\mathcal{S}}_{\circ}$ and $s \in \mathcal{S}_{\circ}$. If $\Phi \triangleright_{\mathcal{V}_S} \emptyset \vdash (t, s) : P$ is tight, then $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = 0 \cdot \mathtt{App} + 0 \cdot \mathtt{Pop} + 0 \cdot \mathtt{Push}$ if and only if $t \in \mathtt{NO}_{\mathtt{CBV}}$.*

Proof. The proof is identical to that of the corresponding lemma for the global state calculus (see Lemma 7.76), *mutatis mutandis*, replacing global state constructs with the corresponding stack constructs. □

Lemma 8.44 (Tight Typability of $\mathtt{CBV}$-Normal Forms). *Let $s \in \mathcal{S}_{\circ}$. If $t \in \mathtt{NO}_{\mathtt{CBV}}$, then there is a derivation $\Phi \triangleright_{\mathcal{V}_S} \emptyset \vdash (t, s) : P$ that is tight.*

Proof. If $t \in \mathtt{NO}_{\mathtt{CBV}}$, then $t \in \mathrm{V}^{\mathcal{S}}_{\circ}$. Therefore, t is a closed abstraction, *i.e.* $t = \lambda x.u$, and we can take $P = \{\!\{\ \}\!\} \times \{l \mapsto [\,]\}_{l \in \mathcal{L}}$, and Φ to be

$$\dfrac{\dfrac{}{\varnothing \vdash \lambda x.u : \{l \mapsto [\,]\}_{l\in\mathcal{L}} \Rightarrow (\{\!\{\ \}\!\} \times S)}\ \mathtt{Abs} \qquad \dfrac{\left(\dfrac{}{\varnothing \vdash s\,(l) : [\,]}\ \mathtt{EStack}\right)_{l\in\mathcal{L}}}{\varnothing \vdash \{l \mapsto s\,(l)\}_{l\in\mathcal{L}} : \{l \mapsto [\,]\}_{l\in\mathcal{L}}}\ \mathtt{State}}{\varnothing \vdash (\lambda x.u, s) : \{\!\{\ \}\!\} \times S}\ \mathtt{Conf}$$

We can conclude since the above type derivation is tight. □

Lemma 8.45. *Let $l \in \mathcal{L}$ and $s \in \mathcal{S}$. There are $\Phi \triangleright_{\mathcal{V}_S} \varnothing \vdash s : S$ and $\Psi \triangleright_{\mathcal{V}_S} \varnothing \vdash v : M$ if and only if there is $\Pi \triangleright_{\mathcal{V}_S} \varnothing \vdash s\{l := v \cdot s\,(l)\} : S\{l := M \cdot S\,(l)\}$. Moreover, $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi)$.*

Proof. The proof is identical to that of Lemma 8.30, *mutatis mutandis*. The only difference is that in the CBV system the pushed term is required to be a value, which does not affect the argument. □

F.1.2.1 Quantitative Soundness

Lemma 8.46 (Weighted Substitution for CBV)**.** *Let $t \in \Lambda^{\mathcal{S}}_{\circ}$, and $v \in \mathrm{V}^{\mathcal{S}}$. If $\Phi \triangleright_{\mathcal{V}_S} \Gamma; x : M \vdash t : \mathbb{M}$ (resp. N) and $\Psi \triangleright_{\mathcal{V}_S} \varnothing \vdash v : M$, then $\Pi \triangleright_{\mathcal{V}_S} \Gamma \vdash t\{x \setminus v\} : \mathbb{M}$ (resp. N), such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi)$.*

Proof. The proof follows by induction on the structure of Φ. We proceed by cases, according to the last rule R of Φ. We only show rules Pop and Push. The remaining cases are identical to those for the CBN $\lambda^{\mathcal{G}}$-calculus (see Lemma 7.81):

- Case $R = \mathtt{Pop}$. Then, $t = \mathtt{pop}(l, \lambda y.u)$, and Φ must be of the form

  $$\dfrac{\Phi_1 \triangleright_{\mathcal{V}_S} \Gamma; y : \mathtt{head}(S\,(l)); x : M \vdash u : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P}{\Gamma \vdash \mathtt{pop}(l, \lambda y.u) : S \Rightarrow P}\ \mathtt{Pop}$$

  and $\mathbb{M} = S \Rightarrow P$. By applying the *i.h.* to Φ_1, there is $\Pi_1 \triangleright_{\mathcal{V}_S} \Gamma; y : \mathtt{head}(S\,(l)) \vdash u\{x \setminus v\} : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_1) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi)$. Therefore, we can take Π to be

  $$\dfrac{\Pi_1 \triangleright_{\mathcal{V}_S} \Gamma; y : \mathtt{head}(S\,(l)) \vdash u\{x \setminus v\} : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P}{\Gamma \vdash \mathtt{pop}(l, \lambda y.u\{x \setminus v\}) : S \Rightarrow P}\ \mathtt{Pop}$$

  since $\mathtt{pop}(l, \lambda y.u\{x \setminus v\}) = \mathtt{pop}(l, \lambda y.u)\{x \setminus v\}$, and

  $$\begin{aligned} \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi) &= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_1) + 1 \cdot \mathtt{Pop} \\ &= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi) + 1 \cdot \mathtt{Pop} \\ &= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi). \end{aligned}$$

- Case $R = \mathtt{Push}$. Then, $t = \mathtt{push}((l,w),\lambda y.r)$, and Φ must be of the form

$$\frac{\Phi_1 \rhd_{\mathcal{V}_S} \Gamma_1; x : M_1 \vdash w : N \qquad \Phi_2 \rhd_{\mathcal{V}_S} \Gamma_2; x : M_2 \vdash r : S\{l := N \cdot S\,(l)\} \Rightarrow P}{(\Gamma_1 + \Gamma_2); x : M_1 \sqcup M_2 \vdash \mathtt{push}((l,w),r) : S \Rightarrow P}\ \mathtt{Push}$$

  $\Gamma = \Gamma_1 + \Gamma_2$, $M = M_1 \sqcup M_2$, and $\mathbb{M} = S \Rightarrow P$. By applying the *i.h.* to Φ_1, we know there exists $\Pi_1 \rhd_{\mathcal{V}_S} \Gamma_1 \vdash w\{x \setminus v\} : N$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_1) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_1)$. By applying the *i.h.* to Φ_2, we know there exists $\Pi_2 \rhd_{\mathcal{V}_S} \Gamma_2 \vdash r\{x \setminus v\} : S\{l := N \cdot S\,(l)\} \Rightarrow P$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_2) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_2) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_2)$. Therefore, we can take Π to be

$$\frac{\Pi_1 \rhd_{\mathcal{V}_S} \Gamma_1 \vdash w\{x \setminus v\} : N \qquad \Pi_2 \rhd_{\mathcal{V}_S} \Gamma_2 \vdash r\{x \setminus v\} : S\{l := N \cdot S\,(l)\} \Rightarrow P}{\Gamma_1 + \Gamma_2 \vdash \mathtt{push}((l,w\{x \setminus v\}),r\{x \setminus v\}) : S \Rightarrow P}\ \mathtt{Push}$$

  since substitution is well-defined on values, values are preserved by substitution, and we have that $\mathtt{push}((l,w\{x \setminus u\}),r\{x \setminus v\}) = \mathtt{push}((l,w),r)\{x \setminus v\}$, and

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi) &= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_2) + 1 \cdot \mathtt{Push} \\
&= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_1) \\
&+ \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_2) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_2) + 1 \cdot \mathtt{Push} \\
&= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_2) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi) + 1 \cdot \mathtt{Push} \\
&= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi).
\end{aligned}$$

□

Lemma 8.47 (Exact Subject Reduction for CBV)**.** *Let $t \in \Lambda^{\mathcal{S}}_{\circ}$, $s \in \mathcal{S}_{\circ}$, and $S \in \{\beta_{\mathtt{v}}, \mathtt{pop}, \mathtt{push}\}$. If $(t,s) \Rightarrow_{\mathtt{CBV}(S)} (u,q)$ and $\Phi \rhd_{\mathcal{V}_S} \emptyset \vdash (t,s) : P$, then there is $\Psi \rhd_{\mathcal{V}_S} \emptyset \vdash (u,q) : P$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi) + 1 \cdot R$, where*

- $R = \mathtt{App}$, *if* $S = \beta_{\mathtt{v}}$;
- $R = \mathtt{Pop}$, *if* $S = \mathtt{pop}$;
- *and* $R = \mathtt{Push}$, *if* $S = \mathtt{push}$.

Proof. The proof follows by induction on the definition of the contexts of the reduction strategy. We only show the case where $\mathtt{V} = \Box$ and t is an operation. The remaining cases are identical to those for the CBV $\lambda^{\mathcal{G}}$-calculus (see Lemma 7.82):

- Case $\mathtt{V} = \Box$:

– If $t = \mathtt{pop}(l, \lambda x.r)$, we have that $(\mathtt{pop}(l, \lambda x.r), s) \Rightarrow_{\mathtt{CBV(pop)}} (r\{x \setminus \mathtt{head}(s\,(l))\}, s\{l := \mathtt{tail}(s\,(l))\})$, and Φ must be of the form

$$\dfrac{\dfrac{\Phi_1 \rhd_{\mathcal{V}_S} x : \mathtt{head}(S\,(l)) \vdash r : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P}{\varnothing \vdash \mathtt{pop}(l, \lambda x.r) : S \Rightarrow P}\,\mathtt{Pop} \qquad \Phi_2 \rhd_{\mathcal{V}_S} \varnothing \vdash s : S}{\varnothing \vdash (\mathtt{pop}(l, \lambda x.r), s) : P}\,\mathtt{Conf}$$

