DEPENDENCY AND LINEARITY ANALYSES IN PURE TYPE SYSTEMS

Pritam Choudhury

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2023

Supervisor of Dissertation

Stephanie Weirich, ENIAC President's Distinguished Professor, Computer and Information Science

Graduate Group Chairperson

Mayur Naik, Professor, Computer and Information Science

Dissertation Committee

Rajeev Alur, Zisman Family Professor, Computer and Information Science
Dominic Orchard, Senior Lecturer, School of Computing, University of Kent and
Department of Computer Science and Technology, University of Cambridge
Benjamin C. Pierce, Henry Salvatori Professor, Computer and Information Science
Steve Zdancewic, Schlein Family President's Distinguished Professor, Computer and Information Science

Dedicated to my Gurudeva, Śrī Śrī Ṭhākura, who showed me the way.

Acknowledgement

One fine day, when I left my home in India for my doctoral studies at Penn, I didn't know how things would turn out. There was expectation, excitement, fear, hope, doubt, anticipation, all playing out in my heart. In the midst of this whirlpool of emotions, the faith of my mother and the spirit of my father kept me grounded. Standing upon these two pillars, I dreamed of creating something new. And thus, began my journey.

Six years have passed since then. During these six years, a lot has happened. I have gone through a diverse range of experiences and have learned from them. The successes and setbacks I had have together made me more resilient. The appreciations and criticisms I received have together made me more humble. The joys and pains I experienced have together made me more mature. When I look back now, I see that my journey has been quite different from what I had expected. But nevertheless, I am happy at how it has turned out. I shall take this opportunity to acknowledge the people without whose participation this journey would not have been possible.

First, I would like to thank my advisor, Stephanie Weirich, who helped me develop my research ideas and supported me for the past several years. I appreciate the freedom she gave me in working on problems I found interesting. Second, I would like to thank my research collaborators, Harley Eades III, Richard Eisenberg, Antoine Voizard and Stephanie Weirich, with whom I worked on projects that led to this dissertation. Third, I would like to thank the members of my Dissertation Committee, Rajeev Alur, Dominic Orchard, Benjamin C. Pierce and Steve Zdancewic, for their comments and feedback on my thesis proposal and the initial draft of this dissertation. Fourth, I would like to thank Scott Weinstein for the many interesting discussions on model theory we had throughout the last few years.

Next, I would like to thank my friends from the Penn PL Club, with whom I had several stimulating conversations on a wide range of topics. I would also like to thank other friends of mine, both here and back home, who lightened up my mood and motivated me on this journey. Thanks are also due to my younger sister who has been a very good friend all along, cheering me up with her kind words of support and encouragement.

I shall take this opportunity to thank two of my teachers from school days, Krishna Debnath and Keshab Chandra Saha, who taught me Maths with utmost care and instilled in me a love for the subject. That love for Maths motivated me to study Computer Science. So, I owe this thesis, in part, to their active encouragement during those formative years.

I shall also take this opportunity to thank Master Felice Macera and other instructors of the Penn TaeKwonDo Club. Joining this club has been one of the best decisions of my life. I shall remain grateful to Master Macera for the kindness he has shown me in teaching martial arts.

Finally, to the city of Philadelphia: thank you, for you have given me more than what I could ask for. Wherever I go, your memories shall always be in my heart. To you, Philly, I will remain indebted forever!

Abstract

DEPENDENCY AND LINEARITY ANALYSES IN PURE TYPE SYSTEMS

Pritam Choudhury

Stephanie Weirich

Dependency analysis is necessary to ensure proper flow of information in programming languages. Linearity analysis is necessary to ensure proper usage of resources in programming languages. These analyses are used for many purposes: for example, analyzing binding-time for partial evaluation of programs, ensuring data security in programs, ensuring deadlock freedom in concurrent programs, etc. What connects these analyses is that both of them need to model at least two different worlds with constrained mutual interaction. To elaborate, a typical dependency analysis may model low-security and high-security worlds with the constraint that information never leaks from the high-security world to the low-security world; a typical linearity analysis may model nonlinear and linear worlds with the constraint that derivations in the nonlinear world cannot make use of assumptions from the linear world. To statically enforce such constraints, dependency and linear type systems are employed. Several dependency and linear type systems have been proposed in literature, especially with respect to simple and polymorphic types. However, with respect to dependent types, the two analyses have not been explored much.

This dissertation explores how dependent types interact with dependency and linearity analyses. More precisely, in this dissertation, we extend dependency and linearity analyses to Pure Type Systems, which include several well-known dependent type systems. We build upon existing work on graded type systems to develop three specific graded calculi for Pure Type Systems: DDC for dependency analysis, GRAD for linearity analysis and LDC for combined dependency and linearity analysis. Our thesis is that these calculi provide a systematic way for analyzing dependency, linearity or a combination of the two in any Pure Type System. We study the metatheoretic properties of these calculi, show soundness of their analyses, discuss their applications and position them in the milieu of other dependency and linear calculi. Overall, our work provides insight into several nuances, hitherto unknown, of dependency and linearity analyses and their interactions with dependent types.

Contents

Acknowledgement iii

Abstract iv

Contents v

List of Tables x

List of Figures xi

Preface xiii

1 Introduction 1

1.1 Contextualizing Our Work on Linearity Analysis . . . 3
1.1.1 Linear Logic . . . 3
1.1.2 Early Linear Simple Type Systems: The Problems . . . 5
1.1.3 Split-Context Linear Simple Type Systems: Problems Resolved . . . 6
1.1.4 Linear Dependent Type Systems: The Challenge . . . 8
1.1.5 Bounded Linear Logic . . . 9
1.1.6 An Abstract Structure to Model Resource Usage . . . 11
1.1.7 Graded-Context Linear Simple Type Systems . . . 13
1.1.8 Linear Dependent Type Systems: Challenge Resolved . . . 16
1.1.9 Accounting Usage in Terms and Types: McBride and Atkey . . . 17
1.1.10 Accounting Usage in Terms and Types: Our GRAD . . . 18
1.1.11 Accounting Usage in Terms and Types: Moon et al. . . . 20
1.1.12 Summary . . . 20

1.2 Contextualizing Our Work on Dependency Analysis . . . 21
1.2.1 Dependency Analysis in Programming Languages . . . 21
1.2.2 An Abstract Structure to Model Dependency . . . 23
1.2.3 Dependency Core Calculus . . . 25
1.2.4 Our Calculi for Dependency Analysis . . . 26
1.2.5 Irrelevance Analysis in Dependent Type Systems . . . 26

1.3 Linearity and Dependency Analyses: A Comparison . . . 28
1.3.1 Preordered Semiring vs. Lattice . . . 28
1.3.2 Comonadic vs. Monadic . . . 30

1.4 Grading in Programming Languages . . . 31
1.4.1 A Tale of Multiple Worlds . . . 32

1.4.2 Graded Type Systems . . . 33
1.4.3 Grading and Type and Effect Systems . . . 34
1.4.4 Grading and Generative Type Abstraction . . . 35
1.5 Practical Significance of Our Work . . . 38
1.6 Contributions . . . 38

2 Dependency Analysis in Simple Type Systems **40**
2.1 Dependency Analysis in Action . . . 41
2.1.1 Information Flow Analysis . . . 41
2.1.2 Binding-Time Analysis . . . 43
2.2 Graded Monadic Calculus . . . 44
2.2.1 Grammar and Type System . . . 44
2.2.2 Equational Theory . . . 45
2.2.3 Graded Monads . . . 46
2.2.4 Categorical Model . . . 48
2.3 DCC and GMC . . . 49
2.3.1 Type System and Equational Theory of DCC . . . 50
2.3.2 Is DCC a Graded Monadic Calculus? . . . 51
2.3.3 Is DCC Just a Graded Monadic Calculus? . . . 52
2.4 Graded Comonadic Calculus . . . 53
2.4.1 Type System and Equational Theory . . . 53
2.4.2 Graded Comonad and Categorical Model . . . 55
2.5 DCC and GCC . . . 55
2.5.1 Protection Judgment, Revisited . . . 56
2.5.2 DCC_e is a Graded Comonadic Calculus . . . 57
2.6 Graded Monadic Comonadic Calculus . . . 58
2.6.1 The Calculus . . . 58
2.6.2 Categorical Model . . . 59
2.7 GMCC and DCC_e . . . 60
2.8 DCC_e: Categorical Semantics . . . 62
2.8.1 Categorical Models for DCC_e . . . 62
2.8.2 Proof of Noninterference . . . 64
2.9 Binding-Time Calculus, λ° . . . 65
2.9.1 The Calculus λ° . . . 66
2.9.2 Categorical Models for λ° . . . 67
2.9.3 Correctness of Binding-Time Analysis in λ° . . . 68
2.9.4 Can We Translate λ° to GMCC? . . . 70
2.10 GMCC_e . . . 71
2.10.1 The Calculus . . . 71
2.10.2 Relating DCC_e to GMCC_e . . . 73
2.10.3 Relating λ° to GMCC_e . . . 75
2.11 Discussions and Related Work . . . 76
2.11.1 Nontermination . . . 76
2.11.2 Algehed's SDCC . . . 77
2.11.3 Relational Semantics of DCC, Revisited . . . 77
2.12 Conclusion . . . 77

3 Dependency Analysis in Pure Type Systems **79**

3.1 Irrelevance Analysis as a Dependency Analysis 81
3.1.1 Run-Time Irrelevance 81
3.1.2 Compile-Time Irrelevance 82
3.1.3 Strong Irrelevant Σ-types 84
3.2 A Simple Dependency Calculus 86
3.2.1 The Calculus 86
3.2.2 Metatheoretic Properties 89
3.2.3 A Syntactic Proof of Noninterference 90
3.2.4 Relation with Sealing Calculus 93
3.3 Dependent Dependency Calculi 94
3.3.1 $\text{DDC}^{\top}$ and DDC: Graded Types and Terms 95
3.3.2 $\text{DDC}^{\top}$: Π-Types 97
3.3.3 $\text{DDC}^{\top}$: Σ-Types and Sum Types 98
3.3.4 Embedding SDC into $\text{DDC}^{\top}$ 99
3.3.5 Analyzing Run-time Irrelevance 100
3.4 DDC: Supporting Compile-Time Irrelevance 102
3.4.1 Towards Compile-Time Irrelevance 102
3.4.2 Core Type System 104
3.4.3 Σ-types 105
3.4.4 Noninterference 107
3.4.5 Consistency of Definitional Equality 108
3.4.6 Type-Soundness 110
3.4.7 Type-Checking in $\text{DDC}^{\top}$ and DDC 111
3.5 Discussions and Related Work 114
3.5.1 Irrelevance Analysis in Dependent Type Theories 114
3.5.2 Graded-Context Type Systems 115
3.5.3 Dependency Analysis and Dependent Type Systems 116
3.6 Conclusion 117

4 Linearity Analysis in Pure Type Systems 118
4.1 Modeling Usage: Preordered Semirings and Variants 119
4.2 A Graded-Context Simple Type System 120
4.2.1 The Basics 120
4.2.2 Type System 122
4.2.3 Type Soundness 124
4.3 Heap Semantics for GLC 125
4.3.1 The Step Judgment 126
4.3.2 Step Rules 127
4.3.3 Heap Semantics vs. Standard Operational Semantics 129
4.3.4 Accounting of Resources 130
4.3.5 Heap Compatibility 131
4.3.6 Graphical and Algebraic Views of Well-Formed Heaps 132
4.3.7 Soundness Theorem 134
4.4 Applications 135
4.4.1 No Usage 135
4.4.2 Linear Usage 136
4.5 GLC and SDC: A Comparison 137
4.5.1 Usage Analysis in SDC 138

4.5.2 Dependency Analysis in GLC . . . 140
4.6 Graded-Context Dependent Type System . . . 140
4.6.1 The Basics . . . 140
4.6.2 Type System . . . 141
4.6.3 Metatheory . . . 144
4.7 Heap Semantics for GRAD . . . 144
4.7.1 A Dependently-Typed Language with Definitions . . . 145
4.7.2 Heap Soundness Theorem . . . 147
4.8 GRAD and DDC$^{\top}$/DDC: A Comparison . . . 149
4.8.1 Technical Similarities and Differences . . . 149
4.8.2 Similarities and Differences in Analyses . . . 150
4.9 Discussions and Related Work . . . 151
4.9.1 GRAD as a General Coeffect Calculus . . . 151
4.9.2 Technical Comparison with QTT and GRTT . . . 152
4.9.3 Heap Semantics for Linear Calculi . . . 153
4.10 Conclusion . . . 154

5 Combined Linearity and Dependency Analysis in Pure Type Systems 155
5.1 Challenges and Resolution . . . 157
5.1.1 Dependency Analysis Through Graded-Context Type Systems . . . 157
5.1.2 Linearity Analysis Through Dependency Calculi . . . 160
5.2 Linearity and Dependency Analyses over Simple Types . . . 160
5.2.1 Type System for Linearity Analysis . . . 160
5.2.2 Metatheory of Linearity Analysis . . . 163
5.2.3 Type System for Dependency Analysis . . . 164
5.2.4 Metatheory of Dependency Analysis . . . 167
5.2.5 SLDC and SDC: A Comparison . . . 168
5.3 Heap Semantics for SLDC . . . 168
5.3.1 Reduction Relation . . . 169
5.3.2 Fair Enforcement of Usage and Dependency Constraints . . . 171
5.3.3 Soundness With Respect To Heap Semantics . . . 172
5.4 Linearity and Dependency Analyses in Pure Type Systems . . . 174
5.4.1 Type System of LDC . . . 174
5.4.2 Metatheory of LDC . . . 177
5.4.3 Heap Semantics for LDC . . . 177
5.5 Adding Unrestricted Usage . . . 179
5.5.1 A Problem and a Modification in Response . . . 180
5.5.2 Metatheory and Soundness of Modified System . . . 182
5.6 LDC: A General Calculus for Linearity and Dependency Analyses . . . 184
5.6.1 Combined Usage and Dependency Analysis . . . 184
5.6.2 Comparison with GRAD and DDC$^{\top}$ . . . 185
5.6.3 No Usage and Run-Time Irrelevance . . . 186
5.7 Discussions and Related Work . . . 186
5.7.1 SLDC and LNL λ-calculus . . . 186
5.7.2 LDC and McBride's System . . . 189
5.7.3 Other Calculi Related to LDC . . . 190
5.8 Conclusion . . . 191

6 Conclusion **192**
6.1 Grading: The Key Ideas of Our Work . . . 193
6.2 Areas That May Benefit from Our Work on Grading . . . 196
6.2.1 Dependent Session Types . . . 196
6.2.2 Generic Type and Effect System for Dependent Types . . . 198
6.3 Epilogue . . . 199

A Dependency Analysis in Simple Type Systems **200**
A.1 Proofs of Lemmas/Theorems Stated in Section 2.2 . . . 200
A.2 Proofs of Lemmas/Theorems Stated in Section 2.3 . . . 207
A.3 Proofs of Lemmas/Theorems Stated in Section 2.4 . . . 208
A.4 Proofs of Lemmas/Theorems Stated in Section 2.5 . . . 208
A.5 Proofs of Lemmas/Theorems Stated in Section 2.6 . . . 209
A.6 Proofs of Lemmas/Theorems Stated in Section 2.7 . . . 220
A.7 Proofs of Lemmas/Theorems Stated in Section 2.8 . . . 226
A.8 Proofs of Lemmas/Theorems Stated in Section 2.9 . . . 244
A.9 Proofs of Lemmas/Theorems Stated in Section 2.10 . . . 248
A.10 Proof of a Proposition Stated in Section 2.11 . . . 258

B Dependency Analysis in Pure Type Systems **261**
B.1 Proofs of Lemmas/Theorems Stated in Section 3.2 . . . 261
B.2 Proofs of Lemmas/Theorems Stated in Section 3.3 . . . 273
B.3 Proofs of Lemmas/Theorems Stated in Section 3.4 . . . 278

C Linearity Analysis in Pure Type Systems **308**
C.1 Proof of a Proposition Stated in Section 4.2 . . . 308
C.2 Proofs of Lemmas/Theorems Stated in Section 4.2 . . . 309
C.3 Proofs of Lemmas/Theorems Stated in Section 4.3 . . . 313
C.4 Proofs of Lemmas/Theorems Stated in Section 4.4 . . . 323
C.5 Proofs of Lemmas/Theorems Stated in Section 4.5 . . . 326
C.6 Proofs of Lemmas/Theorems Stated in Section 4.6 . . . 329
C.7 Proofs of Lemmas/Theorems/Corollaries Stated in Section 4.7 . . . 337
C.8 Proofs of Lemmas/Theorems Stated in Section 4.8 . . . 354

D Combined Linearity and Dependency Analysis in Pure Type Systems **357**
D.1 Proof of a Proposition Stated in Section 5.1 . . . 357
D.2 Proofs of Lemmas/Theorems Stated in Section 5.2 . . . 359
D.3 Proofs of Lemmas/Theorems Stated in Section 5.3 . . . 378
D.4 Proofs of Lemmas/Theorems/Corollaries Stated in Section 5.4 . . . 382
D.5 Proofs of Lemmas/Theorems Stated in Section 5.5 . . . 394
D.6 Proofs of Lemmas/Theorems Stated in Section 5.6 . . . 407
D.7 Proofs of Lemmas/Theorems Stated in Section 5.7 . . . 411

Bibliography **415**

List of Tables

3.1 Well-known type systems seen through PTS formalism 95

List of Figures

1.1 Examples of lattices for modeling dependencies 24

1.2 Examples of non-distributive lattices 29

2.1 Grammar of GMC 45

2.2 Typing rules of GMC (Excerpt) 45

2.3 Equality rules of GMC (Excerpt) 46

2.4 Commutative diagrams for lax monoidal functor 47

2.5 Commutative diagrams for strong endofunctor 48

2.6 Commutative diagram for strong natural transformation 48

2.7 Interpretation of GMC (Excerpt) 48

2.8 Protection rules for DCC 50

2.9 Translation function from GMC to DCC (Excerpt) 52

2.10 Typing rules of GCC (Excerpt) 54

2.11 Equality rules of GCC (Excerpt) 54

2.12 Interpretation of GCC (Excerpt) 55

2.13 Translation function from GCC to DCC_e (Excerpt) 58

2.14 Additional equality rules of GMCC 59

2.15 Grammar of λ° 66

2.16 Typing rules of λ° (Excerpt) 66

2.17 Equality rules of λ° (Excerpt) 67

2.18 Typing rules of $\lambda^\circ(k)$ (Excerpt) 69

2.19 Typing rules of GMCC_e 72

3.1 Grammar and typing rules of Simple Dependency Calculus 87

3.2 Operational semantics of SDC (Excerpt) 90
3.3 Indexed indistiguishability relation for SDC (Excerpt) 91
3.4 Grammar of Dependent Dependency Calculus (Types and terms) 95
3.5 $\text{DDC}^{\top}$ type system (Core rules) 96
3.6 Indexed indistiguishability relation for $\text{DDC}^{\top}$ and DDC (Excerpt) 101
3.7 DDC type system (Core rules) 104
3.8 Indexed definitional equality relation for DDC (Excerpt) 106
3.9 Parallel reduction relation (Excerpt) 109
3.10 Transitive closure of parallel reduction relation 109
3.11 Consistency relation between types 110
3.12 Key typing rules and step rules of ICC^{*} 113
3.13 Translation from a sublanguage of DDC to ICC^{*} 114

4.1 Ordering for tracking linear usage and affine usage 119
4.2 Grammar of GLC 120
4.3 Small-step call-by-name reduction (Excerpt) 124
4.4 Type and term translation from $\text{GLC}(\mathbb{B}_{\geq})$ to $\text{SDC}(\mathbb{B}_{\geq})$ 138
4.5 Term translation from $\textsc{Grad}(\mathbb{B}_{\geq})$ to $\text{DDC}^{\top}(\mathbb{B}_{\geq})$ (Excerpt) 151

5.1 Grammar of $\text{SLDC}(\mathcal{Q}_{\mathbb{N}})$ 161
5.2 Typing rules of $\text{SLDC}(\mathcal{Q}_{\mathbb{N}})$ 161
5.3 Small-step reduction for SLDC (Excerpt) 164
5.4 Typing rules of $\text{SLDC}(\mathcal{L})$ (Excerpt) 165
5.5 Heap semantics for SLDC (Excerpt) 170
5.6 Typing rules of LDC 175
5.7 Equality rules of LDC (Excerpt) 175
5.8 Typing rules with definitions 178
5.9 Compatibility relation with definitions 178
5.10 Grammar of LNL λ-calculus 187
5.11 Typing rules of LNL λ-calculus 188
5.12 β-rules of LNL λ-calculus 188
5.13 Type and term translation from LNL λ-calculus to $\text{SLDC}(\mathcal{Q}_{\text{Lin}})$ 189

Preface

Perspective plays an important role in shaping our understanding of the physical world. The same entity, when viewed from a different perspective, may appear to be very different. The ancient Indian story of the blind men and the elephant is an age-old testimony to this phenomenon. What makes this story interesting is that though each blind man had a unique description of the animal, all of them were equally correct. So in a hypothetical world inhabited solely by these blind men, the question, 'How does an elephant look like?', has multiple correct answers. The answers depend not just on the elephant but also on the men who came in contact with the animal. This story illustrates how our understanding of a physical entity is not just a function of the entity alone but also of factors external to the entity, in particular, our perspective.

Perspective plays an equally important role in understanding program behavior. To provide a concrete instance, consider the question: Are two given programs equivalent? For some given pair of programs, the answer to this question can be both yes and no. For example, considering from a security standpoint, suppose, the two given programs output high-security values of the same type. Then, to an observer with just low-security clearance, the two programs would appear equivalent whereas an observer with high-security clearance may find them to be different. So, the equivalence of two programs is not just a function of the programs themselves but also of factors external to the programs, in particular, the 'perspective of the observer'.

Similar to perspective, environment also plays an important role in shaping our understanding of the physical world. The same entity, when viewed in a different environment, may appear to be very different. There is an ancient Indian story, similar to the above one, that illustrates this point: Four brothers seriously disagree in their descriptions of Kiṃśuka (*Butea monosperma*) tree because they saw the tree during different seasons. So the question, 'How does a Kiṃśuka tree look like?' has multiple correct answers, depending upon the season one sees the tree.

Environment plays an equally important role in understanding program behavior. To provide a concrete instance, consider the question: Can an array be destructively updated? For a given array, the answer to this question can be both yes and no, depending upon the number of pointers to the array. If the array has just one pointer to it, then it can be destructively updated. Otherwise, it should not be destructively updated. Thus, the choice of destructive update of an array is a function of the environment in which it exists, or more precisely, the number of pointers to it.

'Perspective' and 'Environment' are notions central to dependency and linearity analyses respectively. This dissertation formalizes these notions towards analyzing dependency and linearity in Pure Type Systems. This formalization answers important technical questions and helps us understand the two analyses in a new light. In this dissertation, our focus is on dependency and linearity analyses, but we believe that our formalization and the techniques we develop have general value and can be used in other static analyses of Pure Type System programs.

Chapter 1

Introduction

Type systems formalize our intuition of correct programs: upon formalization, a correct program is one that is well-typed. Our intuition of correctness, however, varies depending upon application. For example: in a security system, correctness would entail absence of read access to secret files by public users; in a distributed system, correctness would entail absence of simultaneous write access to a file by multiple users. To formalize such and other similar notions of correctness, we employ dependency type systems and linear type systems. These type systems have a wide variety of applications.

Dependency type systems are good at *controlling information flow.* In the form of security type systems [Heintze and Riecke, 1998, Volpano et al., 1996], they guarantee that low-security outputs do not depend upon high-security inputs. In the form of binding-time type systems [Davies, 2017, Gomard and Jones, 1991], they guarantee that early-bound expressions do not depend upon late-bound ones. Dependency type systems are quite common in practical programming languages. For example, the metaprogramming language MetaOcaml [Calcagno et al., 2003] is based on the dependency type system of Davies [2017]. The Jif extension of Java, employed for ensuring secure information flow, is based on the dependency type system of Myers [1999].

Linear type systems are good at *managing resource usage and sharing.* They can be employed for fine-grained memory management [Chirimar et al., 2000, Fluet et al., 2006]. They can reason about state and enable compiler optimizations like in-place update of memory locations [Wadler, 1990]. They can guarantee absence of deadlocks in distributed systems, via session types [Caires and Pfenning, 2010]. Owing to their utility, linear types have found their way into several programming languages, like Haskell [Bernardy et al., 2018], Granule [Orchard et al., 2019] and Idris 2 [Brady, 2021].

Though dependency type systems and linear type systems serve different purposes, they essentially address the same abstract problem. The problem is to model two different worlds that interact following given constraints. For example: a typical dependency type system needs to model low-security and high-security worlds with the constraint that information from the high-security world never leaks into the low-security

world; a typical linear type system needs to model nonlinear and linear worlds with the constraint that derivations in the nonlinear world cannot make use of assumptions from the linear world. This fundamental similarity connects dependency and linearity analyses and forms the basis of this dissertation.

There are several type systems for dependency analysis [Abadi, 2006, Abadi et al., 1999, Davies, 2017, Hatcliff and Danvy, 1997, Heintze and Riecke, 1998, Shikuma and Igarashi, 2006, Volpano et al., 1996, etc.] and linearity analysis [Abramsky, 1993, Barber, 1996, Benton, 1994, Benton et al., 1993, Brunel et al., 2014, Ghica and Smith, 2014, etc.] in simple and polymorphic types. However, with regard to dependent types, these analyses have not been explored much. But with the growing use of dependent types both in theory and practice, dependency and linearity analyses in these type systems are much needed. The reason behind this need is that the problems these analyses address in simple and polymorphic type systems also arise in dependent type systems. For example, analyzing information flow, analyzing binding-time, reasoning about state, etc. are important in dependent type systems as well.

To address this need, we extend dependency and linearity analyses to Pure Type Systems. A Pure Type System (PTS) [Barendregt, 1993] is a general formalism that captures many well-known dependent type systems like λP, Calculus of Constructions (CoC), Type-in-Type, etc. (in addition to capturing simple and polymorphic type systems like Simply-Typed λ-Calculus and Polymorphic λ-Calculus). In this dissertation, we design the following calculi for Pure Type Systems: DDC for dependency analysis, GRAD for linearity analysis and LDC for combined dependency and linearity analysis. We demonstrate various applications of these calculi. We use DDC for enforcing security and binding-time constraints in dependently-typed programs. We use GRAD for enforcing constraints on variable use and reasoning about state in dependently-typed programs. LDC brings DDC and GRAD under the same umbrella (roughly speaking) and combines their individual applications.