Notice that $s = s\{l := \mathtt{head}(s\,(l)) \cdot \mathtt{tail}(s\,(l))\}$, and also $S = S\{l := \mathtt{head}(S\,(l)) \cdot \mathtt{tail}(S\,(l))\}$. Thus, by applying Lemma 8.45 to Φ_2, there are $\Pi_1 \rhd_{\mathcal{V}_S} \varnothing \vdash \mathtt{head}(s\,(l)) : \mathtt{head}(S\,(l))$ and $\Psi_2 \rhd_{\mathcal{V}_S} \varnothing \vdash s\{l := \mathtt{tail}(s\,(l))\} : S\{l := \mathtt{tail}(S\,(l))\}$, such that $\#_{\mathtt{App,Pop,Push}}(\Phi_2) = \#_{\mathtt{App,Pop,Push}}(\Pi_1) + \#_{\mathtt{App,Pop,Push}}(\Psi_2)$. By applying Lemma 7.81 to Φ_1 and Π_1, there is $\Psi_1 \rhd_{\mathcal{V}_S} \varnothing \vdash r\{x \setminus \mathtt{head}(s\,(l))\} : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P$, such that $\#_{\mathtt{App,Pop,Push}}(\Psi_1) = \#_{\mathtt{App,Pop,Push}}(\Phi_1) + \#_{\mathtt{App,Pop,Push}}(\Pi_1)$. Therefore, we can take Ψ to be

$$\dfrac{\begin{array}{c}\Psi_1 \rhd_{\mathcal{V}_S} \varnothing \vdash r\{x \setminus \mathtt{head}(s\,(l))\} : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P \\ \Psi_2 \rhd_{\mathcal{V}_S} \varnothing \vdash s\{l := \mathtt{tail}(s\,(l))\} : S\{l := \mathtt{tail}(S\,(l))\}\end{array}}{\varnothing \vdash (r\{x \setminus \mathtt{head}(s\,(l))\}, s\{l := \mathtt{tail}(s\,(l))\}) : P}\,\mathtt{Conf}$$

since

$$\begin{aligned}\#_{\mathtt{App,Pop,Push}}(\Phi) &= \#_{\mathtt{App,Pop,Push}}(\Phi_1) + \#_{\mathtt{App,Pop,Push}}(\Phi_2) + 1 \cdot \mathtt{Pop} \\ &= \#_{\mathtt{App,Pop,Push}}(\Phi_1) + \#_{\mathtt{App,Pop,Push}}(\Pi_1) + \#_{\mathtt{App,Pop,Push}}(\Psi_2) + 1 \cdot \mathtt{Pop} \\ &= \#_{\mathtt{App,Pop,Push}}(\Psi_1) + \#_{\mathtt{App,Pop,Push}}(\Psi_2) + 1 \cdot \mathtt{Pop} \\ &= \#_{\mathtt{App,Pop,Push}}(\Psi) + 1 \cdot \mathtt{Pop},\end{aligned}$$

and $R = \mathtt{Pop}$.

– If $t = \mathtt{push}((l, v), r)$, then $(\mathtt{push}((l, v), r), s) \Rightarrow_{\mathtt{CBV(push)}} (r, s\{l := v \cdot s\,(l)\})$, and Φ must be of the form

$$\dfrac{\dfrac{\Phi_1 \rhd_{\mathcal{V}_S} \varnothing \vdash v : M \quad \Phi_2 \rhd_{\mathcal{V}_S} \varnothing \vdash r : S\{l := M \cdot S\,(l)\} \Rightarrow P}{\varnothing \vdash \mathtt{push}((l, v), r) : S \Rightarrow P}\,\mathtt{Push} \qquad \Phi_3 \rhd_{\mathcal{V}_S} \varnothing \vdash s : S}{\varnothing \vdash (\mathtt{push}((l, v), r), s) : P}\,\mathtt{Conf}$$

By applying Lemma 8.45 to Φ_3 and Φ_1, there is $\Psi_3 \rhd_{\mathcal{V}_S} \varnothing \vdash s\{l := v \cdot s\,(l)\} : S\{l := M \cdot S\,(l)\}$, such that $\#_{\mathtt{App,Pop,Push}}(\Phi_3) + \#_{\mathtt{App,Pop,Push}}(\Phi_1) = \#_{\mathtt{App,Pop,Push}}(\Psi_3)$. Therefore, we can take Ψ to be

$$\dfrac{\Phi_2 \rhd_{\mathcal{V}_S} \varnothing \vdash r : S\{l := M \cdot S\,(l)\} \Rightarrow P \quad \Psi_3 \rhd_{\mathcal{V}_S} \varnothing \vdash s\{l := v \cdot s\,(l)\} : S\{l := M \cdot S\,(l)\}}{\varnothing \vdash (r, s\{l := v\}) : P}\,\mathtt{Conf}$$

since

$$\begin{aligned}\#_{\mathtt{App,Pop,Push}}(\Phi) &= \#_{\mathtt{App,Pop,Push}}(\Phi_1) + \#_{\mathtt{App,Pop,Push}}(\Phi_2) + 1 \cdot \mathtt{Push} + \#_{\mathtt{App,Pop,Push}}(\Phi_3) \\ &= \#_{\mathtt{App,Pop,Push}}(\Phi_2) + \#_{\mathtt{App,Pop,Push}}(\Psi_3) + 1 \cdot \mathtt{Push} \\ &= \#_{\mathtt{App,Pop,Push}}(\Psi) + 1 \cdot \mathtt{Push},\end{aligned}$$

and $R = \mathtt{Push}$.

□

Theorem 8.48 (Exact Soundness for CBV). *Let $t \in \Lambda^{\mathcal{S}}_{\circ}$. If there is $\Phi \rhd_{\mathcal{V}_{\mathcal{S}}} \varnothing \vdash (t,s) : P$ tight, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = n \cdot \mathtt{App} + o \cdot \mathtt{Pop} + u \cdot \mathtt{Push}$, then $(t,s) \overset{n\cdot\beta_{\mathtt{v}}+o\cdot\mathtt{pop}+u\cdot\mathtt{push}}{\Rightarrow}_{\mathtt{CBV}} (u,q)$, such that $u \in \mathtt{NO}_{\mathtt{CBV}}$.*

Proof. The proof is identical to that of Theorem 7.83, *mutatis mutandis*. It proceeds by induction on $n+o+u$, using exact subject reduction (Lemma 8.47) and the characterization of zero-weight derivations (Lemma 8.43). The only difference is that global state operations and typing rules are replaced by the corresponding stack operations and rules. □

F.1.2.2 Quantitative Completeness

Lemma 8.49 (Weighted Anti-Substitution for CBV). *Let $t \in \Lambda^{\mathcal{S}}$ and $v \in \mathrm{V}^{\mathcal{S}}_{\circ}$. If $\Phi \rhd_{\mathcal{V}_{\mathcal{S}}} \Gamma \vdash t\{x \setminus v\} : \mathbb{M}$ (resp. N), then $\Psi \rhd_{\mathcal{V}_{\mathcal{S}}} \Gamma; x : M \vdash t : \mathbb{M}$ (resp. N) and $\Pi \rhd_{\mathcal{V}_{\mathcal{S}}} \varnothing \vdash v : M$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi)$.*

Proof. The proof follows by induction on the structure of Φ. We only show the cases where t is an operation. The remaining cases are identical to those for the CBV $\lambda^{\mathcal{G}}$-calculus (see Lemma 7.84):

- Case $t = \mathtt{pop}(l, \lambda y.u)$. Then, $t\{x \setminus v\} = \mathtt{pop}(l, \lambda y.u\{x \setminus v\})$, and Φ must be of the form:

$$\frac{\Phi' \rhd_{\mathcal{V}_{\mathcal{S}}} \Gamma; y : \mathtt{head}(S\,(l)) \vdash u\{x \setminus v\} : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P}{\Gamma \vdash \mathtt{pop}(l, \lambda y.u\{x \setminus v\}) : S \Rightarrow P}\ \mathtt{Pop}$$

  and $\mathbb{M} = S \Rightarrow P$. By applying the *i.h.* to u, we know $\Psi' \rhd_{\mathcal{V}_{\mathcal{S}}} \Gamma; y : \mathtt{head}(S\,(l)); x : M \vdash u : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P$ exists and $\Pi' \rhd_{\mathcal{N}_{\mathcal{S}}} \varnothing \vdash v : M$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi') = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi') + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi')$. Therefore, we can conclude by taking $\Pi = \Pi'$ and Ψ to be

$$\frac{\Psi' \rhd_{\mathcal{V}_{\mathcal{S}}} \Gamma; y : \mathtt{head}(S\,(l)); x : M \vdash u : S\{l := \mathtt{tail}(S\,(l))\} \Rightarrow P}{\Gamma; x : M \vdash \mathtt{pop}(l, \lambda y.u) : S \Rightarrow P}\ \mathtt{Pop}$$

  since

$$\begin{aligned} \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) &= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi') + 1 \cdot \mathtt{Pop} \\ &= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi') + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi') + 1 \cdot \mathtt{Pop} \\ &= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi). \end{aligned}$$

- Case $t = \mathtt{push}((l,w),u)$. Then, $t\{x \setminus v\} = \mathtt{push}((l, w\{x \setminus u\}), u\{x \setminus v\})$, and Φ must be of the form

$$\frac{\Phi_1 \rhd_{\mathcal{V}_{\mathcal{S}}} \Gamma_1 \vdash w\{x \setminus v\} : N \qquad \Phi_2 \rhd_{\mathcal{V}_{\mathcal{S}}} \Gamma_2 \vdash u\{x \setminus v\} : S\{l := N \cdot S\,(l)\} \Rightarrow P}{\Gamma_1 + \Gamma_2 \vdash \mathtt{push}((l, w\{x \setminus v\}), u\{x \setminus v\}) : S \Rightarrow P}\ \mathtt{Push}$$

$\Gamma = \Gamma_1 + \Gamma_2$, and $\mathbb{M} = S \Rightarrow P$. By applying the *i.h.* to w, we know there exist $\Psi_1 \triangleright_{\mathcal{V}_S} \Gamma_1; x : M_1 \vdash w : N$ and $\Pi_1 \triangleright_{\mathcal{V}_S} \emptyset \vdash v : M_1$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_1) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_1)$. By applying the *i.h.* to u, we know there exist $\Psi_2 \triangleright_{\mathcal{N}_S} \Gamma_2; x : M_2 \vdash u : S\{l := N \cdot S\,(l)\} \Rightarrow P$ and $\Pi_2 \triangleright_{\mathcal{N}_S} \emptyset \vdash v : M_2$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_2) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_2) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_2)$. By Lemma 8.40, there exists a type derivation $\Pi' \triangleright_{\mathcal{V}_S} \emptyset \vdash v : M_1 \sqcup M_2$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi') = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_2)$. Therefore, we can conclude by taking $M = M_1 \sqcup M_2$, $\Pi = \Pi'$ and Ψ to be

$$\frac{\Psi_1 \triangleright_{\mathcal{V}_S} \Gamma_1; x : M_1 \vdash w : N \qquad \Psi_2 \triangleright_{\mathcal{V}_S} \Gamma_2; x : M_2 \vdash u : S\{l := N \cdot S\,(l)\} \Rightarrow P}{(\Gamma_1 + \Gamma_2); x : M_1 \sqcup M_2 \vdash \mathtt{push}((l, w), u) : S \Rightarrow P}\ \mathtt{Push}$$

since

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) &= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_2) + 1 \cdot \mathtt{Push} \\
&= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_1) \\
&+ \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_2) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_2) + 1 \cdot \mathtt{Push} \\
&= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_2) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi') + 1 \cdot \mathtt{Push} \\
&= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi).
\end{aligned}$$

□

Lemma 8.50 (Exact Subject Expansion for CBV)**.** *Let $t, u \in \Lambda^{\mathcal{S}}_{\circ}$, $s \in \mathcal{S}_{\circ}$, and $S \in \{\beta_{\mathtt{v}}, \mathtt{pop}, \mathtt{push}\}$. If $(t, s) \Rightarrow_{\mathtt{CBV}(S)} (u, q)$ and $\Phi \triangleright_{\mathcal{V}_S} \emptyset \vdash (u, q) : P$, then there is $\Psi \triangleright_{\mathcal{V}_S} \emptyset \vdash (t, s) : P$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) + 1 \cdot R = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi)$, where*

- $R = \mathtt{App}$, *if* $S = \beta_{\mathtt{v}}$;
- $R = \mathtt{Pop}$, *if* $S = \mathtt{pop}$;
- *and* $R = \mathtt{Push}$, *if* $S = \mathtt{push}$.