The extension of dependency and linearity analyses to dependent type systems is a challenging problem. The main challenge lies in extending the analyses from just terms in simple/polymorphic type systems to both terms and types in dependent type systems. The difficulty of the challenge, particularly in relation to linearity analysis, is well-known [McBride, 2016]. With regard to dependency analysis, the problem did not receive much attention, presumably because there are some calculi [Abel and Scherer, 2012, Bernardy and Guilhem, 2013, Mishra-Linger and Sheard, 2008, etc.] for analyzing specific dependencies in dependent type systems. Our interest, however, lies in a general dependency calculus for dependent type systems, one that can be used for a variety of analyses like security, binding-time, etc. The problem of designing such a calculus has been open.

In extending linearity and dependency analyses to dependent type systems, we draw upon existing literature on the two analyses. To appreciate the novelty of our work, it is important to understand it in the context of this existing literature. As such, we shall briefly review the milieu of linearity and dependency analyses, with special emphasis on type systems that acted as precursors to our work. We start with linearity analysis because it has a richer literature than dependency analysis.

1.1 Contextualizing Our Work on Linearity Analysis

1.1.1 Linear Logic

Linearity analysis is based on linear logic [Girard, 1987]. Linear logic is a logic of state [Girard, 1995]. Unlike traditional classical or intuitionistic logic where true propositions are always true, in linear logic, true propositions may change their state and become false. This flexibility enables linear logic to reason about dynamic practical situations that are not readily captured by traditional logics. For example, consider the situation: Whenever I move out of my office, the statement 'I am in my office' changes its truth value. Traditional logics do not provide a straightforward way to reason about such statements. But in linear logic, we may model moving out of office through the axiom, $\texttt{moveOut} : \forall x.\texttt{in}(x) \multimap \texttt{out}(x)$, which means that $\texttt{moveOut}(x)$ is a transformation that consumes proposition $\texttt{in}(x)$, denoting x is in office, and produces proposition $\texttt{out}(x)$, denoting x is out of office (x here stands for individuals). This idea of treating *propositions as resources* that are consumed and produced forms the basis of reasoning about state in linear logic.

Now that propositions are resources, they should not be arbitrarily copied or discarded because otherwise, they lose their meaning. For example, consider the situation: I am in my office and I move out of my office. This situation may be represented as: $\texttt{in}(x) \otimes \texttt{moveOut}(x)$ (implying $\texttt{out}(x)$). If copying were allowed, we would run into a contradiction: $\texttt{in}(x) \otimes \texttt{moveOut}(x) \multimap \texttt{in}(x) \otimes \texttt{in}(x) \otimes \texttt{moveOut}(x) \multimap \texttt{in}(x) \otimes \texttt{out}(x)$. Thus, arbitrary copying and discarding of propositions are prohibited in linear logic. Linear logic does so by forgoing the structural rules of contraction and weakening respectively. Recall that contraction allows replacement of multiple copies of the same premise in the context (of a judgment) with a single copy of that premise while weakening allows addition of extra premises in the context (of a judgment). For example, in a natural deduction system, contraction allows the derivation of $A \vdash B$ from $A, A \vdash B$ while weakening allows the derivation of $A, C \vdash B$ from $A \vdash B$, where A, B and C are propositions. Linear logic, however, retains the structural rule of exchange which allows reordering of the premises in the context. For example, in linear logic, $B, A \vdash C$ follows from $A, B \vdash C$.

Since linear logic forgoes some of the structural rules, it is also referred to as a substructural logic [Schroeder-Heister and Došen, 1994]. Note here that linear logic is not the only substructural logic; there are other substructural logics that forgo a different set of structural rules. For example, relevance logic forgoes the weakening rule only, noncommutative logic forgoes all the above three structural rules, bounded linear logic (to be introduced in Section 1.1.5) forgoes the contraction rule only, etc. For an in-depth exposition on substructural logics, see Restall [1999].

While forgoing the structural rules of contraction and weakening is necessary for reasoning about resources, it also limits the expressive power of the logic, as explained next. From a resource perspective, the intuitionistic implication, $f : A \to B$, may be read as: f consumes some number of copies (can also be 0) of A and produces one copy of B. The linear implication, $g : A \multimap B$, on the other hand, requires that g consume exactly one copy of A. Owing to this difference, the intuitionistic implication cannot be expressed through the linear implication unless copying and discarding of resources are allowed. To resolve this impasse, linear logic

introduces copying and discarding, but in a restricted manner via the modality, !, also referred to as an exponential. Resources appearing under this exponential modality can be copied and discarded arbitrarily, as shown below.

$$\frac{\Gamma, !A, !A \vdash B}{\Gamma, !A \vdash B}\ (Contract) \qquad\qquad \frac{\Gamma \vdash B}{\Gamma, !A \vdash B}\ (Weaken)$$

Here, Γ is a multiset of propositions. Note that only resources appearing under the !-modality, also referred to as *nonlinear* resources, may be copied or discarded; other resources are *linear* and cannot be copied or discarded. To give an example, the function $f' : !A \multimap B \multimap C$, in its definition, may copy or discard A, but not B. Now, even though linear and nonlinear resources behave differently, they do interact via the rules shown below, called promotion and dereliction.

$$\frac{!\Gamma \vdash B}{!\Gamma \vdash !B}\ (Promote) \qquad\qquad \frac{\Gamma, A \vdash B}{\Gamma, !A \vdash B}\ (Derelict)$$

Here, $!\Gamma$ is a multiset of the form $!A_1, !A_2, \ldots, !A_n$. The intuition behind the promotion rule is that if every hypothesis in the derivation of a proposition is nonlinear, then that proposition may be used nonlinearly. The intuition behind the dereliction rule is that any nonlinear resource may be used linearly. Note that the converse is not true in general: a linear resource cannot be used nonlinearly, unless it is derived using nonlinear resources only. The above four rules formalize the behavior of the !-modality. With this modality, linear logic regains its lost expressive power. As a matter of fact, it subsumes standard intuitionistic logic. So, one may view linear logic as an intuitionistic logic that can, additionally, reason about state. Note here that there is a classical version of linear logic as well but we don't consider the classical version in this dissertation.

Now, programming languages need an intuitionistic logic [Martin-Löf, 1982] and the ability to reason about state. So it comes as no surprise that soon after the introduction of linear logic, type systems based on the logic [Abramsky, 1993, Lafont, 1988, Lincoln and Mitchell, 1992, Wadler, 1990, 1994, etc.] were proposed for programming. Since the introduction of these early linear type systems, three decades have passed. During these decades, a lot of research has happened in the arena of linear type systems. Thanks to this research, linear types and their variants have found their way into mainstream programming languages like Haskell, Rust, etc. However, there are still many interesting open problems in this arena. As such, the research on linear type systems is consistently up and running [Atkey, 2018, Choudhury et al., 2021, McBride, 2016, Moon et al., 2021, etc]. To understand where our work stands in the midst of all this research on linear type systems, we first need to draw an outline of the trajectory of the research itself.

1.1.2 Early Linear Simple Type Systems: The Problems

The problem of designing a linear type system, even for simple types, is far from trivial. Only with trial and error, did researchers come up with systems that are now regarded as standards. These standard systems may be easy to understand and work with but they were certainly not easy to come up with. So, we would like to caution the reader against hindsight bias as we discuss the evolution of linear type systems.

Linear type systems need to model two types of resources: linear resources that must be used exactly once and nonlinear resources that may be used any number (including 0) of times. How can the same type system model both linear and nonlinear resources? This question perplexed early researchers on linear type systems.

The solution proposed by Abramsky [1993] in his Linear Term Calculus was to let all resources (i.e., propositions) be linear by default and then mark the nonlinear ones among them with a type former, !. To illustrate, in Abramsky's Linear Term Calculus, the resource $x : A$ is linear by default; but if $A = !B$, then it may be used as a nonlinear resource. The variable rule of the calculus, shown below, illustrates the default linear nature of resources. Observe that this rule does not permit contraction or weakening, which are restricted to nonlinear resources.

$$\textsc{LTC-Var}\quad \frac{}{x : A \vdash x : A}$$

Several early linear type systems [Benton et al., 1993, Lincoln and Mitchell, 1992, Wadler, 1994, etc.] followed this approach of Abramsky's Linear Term Calculus. While this approach works, it also has some issues:

1. This approach leads to a problematic promotion rule. To elaborate, the promotion rule of Abramsky's system, shown below, does not commute with substitution.

   $$\textsc{LTC-Promote}\quad \frac{!\Gamma \vdash b : B}{!\Gamma \vdash !b : !B}$$

   Here, $!\Gamma$ is a multiset of the form $x_1 : !A_1, x_2 : !A_2, \ldots, x_n : !A_n$. To see why promotion does not commute with substitution, consider the following judgments: $y : !B \vdash y : !B$ and $z : A \multimap !B, x : A \vdash z\,x : !B$. With regard to these judgments, promotion followed by substitution gives us: $z : A \multimap !B, x : A \vdash !(z\,x) : !!B$. But if we first substitute, we get stuck at $z : A \multimap !B, x : A \vdash z\,x : !B$ because we cannot apply rule LTC-Promote to this judgment. Owing to this noncommutativity, substitution is inadmissible in Abramsky's system, and as such, is added as an explicit typing rule.

   Later authors fixed this problem by baking substitution directly into the promotion rule. For example, in the Intuitionistic Linear Type System of Benton et al. [1993], the promotion rule is simultaneously a promotion rule and a substitution rule, as shown below.

$$\frac{\Gamma_1 \vdash a_1 : !A_1 \quad \ldots \quad \Gamma_n \vdash a_n : !A_n \qquad x_1 : !A_1, \ldots, x_n : !A_n \vdash b : B}{\Gamma_1, \ldots, \Gamma_n \vdash \textbf{promote } a_1, \ldots, a_n \textbf{ for } x_1, \ldots, x_n \textbf{ in } b : !B} \text{ ILT-PROMOTE}$$

This fix makes substitution admissible, but at the cost of a complex promotion rule.

2. This approach leads to verbose type systems. Such type systems require explicit term-level constructs (and therefore typing rules) for contraction, weakening, promotion and dereliction. The contraction, weakening and dereliction rules of the Intuitionistic Linear Type System of Benton et al. [1993] are shown below:

$$\frac{\Gamma_1 \vdash a : !A \qquad \Gamma_2, x : !A, y : !A \vdash b : B}{\Gamma_1, \Gamma_2 \vdash \textbf{copy } a \textbf{ as } x, y \textbf{ in } b : B} \text{ ILT-CONTRACT}$$

$$\frac{\Gamma_1 \vdash a : !A \qquad \Gamma_2 \vdash b : B}{\Gamma_1, \Gamma_2 \vdash \textbf{discard } a \textbf{ in } b : B} \text{ ILT-WEAKEN}$$

$$\frac{\Gamma \vdash a : !A}{\Gamma \vdash \textbf{derelict } a : A} \text{ ILT-DERELICT}$$

Now, rule ILT-PROMOTE may be thought of as an introduction rule for !, with rules ILT-CONTRACT, ILT-WEAKEN, and ILT-DERELICT being the elimination rules. Having three elimination rules for a type former leads to an involved semantics. Refer to the operational semantics of the Intuitionistic Linear Type System given by Benton et al. [1993]; in particular, note the long list of commuting conversions presented in Figure 6 of the paper.

On a closer analysis, one finds that the root cause of both the issues described above is the safe but restrictive assumption that all resources are linear by default. The assumption is safe because nonlinear resources can be treated linearly. But the assumption is also restrictive because nonlinear resources need to be identified and treated differently. This observation, then, leads us to a natural question: Can we do away with the above assumption? Put differently, is there a replacement for the above assumption? Now, one replacement might be to distinguish between linear and nonlinear resources from the very outset, as opposed to treating resources as linear by default. This alternative approach, though simple, turned out to be effective in resolving the issues mentioned above. As a matter of fact, this approach laid the foundation for the later and now standard linear simple type systems, the Linear Nonlinear (LNL) λ-calculus of Benton [1994] and the Dual Intuitionistic Linear Logic (DILL) of Barber [1996].

1.1.3 Split-Context Linear Simple Type Systems: Problems Resolved

Girard [1993] designed a logic called **LU**, which unifies classical, intuitionistic and linear logics into a single system by dividing sequents into two zones: one that is maintained as in classical logic and the other that is maintained as in linear logic. Inspired by **LU**, both Benton [1994] and Barber [1996] designed linear type systems where contextual assumptions of typing judgments are split into two zones: nonlinear and linear. The idea behind the splitting is that while assumptions in the nonlinear zone could be used without

restriction, those in the linear zone had to be used linearly. With this simple modification, Benton [1994] and Barber [1996] were able to resolve the issues their predecessors faced:

1. The promotion rule in their systems is quite straightforward: promote whenever the linear zone in the context is empty. See below the promotion rule of DILL of Barber [1996], rule DILL-ExpIntro.

$\boxed{\Delta\ ;\ \Gamma \vdash a : A}$ *(Introduction and elimination rules for !)*

$$\text{DILL-ExpIntro}\quad \frac{\Delta\ ;\ \varnothing \vdash a : A}{\Delta\ ;\ \varnothing \vdash !a : !A}
\qquad
\text{DILL-ExpElim}\quad \frac{\Delta\ ;\ \Gamma_1 \vdash a : !A \qquad \Delta, x : A\ ;\ \Gamma_2 \vdash b : B}{\Delta\ ;\ \Gamma_1, \Gamma_2 \vdash \mathbf{let}\ !x\ \mathbf{be}\ a\ \mathbf{in}\ b : B}$$

Here Δ and Γ denote the nonlinear and linear zones of the context respectively. Note that in contrast to three elimination rules for ! in the single-context system of Benton et al. [1993] (rules ILT-Contract, ILT-Weaken, and ILT-Derelict), DILL has just one elimination rule for !, rule DILL-ExpElim.

2. The systems of Benton [1994] and Barber [1996] do not have explicit constructs for weakening and contraction. Weakening and contraction are carried out implicitly through the variable rules. Observe the two variable rules of DILL of Barber [1996], shown below, to see how.

$$\text{DILL-VarL}\quad \frac{}{\Delta\ ;\ x : A \vdash x : A}
\qquad
\text{DILL-VarNL}\quad \frac{x : A \in \Delta}{\Delta\ ;\ \varnothing \vdash x : A}$$

In short, by splitting the context of a typing judgment into two zones, LNL λ-calculus and DILL provide a simple and clean formalism for linearity analysis in simply-typed setting, something that was missing from the earlier linear simple type systems. Note here that though both LNL λ-calculus and DILL are split-context systems, there are still some differences between the two. These differences are not important with respect to the discussion in this chapter, but we shall come back to them in Chapter 5.

Before moving ahead, let us reflect on the reasons behind the simplicity of split-context systems compared to their single-context counterparts:

1. **Separation:** Unlike single-context systems, split-context systems have an *intrinsic* mechanism for separating nonlinear resources from linear ones. Owing to this separation, split-context systems can easily implement promotion. For example, compare rule DILL-ExpIntro with rules LTC-Promote and ILT-Promote. While both rules LTC-Promote and ILT-Promote need to ensure that a premise judgment has context of the form $x_1 : !A_1, \ldots, x_n : !A_n$, rule DILL-ExpIntro just needs to ensure that the linear zone of the context is empty.

2. **Flexibility:** Separation of resources helps make split-context systems more flexible. To elaborate, these systems can allow resource-agnostic and resource-conscious reasoning simultaneously. For example, compare the two variable rules DILL-VarL and DILL-VarNL of Barber [1996] with the sole variable rule LTC-Var of Abramsky [1993]. Rules DILL-VarL and DILL-VarNL are resource-agnostic with respect to the left zone of the context while being resource-conscious with respect to the right zone. Rule LTC-Var, on the other hand, is resource-conscious through and through. Thanks to this flexibility, Barber [1996] does not need explicit rules for contraction and weakening of nonlinear resources, unlike Abramsky [1993].

Separation and flexibility are two motifs that we shall come back to again and again in this dissertation. But for now, we move on to linear dependent type systems.

1.1.4 Linear Dependent Type Systems: The Challenge

LNL λ-calculus and DILL are effective in tracking linearity in simple type systems. So in due course, these systems were extended to dependent types. Krishnaswami et al. [2015] extended LNL λ-calculus to dependent types while Vákár [2015], building upon Cervesato and Pfenning [2002], did the same for DILL. The systems of Krishnaswami et al. [2015] and Vákár [2015] are interesting because they bring linearity and dependent types together. However, they do so only after imposing a major restriction: *types cannot depend upon linear assumptions.*

One might say, with this restriction in place, these systems are not really combining linearity with dependent types. For example, consider the following functions from Agda Standard Library [Agda Team, 2021], with their types refined: `fromℕ : (n : ℕ) ⊸ Fin (suc n)` and `_ℕ-ℕ_ : (n : ℕ) ⊸ Fin (suc n) ⊸ ℕ`. Here, ℕ is the type of natural numbers and `Fin n` is the type of finite numbers less than `n`. The function `fromℕ` takes a natural number and returns the 'same' number. The function `_ℕ-ℕ_` takes a natural number `n` and a finite number less than or equal to `n` and returns the difference between the two. The types of these functions, as given above, cannot be expressed in the systems under discussion because the types depend upon the linear assumption, `n : ℕ`. Thus, these systems leave something to be desired with regard to their combination of linearity and dependent types.

Now, if we carefully look at the systems of Krishnaswami et al. [2015] and Vákár [2015], we realize that the restriction they place on types is not their choice per se but is forced upon them by the simple type systems they extend. To see why, consider the application rule of DILL (rule DILL-App) and a hypothetical extension (rule DILL-DApp) of this rule to dependent types:

$$\text{DILL-App}\quad \frac{\Delta\ ;\ \Gamma_1 \vdash b : A \multimap B \qquad \Delta\ ;\ \Gamma_2 \vdash a : A}{\Delta\ ;\ \Gamma_1, \Gamma_2 \vdash b\ a : B} \qquad\qquad \text{DILL-DApp}\quad \frac{\Delta\ ;\ \Gamma_1 \vdash b : (x : A) \multimap B \qquad \Delta\ ;\ \Gamma_2 \vdash a : A}{\Delta\ ;\ \Gamma_1, \Gamma_2 \vdash b\ a : B\{a/x\}}$$

Observe that in rule DILL-DApp, the type A can depend neither upon Γ_1 nor upon Γ_2 because Δ, Γ_1 and Γ_2 are mutually disjoint. As such, A can depend only upon Δ. In other words, A cannot depend upon any linear assumptions. Owing to this restriction, in type $(x : A) \multimap B$, the body B should not depend upon the linear assumption, $x : A$; meaning, the type $(x : A) \multimap B$ is essentially $A \multimap B$. This example shows why these systems cannot accommodate linear dependent types.

To summarize, though rule DILL-App works well in a simply-typed system, its extension, rule DILL-DApp, does not work well in a dependently-typed system because it forces types to not depend upon linear assumptions. This case in point suggests that for extending linearity analysis to dependent types, split-context linear simple type systems are not ideal starting points. Put differently, one needs to start out with linear simple type systems that are more flexible. Fortunately, the foundation for such type systems had already been laid down by bounded linear logic [Girard et al., 1992].

1.1.5 Bounded Linear Logic

Linear logic divides resources into two categories: linear and nonlinear. While linear resources must be used exactly once, nonlinear resources may be used any number of times. Bounded linear logic (BLL) [Girard et al., 1992] brings precision into the category of nonlinear resources by introducing fine-grained distinctions on use of such resources. For example, BLL distinguishes resources that are used at most n_1 times from those that are used at most n_2 times, where $n_1, n_2 \in \mathbb{N}$ and $n_1 \neq n_2$. BLL introduces such distinctions to characterize resource-bound computations, or more specifically, polynomial-time computations.

To characterize polynominal-time computability, Girard et al. [1992], building upon Nerode et al. [1989], introduced bounded linear logic where the !-modality of linear logic is graded with polynomials (having natural number coefficients) that represent bounds on usage. For example, a resource A that may be used at most p times is represented as $!_p A$, where p is a polynomial. The inference rules of BLL mirror those of linear logic. However, owing to grading, the inference rules involving the !-modality, i.e., promotion, dereliction, contraction and weakening, get modified in BLL. Below, we present simplified versions of the modified rules in BLL [Girard et al., 1992]. (The exact rules are more complex and not necessary for the purpose of this discussion.)

$$\frac{!_{\overrightarrow{y}}\Gamma \vdash B}{!_{x\overrightarrow{y}}\Gamma \vdash !_x B}\ (BPromote) \qquad \frac{\Gamma, A \vdash B}{\Gamma, !_1 A \vdash B}\ (BDerelict)$$

$$\frac{\Gamma, !_x A, !_y A \vdash B}{\Gamma, !_{x+y} A \vdash B}\ (BContract) \qquad \frac{\Gamma \vdash B}{\Gamma, A \vdash B}\ (BWeaken)$$

Here, x, y are variables that represent polynomials; $\overrightarrow{y} = y_1, \ldots, y_n$ denotes a vector of variables; $x\overrightarrow{y}$ denotes scalar multiplication of vector $\overrightarrow{y}$ by x; and finally, given $\Gamma = A_1, \ldots, A_n$ and $\overrightarrow{y} = y_1, \ldots, y_n$, context $!_{\overrightarrow{y}}\Gamma$ is a shorthand for $!_{y_1} A_1, \ldots, !_{y_n} A_n$. Note the precision with respect to resource usage in rules *BPromote*, *BDerelict* and *BContract* of bounded linear logic, compared to rules *Promote*, *Derelict* and *Contract* of linear logic, presented in Section 1.1.1. (We shall come to rule *BWeaken* shortly.) This precision is made possible by the grading of the !-modality.

Grading modalities to make finer distinctions in a language is a powerful technique that has found applications in several disciplines including computer science, logic, linguistics and philosophy. This technique is a cornerstone of this dissertation. All the novel linearity and dependency calculi presented in this dissertation build upon calculi that use this technique. We shall discuss the nuances of this technique and their ramifications throughout the following chapters. Even in this chapter, as we move along, we shall see some of the benefits accruing from grading modalities. We shall take up a full-fledged discussion on grading in Section 1.4, after we finish the ongoing discussion on linearity and dependency analyses.

We see above that BLL enables fine-grained reasoning about resource usage through the graded exponential modality. Undoubtedly, compared to linear logic, BLL can express finer distinctions between resources. However, BLL does not subsume linear logic. In particular, BLL cannot model the !-modality of linear logic, which represents unrestricted, and thus unbounded, usage. This failure, however, should not be seen as a shortcoming since unfettered exponential modality can be used to express computations that have 'no realistic time bounds' [Girard et al., 1992]. But if we move away from the domain of complexity analysis to that of programming languages, where we are at ease even with nonterminating computations, we may wish to allow unrestricted usage alongside bounded usage. That would enable fine-grained reasoning about usage without compromising on the benefits accorded by linear logic. To this end, we introduce the infinite cardinal ω, a solution of the equation $x = x + 1$, to model unrestricted usage. (Strictly speaking, we should have used the symbol $\aleph_0$ in lieu of ω. But we go with ω, following existing literature on linear type systems.) The exponential modality, !, of linear logic is, then, regarded as $!_\omega$.

BLL, extended with $!_\omega$, may seem to be a more expressive version of linear logic; however, it is not quite so. The reason is that BLL allows arbitrary discarding of resources whereas linear logic does not. Compare the weakening rules of the two logics: *BWeaken* (shown above) and *Weaken* (shown in Section 1.1.1). While BLL allows unfettered weakening, linear logic allows weakening only under the !-modality. This difference between the two logics has important consequences. One consequence is that judgments that are valid in one may not be so in the other. For example: for any proposition A, the judgment $\varnothing \vdash A \otimes A \multimap A$ is valid in BLL but not so in linear logic. This difference is, in some sense, irreconcilable because BLL and linear logic are substructural logics that treat the weakening rule differently.

Now, from a programming languages perspective, one may be interested in analyzing exact usage (by disallowing weakening) or bounded usage (by allowing weakening) or both. Towards this end, one can design a type system for analyzing exact usage and another one for analyzing bounded usage. While such an approach works, it is not very economical because it does not take advantage of the commonalities between the two analyses. A better approach would be to design a single type system that can take care of both the analyses. This is where the idea of *parametrization* comes in handy.

Parametrization enables a type system to handle multiple analyses that have differences among them but nevertheless, share a common structure. That common structure is usually formalized in terms of an abstract algebraic structure like semiring, lattice, etc. A type system, parametrized by an abstract algebraic structure, can potentially carry out different analyses upon different concrete instantiations of that abstract algebraic structure. Parametrization of the type system by an abstract algebraic structure marks a key step in the

evolution of linear type systems. All the early linear type systems inspired by BLL [Brunel et al., 2014, Ghica and Smith, 2014, Petricek et al., 2014] employ parametrization to their benefit. We shall see some of these benefits as we move ahead. In part, the benefits ensue from the very nature of the abstract algebraic structure used for parametrizing these type systems. So next, we shall motivate this algebraic structure.

1.1.6 An Abstract Structure to Model Resource Usage

BLL can employ natural numbers to model resource usage. However, as we pointed out earlier, BLL, more generally, employs polynomials with natural number coefficients for this purpose. So we may ask: is there an even more general structure that might be used for the same purpose? A closer look at the rules that account resource usage in BLL gives us a hint of such a structure. A structure modeling resource usage in BLL needs to support addition (see rule *BContract*) and multiplication (see rule *BPromote*) the way we understand these operations. The structure also needs to have an identity element for multiplication (see rule *BDerelict*). Now going through abstract algebra, we find that a semiring [Golan, 1999] closely fits these requirements: a semiring supports addition and multiplication and also has identity elements for both these operations.

Formally, a *semiring* $\mathcal{Q}$ is a set Q with two binary operators, $+$ and $\cdot$, called addition and multiplication respectively, and two constants, 0 and 1, such that:

- addition and multiplication are associative, meaning, $q_1 + (q_2 + q_3) = (q_1 + q_2) + q_3$ and $q_1 \cdot (q_2 \cdot q_3) = (q_1 \cdot q_2) \cdot q_3$, for all $q_1, q_2, q_3 \in Q$;
- addition is commutative, meaning, $q_1 + q_2 = q_2 + q_1$, for all $q_1, q_2 \in Q$;
- 0 and 1 are identity elements for addition and multiplication respectively, meaning, $0 + q = q = q + 0$ and $1 \cdot q = q = q \cdot 1$, for all $q \in Q$;
- multiplication is left and right distributive over addition, meaning, $q_1 \cdot (q_2 + q_3) = q_1 \cdot q_2 + q_1 \cdot q_3$ and $(q_1 + q_2) \cdot q_3 = q_1 \cdot q_3 + q_2 \cdot q_3$, for all $q_1, q_2, q_3 \in Q$; and
- 0 is an annihilator for multiplication, meaning, $0 \cdot q = 0 = q \cdot 0$, for all $q \in Q$.