Proof. The proof follows by induction on the definition of CBV evaluation contexts and is the stack-based analogue of the CBV $\lambda^{\mathcal{G}}$ subject expansion proof. We only show the case where $\mathtt{V} = \Box$ and t is an operation:

- Case $\mathtt{V} = \Box$:
  - If $t = \mathtt{pop}(l, \lambda x.r)$, we have that $(\mathtt{pop}(l, \lambda x.r), s) \Rightarrow_{\mathtt{CBV}(\mathtt{pop})} (r\{x \setminus \mathtt{head}(s\,(l))\}, s\{l := \mathtt{tail}(s\,(l))\})$, and Φ is of the form

$$\frac{\Phi_1 \triangleright_{\mathcal{V}_S} \emptyset \vdash r\{x \setminus \mathtt{head}(s\,(l))\} : S \Rightarrow P \qquad \Phi_2 \triangleright_{\mathcal{V}_S} \emptyset \vdash s\{l := \mathtt{tail}(s\,(l))\} : S}{\emptyset \vdash (r\{x \setminus \mathtt{head}(s\,(l))\}, s\{l := \mathtt{tail}(s\,(l))\}) : P}\ \mathtt{Conf}$$

By applying Lemma 8.49 to Φ_1, there exist $\Psi_1 \triangleright_{\mathcal{V}_S} x : M \vdash r : S \Rightarrow P$ and $\Pi_2 \triangleright_{\mathcal{V}_S} \varnothing \vdash \mathtt{head}(s\,(l)) : M$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_1) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_2)$. Let $R = S\{l := M \cdot S\,(l)\}$. Then, $\mathtt{head}(R\,(l)) = M$, $\mathtt{tail}(R\,(l)) = S\,(l)$, and $R\{l := \mathtt{tail}(R\,(l))\} = S$. Thus, by applying Lemma 8.45 to Φ_2 and Π_2, there is $\Psi_2 \triangleright_{\mathcal{V}_S} \varnothing \vdash s : R$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_2) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_2) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_2)$. Therefore, we can take Ψ to be

$$\dfrac{\dfrac{\Psi_1 \triangleright_{\mathcal{V}_S} x : M \vdash r : R\{l := \mathtt{tail}(R\,(l))\} \Rightarrow P}{\varnothing \vdash \mathtt{pop}(l, \lambda x.r) : R \Rightarrow P}\,\mathtt{Pop} \qquad \Psi_2 \triangleright_{\mathcal{V}_S} \varnothing \vdash s : R}{\varnothing \vdash (\mathtt{pop}(l, \lambda x.r), s) : P}\,\mathtt{Conf}$$

since

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) + 1 \cdot \mathtt{Pop} &= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_2) + 1 \cdot \mathtt{Pop} \\
&= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_2) \\
&+ \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_2) + 1 \cdot \mathtt{Pop} \\
&= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_2) + 1 \cdot \mathtt{Pop} \\
&= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi),
\end{aligned}$$

and $R = \mathtt{Pop}$.

– If $t = \mathtt{push}((l, v), r)$, then $(\mathtt{push}((l, v), r), s) \Rightarrow_{\mathtt{CBV}(\mathtt{push})} (r, s\{l := v \cdot s\,(l)\})$, and Φ is of the form

$$\dfrac{\Phi_1 \triangleright_{\mathcal{V}_S} \varnothing \vdash r : S \Rightarrow P \qquad \Phi_2 \triangleright_{\mathcal{V}_S} \varnothing \vdash s\{l := v \cdot s\,(l)\} : S}{\varnothing \vdash (r, s\{l := v \cdot st\,(l)\}) : P}\,\mathtt{Conf}$$

Let $R = S\{l := L\}$ and $R\{l := S\,(l)\} = R\{l := M \cdot L\} = R\{l := M \cdot R\,(l)\} = S$. Then, by applying Lemma 8.45 to Φ_2, there are $\Pi_1 \triangleright_{\mathcal{V}_S} \varnothing \vdash v : M$ and $\Psi_2 \triangleright_{\mathcal{V}_S} \varnothing \vdash s : R$, such that $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_2) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_1) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_2)$. Therefore, we can take Ψ to be

$$\dfrac{\dfrac{\Pi_1 \triangleright_{\mathcal{V}_S} \varnothing \vdash v : M \qquad \Phi_1 \triangleright_{\mathcal{V}_S} \varnothing \vdash r : R\{l := M \cdot R\,(l)\} \Rightarrow P}{\varnothing \vdash \mathtt{push}((l, v), r) : R \Rightarrow P}\,\mathtt{Push} \qquad \Psi_2 \triangleright_{\mathcal{V}_S} \varnothing \vdash s : R}{\varnothing \vdash (\mathtt{push}((l, v), r), s) : P}\,\mathtt{Conf}$$

since

$$\begin{aligned}
\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) + 1 \cdot \mathtt{Push} &= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_2) + 1 \cdot \mathtt{Push} \\
&= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi_1) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi_2) + \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Pi_1) \\
&+ 1 \cdot \mathtt{Push} \\
&= \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi),
\end{aligned}$$

and $R = \mathtt{Push}$.

The remaining cases are identical to those for the CBV $\lambda^{\mathcal{G}}$-calculus (see Lemma 7.85). □

Theorem 8.51 (Exact Completeness for CBV). *Let* $t \in \Lambda_{\circ}^{\mathcal{S}}$. *If* $(t,s)^{n\cdot\beta_{\mathsf{v}}+o\cdot\mathtt{pop}+u\cdot\mathtt{push}}\Rightarrow_{\mathtt{CBV}}(u,q)$, *such that* $u \in \mathtt{NO}_{\mathtt{CBV}}$, *then there is* $\Phi \rhd_{\mathcal{V}_{\mathcal{S}}} \emptyset \vdash (t,s) : P$ *tight, such that* $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = n\cdot\mathtt{App} + o\cdot\mathtt{Pop} + u\cdot\mathtt{Push}$.

Proof. The proof is identical to that of Theorem 7.86, *mutatis mutandis*. The argument proceeds by induction on $n+o+u$, using exact subject expansion (Lemma 8.50) and the tight typability of CBV-normal forms (Lemma 8.44). The only difference is that global state operations and typing rules are replaced by the corresponding stack operations and rules. □

F.1.3 Subsuming Call-By-Name and Call-By-Value

Lemma 8.55 (CBPV Multiset-Typings Split and Merge). *Let* $t \in !\Lambda^{\mathcal{S}}$. *Then, there is* $\Phi \rhd_{\mathcal{P}_{\mathcal{S}}} \Gamma \vdash t : M$, *such that* $M = \sqcup_{i\in\mathtt{I}} M_i$, *if and only if there are* $\Phi_i \rhd_{\mathcal{P}_{\mathcal{S}}} \Gamma_i \vdash t : M_i$, *such that* $\Gamma = +_{i\in\mathtt{I}}\Gamma_i$, *and* $\#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Phi) = +_{i\in\mathtt{I}}\#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Phi_i)$.

Proof. The proof is identical to that of the corresponding lemma for global state (Lemma 7.93), *mutatis mutandis*. In particular, the argument proceeds by induction on the structure of the typing derivation and relies solely on the additive nature of multiset types and typing contexts. The presence of stacks does not affect the splitting and merging properties, which are purely structural. □

Lemma 8.56 (Contexts and Free Variables for CBPV). *If* $\Phi \rhd_{\mathcal{P}_{\mathcal{S}}} \Gamma \vdash t : \mathbb{A}$ *(resp. M), then* $\mathtt{dom}(\Gamma) \subseteq \mathtt{fv}(t)$.

Proof. The proof is identical to that of the corresponding lemma for global state (Lemma 7.94), *mutatis mutandis*. The argument proceeds by induction on the structure of the typing derivation and relies solely on the variable-binding structure of CBPV terms. The differences between the global state and stack constructs do not change the argument. □

Lemma 8.57 (CBPV Values Are Not Effectful). *Let* $t \in !\Lambda^{\mathcal{S}}$ *and* $\Phi \rhd_{\mathcal{P}_{\mathcal{S}}} \emptyset \vdash t : S \Rightarrow (A \times Q)$. *If* $t \in !\mathrm{V}^{\mathcal{S}}$, *then* $Q = S$ *and* Φ *ends with rule* `Lift`.

Proof. The proof is identical to that of the corresponding lemma for global state (Lemma 7.95), *mutatis mutandis*. The argument relies solely on the syntactic characterization of CBPV values and the structure of CBPV typing derivations. Stack-related operations do not affect the classification of values or the form of effectful types. □

Lemma 8.58 (Zero Weight Tight Type Derivations Characterize CBPV-Normal Forms). *Let* $t \in \Lambda_{\circ}^{\mathcal{S}}$ *and* $s \in \mathcal{S}_{\circ}$. *If* $\Phi \rhd_{\mathcal{P}_{\mathcal{S}}} \emptyset \vdash (t,s) : P$ *is tight, then* $\#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Phi) = 0\cdot\mathtt{App} + 0\cdot\mathtt{Der} + 0\cdot\mathtt{Pop} + 0\cdot\mathtt{Push}$ *if and only if* $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$.