Note that multiplication in a semiring need not be commutative. The following are some examples of semirings: $\mathbb{N}$ and $\mathbb{N} \cup \{\omega\}$ and $\{0, \omega\}$, with addition and multiplication defined as usual; the set of polynomials in one variable, say X, with coefficients drawn from $\mathbb{N}$; the powerset of any set, with addition and multiplication defined as set intersection and set union respectively.

A semiring is a general algebraic structure to model resource usage. As such, it can be used as a parameter to a type system geared for analyzing resource usage. By varying the parameter, such a type system can analyze various notions of usage. Below, we elaborate the mechanism of analyzing resource usage through semirings. With an abstract semiring $\mathcal{Q}$ (symbol motivated by Quantitative applications) as the *parameter*,

a type system would employ elements of $\mathcal{Q}$ to grade the !-modality and then, employ this graded modality to carry out accounting of resources via the semiring operations. Intuitively, in such a type system, the graded modal type $!_q A$ would represent q copies of A, for any $q \in \mathcal{Q}$. Now, to get an idea of the accounting process, we consider the rules that specify the behavior of the graded !-modality:

$$\frac{!_{\overrightarrow{r}}\Gamma \vdash B}{!_{q\cdot\overrightarrow{r}}\Gamma \vdash !_q B}\ (QPromote) \qquad \frac{\Gamma, !_q A, !_r A \vdash B}{\Gamma, !_{q+r} A \vdash B}\ (QContract)$$

$$\frac{\Gamma, A \vdash B}{\Gamma, !_1 A \vdash B}\ (QDerelict) \qquad \frac{\Gamma \vdash B}{\Gamma, !_0 A \vdash B}\ (QWeaken)$$

Rules *QPromote* and *QContract* are similar to rules *BPromote* and *BContract* respectively, with x, y replaced by $q, r \in \mathcal{Q}$. Rule *QDerelict* is typographically the same as rule *BDerelict*; however, 1 in rule *QDerelict* is the multiplicative identity of $\mathcal{Q}$, and not necessarily $1 \in \mathbb{N}$. Next, rule $QWeaken$ is different from rule $BWeaken$ because rule $QWeaken$ does not allow unfettered weakening. Note here that 0 in rule $QWeaken$ is the additive identity of $\mathcal{Q}$.

Why is rule $QWeaken$ different from rule $BWeaken$? To understand why, we need to recall the motivation behind designing a parametrized type system. The key motivation is to have a single type system that can take care of multiple usage analyses, such as, analyses of exact usage and bounded usage. Now, if unfettered weakening is allowed, then the system can no longer analyze exact usage. This is why rule $QWeaken$ is used, in lieu of rule $BWeaken$, for the purpose of weakening. But then, one may point out, and rightly so, that the system, as described by the above four rules, is not suited for bounded usage analysis. This criticism, while true, can be addressed by introducing a notion of *preorder* in the parametrizing structure itself. More specifically, the parametrizing structure, when generalized from a semiring to a preordered semiring, can help the type system analyze both exact and bounded usage. Below, we describe how.

A *preordered semiring* is a semiring with a preorder, denoted $<:$, which respects the binary operations, meaning, if $q_1 <: q_2$, then $q + q_1 <: q + q_2$ and $q \cdot q_1 <: q \cdot q_2$ and $q_1 \cdot q <: q_2 \cdot q$, for all $q \in \mathcal{Q}$. The preorder formalizes our intuition regarding less and more resources. The intuitive meaning of $q <: r$ is that q contains more resources than r; so it is safe to replace the hypothesis $!_r A$ with $!_q A$ in any derivation. This intuition motivates the subusage rule shown below:

$$\frac{\Gamma, !_r A \vdash B \quad q <: r}{\Gamma, !_q A \vdash B}\ (QSubusage)$$

With this extra rule in place, the same type system can analyze both exact and bounded usage. While analyzing exact usage, the parameter is set to a semiring with discrete order, in which case the above rule becomes superfluous. While analyzing bounded usage, the parameter is set to a semiring with order: $q <: r \triangleq \exists q_0, q = q_0 + r$, in which case the above rule, together with rule $QWeaken$, allows unfettered weakening.

A preordered semiring is an excellent structure to model resource usage. A type system that is parametrized over a preordered semiring can not only analyze fine-grained notions of usage but also accommodate usage analyses with conflicting constraints, as illustrated above. To emphasize, type systems parametrized over

preordered semirings are very *versatile* with regard to usage analyses. We described above how they can analyze both exact and bounded usage. But note that they are not limited to these two analyses only; they are capable of many other analyses, some of which are discussed in Chapter 4.

Owing to their versatility, type systems parametrized over preordered semirings have found many applications. The earliest of such type systems, introduced around the same time by Brunel et al. [2014], Ghica and Smith [2014] and Petricek et al. [2014], have a diverse range of applications: timing analysis, liveness analysis, complexity analysis, etc. One particular application that interests us and motivated this discussion in the first place is that these type systems show a way to resolve the challenge that obstructs the design of linear dependent type systems, discussed in Section 1.1.4. To understand how these type systems help resolve the challenge, we first need to acquaint ourselves with the novelties they bring to the table. Hereafter, we refer to these type systems as graded-context linear simple type systems, for reasons that will soon become apparent.

1.1.7 Graded-Context Linear Simple Type Systems

The early linear simple type systems inspired by BLL, i.e., Brunel et al. [2014], Ghica and Smith [2014] and Petricek et al. [2014], are all parametrized over preordered semirings or similar algebraic structures. These systems, notwithstanding their technical differences, employ the same gadgets for usage analysis. These gadgets make the systems very flexible and additionally, show a way to resolve the challenge in designing linear dependent type systems. We introduce these gadgets below.

1. **Graded Contexts:** These systems employ graded contexts, meaning, they use the elements of the parametrizing structure to grade assumptions in contexts of typing judgments. Concretely, a graded context looks like $x_1 :^{q_1} A_1, x_2 :^{q_2} A_2, \dots, x_n :^{q_n} A_n$, where grade q_i $(i = 1, 2, \dots, n)$ is an element of the parametrizing structure. Note that a graded context, Γ, can be decomposed into a ungraded context, $\lfloor \Gamma \rfloor$, and a vector of grades, $\overline{\Gamma}$. The intuitive meaning of the graded context $x_1 :^{q_1} A_1, x_2 :^{q_2} A_2, \dots, x_n :^{q_n} A_n$ is the same as that of the standard context $x_1 : !_{q_1} A_1, x_2 : !_{q_2} A_2, \dots, x_n : !_{q_n} A_n$. However, there is a very good reason behind employing graded contexts in place of standard ones, as we shall see shortly.

   Graded contexts may be seen as generalization of split contexts, we saw in Section 1.1.3. While split contexts allow only two zones (nonlinear and linear), graded contexts allow as many zones as the number of elements of the parametrizing structure. For example, over $\mathbb{N}$ with discrete order, the graded context $x_1 :^1 A_1, x_2 :^2 A_2, x_3 :^3 A_3$ may be seen as a context with $\mathbb{N}$ zones, where the zones corresponding to $1, 2$ and 3 contain the assumptions $x_1 : A_1, x_2 : A_2$ and $x_3 : A_3$ respectively, while all the other zones are empty.

2. **Graded Types:** Ghica and Smith [2014] and Petricek et al. [2014] employ graded types in their systems, for example, types of the form ${}^q A \to B$, where q is a grade. The intuitive meanings of graded types ${}^q A \to B$ and ${}^q A \otimes B$ are the same as those of types $(!_q A) \to B$ and $(!_q A) \otimes B$ respectively.

Notwithstanding this intuitive equivalence, graded types are, in fact, more flexible compared to their graded modal counterparts. Note that types that include grade annotations via modalities only are referred to as graded modal types, and not as graded types. Now, the flexibility of graded types over graded modal types is not quite apparent in a simply-typed setting; we need to move to a dependently-typed setting to appreciate it. Consider the following dependent types: $\Pi x :!_q A.B$ and $\Pi x :^q A.B$. Both of them are types of functions that use their arguments q times. However, given the dependent setting, the codomain type B can also use the argument x. And here in, lies the difference between the two! Compared to the graded type $\Pi x :^q A.B$, the type $\Pi x :!_q A.B$ imposes more restrictions as to how x may be used in B. We shall discuss the technical reason behind such restrictions in Section 4.6.2.

3. **Context Operations:** These systems employ context operations for accounting usage, i.e., their typing rules add contexts and multiply context by grades. Context operations are defined by lifting the corresponding operations of the parametrizing structure to the level of contexts. For example, for contexts Γ_1 and Γ_2, graded over a preordered semiring $\mathcal{Q}$, and given $\lfloor\Gamma_1\rfloor = \lfloor\Gamma_2\rfloor$, the graded context $\Gamma_1 + \Gamma_2$ is defined by pointwise addition of grades; and for any $q \in \mathcal{Q}$, the graded context $q \cdot \Gamma_1$ is defined by pre-multiplying every grade in Γ_1 by q.

   To get an idea of how context operations are employed in typing rules, we look at sample abstraction and application rules:

$$\textsc{BLL-Abs}\ \frac{\Gamma, x :^q A \vdash b : B}{\Gamma \vdash \lambda^q x : A.b : {}^q A \to B} \qquad \textsc{BLL-App}\ \frac{\Gamma_1 \vdash b : {}^q A \to B \qquad \Gamma_2 \vdash a : A \qquad \lfloor\Gamma_1\rfloor = \lfloor\Gamma_2\rfloor}{\Gamma_1 + q \cdot \Gamma_2 \vdash b\ a^q : B}$$

   Note that the above rules employ graded contexts and graded types. ${}^q A \to B$ is the type of functions that use their argument q times. As such, in rule BLL-App, to type-check the application $b\ a^q$, the context of the argument a, i.e. Γ_2, is multiplied by q and then added to the context of the function b, i.e. Γ_1. We shall discuss these and other typing rules in detail in Section 4.2.2.

Graded contexts, graded types and context operations make the type systems inspired by BLL [Brunel et al., 2014, Ghica and Smith, 2014, Petricek et al., 2014] more flexible than the split-context type systems [Barber, 1996, Benton, 1994]. These graded-context type systems, unlike the split-context ones, are not limited to linearity analysis. As pointed out earlier, by varying the parametrizing structure, these systems can carry out other analyses too. Moreover, even with regard to linearity analysis, graded-context type systems are better than split-context type systems. Below, we explain how and why. First, note that for linearity analysis, graded-context type systems instantiate the parametrizing structure to the *linearity preordered semiring*, $\mathcal{Q}_{\text{Lin}}$, defined as: $\{0, 1, \omega\}$ with $1 + 1 = \omega + 1 = 1 + \omega = \omega + \omega = \omega$ and $\omega \cdot \omega = \omega$ and $\omega <: 0$ and $\omega <: 1$. Here, 0, 1 and ω indicate no usage, linear usage and unrestricted usage respectively. Note that in $\mathcal{Q}_{\text{Lin}}$, $\neg(1 <: 0)$ because otherwise 1 would indicate affine (at most once) usage and not linear (exactly once) usage. Next, we describe how a graded-context type system, instantiated to $\mathcal{Q}_{\text{Lin}}$, is better than a split-context type system:

1. **Analyzes No Usage:** The split-context type systems can analyze linear and nonlinear usages only. Graded-context type systems, in addition to analyzing linear and nonlinear usages through grades 1 and ω respectively, can also analyze no usage through grade 0. This makes graded-context type systems more expressive: through types ${}^0A \to B$, they can represent functions that do not make use of their arguments; through types ${}^0A \otimes B$, they can represent pairs whose first components are unusable. Note here that the types ${}^0A \to B$ and ${}^0A \otimes B$ are different from the types $!A \to B$ and $!A \otimes B$, which are expressible in both split-context and graded-context type systems. Analysis of no usage is particularly important in dependent type systems. We shall discuss more about this analysis in Chapter 4.

2. **No Extra Checks:** The split-context type systems, due to their rigid division of the context into two zones, need to be cautious with the promotion rule. Recall the promotion rule of DILL from Section 1.1.3 and compare it with the promotion rule of a graded-context type system:

$$\text{DILL-ExpIntro}\ \frac{\Delta\ ;\ \varnothing \vdash a : A}{\Delta\ ;\ \varnothing \vdash !a : !A} \qquad\qquad \text{BLL-ExpIntroL}\ \frac{\Gamma \vdash a : A}{\omega \cdot \Gamma \vdash !_\omega a : !_\omega A}$$

Rule DILL-ExpIntro needs to ensure that the linear zone of the context in the premise judgment is empty. But rule BLL-ExpIntroL requires no such restriction. In fact, the context in the premise judgment of rule BLL-ExpIntroL may contain both linear and nonlinear assumptions (i.e., assumptions with grades $q = 1$ and $q \in \{0, \omega\}$ respectively). Multiplication of the context by ω in this rule ensures that every contextual assumption in the conclusion judgment is nonlinear.