Proof. The proof is identical to that of the corresponding lemma for the $\lambda!^{\mathcal{G}}$-calculus (Lemma 7.96), *mutatis mutandis*, replacing global state constructs with the corresponding stack constructs. □

Lemma 8.59 (Tight Typability of CBPV-Normal Forms)**.** *Let $t \in \Lambda^{\mathcal{S}}_{\circ}$. If $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$, then there is a derivation $\Phi \triangleright_{\mathcal{P}_{\mathcal{S}}} \emptyset \vdash (t, s) : P$ that is tight.*

Proof. The proof is identical to that of the corresponding lemma for the $\lambda!^{\mathcal{G}}$-calculus (Lemma 7.97), *mutatis mutandis*, replacing global state constructs with the corresponding stack constructs. □

Lemma 8.60. *Let $l \in \mathcal{L}$ and $s \in \mathcal{S}$. There are $\Phi \triangleright_{\mathcal{P}_{\mathcal{S}}} \emptyset \vdash s : S$ and $\Psi \triangleright_{\mathcal{P}_{\mathcal{S}}} \emptyset \vdash f : L$ if and only if there is $\Pi \triangleright_{\mathcal{P}_{\mathcal{S}}} \emptyset \vdash s\{l := f\} : S\{l := L\}$. Moreover, $\#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Phi) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Psi) = \#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Pi)$.*

Proof. The proof is identical to that for the $\lambda^{\mathcal{S}}$-calculus (Lemma 8.45), *mutatis mutandis*. It relies only on the structural decomposition of state typings and on the additivity of typing weights, which are unchanged in the CBPV setting. □

F.1.3.1 Quantitative Soundness

Lemma 8.61 (Weighted Substitution for CBPV)**.** *Let $t \in !\Lambda^{\mathcal{S}}$ and $v \in !\mathrm{V}^{\mathcal{S}}$. If $\Phi \triangleright_{\mathcal{P}_{\mathcal{S}}} \Gamma; x : M \vdash t : \mathbb{A}$ (resp. N) and $\Psi \triangleright_{\mathcal{P}_{\mathcal{S}}} \emptyset \vdash v : M$, then $\Pi \triangleright_{\mathcal{P}_{\mathcal{S}}} \Gamma \vdash t\{x \setminus v\} : \mathbb{A}$ (resp. N), such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Pi) = \#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Phi) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Psi)$.*

Proof. The proof follows by induction on the structure of Φ and is identical to that of the $\lambda!^{\mathcal{G}}$-calculus (Lemma 7.100), *mutatis mutandis*. The argument relies only on the compositionality of substitution and the additivity of typing weights, which are preserved in the stack-based CBPV setting. □

Lemma 8.62 (Exact Subject Reduction for CBPV)**.** *Let $t \in !\Lambda^{\mathcal{S}}_{\circ}$, $s \in \mathcal{S}_{\circ}$, and $S \in \{\beta_{\mathtt{v}}, \beta_{!}, \mathtt{pop}, \mathtt{push}\}$. If $(t, s) \Rightarrow_{\mathtt{CBPV}(S)} (u, q)$ and $\Phi \triangleright_{\mathcal{P}_{\mathcal{S}}} \emptyset \vdash (t, s) : P$, then there is $\Psi \triangleright_{\mathcal{P}_{\mathcal{S}}} \emptyset \vdash (u, q) : P$, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Phi) = \#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Psi) + 1 \cdot R$, where*

- *$R = \mathtt{App}$, if $S = \beta_{\mathtt{v}}$;*
- *$R = \mathtt{Der}$, if $S = \beta_{!}$;*
- *$R = \mathtt{Pop}$, if $S = \mathtt{pop}$;*
- *and $R = \mathtt{Push}$, if $S = \mathtt{push}$.*

Proof. The proof follows by induction on the definition of the contexts of the reduction strategy and is identical to that for the $\lambda!^{\mathcal{G}}$-calculus (Lemma 7.101), *mutatis mutandis*. □

Theorem 8.63 (Exact Soundness for CBPV). *Let $t \in \Lambda^{\mathcal{S}}_{\circ}$. If there is $\Phi \rhd_{\mathcal{P}_{\mathcal{S}}} \emptyset \vdash (t,s) : P$ tight, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Phi) = n \cdot \mathtt{App} + m \cdot \mathtt{Der} + o \cdot \mathtt{Pop} + u \cdot \mathtt{Push}$, then $(t,s) \overset{n\cdot\beta_{\mathtt{v}}+m\cdot\beta_{!}+o\cdot\mathtt{pop}+u\cdot\mathtt{push}}{\Rightarrow\!\!\!\!\!\!\Rightarrow}_{\mathtt{CBPV}} (u,q)$, such that $u \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$.*

Proof. The proof is identical to that of Theorem 7.102, *mutatis mutandis*. The argument proceeds by induction on $n+m+o+u$, using exact subject reduction (Lemma 8.62) and the characterization of zero-weight tight derivations (Lemma 8.58). The only difference is that global state operations and typing rules are replaced by the corresponding stack operations and rules. □

F.1.3.2 Quantitative Completeness

Lemma 8.64 (Typability Implies Clash-Freeness). *Let $t \in !\Lambda^{\mathcal{S}}_{\circ}$ and $s \in \mathcal{S}_{\circ}$. If $\Phi \rhd_{\mathcal{P}_{\mathcal{S}}} \emptyset \vdash (t,s) : P$, then t is clash-free.*

Proof. The proof is identical to that of Lemma 7.103, *mutatis mutandis*, replacing global state constructs with the corresponding stack constructs. □

Lemma 8.65 (Typability Characterizes CBPV Clash-Free Normal Forms). *Let $t \in !\Lambda^{\mathcal{S}}_{\circ}$ and $s \in \mathcal{S}_{\circ}$. Then, $t \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$ if and only if $t \in \mathtt{NO}_{\mathtt{CBPV}}$ and (t,s) is $\mathcal{P}_{\mathcal{S}}$-typable.*

Proof. The proof is identical to that for the $\lambda!^{\mathcal{G}}$-calculus (see Lemma 7.104). □

Lemma 8.66 (Weighted Anti-Substitution for CBPV). *Let $t \in !\Lambda^{\mathcal{S}}$, and $v \in !\mathrm{V}^{\mathcal{S}}_{\circ}$. If $\Phi \rhd_{\mathcal{P}_{\mathcal{S}}} \Gamma \vdash t\{x \setminus v\} : \mathbb{A}$ (resp. N), then $\Psi \rhd_{\mathcal{P}_{\mathcal{S}}} \Gamma; x : M \vdash t : \mathbb{A}$ (resp. N) and $\Pi \rhd_{\mathcal{P}_{\mathcal{S}}} \emptyset \vdash v : M$, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Phi) = \#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Psi) + \#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Pi)$.*

Proof. The proof follows by induction on the structure of t and identical to the one for the $\lambda!^{\mathcal{G}}$-calculus (see Lemma 7.105), *mutatis mutandis*. □

Lemma 8.67 (Exact Subject Expansion for CBPV). *Let $t \in !\Lambda^{\mathcal{S}}_{\circ}$, $s \in \mathcal{S}_{\circ}$, and $S \in \{\beta_{\mathtt{v}}, \beta_{!}, \mathtt{pop}, \mathtt{push}\}$. If $(t,s) \Rightarrow_{\mathtt{CBPV}(S)} (u,q)$ and $\Phi \rhd_{\mathcal{P}_{\mathcal{S}}} \emptyset \vdash (u,q) : P$, then there is $\Psi \rhd_{\mathcal{P}_{\mathcal{S}}} \emptyset \vdash (t,s) : P$, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Phi) + 1 \cdot R = \#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Psi)$, where*

- *$R = \mathtt{App}$, if $S = \beta_{\mathtt{v}}$;*
- *$R = \mathtt{Der}$, if $S = \beta_{!}$;*
- *$R = \mathtt{Pop}$, if $S = \mathtt{pop}$;*
- *and $R = \mathtt{Push}$, if $S = \mathtt{push}$.*

Proof. The proof follows by induction on the definition of the contexts of the reduction strategy and is identical to that for the $\lambda!^{\mathcal{G}}$-calculus (see Lemma 7.106), *mutatis mutandis*. □

Theorem 8.68 (Exact Completeness for CBPV). *Let $t \in \Lambda^{\mathcal{S}}_{\circ}$. If $(t,s) \overset{n\cdot\beta_{\mathtt{v}}+m\cdot\beta_{!}+o\cdot\mathtt{pop}+u\cdot\mathtt{push}}{\twoheadrightarrow_{\mathtt{CBPV}}} (u,q)$, such that $u \in \mathtt{NO}^{\mathtt{CF}}_{\mathtt{CBPV}}$, then there is $\Phi \rhd_{\mathcal{P}_{\mathcal{S}}} \varnothing \vdash (t,s) : P$ tight, such that $\#_{\mathtt{App},\mathtt{Der},\mathtt{Pop},\mathtt{Push}}(\Phi) = n \cdot \mathtt{App} + m \cdot \mathtt{Der} + o \cdot \mathtt{Pop} + u \cdot \mathtt{Push}$.*

Proof. The proof is identical to that of Theorem 7.107, *mutatis mutandis*. The argument proceeds by induction on $n+m+o+u$, using exact subject expansion (Lemma 8.67) and the tight typability of clash-free normal forms (Lemma 8.59). The only difference is that global state operations and typing rules are replaced by the corresponding stack operations and rules. □

F.1.3.3 Soundness and Completeness of Translations

Lemma 8.69 (Soundness of Term and Stack Translations). *Let $t \in \Lambda^{\mathcal{S}}$.*