3. **Less Restrictive:** Split-context type systems, even though flexible, impose certain restrictions. In split-context systems, an assumption that appears in the linear zone of the context of a premise judgment cannot appear in the context of any other premise judgment of the same typing rule. But in graded-context type systems, an assumption can appear at different grades in different premise judgments of the same typing rule. A comparison of the application rule of the two systems makes this point clear:

$$\text{DILL-App}\ \frac{\Delta\ ;\ \Gamma_1 \vdash b : A \multimap B \qquad \Delta\ ;\ \Gamma_2 \vdash a : A}{\Delta\ ;\ \Gamma_1, \Gamma_2 \vdash b\ a : B} \qquad\qquad \text{BLL-AppL}\ \frac{\Gamma'_1 \vdash b : A \multimap B \qquad \Gamma'_2 \vdash a : A \qquad \lfloor \Gamma'_1 \rfloor = \lfloor \Gamma'_2 \rfloor}{\Gamma'_1 + \Gamma'_2 \vdash b\ a : B}$$

In rule DILL-App, Δ, Γ_1 and Γ_2 are mutually disjoint but in rule BLL-AppL, Γ'_1 and Γ'_2 have the same underlying set of assumptions, possibly held at different grades. So, an assumption $z : C$ can appear as $z :^1 C$ and $z :^0 C$ in Γ'_1 and Γ'_2 respectively but an assumption $z : C$ cannot simultaneously appear in Γ_1 and Δ. Again, an assumption $z : C$ can appear as $z :^1 C$ and $z :^1 C$ in Γ'_1 and Γ'_2 respectively but an assumption $z : C$ cannot simultaneously appear in Γ_1 and Γ_2.

Observe how graded contexts and context operations ensure proper accounting even when: an assumption is simultaneously used as a linear resource and as a nonlinear resource in the premise judgments of a typing rule; or an assumption is used as a linear resource in multiple premise judgments of a typing

rule. This flexibility is not offered by split-context type systems, where contextual assumptions are rigidly attached to zones whose boundaries they cannot transgress. In other words, in split-context type systems, a contextual assumption and the zone it belongs to are coupled together, which entangles typing and resource accounting. A graded-context type system decouples a contextual assumption and its usage constraint, thereby enabling the same assumption to appear at different grades in different premise judgments of a typing rule. This decoupling makes typing and resource accounting fairly orthogonal to one another. Owing to this orthogonality, accounting is smoother in graded-context type systems than in split-context ones.

The extra flexibility offered by graded-context type systems has been exploited in many applications, as we pointed out before. Next, we shall see how this flexibility helps resolve the challenge facing the design of linear dependent type systems.

1.1.8 Linear Dependent Type Systems: Challenge Resolved

The extra flexibility offered by graded-context type systems, mentioned in the last subsection, is not very significant with regard to linearity analysis in a simply-typed setting. However, this extra flexibility becomes significant in a dependently-typed setting. To see why, compare rule DILL-DApp (from Section 1.1.4) with rule BLL-DAppL (extension of rule BLL-AppL to dependent types):

$$\text{DILL-DApp}\quad \frac{\Delta\ ;\ \Gamma_1 \vdash b : (x : A) \multimap B \qquad \Delta\ ;\ \Gamma_2 \vdash a : A}{\Delta\ ;\ \Gamma_1, \Gamma_2 \vdash b\ a : B\{a/x\}}$$

$$\text{BLL-DAppL}\quad \frac{\Gamma_1' \vdash b : (x : A) \multimap B \qquad \Gamma_2' \vdash a : A \qquad \lfloor \Gamma_1' \rfloor = \lfloor \Gamma_2' \rfloor}{\Gamma_1' + \Gamma_2' \vdash b\ a : B\{a/x\}}$$

As discussed in Section 1.1.4, in rule DILL-DApp, the type A cannot depend upon Γ_1 or Γ_2. But in rule BLL-DAppL, A can depend upon Γ_1' (or Γ_2') because Γ_1' and Γ_2' contain the same underlying set of assumptions. This flexibility makes it possible for types to depend upon linear assumptions. For instance, in rule BLL-DAppL, A can depend upon a linear assumption in Γ_2', say $z :^1 C$, because $z : C$ also appears in Γ_1', possibly at a different grade. However, in rule DILL-DApp, A cannot depend upon any assumption in Γ_2 (or in Γ_1) because Δ, Γ_1 and Γ_2 are mutually disjoint. Observe how decoupling assumptions from their usage constraints drives this difference between the two rules. Thanks to this decoupling, rule BLL-DAppL does not need to impose the restrictions on types that rule DILL-DApp does.

In this way, graded-context type systems show how to resolve the challenge facing the design of linear dependent type systems, a way out of the restriction put on types by the systems of Krishnaswami et al. [2015] and Vákár [2015]. Recall that Krishnaswami et al. [2015] and Vákár [2015] extend the split-context simple type systems of Benton [1994] and Barber [1996] and as such, are forced to restrict types to not depend upon linear assumptions. From the above discussion, we see that unlike the split-context simple type

systems, the graded-context simple type systems [Brunel et al., 2014, Ghica and Smith, 2014, Petricek et al., 2014] do not force such a restriction. This observation was first made by McBride [2016].

McBride [2016] extends the graded-context simple type system of Petricek et al. [2014] to dependent types. He does away with the restriction that types can only depend upon nonlinear assumptions. In his system, types can depend upon both linear and nonlinear assumptions. One consequence of lifting this restriction is that his system allows types of the form $(x : A) \multimap B$, where B depends upon the linear assumption, $x : A$. Recall that the split-context dependent type systems do not allow such types. Undoubtedly, McBride's system takes a step in the right direction in combining linearity with dependent types. But allowing types of the form $(x : A) \multimap B$, where B depends upon $x : A$, raises a complex question.

In the split-context dependent type systems, all linear functions have non-dependent types of the form $A \multimap B$. However, in McBride's system, linear functions can have types of the form $(x : A) \multimap B$, where B depends upon $x : A$. But then, given $f : (x : A) \multimap B$, the function f uses the argument $x : A$ exactly once while the type B depends upon, and therefore also uses, the same argument. If both a type and a term use a linear assumption, then how is that linear assumption being used only once? Observe that this question arises because the restriction on types enforced in the split-context dependent type systems is lifted in McBride's system. This question is, in fact, an instance of a more general question: If types can also use resources, then how should one account for resource usage in terms and in types? This question is complex because it engenders a whole set of questions on resource usage:

Q.1 Should usage in types be treated the same way as usage in terms?

Q.2 Should usage of an assumption in a type and a term having that type be related?

Q.3 Should usage signify the usage in term or the usage in type or some combination of both?

Thanks to the orthogonality of typing and accounting in graded-context type systems, these questions can be answered in multiple ways. The multiple answers to these questions have led to the development of multiple linear dependent type systems [Atkey, 2018, Choudhury et al., 2021, Moon et al., 2021]. Our answers to these questions form the basis of our linear dependent type system, GRAD [Choudhury et al., 2021]. To appreciate the answers we give, one needs to know the answers that others gave. So next, we look at the various known answers to the above questions.

1.1.9 Accounting Usage in Terms and Types: McBride and Atkey

McBride [2016] realized that the graded-context simple type systems can be extended to dependent types without requiring to impose the restriction that types depend upon nonlinear assumptions only. Then, when confronted with the question whether usage in types should be treated the same way as usage in terms (vide Q.1), McBride [2016] answered: no, because usage in types is 'contemplative' whereas usage in terms is 'consumptive'. In other words, according to McBride, usage in types does not count as real usage, only

usage in terms does. For example, given type $(x : A) \multimap B$, the linear assumption $x : A$ may be used any number of times in the body B (because such usage is 'contemplative') but exactly once in functions having this type (because such usage is 'consumptive'). So then, in McBride's system, usage signifies usage in terms (vide Q.3) and this usage, which is consumptive, is not related to usage in types, which is contemplative (vide Q.2). Having thus answered the questions on usage in terms and in types, McBride [2016] designed the first-of-its-kind linear dependent type system.

McBride's insights are brilliant. But unfortunately, his system does not admit substitution [Atkey, 2018]. In a way, his system is too general to admit substitution. Atkey [2018] restricted his system in several ways and proposed a modified system, QTT. QTT adopts McBride's doctrine of contemplative use versus consumptive use and answers the questions Q.1-Q.3 the same way as McBride [2016]. However, owing to the restrictions imposed, QTT admits substitution. Atkey [2018] shows soundness of QTT via categorical models.

Since McBride's system does not admit substitution, QTT may be regarded as the first linear dependent type system that allows types to depend upon linear assumptions. This way QTT fulfilled the long-cherished dream of truly combining linearity with dependent types. Undoubtedly, QTT is a milestone in the area of linear dependent type systems. But nevertheless, QTT has some issues:

1. **Non-uniformity:** Types and terms are treated very differently in QTT: types live in a resource-agnostic world while terms live in a resource-aware world. This dichotomy is reflected in the distinct forms of typing judgments, $\Gamma \vdash A :^0 A'$ and $\Gamma \vdash a :^1 A$, which QTT employs to type-check types and terms respectively. To give a concrete example, in QTT, the type $(y : B) \multimap C$ and the term $\lambda y : B.c$ are type-checked via judgments $\Gamma \vdash A :^0 A'$ and $\Gamma \vdash a :^1 A$ respectively. This non-uniform treatment of types and terms makes the calculus more complex than is necessary.

2. **Restrictions on Parametrizing Structure:** QTT needs to impose restrictions on the parametrizing structure to admit substitution. QTT is parametrized over semirings (without order), $\mathcal{Q}$, that satisfy two special properties: $q_1 + q_2 = 0 \Rightarrow q_1 = q_2 = 0$, and $q_1 \cdot q_2 = 0 \Rightarrow q_1 = 0 \vee q_2 = 0$, for all $q_1, q_2 \in \mathcal{Q}$. These restrictions on the parametrizing structure are not limiting with regard to linearity analysis but they do limit other analyses, such as, analysis of information flow.

1.1.10 Accounting Usage in Terms and Types: Our GraD

To address the above-mentioned issues of QTT, we designed an alternative linear dependent type system, GRAD [Choudhury et al., 2021]. GRAD is a Pure Type System that is parametrized over an arbitrary preordered semiring. (Note that GRAD, as presented in this dissertation, is a generalization of GRAD, as presented in Choudhury et al. [2021], which is a Type-in-Type calculus parametrized over partially ordered semirings.) GRAD can be used for linearity and several other analyses (see Chapter 4). The key distinction between GRAD and QTT is that in GRAD, both types and terms live in a resource-aware world. As such, GRAD employs a single form of typing judgment, $\Gamma \vdash a : A$, to type-check both types and terms. This

uniform treatment of types and terms makes GRAD a simpler system, compared to QTT. Further, unlike QTT, GRAD does not need to impose any restrictions on the parametrizing structure to admit substitution.

Compared to QTT, GRAD may be seen as a simpler extension of the graded-context simple type systems [Brunel et al., 2014, Ghica and Smith, 2014, Petricek et al., 2014] to dependent types. Both GRAD and the graded-context simple type systems employ only one form of typing judgment, $\Gamma \vdash a : A$. Neither GRAD nor the graded-context simple type systems need to restrict the parametrizing structure to admit substitution. The accounting mechanism of GRAD is also similar to that of the graded-context simple type systems, as far as usage in terms is concerned.

Now, when it comes to accounting usage in types, GRAD addresses the questions Q.1-Q.3 as follows:

1. In McBride's system and QTT, usage in a type does not ever count. In GRAD, usage in a type does not count whenever it appears as a type in a typing judgment but usage in a type does count whenever it appears as a term in a typing judgment. For example: usage in A does not count in the judgment $\Gamma \vdash a : A$, but usage in A does count in the judgment $\Gamma' \vdash A : B$. More precisely, in GRAD, given any typing judgment $\Gamma \vdash a : A$, the graded-context Γ denotes the resources used by a, irrespective of whether a is a term or a type. This is how GRAD answers Q.1.

2. In McBride's system and QTT, since usage in types does not count, the question whether usage in types is related to usage in terms (vide Q.2) does not make sense. But this question makes perfect sense in case of GRAD. Given the accounting principle followed by GRAD, I believe that both an affirmative and a negative answer to this question would work out. However, for the sake of simplicity and expressiveness (see Section 4.6.2), GRAD chooses the latter, i.e., GRAD *chooses not* to relate usage in types with that in terms. To illustrate, in GRAD, the assumption $x : A$ may be used any number of times in the body B of type $(x : A) \multimap B$, even though terms having this type may use this assumption once only.

3. In McBride's system and QTT, since usage in types does not count, the question what usage signifies (vide Q.3) has a straightforward answer. But given the accounting principle followed by GRAD, this question can be answered in more than one way. For example, one can imagine types of the form $\Pi x :^{q,r} A.B$, which require terms to use q copies of $x : A$ while the body B uses r copies of $x : A$. With such types around, one can record usage of assumptions both in types and in terms. However, for the sake of simplicity, GRAD *chooses* to restrict itself to *recording* usage in terms only. Put differently, though GRAD accounts for usage in types, it does not record those usages. So, Π-types in GRAD are of the form $\Pi x :^{q} A.B$ (and not $\Pi x :^{q,r} A.B$), where q signifies usage of the assumption $x : A$ in terms having this type.

The design choices described above make GRAD an elegant calculus: it has a simple, uniform and minimalist presentation.