1. *If $\Phi \rhd_{\mathcal{N}_{\mathcal{S}}} \Gamma \vdash t$ (resp. t or f) $: \mathbb{A}$ (resp. M or L), then $\Psi \rhd_{\mathcal{P}_{\mathcal{S}}} \Gamma \vdash (t)^{\mathtt{CBN}}$ (resp. $!(t)^{\mathtt{CBN}}$ or $(f)^{\mathtt{CBN}}$) $:$ $\mathbb{A}$ (resp. M or L). Moreover, $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi)$.*
2. *If $\Phi \rhd_{\mathcal{V}_{\mathcal{S}}} \Gamma \vdash t$ (resp. t or f) $: \mathbb{M}$ (resp. M or L), then $\Psi \rhd_{\mathcal{P}_{\mathcal{S}}} (\Gamma)^{\mathtt{G}_{\mathtt{CBV}}} \vdash (t)^{\mathtt{CBV}}$ (resp. $(t)^{\mathtt{CBV}}$ or $(f)^{\mathtt{CBV}}$) $:$ $(\mathbb{M})^{\mathtt{G}_{\mathtt{CBV}}}$ (resp. $(M)^{\mathtt{G}_{\mathtt{CBV}}}$ or $(L)^{\mathtt{G}_{\mathtt{CBV}}}$). Moreover, $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi)$.*

Proof. The proof proceeds by induction on the structure of the typing derivation Φ. Each typing rule of $\mathcal{N}_{\mathcal{S}}$ (resp. $\mathcal{V}_{\mathcal{S}}$) is mapped in a structure-preserving way to the corresponding rule of $\mathcal{P}_{\mathcal{S}}$ under the translations $(\cdot)^{\mathtt{CBN}}$ (resp. $(\cdot)^{\mathtt{CBV}}$ and $(\cdot)^{\mathtt{G}_{\mathtt{CBV}}}$). In all cases, the translated derivation preserves the typing judgment and does not introduce additional occurrences of $\mathtt{App}$, $\mathtt{Pop}$, or $\mathtt{Push}$. The argument is identical to that of the $\lambda!^{\mathcal{G}}$-calculus (see Lemma 7.111), with global state constructs replaced by stack constructs. □

Lemma 8.70 (Completeness of CBN Term and Stack Translations). *Let $t \in \Lambda^{\mathcal{S}}$. If $\Psi \rhd_{\mathcal{P}_{\mathcal{S}}} \Gamma \vdash (t)^{\mathtt{CBN}}$ (resp. $!(t)^{\mathtt{CBN}}$ or $(f)^{\mathtt{CBN}}$) $:$ $\mathbb{A}$ (resp. M or L), then $\Phi \rhd_{\mathcal{N}_{\mathcal{S}}} \Gamma \vdash t$ (resp. t or f) $:$ $\mathbb{A}$ (resp. M or L). Moreover, $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi)$.*

Proof. The proof proceeds by induction on the structure of the CBPV typing derivation Ψ. Each typing rule used to derive $\Psi \rhd_{\mathcal{P}_{\mathcal{S}}} \Gamma \vdash (t)^{\mathtt{CBN}} : \mathbb{A}$ corresponds uniquely to a typing rule of $\mathcal{N}_{\mathcal{S}}$ for t. By inverting the translation $(\cdot)^{\mathtt{CBN}}$ at each step, we reconstruct a derivation $\Phi \rhd_{\mathcal{N}_{\mathcal{S}}} \Gamma \vdash t : \mathbb{A}$ with the same typing context and type. No additional occurrences of $\mathtt{App}$, $\mathtt{Pop}$, or $\mathtt{Push}$ are introduced in this reconstruction, so the weight is preserved. The argument is identical to that of the $\lambda!^{\mathcal{G}}$-calculus (see Lemma 7.112), with global state constructs replaced by stack constructs. □

Lemma 8.71 (Soundness and Completeness of CBN State Translations). *Let $s \in \mathcal{S}$. Then, $\Phi \rhd_{\mathcal{N}_{\mathcal{S}}} \Gamma \vdash s : S$ if and only if $\Psi \rhd_{\mathcal{P}_{\mathcal{S}}} \Gamma \vdash (s)^{\mathtt{CBN}} : S$. Moreover, $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi)$.*

Proof. Both directions are proved by induction on the structure of the state $s \in \mathcal{S}$.

$\Rightarrow$) Assume $\Phi \triangleright_{\mathcal{N}_\mathcal{S}} \Gamma \vdash s : S$. Each typing rule for states in $\mathcal{N}$ is preserved by the translation $(\cdot)^{\mathtt{CBN}}$. For each stack component, soundness of term translations (Lemma 8.69) yields a corresponding CBPV typing. Reassembling these derivations yields $\Psi \triangleright_{\mathcal{P}_\mathcal{S}} \Gamma \vdash (s)^{\mathtt{CBN}} : S$.

$\Leftarrow$) Conversely, assume $\Psi \triangleright_{\mathcal{P}_\mathcal{S}} \Gamma \vdash (s)^{\mathtt{CBN}} : S$. By induction on the structure of s and by completeness of term translations (Lemma 8.70), each translated stack component corresponds to a well-typed source component. Reconstructing the original state typing yields $\Phi \triangleright_{\mathcal{N}_\mathcal{S}} \Gamma \vdash s : S$.

In both directions, no occurrences of `App`, `Pop`, or `Push` are introduced or removed by the translation, so the size equality follows immediately. □

Lemma 8.72 (Soundness of CBV State Translations)**.** *Let $s \in \mathcal{S}$. If $\Phi \triangleright_{\mathcal{V}_\mathcal{S}} \Gamma \vdash s : S$, then $\Psi \triangleright_{\mathcal{P}_\mathcal{S}} (\Gamma)^{\mathtt{CBV}} \vdash (s)^{\mathtt{CBV}} : (S)^{\mathtt{CBV}}$. Moreover, $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi)$.*

Proof. The proof follows by induction on the structure of the state $s \in \mathcal{S}$. Each typing rule for states in $\mathcal{V}$ is preserved by the translation $(\cdot)^{\mathtt{CBV}}$. For each state component, soundness of CBV term translations (Lemma 8.69) yields a corresponding CBPV typing. Reassembling these derivations gives $\Psi \triangleright_{\mathcal{P}_\mathcal{S}} (\Gamma)^{\mathtt{CBV}} \vdash (s)^{\mathtt{CBV}} : (S)^{\mathtt{CBV}}$. Since state typing rules do not contribute to `App`, `Pop`, or `Push`, the size equality holds immediately. □

Theorem 8.73 (Soundness and Completeness of CBN Configuration Translations)**.** *Let $t \in \Lambda^{\mathcal{S}}$ and $s \in \mathcal{S}$. Then, $\Phi \triangleright_{\mathcal{N}_\mathcal{S}} \Gamma \vdash (t,s) : A \times Q$ if and only if $\Psi \triangleright_{\mathcal{P}_\mathcal{S}} \Gamma \vdash ((t,s))^{\mathtt{CBN}} : A \times Q$. Moreover, $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi)$.*

Proof. The proof follows smoothly from the soundness and completeness of CBN term translations (Lemmas 8.69 and 8.70) together with soundness and completeness of state translations (Lemma 8.71). The result follows by reassembling the corresponding term and state derivations using the configuration typing rule. Weight preservation is immediate. □

Theorem 8.74 (Soundness of CBV Configuration Translations)**.** *Let $t \in \Lambda^{\mathcal{S}}$ and $s \in \mathcal{S}$. If $\Phi \triangleright_{\mathcal{V}_\mathcal{S}} \Gamma \vdash (t,s) : P$, then $\Psi \triangleright_{\mathcal{P}_\mathcal{S}} \Gamma \vdash ((t,s))^{\mathtt{CBV}} : (P)^{G_{\mathtt{CBV}}}$. Moreover, $\#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Phi) = \#_{\mathtt{App},\mathtt{Pop},\mathtt{Push}}(\Psi)$.*

Proof. The proof follows smoothly from soundness of CBV term translations (Lemma 8.69) and soundness of CBV state translations (Lemma 8.72), by composing the translated derivations using the configuration typing rule. □

Bibliography

[1] P. Gardner, "Discovering needed reductions using type theory," in *Theoretical Aspects of Computer Software, International Conference TACS '94, Sendai, Japan, April 19-22, 1994, Proceedings*, ser. Lecture Notes in Computer Science, M. Hagiya and J. C. Mitchell, Eds., vol. 789. Springer, 1994, pp. 555–574. [Online]. Available: https://doi.org/10.1007/3-540-57887-0_115 [Cited on pages 3, 4, 6, 7, 27, and 40.]

[2] A. J. Kfoury, "A linearization of the lambda-calculus and consequences," *J. Log. Comput.*, vol. 10, no. 3, pp. 411–436, 2000. [Online]. Available: https://doi.org/10.1093/logcom/10.3.411 [Cited on pages 4, 27, and 40.]

[3] D. d. Carvalho, "Sémantiques de la logique linéaire et temps de calcul," Ph.D. dissertation, Aix-Marseille 2 2007, 2007, thèse de doctorat dirigée par Ehrhard, Thomas Mathématiques discrètes et fondements de l'informatique Aix-Marseille 2 2007. [Online]. Available: http://www.theses.fr/2007AIX22066 [Cited on pages 4, 7, 28, and 40.]

[4] D. de Carvalho, "Execution time of λ-terms via denotational semantics and intersection types," *Math. Struct. Comput. Sci.*, vol. 28, no. 7, pp. 1169–1203, 2018. [Online]. Available: https://doi.org/10.1017/S0960129516000396 [Cited on pages 4, 28, and 40.]

[5] B. Accattoli, S. Graham-Lengrand, and D. Kesner, "Tight typings and split bounds, fully developed," *J. Funct. Program.*, vol. 30, p. e14, 2020. [Online]. Available: https://doi.org/10.1017/S095679682000012X [Cited on pages 4, 7, 39, and 40.]

[6] A. Bucciarelli, D. Kesner, and D. Ventura, "Non-idempotent intersection types for the lambda-calculus," *Log. J. IGPL*, vol. 25, no. 4, pp. 431–464, 2017. [Online]. Available: https://doi.org/10.1093/jigpal/jzx018 [Cited on pages 7 and 40.]

[7] A. Bernadet and S. Lengrand, "Non-idempotent intersection types and strong normalisation," *Log. Methods Comput. Sci.*, vol. 9, no. 4, 2013. [Online]. Available: https://doi.org/10.2168/LMCS-9(4:3)2013 [Cited on pages 3, 4, 6, and 7.]

[8] D. Kesner and D. Ventura, "A resource aware computational interpretation for herbelin's syntax," in *Theoretical Aspects of Computing - ICTAC 2015 - 12th International Colloquium Cali, Colombia, October 29-31, 2015, Proceedings*, ser. Lecture Notes in Computer Science, M. Leucker, C. Rueda, and F. D. Valencia, Eds., vol. 9399. Springer, 2015, pp. 388–403. [Online]. Available: https://doi.org/10.1007/978-3-319-25150-9_23 [Cited on page 4.]