1.1.11 Accounting Usage in Terms and Types: Moon et al.

Soon after the introduction of GRAD [Choudhury et al., 2021], another linear dependent type system, based on a different accounting principle, was proposed by Moon et al. [2021]. Moon et al. [2021] proposed a system, GRTT, where:

1. Usages in types and term are simultaneously accounted in any typing judgment. To provide an example, given a standard typing judgment, $x_1 : A_1, x_2 : A_2, x_3 : A_3 \vdash b : B$, GRTT accounts: the usage of x_1 in A_2, A_3, b and B; the usage of x_2 in A_3, b and B; and the usage of x_3 in b and B. In contrast, given the typing judgment, $x_1 : A_1, x_2 : A_2, x_3 : A_3 \vdash b : B$, GRAD just accounts the usage of x_1, x_2 and x_3 in b. Note the difference in the treatment of usage in types (vide Q.1) in GRTT and in GRAD.

2. Usages of assumptions in both types and terms are recorded. For example, GRTT has types of the form $\Pi x :^{q,r} A.B$ where q and r (r not related to q) denote usages of the assumption $x : A$ in the body of functions (having this type) and the codomain B respectively. On the other hand, Π-types in GRAD are of the form $\Pi x :^{q} A.B$, where q denotes usage of the assumption $x : A$ in the body of functions having this type. Note that GRTT is similar to GRAD in that the usage of an assumption in the body of a type and a term having that type are not related (vide Q.2). However, GRTT is also different from GRAD in that the usages of an assumption in the body of a type and a term having that type are both recorded in GRTT (vide Q.3).

As we see above, compared to GRAD, GRTT employs elaborate usage accounting and recording mechanisms in the language. These elaborate mechanisms, though helpful in several analyses, makes GRTT a complex calculus: the typing rules of GRTT are particularly intricate. In this regard, GRAD is a much simpler calculus that combines linearity with dependent types while following a minimalist design principle.

1.1.12 Summary

By now, we have seen three sets of answers to questions Q.1-Q.3, given by QTT, GRAD and GRTT respectively. All these sets of answers are interesting and provide unique perspectives on the problem of usage accounting in dependent type systems. However, they do not cover the whole space of possible answers to these questions. As we indicated in our discussion in Section 1.1.10, there are some alternative answers, with respect to GRAD, that we did not explore. This richness of alternative answers goes on to show how flexible graded-context type systems are, especially in a dependent setting. These type systems do not force a rigid accounting regimen but give room to multiple accounting principles that may even be in conflict with one another. Thanks to graded-context type systems, we already have three different calculi for combining linearity with dependent types.

Before we wind up this section, let's recap the key ideas we have come across here. In this section, we went over the evolution of linear type systems. We saw three distinct stages in this evolution, marked by single-context type systems, split-context type systems and graded-context type systems, in that order. The

key trend in this evolution is a move towards flexibility. Graded-context type systems are more flexible than split-context type systems, which are in turn more flexible than single-context type systems. This move towards flexibility resulted in better type systems: type systems that are versatile, more expressive and less restrictive. In this move, grading has played a driving role. Grading, in its various manifestations, like graded contexts, graded modalities and graded types, enabled smooth accounting of resource usage in simple type systems by decoupling typing and accounting to a fair extent. This decoupling facilitated smooth integration of linearity and dependent types. Furthermore, it provided the leeway to implement multiple strategies for accounting usage in dependent type systems. The three strategies we came across, embodied by the calculi QTT [Atkey, 2018], GraD [Choudhury et al., 2021] and Grtt [Moon et al., 2021], represent the state of the art in accounting usage in dependent type systems.

We started this section with the goal of contextualizing our work on linearity analysis. With this goal in mind, we traversed the literature on linear type systems, starting with simple types and then moving on to dependent types. We deliberated upon the key ideas and insights that shaped the development of linear type systems to their current form. This background provides the necessary context to our work on linearity analysis. In particular, it positions our calculus, GraD, in the field of linear type systems. This section also presents an outline of our calculus itself. The full calculus, with all its details, is presented in Chapter 4.

Next, we move on to contextualizing our work on dependency analysis.

1.2 Contextualizing Our Work on Dependency Analysis

1.2.1 Dependency Analysis in Programming Languages

Dependency analysis is the analysis of dependence, or lack thereof, of an entity upon another. The entities are primarily programs or parts of programs, but they can also be abstract, like security clearance levels in an organization, stages in a compilation process, etc. Broadly speaking, an entity depends upon another one if the latter influences the behavior of the former. On the other hand, an entity is independent of another one if the latter does not interfere in the behavior of the former. For example, consider the following λ-terms: $(\lambda x.x)\,(2+2)$ and $(\lambda x.5)\,(2+2)$. The argument $(2+2)$ dictates the normal form of the first term whereas it plays no role in deciding the normal form of the second term. So, we say that the first term depends upon the argument whereas the second one does not. What this means is that in the second term, we can replace the argument $(2+2)$ with any other terminating computation, while maintaining the same normal form for the term as a whole.

Dependency analysis is useful and powerful precisely because it provides this guarantee: if an entity does not depend upon another one, then variations in the latter does not affect the former. This is the well-known principle of *noninterference*. First proposed in the realm of computer security [Goguen and Meseguer, 1982, Jones and Lipton, 1975], this principle has far-reaching implications in a variety of applications:

1. **Security Applications:** Via this principle, dependency analysis, in the form of information flow analysis [Heintze and Riecke, 1998, Smith and Volpano, 1998], ensures absence of information leaks from high-security data to low-security variables. Phrased abstractly in terms of security levels, information flow analysis ensures that level **Low**, denoting low-security computations, *does not depend* upon level **High**, denoting high-security computations.

2. **Metaprogramming Applications:** Via this principle, dependency analysis, in the form of binding-time analysis [Davies, 2017, Hatcliff and Danvy, 1997, Jones et al., 1993], enables sound partial evaluation of programs, i.e., programs can be evaluated in multiple stages even when each stage can potentially optimize based on inputs received from earlier stages. Phrased abstractly in terms of stages of evaluation, binding-time analysis ensures that stage **Early**, denoting early-stage computations, *does not depend* upon stage **Later**, denoting later-stage computations.

3. **Optimization Applications:** Via this principle, dependency analysis, in the form of irrelevance analysis [Mishra-Linger and Sheard, 2008, Pfenning, 2001], facilitates proper erasure of parts of programs that are irrelevant to run-time computation. Phrased abstractly in terms of modes of use, irrelevance analysis ensures that mode **Relevant**, denoting computations relevant at run time, *does not depend* upon mode **Irrelevant**, denoting computations irrelevant at run time.

In short, it is the principle of noninterference that makes dependency analysis indispensable to the above and several other applications [Abadi et al., 1996, Palsberg and Ørbæk, 1995, Tang and Jouvelot, 1995, Tip, 1995, Tofte and Talpin, 1997, etc.] in computer science.

Dependency analysis is an important problem and also a challenging one. Designing a technique that statically decides whether a program depends upon an argument requires due deliberation. Below, we elaborate why.

1. At first, one might propose the following straightforward technique for dependency analysis: a program does not depend upon an argument if the variable bound to that argument does not appear in the body of the program. We can apply this technique on the examples presented above: $(\lambda x.5)\ (2+2)$ does not depend upon $(2+2)$ because x does not appear in the body of the term whereas $(\lambda x.x)\ (2+2)$ depends upon $(2+2)$ because x appears in the body of the term. While this technique works for these examples and is also correct in general, nevertheless, it is very limited in scope. To understand why, consider the λ-term: $(\lambda y.5)\ ((\lambda x.x)\ (2+2))$. This term, too, does not depend upon the argument $(2+2)$. However, the proposed technique would fail to identify this non-dependence because x appears in the body of the program. Thus, a straightforward presence/absence analysis of variables does not provide a satisfactory solution to the problem of dependency analysis.

2. The problem of dependency analysis is also not as innocuous as it seems at first sight; it can indeed be quite subtle. Consider the λ-term: $(\lambda x.\mathbf{if}\ x\ \mathbf{then}\ 2\ \mathbf{else}\ 5)\ \mathbf{true}$. Clearly, this term depends upon the argument, **true**. But observe that both the branches of the conditional just return constants. This example shows that even if a program does not explicitly make use of an argument in its return values, it may still implicitly depend upon that argument [Denning, 1976].

Thus, dependency analysis requires a principled and systematic approach. A critic might argue that one could, for example, given a program and an argument, invariably employ some sort of *local analysis* to determine whether the program depends upon the argument. This criticism can be countered by pointing out that such an analysis would be tedious because one would need to carry it out for every program and argument pair. So instead of a local analysis, we would prefer a *global analysis*, whereby the language itself is endowed with an abstract notion of dependency. This way the very type system of the language can inform us whether a program depends upon an argument. This is what motivates dependency type systems. In this section, we shall introduce multiple dependency type systems. But before that, we need to understand the abstract structure of dependency, which forms the foundation of these type systems.

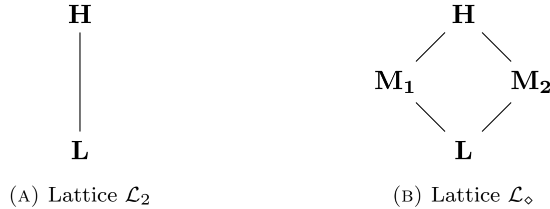
1.2.2 An Abstract Structure to Model Dependency

The three example applications of dependency analysis discussed above give us an idea about the abstract structure of dependency. All the three examples have two dependency levels, call them ℓ_1 and ℓ_2, with the constraint that ℓ_2 may depend upon ℓ_1 but not vice-versa. In case of information flow analysis, $\ell_1 = \textbf{Low}$ and $\ell_2 = \textbf{High}$; in case of binding-time analysis, $\ell_1 = \textbf{Early}$ and $\ell_2 = \textbf{Later}$; in case of irrelevance analysis, $\ell_1 = \textbf{Relevant}$ and $\ell_2 = \textbf{Irrelevant}$. Though all the above examples consider two dependency levels only, we can imagine the analyses being extended to an arbitrary number of dependency levels, with constraints among them. These constraints induce an ordering on the dependency levels, with $\ell_1 \sqsubseteq \ell_2$ if and only if ℓ_2 may depend upon ℓ_1.

The ordering relation, $\sqsubseteq$, on dependency levels is generally reflexive and transitive. The relation is reflexive because any level ℓ may depend upon itself. The relation is transitive because if ℓ_3 may depend upon ℓ_2, which in turn may depend upon ℓ_1, then ℓ_3 may indeed depend upon ℓ_1. Thus, the relation $\sqsubseteq$ is a preorder. Now, if ℓ_1 and ℓ_2 have a mutual dependence, then it makes sense to merge the two levels into a single one. So, without loss of generality, we can assume that $\sqsubseteq$ is a partial order. Now, given any partial ordering of dependency levels, we can convert it into a lattice by adding the missing 'joins' and 'meets'. This gives us a lattice model of dependency levels.

A *lattice* $\mathcal{L}$ [Birkhoff, 1967] is a partially-ordered set, L, where any pair of elements, $\ell_1, \ell_2 \in L$, has a least upper bound (also called join), denoted $\ell_1 \sqcup \ell_2$, and a greatest lower bound (also called meet), denoted $\ell_1 \sqcap \ell_2$. A lattice may also be *equivalently* defined as a set L with two binary operators, $\sqcup$ and $\sqcap$, called join and meet respectively, such that:

- join and meet are commutative, meaning, $\ell_1 \sqcup \ell_2 = \ell_2 \sqcup \ell_1$ and $\ell_1 \sqcap \ell_2 = \ell_2 \sqcap \ell_1$, for all $\ell_1, \ell_2 \in L$;
- join and meet are associative, meaning, $\ell_1 \sqcup (\ell_2 \sqcup \ell_3) = (\ell_1 \sqcup \ell_2) \sqcup \ell_3$ and $\ell_1 \sqcap (\ell_2 \sqcap \ell_3) = (\ell_1 \sqcap \ell_2) \sqcap \ell_3$, for all $\ell_1, \ell_2, \ell_3 \in L$;
- join and meet satisfy the absorption laws, meaning, $\ell \sqcup (\ell \sqcap \ell_0) = \ell = \ell \sqcap (\ell \sqcup \ell_0)$, for all $\ell, \ell_0 \in L$.



(A) Lattice $\mathcal{L}_2$ (B) Lattice $\mathcal{L}_\diamond$

Figure 1.1: Examples of lattices for modeling dependencies

With regard to this algebraic definition of lattice, the order relation may be defined as: $\ell_1 \sqsubseteq \ell_2 \triangleq \ell_1 \sqcup \ell_2 = \ell_2$. In this dissertation, we shall mostly use the algebraic definition of lattice and the derived order relation. Note that a lattice $\mathcal{L}$ is said to be bounded if there exists elements, $\bot, \top \in \mathcal{L}$, called bottom and top respectively, such that $\bot \sqsubseteq \ell$ and $\ell \sqsubseteq \top$, for all $\ell \in \mathcal{L}$. For the purpose of this dissertation, it is harmless to assume that all lattices are bounded because lattices that are not bounded can be turned into bounded lattices by simply adding missing top and bottom elements.

The lattice model of dependency levels was first proposed by Denning [1976], in her work on information flow analysis. The lattice model forms the cornerstone of dependency analysis. To better understand the model, we look at some examples:

- Information classification systems of government organizations can be represented as dependency lattices [Denning, 1976]. For example, the classification levels of the US government, viz. Unclassified, Confidential, Secret and Top Secret, denoted **U**, **C**, **S** and **TS** respectively, can be represented as: $\mathbf{U} \sqsubseteq \mathbf{C} \sqsubseteq \mathbf{S} \sqsubseteq \mathbf{TS}$. Any information flow that goes against the order of this lattice model would be considered a security breach.