[9] D. Kesner and P. Vial, "Types as resources for classical natural deduction," in *2nd International Conference on Formal Structures for Computation and Deduction, FSCD 2017, Oxford, UK, September 3-9, 2017*, ser. LIPIcs, D. Miller, Ed., vol. 84. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, pp. 24:1–24:17. [Online]. Available: https://doi.org/10.4230/LIPIcs.FSCD.2017.24

[10] ——, "Non-idempotent types for classical calculi in natural deduction style," *Log. Methods Comput. Sci.*, vol. 16, no. 1, 2020. [Online]. Available: https://doi.org/10.23638/LMCS-16(1:3)2020 [Cited on page 4.]

[11] ——, "Consuming and persistent types for classical logic," in *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, H. Hermanns, L. Zhang, N. Kobayashi, and D. Miller, Eds. ACM, 2020, pp. 619–632. [Online]. Available: https://doi.org/10.1145/3373718.3394774 [Cited on page 7.]

[12] B. Accattoli, S. Graham-Lengrand, and D. Kesner, "Tight typings and split bounds," *Proc. ACM Program. Lang.*, vol. 2, no. ICFP, pp. 94:1–94:30, 2018. [Online]. Available: https://doi.org/10.1145/3236789 [Cited on pages 39 and 40.]

[13] B. Accattoli, G. Guerrieri, and M. Leberle, "Types by need," in *Programming Languages and Systems - 28th European Symposium on Programming, ESOP 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*, ser. Lecture Notes in Computer Science, L. Caires, Ed., vol. 11423. Springer, 2019, pp. 410–439. [Online]. Available: https://doi.org/10.1007/978-3-030-17184-1_15 [Cited on page 40.]

[14] B. Accattoli, U. D. Lago, and G. Vanoni, "The space of interaction," in *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*. IEEE, 2021, pp. 1–13.

[15] ——, "Multi types and reasonable space," *Proc. ACM Program. Lang.*, vol. 6, no. ICFP, pp. 799–825, 2022. [Cited on page 4.]

[16] A. Bernadet and S. J. Lengrand, "Non-idempotent intersection types and strong normalisation," *Logical Methods in Computer Science*, vol. 9, 2013. [Cited on pages 4 and 40.]

[17] D. Kesner, "Reasoning about call-by-need by means of types," in *Foundations of Software Science and Computation Structures - 19th International Conference, FOSSACS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, ser. Lecture Notes in Computer Science, B. Jacobs and C. Löding, Eds., vol. 9634. Springer, 2016, pp. 424–441. [Online]. Available: https://doi.org/10.1007/978-3-662-49630-5_25 [Cited on page 4.]

[18] G. Huet and J.-J. Lévy, "Computations in orthogonal rewriting systems - Part I and Part II," in *Computational Logic, Essays in Honor of Alan Robinson*, J.-L. Lassez and G. Plotkin, Eds. MIT Press, 1991, pp. 395–443. [Cited on page 4.]

[19] R. C. Sekar and I. V. Ramakrishnan, "Programming in equational logic: Beyond strong sequentiality," *Inf. Comput.*, vol. 104, no. 1, pp. 78–109, 1993.

[20] H. Cirstea and C. Kirchner, "The rewriting calculus - part I," *Log. J. IGPL*, vol. 9, no. 3, pp. 339–375, 2001.

[21] W. Kahl, "Basic pattern matching calculi: a fresh view on matching failure," in *Functional and Logic Programming, 7th International Symposium, FLOPS 2004, Nara, Japan, April 7-9, 2004, Proceedings*, ser. Lecture Notes in Computer Science, Y. Kameyama and P. J. Stuckey, Eds., vol. 2998. Springer, 2004, pp. 276–290.

[22] C. B. Jay and D. Kesner, "First-class patterns," *J. Funct. Program.*, vol. 19, no. 2, pp. 191–225, 2009.

[23] Z. Khasidashvili, "Expression reduction systems," in *Proceedings of IN Vekua Institute of Applied Mathematics*, vol. 36, Tbilisi, 1990.

[24] D. Kesner, L. Puel, and V. Tannen, "A typed pattern calculus," *Inf. Comput.*, vol. 124, no. 1, pp. 32–61, 1996.

[25] S. Cerrito and D. Kesner, "Pattern matching as cut elimination," in *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*. IEEE Computer Society, 1999, pp. 98–108.

[26] N. Zeilberger, "Focusing and higher-order abstract syntax," in *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, G. C. Necula and P. Wadler, Eds. ACM, 2008, pp. 359–369.

[27] N. R. Krishnaswami, "Focusing on pattern matching," in *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, Savannah, GA, USA, January 21-23, 2009*, Z. Shao and B. C. Pierce, Eds. ACM, 2009, pp. 366–378. [Cited on page 4.]

[28] S. Alves, D. Kesner, and D. Ventura, "A Quantitative Understanding of Pattern Matching," in *25th International Conference on Types for Proofs and Programs (TYPES 2019)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), M. Bezem and A. Mahboubi, Eds., vol. 175. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, pp. 3:1–3:36. [Cited on pages 4, 7, 8, and 65.]

[29] A. Bucciarelli, D. Kesner, and S. R. D. Rocca, "Observability for pair pattern calculi," in *13th International Conference on Typed Lambda Calculi and Applications, TLCA 2015, July 1-3, 2015, Warsaw, Poland*, ser. LIPIcs, T. Altenkirch, Ed., vol. 38. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015, pp. 123–137. [Cited on pages 4 and 7.]

[30] ——, "Solvability = typability + inhabitation," *Log. Methods Comput. Sci.*, vol. 17, no. 1, 2021. [Online]. Available: https://lmcs.episciences.org/7141 [Cited on pages 4, 7, 8, and 65.]

[31] E. Moggi, "Notions of computation and monads," *Inf. Comput.*, vol. 93, no. 1, pp. 55–92, 1991. [Online]. Available: https://doi.org/10.1016/0890-5401(91)90052-4 [Cited on pages 5, 45, 73, and 103.]

[32] P. B. Levy, *Call-By-Push-Value: A Functional/Imperative Synthesis*, ser. Semantics Structures in Computation. Springer, 2004, vol. 2. [Cited on pages 5 and 24.]

[33] ——, "Call-by-push-value: Decomposing call-by-value and call-by-name," *High. Order Symb. Comput.*, vol. 19, no. 4, pp. 377–414, 2006. [Online]. Available: https://doi.org/10.1007/s10990-006-0480-6 [Cited on pages 5 and 21.]

[34] J. Girard, "Linear logic," *Theor. Comput. Sci.*, vol. 50, pp. 1–102, 1987. [Online]. Available: https://doi.org/10.1016/0304-3975(87)90045-4 [Cited on pages 5, 21, 27, and 30.]

[35] T. Ehrhard and G. Guerrieri, "The bang calculus: an untyped lambda-calculus generalizing call-by-name and call-by-value," in *Proceedings of the 18th International Symposium on*

Principles and Practice of Declarative Programming, Edinburgh, United Kingdom, September 5-7, 2016, J. Cheney and G. Vidal, Eds. ACM, 2016, pp. 174–187. [Online]. Available: https://doi.org/10.1145/2967973.2968608 [Cited on pages 5, 7, 21, 24, and 28.]

[36] G. D. Plotkin and J. Power, "Adequacy for algebraic effects," in *Foundations of Software Science and Computation Structures, 4th International Conference, FOSSACS 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2-6, 2001, Proceedings*, ser. Lecture Notes in Computer Science, F. Honsell and M. Miculan, Eds., vol. 2030. Springer, 2001, pp. 1–24. [Online]. Available: https://doi.org/10.1007/3-540-45315-6_1 [Cited on pages 6, 51, and 57.]

[37] ——, "Notions of computation determine monads," in *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings*, ser. Lecture Notes in Computer Science, M. Nielsen and U. Engberg, Eds., vol. 2303. Springer, 2002, pp. 342–356. [Online]. Available: https://doi.org/10.1007/3-540-45931-6_24 [Cited on pages 6, 45, 51, and 54.]

[38] T. Uustalu, "Stateful runners of effectful computations," in *The 31st Conference on the Mathematical Foundations of Programming Semantics, MFPS 2015, Nijmegen, The Netherlands, June 22-25, 2015*, ser. Electronic Notes in Theoretical Computer Science, D. R. Ghica, Ed., vol. 319. Elsevier, 2015, pp. 403–421. [Online]. Available: https://doi.org/10.1016/j.entcs.2015.12.024 [Cited on pages 6, 59, 104, and 156.]

[39] F. Abou-Saleh and D. Pattinson, "Comodels and effects in mathematical operational semantics," in *Foundations of Software Science and Computation Structures - 16th International Conference, FOSSACS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, ser. Lecture Notes in Computer Science, F. Pfenning, Ed., vol. 7794. Springer, 2013, pp. 129–144. [Online]. Available: https://doi.org/10.1007/978-3-642-37075-5_9 [Cited on pages 6 and 58.]

[40] U. de'Liguoro and R. Treglia, "Intersection types for a λ-calculus with global store," in *PPDP 2021: 23rd International Symposium on Principles and Practice of Declarative Programming, Tallinn, Estonia, September 6-8, 2021*, N. Veltri, N. Benton, and S. Ghilezan, Eds. ACM, 2021, pp. 5:1–5:11. [Online]. Available: https://doi.org/10.1145/3479394.3479400 [Cited on page 6.]

[41] G. Guerrieri, W. B. Heijltjes, and J. W. N. Paulus, "A deep quantitative type system," in *29th EACSL Annual Conference on Computer Science Logic, CSL 2021, January 25-28, 2021, Ljubljana,*

Slovenia (Virtual Conference), ser. LIPIcs, C. Baier and J. Goubault-Larrecq, Eds., vol. 183. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pp. 24:1–24:24. [Online]. Available: https://doi.org/10.4230/LIPIcs.CSL.2021.24

[42] U. de'Liguoro and R. Treglia, "From semantics to types: The case of the imperative λ-calculus," *Theor. Comput. Sci.*, vol. 973, p. 114082, 2023. [Online]. Available: https://doi.org/10.1016/j.tcs.2023.114082

[43] S. Alves, D. Kesner, and M. Ramos, "Quantitative global memory," in *Logic, Language, Information, and Computation - 29th International Workshop, WoLLIC 2023, Halifax, NS, Canada, July 11-14, 2023, Proceedings*, ser. Lecture Notes in Computer Science, H. H. Hansen, A. Scedrov, and R. J. G. B. de Queiroz, Eds., vol. 13923. Springer, 2023, pp. 53–68. [Online]. Available: https://doi.org/10.1007/978-3-031-39784-4_4 [Cited on pages 40, 85, 88, 89, and 165.]

[44] ——, "A quantitative approach to global state composition," *Mathematical Structures in Computer Science*, vol. 35, p. e28, 2025. [Cited on pages 6, 20, 85, 88, 89, and 110.]

[45] A. Bucciarelli, D. Kesner, A. Ríos, and A. Viso, "The bang calculus revisited," *Inf. Comput.*, vol. 293, p. 105047, 2023. [Online]. Available: https://doi.org/10.1016/j.ic.2023.105047 [Cited on pages 7, 24, 28, and 40.]

[46] V. Arrial, G. Guerrieri, and D. Kesner, "Quantitative inhabitation for different lambda calculi in a unifying framework," *Proc. ACM Program. Lang.*, vol. 7, no. POPL, pp. 1483–1513, 2023. [Online]. Available: https://doi.org/10.1145/3571244 [Cited on page 7.]

[47] U. D. Lago, C. Faggian, and S. R. D. Rocca, "Intersection types and (positive) almost-sure termination," *Proc. ACM Program. Lang.*, vol. 5, no. POPL, pp. 1–32, 2021. [Online]. Available: https://doi.org/10.1145/3434313 [Cited on pages 7 and 40.]

[48] F. Gavazzo, R. Treglia, and G. Vanoni, "Monadic intersection types, relationally," in *Programming Languages and Systems - 33rd European Symposium on Programming, ESOP 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings, Part I*, ser. Lecture Notes in Computer Science, S. Weirich, Ed., vol. 14576. Springer, 2024, pp. 22–51. [Online]. Available: https://doi.org/10.1007/978-3-031-57262-3_2 [Cited on pages 7 and 178.]

[49] Z. Galal, F. Gavazzo, R. Treglia, and G. Vanoni, "Monadic intersection types, relationally, and ordered," *ACM Trans. Program. Lang. Syst.*, vol. 47, no. 4, Dec. 2025. [Online]. Available: https://doi.org/10.1145/3777483 [Cited on page 7.]

[50] W. Heijltjes, "Quantitative types for the functional machine calculus," in *10th International Conference on Formal Structures for Computation and Deduction, FSCD 2025, July 14-20, 2025, Birmingham, UK*, ser. LIPIcs, M. Fernández, Ed., vol. 337. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025, pp. 24:1–24:20. [Online]. Available: https://doi.org/10.4230/LIPIcs.FSCD.2025.24 [Cited on page 8.]

[51] G. D. Plotkin and M. Pretnar, "Handling algebraic effects," *Log. Methods Comput. Sci.*, vol. 9, no. 4, 2013. [Online]. Available: https://doi.org/10.2168/LMCS-9(4:23)2013 [Cited on pages 8 and 57.]

[52] A. J. Power and O. Shkaravska, "From comodels to coalgebras: State and arrays," in *Proceedings of the Workshop on Coalgebraic Methods in Computer Science, CMCS 2004, Barcelona, Spain, March 27-29, 2004*, ser. Electronic Notes in Theoretical Computer Science, J. Adámek and S. Milius, Eds., vol. 106. Elsevier, 2004, pp. 297–314. [Online]. Available: https://doi.org/10.1016/j.entcs.2004.02.041 [Cited on page 8.]

[53] R. Garner, "The costructure-cosemantics adjunction for comodels for computational effects," *Math. Struct. Comput. Sci.*, vol. 32, no. 4, pp. 374–419, 2022. [Online]. Available: https://doi.org/10.1017/S0960129521000219 [Cited on page 8.]

[54] ——, "Stream processors and comodels," *Log. Methods Comput. Sci.*, vol. 19, no. 1, 2023. [Online]. Available: https://doi.org/10.46298/lmcs-19(1:2)2023 [Cited on page 8.]

[55] D. Ahman and A. Bauer, "Runners in action," in *Programming Languages and Systems - 29th European Symposium on Programming, ESOP 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings*, ser. Lecture Notes in Computer Science, P. Müller, Ed., vol. 12075. Springer, 2020, pp. 29–55. [Online]. Available: https://doi.org/10.1007/978-3-030-44914-8_2 [Cited on page 8.]

[56] T. Uustalu and N. F. W. Voorneveld, "Algebraic and coalgebraic perspectives on interaction laws," in *Programming Languages and Systems - 18th Asian Symposium, APLAS 2020, Fukuoka, Japan, November 30 - December 2, 2020, Proceedings*, ser. Lecture Notes in Computer Science, B. C. d. S. Oliveira, Ed., vol. 12470. Springer, 2020, pp. 186–205. [Online]. Available: https://doi.org/10.1007/978-3-030-64437-6_10 [Cited on page 8.]

[57] A. Church, "A set of postulates for the foundation of logic," *Annals of Mathematics*, vol. 33, no. 2, pp. 346–366, 1932. [Online]. Available: http://www.jstor.org/stable/1968337 [Cited on page 15.]

[58] P. J. Landin, "Correspondence between ALGOL 60 and church's lambda-notation: part I," *Commun. ACM*, vol. 8, no. 2, pp. 89–101, 1965. [Online]. Available: https://doi.org/10.1145/363744.363749 [Cited on page 15.]

[59] ——, "A correspondence between ALGOL 60 and church's lambda-notations: Part II," *Commun. ACM*, vol. 8, no. 3, pp. 158–167, 1965. [Online]. Available: https://doi.org/10.1145/363791.363804 [Cited on page 15.]

[60] H. P. Barendregt, *The lambda calculus - its syntax and semantics*, ser. Studies in logic and the foundations of mathematics. North-Holland, 1985, vol. 103. [Cited on pages 15, 18, 21, and 27.]

[61] A. M. Turing, "Computability and λ-definability," *The Journal of Symbolic Logic*, vol. 2, no. 4, pp. 153–163, 1937. [Online]. Available: http://www.jstor.org/stable/2268280 [Cited on page 18.]

[62] G. D. Plotkin, "Call-by-name, call-by-value and the lambda-calculus," *Theor. Comput. Sci.*, vol. 1, no. 2, pp. 125–159, 1975. [Online]. Available: https://doi.org/10.1016/0304-3975(75)90017-1 [Cited on pages 18, 21, and 65.]

[63] Z. M. Ariola and M. Felleisen, "The call-by-need lambda calculus," *J. Funct. Program.*, vol. 7, no. 3, pp. 265–301, 1997. [Online]. Available: https://doi.org/10.1017/s0956796897002724 [Cited on page 21.]

[64] J. Maraist, M. Odersky, D. N. Turner, and P. Wadler, "Call-by-name, call-by-value, call-by-need and the linear lambda calculus," *Theor. Comput. Sci.*, vol. 228, no. 1-2, pp. 175–210, 1999. [Online]. Available: https://doi.org/10.1016/S0304-3975(98)00358-2 [Cited on page 21.]

[65] P. B. Levy, "Call-by-push-value: A subsuming paradigm," in *Typed Lambda Calculi and Applications, 4th International Conference, TLCA'99, L'Aquila, Italy, April 7-9, 1999, Proceedings*, ser. Lecture Notes in Computer Science, J. Girard, Ed., vol. 1581. Springer, 1999, pp. 228–242. [Online]. Available: https://doi.org/10.1007/3-540-48959-2_17 [Cited on pages 21, 22, and 24.]

[66] G. Guerrieri and G. Manzonetto, "The bang calculus and the two girard's translations," in *Proceedings Joint International Workshop on Linearity & Trends in Linear Logic and Applications, Linearity-TLLA@FLoC 2018, Oxford, UK, 7-8 July 2018*, ser. EPTCS, T. Ehrhard, M. Fernández, V. de Paiva, and L. T. de Falco, Eds., vol. 292, 2018, pp. 15–30. [Online]. Available: https://doi.org/10.4204/EPTCS.292.2 [Cited on pages 24 and 28.]

[67] M. Coppo and M. Dezani-Ciancaglini, "A new type assignment for λ-terms," *Arch. Math. Log.*, vol. 19, no. 1, pp. 139–156, 1978. [Online]. Available: https://doi.org/10.1007/BF02011875 [Cited on page 27.]

[68] ——, "An extension of the basic functionality theory for the λ-calculus," *Notre Dame J. Formal Log.*, vol. 21, no. 4, pp. 685–693, 1980. [Online]. Available: https://doi.org/10.1305/ndjfl/1093883253 [Cited on page 27.]

[69] H. Barendregt, M. Coppo, and M. Dezani-Ciancaglini, "A filter lambda model and the completeness of type assignment," *J. Symb. Log.*, vol. 48, no. 4, pp. 931–940, 1983. [Online]. Available: https://doi.org/10.2307/2273659 [Cited on page 27.]

[70] F. Alessi, F. Barbanera, and M. Dezani-Ciancaglini, "Intersection types and lambda models," *Theor. Comput. Sci.*, vol. 355, no. 2, pp. 108–126, 2006. [Online]. Available: https://doi.org/10.1016/j.tcs.2006.01.004 [Cited on page 27.]

[71] H. P. Barendregt, W. Dekkers, and R. Statman, *Lambda Calculus with Types*, ser. Perspectives in logic. Cambridge University Press, 2013. [Online]. Available: http://www.cambridge.org/de/academic/subjects/mathematics/logic-categories-and-sets/lambda-calculus-types [Cited on page 27.]

[72] S. Ghilezan, "Strong normalization and typability with intersection types," *Notre Dame J. Formal Log.*, vol. 37, no. 1, pp. 44–52, 1996. [Online]. Available: https://doi.org/10.1305/ndjfl/1040067315

[73] S. van Bakel, "Intersection type assignment systems," *Theor. Comput. Sci.*, vol. 151, no. 2, pp. 385–435, 1995. [Online]. Available: https://doi.org/10.1016/0304-3975(95)00073-6 [Cited on page 27.]

[74] A. Bucciarelli, T. Ehrhard, and G. Manzonetto, "Not enough points is enough," in *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*, ser. Lecture Notes in Computer Science, J. Duparc and T. A. Henzinger, Eds., vol. 4646. Springer, 2007, pp. 298–312. [Online]. Available: https://doi.org/10.1007/978-3-540-74915-8_24 [Cited on page 28.]

[75] C. L. Ong, "Quantitative semantics of the lambda calculus: Some generalisations of the relational model," in *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*. IEEE Computer Society, 2017, pp. 1–12. [Online]. Available: https://doi.org/10.1109/LICS.2017.8005064 [Cited on page 28.]

[76] J. Girard, "Normal functors, power series and λ-calculus," *Ann. Pure Appl. Log.*, vol. 37, no. 2, pp. 129–177, 1988. [Online]. Available: https://doi.org/10.1016/0168-0072(88)90025-5 [Cited on page 28.]

[77] T. Ehrhard, "Call-by-push-value from a linear logic point of view," in *Programming Languages and Systems - 25th European Symposium on Programming, ESOP 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, ser. Lecture Notes in Computer Science, P. Thiemann, Ed., vol. 9632. Springer, 2016, pp. 202–228. [Online]. Available: https://doi.org/10.1007/978-3-662-49498-1_9 [Cited on page 28.]

[78] B. Accattoli and G. Guerrieri, "Types of fireballs," in *Programming Languages and Systems - 16th Asian Symposium, APLAS 2018, Wellington, New Zealand, December 2-6, 2018, Proceedings*, ser. Lecture Notes in Computer Science, S. Ryu, Ed., vol. 11275. Springer, 2018, pp. 45–66. [Online]. Available: https://doi.org/10.1007/978-3-030-02768-1_3 [Cited on page 40.]

[79] ——, "The theory of call-by-value solvability," *Proc. ACM Program. Lang.*, vol. 6, no. ICFP, pp. 855–885, 2022. [Online]. Available: https://doi.org/10.1145/3547652

[80] D. Kesner and A. Viso, "Encoding tight typing in a unified framework," in *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference)*, ser. LIPIcs, F. Manea and A. Simpson, Eds., vol. 216. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pp. 27:1–27:20. [Online]. Available: https://doi.org/10.4230/LIPIcs.CSL.2022.27

[81] S. Alves, D. Kesner, and D. Ventura, "A quantitative understanding of pattern matching," in *25th International Conference on Types for Proofs and Programs, TYPES 2019, June 11-14, 2019, Oslo, Norway*, ser. LIPIcs, M. Bezem and A. Mahboubi, Eds., vol. 175. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 3:1–3:36. [Online]. Available: https://doi.org/10.4230/LIPIcs.TYPES.2019.3 [Cited on page 40.]

[82] B. Accattoli, "(in)efficiency and reasonable cost models," in *12th Workshop on Logical and Semantic Frameworks, with Applications, LSFA 2017, Brasília, Brazil, September 23-24, 2017*, ser. Electronic Notes in Theoretical Computer Science, S. Alves and R. Wasserman, Eds., vol. 338. Elsevier, 2017, pp. 23–43. [Online]. Available: https://doi.org/10.1016/j.entcs.2018.10.003 [Cited on page 40.]

[83] E. Moggi, "Computational lambda-calculus and monads," in *4th Annual Symposium on Logic in Computer Science, (LICS), 1989, Pacific Grove, California, USA*. IEEE Computer Society,

1989, pp. 14–23. [Online]. Available: https://doi.org/10.1109/LICS.1989.39155 [Cited on pages 45 and 73.]

[84] G. D. Plotkin and J. Power, "Algebraic operations and generic effects," *Appl. Categorical Struct.*, vol. 11, no. 1, pp. 69–94, 2003. [Online]. Available: https://doi.org/10.1023/A:1023064908962 [Cited on pages 45 and 51.]

[85] G. D. Plotkin and A. J. Power, "Computational effects and operations: An overview," in *Proceedings of the Workshop on Domains VI 2002, Birmingham, UK, September 16-19, 2002*, ser. Electronic Notes in Theoretical Computer Science, M. Escardó and A. Jung, Eds., vol. 73. Elsevier, 2002, pp. 149–163. [Online]. Available: https://doi.org/10.1016/j.entcs.2004.08.008 [Cited on page 45.]

[86] N. Benton, J. Hughes, and E. Moggi, "Monads and effects," in *Applied Semantics, International Summer School, APPSEM 2000, Caminha, Portugal, September 9-15, 2000, Advanced Lectures*, ser. Lecture Notes in Computer Science, G. Barthe, P. Dybjer, L. Pinto, and J. Saraiva, Eds., vol. 2395. Springer, 2000, pp. 42–122. [Online]. Available: https://doi.org/10.1007/3-540-45699-6_2 [Cited on pages 45 and 56.]

[87] M. Hyland, P. B. Levy, G. D. Plotkin, and J. Power, "Combining algebraic effects with continuations," *Theor. Comput. Sci.*, vol. 375, no. 1-3, pp. 20–40, 2007. [Online]. Available: https://doi.org/10.1016/j.tcs.2006.12.026 [Cited on pages 45 and 56.]

[88] G. D. Plotkin and M. Pretnar, "Handlers of algebraic effects," in *Programming Languages and Systems, 18th European Symposium on Programming, ESOP 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, ser. Lecture Notes in Computer Science, G. Castagna, Ed., vol. 5502. Springer, 2009, pp. 80–94. [Online]. Available: https://doi.org/10.1007/978-3-642-00590-9_7 [Cited on pages 45 and 132.]

[89] M. Hyland, G. D. Plotkin, and J. Power, "Combining effects: Sum and tensor," *Theor. Comput. Sci.*, vol. 357, no. 1-3, pp. 70–99, 2006. [Online]. Available: https://doi.org/10.1016/j.tcs.2006.03.013 [Cited on pages 45, 48, 51, and 60.]

[90] S. Mac Lane, *Categories for the Working Mathematician*. Springer New York, 1978. [Online]. Available: http://dx.doi.org/10.1007/978-1-4757-4721-8 [Cited on page 47.]

[91] F. Gavazzo, "Coinductive equivalences and metrics for higher-order languages with algebraic effects. (equivalences coinductives et métriques pour les langages d'ordre

supérieur avec des effets algébriques)," Ph.D. dissertation, University of Bologna, Italy, 2019. [Online]. Available: https://tel.archives-ouvertes.fr/tel-02386201 [Cited on pages 47, 48, 51, and 55.]

[92] M. Hyland and J. Power, "The category theoretic understanding of universal algebra: Lawvere theories and monads," in *Computation, Meaning, and Logic: Articles dedicated to Gordon Plotkin*, ser. Electronic Notes in Theoretical Computer Science, L. Cardelli, M. Fiore, and G. Winskel, Eds., vol. 172. Elsevier, 2007, pp. 437–458. [Online]. Available: https://doi.org/10.1016/j.entcs.2007.02.019 [Cited on page 48.]

[93] F. W. Lawvere, "Functorial semantics of algebraic theories," *Proceedings of the National Academy of Sciences*, vol. 50, no. 5, p. 869–872, Nov. 1963. [Online]. Available: http://dx.doi.org/10.1073/pnas.50.5.869 [Cited on page 48.]

[94] J. Słomiński, *The theory of abstract algebras with infinitary operations*. Instytut Matematyczny Polskiej Akademi Nauk, 1959. [Online]. Available: http://eudml.org/doc/268556 [Cited on page 54.]

[95] H. Gaifman, "Infinite boolean polynomials i," *Fundamenta Mathematicae*, vol. 54, no. 3, pp. 229–250, 1964. [Online]. Available: http://eudml.org/doc/213758 [Cited on page 54.]

[96] A. Hales, "On the non-existence of free complete boolean algebras," *Fundamenta Mathematicae*, vol. 54, no. 1, pp. 45–66, 1964. [Online]. Available: http://eudml.org/doc/213773 [Cited on page 54.]

[97] A. Bauer and M. Pretnar, "Programming with algebraic effects and handlers," *J. Log. Algebraic Methods Program.*, vol. 84, no. 1, pp. 108–123, 2015. [Online]. Available: https://doi.org/10.1016/j.jlamp.2014.02.001 [Cited on page 57.]

[98] M. Pretnar, "An introduction to algebraic effects and handlers. invited tutorial paper," in *The 31st Conference on the Mathematical Foundations of Programming Semantics, MFPS 2015, Nijmegen, The Netherlands, June 22-25, 2015*, ser. Electronic Notes in Theoretical Computer Science, D. R. Ghica, Ed., vol. 319. Elsevier, 2015, pp. 19–35. [Online]. Available: https://doi.org/10.1016/j.entcs.2015.12.003

[99] A. Bauer, "What is algebraic about algebraic effects and handlers?" *CoRR*, vol. abs/1807.05923, 2018. [Online]. Available: http://arxiv.org/abs/1807.05923 [Cited on page 57.]

[100] G. D. Plotkin and J. Power, "Tensors of comodels and models for operational semantics," in *Proceedings of the 24th Conference on the Mathematical Foundations of Programming Semantics, MFPS 2008, Philadelphia, PA, USA, May 22-25, 2008*, ser. Electronic Notes in Theoretical Computer Science, A. Bauer and M. W. Mislove, Eds., vol. 218. Elsevier, 2008, pp. 295–311. [Online]. Available: https://doi.org/10.1016/j.entcs.2008.10.018 [Cited on pages 57, 58, and 61.]

[101] S. Alves, D. Kesner, and M. Ramos, "Extending the quantitative pattern-matching paradigm (full version)," 2024. [Online]. Available: https://arxiv.org/abs/2408.11007 [Cited on pages 65, 74, 75, and 79.]

[102] M. Florido and L. Damas, "Linearization of the lambda-calculus and its relation with intersection type systems," *J. Funct. Program.*, vol. 14, no. 5, pp. 519–546, 2004. [Online]. Available: https://doi.org/10.1017/S0956796803004970 [Cited on pages 85 and 86.]

[103] S. Alves, M. Fernández, M. Florido, and I. Mackie, "Linearity: A roadmap," *J. Log. Comput.*, vol. 24, no. 3, pp. 513–529, 2014. [Online]. Available: https://doi.org/10.1093/logcom/exs020 [Cited on page 85.]

[104] R. Atkey, "Parameterised notions of computation," *J. Funct. Program.*, vol. 19, no. 3-4, pp. 335–376, 2009. [Online]. Available: https://doi.org/10.1017/S095679680900728X [Cited on page 104.]

[105] Terese, *Term rewriting systems*, ser. Cambridge tracts in theoretical computer science. Cambridge University Press, 2003, vol. 55. [Cited on page 218.]