  Sometimes, we may want to classify information into just two levels, low-security and high-security, denoted **L** and **H** respectively, with the constraint that high-security information is not leaked to individuals with just low-security clearance. In such cases, we may use a simplified lattice with just two elements, **L** and **H**, ordered as $\mathbf{L} \sqsubseteq \mathbf{H}$. We call this lattice $\mathcal{L}_2$. The Hasse diagram of $\mathcal{L}_2$ is shown in Figure 1.1(A). Observe that $\mathcal{L}_2$ is the simplest nontrivial lattice. This makes it an ideal candidate for examples involving lattices throughout this dissertation.

- Access privileges to private data can be represented as dependency lattices. For example, consider the medical and job data of an individual, which may be accessed only by the concerned doctor and supervisor respectively. To model these restrictions, we use a lattice with four elements: **L** denoting the general public, $\mathbf{M_1}$ and $\mathbf{M_2}$ denoting the doctor and supervisor, and **H** denoting the individual. The lattice is ordered as: $\mathbf{L} \sqsubseteq \mathbf{M_1} \sqsubseteq \mathbf{H}$ and $\mathbf{L} \sqsubseteq \mathbf{M_2} \sqsubseteq \mathbf{H}$. We denote this lattice as $\mathcal{L}_\diamond$, inspired by its Hasse diagram, which appears in Figure 1.1(B). Any information flow that goes against the order imposed by $\mathcal{L}_\diamond$ would be considered a privacy breach.

The key idea behind the lattice model of dependency is that for any pair of lattice elements ℓ_1 and ℓ_2, level ℓ_2 can depend upon level ℓ_1 if and only if $\ell_1 \sqsubseteq \ell_2$. This way the lattice model provides an abstract structure to reason about dependency constraints.

Now, one can take a particular set of dependency constraints, represented as a lattice, and design a dependency calculus that enforces these constraints. There are many calculi [Davies, 2017, Hatcliff and Danvy, 1997, Smith and Volpano, 1998, etc.] that analyze specific dependencies in this manner. However, our interest lies in a general calculus of dependency, one that can analyze a wide range of dependencies that appear in applications like security, metaprogramming, optimization, etc. Such a calculus would not be fine-tuned to a specific dependency lattice but would be able to reason about arbitrary dependency lattices. One such calculus and the first of its kind is the Dependency Core Calculus of Abadi et al. [1999], which is parametrized over an *abstract* dependency lattice.

1.2.3 Dependency Core Calculus

More than two decades ago, Abadi et al. [1999] showed that several dependency analyses [Abadi et al., 1996, Hatcliff and Danvy, 1997, Heintze and Riecke, 1998, Tang and Jouvelot, 1995, Volpano et al., 1996] can be seen as instances of a general Dependency Core Calculus, DCC. Over the years, DCC has served as a foundational framework for dependency analysis in the field of programming languages and has led to extensive research [Algehed, 2018, Algehed and Bernardy, 2019, Bowman and Ahmed, 2015, Shikuma and Igarashi, 2006, Tse and Zdancewic, 2004, etc.] on this topic.

DCC is a simply-typed calculus that is parametrized over an abstract dependency lattice. The calculus is a minor extension of Moggi's computational metalanguage [Moggi, 1991]. Recall that the computational metalanguage is a general calculus for analyzing computational effects like nontermination, exceptions, input/output, etc. What makes DCC special is that it integrates the lattice model of dependency with the computational metalanguage by grading the *monadic modality*, T, of the latter with elements of the former. This integration enables the type system of DCC to enforce dependency constraints modeled by an arbitrary lattice. For example: given lattice $\mathcal{L}_2$, where $\mathbf{L} \sqsubseteq \mathbf{H}$, DCC forces any function of type $T_{\mathbf{H}}\,\mathbf{Bool} \to T_{\mathbf{L}}\,\mathbf{Bool}$ to be constant. Here, $T_{\mathbf{H}}\,\mathbf{Bool}$ and $T_{\mathbf{L}}\,\mathbf{Bool}$ denote high-secure and low-secure booleans respectively. Note that a non-constant function of type $T_{\mathbf{H}}\,\mathbf{Bool} \to T_{\mathbf{L}}\,\mathbf{Bool}$ would leak information from $\mathbf{H}$ to $\mathbf{L}$, thereby breaching security.

Owing to its generality, over the years, DCC has become 'the calculus' for dependency analysis in simple type systems. However, in spite of its success, DCC has its limitations:

1. DCC, like the computational metalanguage, is a monadic calculus. But unlike the computational metalanguage, DCC has a nonstandard monadic bind rule, which relies upon an auxiliary protection judgment, which in turn has an unclear semantics.

2. Owing to the nonstandard bind rule, it is unclear how DCC might be extended to richer type systems, for example, dependent type systems.

3. Being of a monadic nature, DCC cannot capture dependency analyses that possess a comonadic nature, for example, the binding-time calculus, λ°, of Davies [2017]. Abadi et al. [1999] were aware of this limitation of DCC and pointed it out in their paper.

Our goal in this dissertation is to design a general dependency calculus for pure type systems. To the best of our knowledge, no such calculus has been proposed in literature. Given that DCC is a general dependency calculus for simple type systems, we thought that we could extend DCC to dependent types. However, with the above limitations of DCC, we did not find a way to do so. This, in turn, motivated us to take a fresh look at DCC and address its limitations in the first place.

1.2.4 Our Calculi for Dependency Analysis

We address the above limitations of DCC by designing an alternative dependency calculus, $\mathrm{GMCC_e}$, that is inspired by standard ideas from category theory. $\mathrm{GMCC_e}$ is both monadic and comonadic in nature and subsumes both monadic DCC and comonadic λ°. Our construction explains the nonstandard bind rule and the protection judgment of DCC in terms of standard categorical concepts. It also leads to a novel technique for proving correctness of dependency analysis. We use this technique to present alternative proofs of correctness for DCC and λ°. We shall present the details of $\mathrm{GMCC_e}$ in Chapter 2.

$\mathrm{GMCC_e}$ is a simply-typed dependency calculus. In Chapter 3, we extend $\mathrm{GMCC_e}$ to pure type systems. We design two calculi, $\mathrm{DDC^\top}$ and DDC, for dependency analysis in pure type systems. Both $\mathrm{DDC^\top}$ and DDC can analyze dependencies, over an arbitrary lattice, in pure type systems. However, DDC is more general than $\mathrm{DDC^\top}$ because not only does it analyze dependencies but also it internalizes dependency analysis in typing itself. In other words, if dependency analysis in DDC were incorrect, then typing itself would be unsound. In most dependency calculi, including $\mathrm{DDC^\top}$, correctness of dependency analysis does not affect typing. Internalizing dependency analysis in typing has several benefits that we point out in Chapter 3.

A dependency analysis that is particularly useful in dependent type systems is irrelevance analysis. Now, irrelevance analysis in dependent type systems is an extensively studied problem with a rich literature. Several calculi [Abel and Scherer, 2012, Barras and Bernardo, 2008, Miquel, 2001, Mishra-Linger, 2008, Pfenning, 2001, Tejiščák, 2020, etc.] address this problem, in its various forms. However, these calculi do not draw the connection between a general dependency analysis and irrelevance analysis (in dependent type systems). In this dissertation, we establish this connection by employing our general dependency calculi, $\mathrm{DDC^\top}$ and DDC, for analyzing irrelevance. We find that our calculi can analyze fine-grained notions of irrelevance and resolve some open problems in this area. Next, we look at the problem of irrelevance analysis in dependent type systems and review the corresponding literature with the aim of contextualizing our contributions in this area.

1.2.5 Irrelevance Analysis in Dependent Type Systems

Irrelevance analysis is very useful in dependent type systems, in the context of both run-time evaluation and type-checking.

First, we describe the utility of irrelevance analysis in the context of run-time evaluation. Significant parts of a dependently-typed program may be present only to satisfy the type-checker, and would therefore be irrelevant

to run-time evaluation. For example, proofs appearing in dependently-typed programs, while necessary for type-checking, are usually irrelevant to run-time evaluation. For instance, consider the following function from Agda Standard Library [Agda Team, 2021], `inject≤ : Fin m → m ≤ n → Fin n`. This function takes a finite number less than `m` and a proof that `m ≤ n` and returns back the 'same' finite number. For typing this function, a proof that `m` is less than or equal to `n` is necessary. But while evaluating this function, that proof becomes irrelevant. So, after type-checking and before evaluation, the proof may be erased from the definition of `inject≤`. Such erasure results in better run-time performance.

Next, we describe the utility of irrelevance analysis in the context of type-checking. Consider the following contrived Agda function: `phantom = λ x . if (fib 28 == 317810) then Nat else Bool : Nat → Set`, which ignores its argument and just returns either `Nat` or `Bool` based on the condition whether the Fibonacci number, F_{28}, equals 317810. Using `phantom`, we define: `conv = λ y . y : phantom 0 → phantom 1`. The function `conv` is essentially an identity function. We can immediately see that `conv` should type-check: since `phantom` ignores its argument, we have the equality, `phantom m = phantom n`, for any `m`, `n` ∈ `Nat`. However, without irrelevance analysis, the type-checker needs to reduce `phantom 0` and `phantom 1` to find out whether they are the same type. With a computationally intensive conditional, such a reduction may take a long time! This contrived example shows that irrelevance analysis can improve type-checking performance. We shall take up more realistic examples illustrating the benefits of irrelevance analysis to type-checking in Chapter 3, after we have explained the nuts and bolts of dependency analysis.

Analysis of irrelevance with regard to type-checking and run-time evaluation are referred to as *compile-time irrelevance* and *run-time irrelevance* analyses respectively. Formally, run-time irrelevance analysis identifies sub-expressions that do not affect the outcome of evaluation while compile-time irrelevance analysis identifies sub-expressions that are not needed for type-checking. Both run-time irrelevance and compile-time irrelevance analyses are dependency analyses that enforce the constraint that level **Relevant**, denoting relevant expressions, does not depend upon level **Irrelevant**, denoting irrelevant expressions. We present the details of these analyses with regard to $\text{DDC}^{\top}$ and DDC in Chapter 3.

Analysis of run-time irrelevance and compile-time irrelevance is a well-studied problem in the design of dependent type systems. In some systems, the focus is only on support for run-time irrelevance: see Mishra-Linger and Sheard [2008], Tejiščák [2020]. In other systems, the focus is on compile-time irrelevance: see Abel and Scherer [2012], Pfenning [2001]. Some systems support both, but require them to overlap: see Barras and Bernardo [2008], Miquel [2001], Mishra-Linger [2008]. The system of Moon et al. [2021] we saw earlier (in Section 1.1.11) does not require them to overlap but does not make use of compile-time irrelevance in the typing rules.

Our dependency calculus, DDC, is the only system we are aware of that analyzes run-time irrelevance and compile-time irrelevance separately and makes use of the latter while checking for equality of types in the conversion rule. Further, DDC analyzes these irrelevances in the presence of strong Σ-types with erasable first components. Recall that Σ-types whose terms are eliminated via projections are called strong Σ-types whereas Σ-types whose terms are eliminated via pattern-matching are called weak Σ-types. To the best of our knowledge, no prior work has been able to model strong Σ-types with erasable first components in a

system that analyzes compile-time irrelevance. The exact nature of the problem posed by such Σ-types and the way we resolve it is discussed in detail in Chapter 3.

We started this section with the goal of contextualizing our work on dependency analysis. With this goal in mind, we overviewed the problem of dependency analysis and the standard calculus on the subject. We went over the limitations of this calculus and the way we address those limitations. Addressing those limitations paved the path to our general dependency calculi for pure type systems. We discussed the novelty of these calculi, especially with regard to irrelevance analysis in dependent type systems. In sum, this section contextualizes our work in the broad field of dependency analysis.

Now that we have contextualized our works on both linearity and dependency analyses, we draw a comparison between the two analyses.

1.3 Linearity and Dependency Analyses: A Comparison

In this dissertation, we design calculi for linearity and dependency analyses in pure type systems. One question that may be asked after such a design is whether these calculi can be unified into a single calculus. In Chapter 5, we answer this question in the affirmative. We design a unified calculus, LDC, for combined linearity and dependency analysis in pure type systems.

Unifying linearity and dependency analyses is a challenging problem. We are not aware of any system in literature that unifies these analyses, even in a simply-typed setting. To understand the challenges in unifying these analyses, we need to carefully observe how they are similar to and different from one another.

1.3.1 Preordered Semiring vs. Lattice

In the beginning of this chapter, we pointed out the fundamental similarity connecting dependency and linearity analyses: both of them need to model two different worlds that interact following given constraints. As we dived deeper into these analyses, we saw that they need not be restricted to model two worlds only but can be generalized to model an arbitrary number of worlds. We also saw that the constraints upon interactions among these worlds can be modeled by abstract algebraic structures. For linearity analysis, the algebraic structures are preordered semirings. For dependency analysis, the algebraic structures are lattices. These structures, while similar in some ways, are also different in other ways.

Both preordered semirings and lattices are algebraic structures with two binary operators and a binary order relation. Additionally, lattices, like preordered semirings, have identity elements for the binary operators. There are other similarities as well, for example, the binary operators in both the structures are associative. Nevertheless, there are also significant differences between the two structures:

1. One crucial distinction is that while semirings need to ensure that one operator (multiplication) distributes over the other (addition), lattices do not need to ensure any such property. As a matter of

(A) Lattice M_3 (B) Lattice N_5

Figure 1.2: Examples of non-distributive lattices

fact, there are lattices where neither operator distributes over the other. The simplest of such lattices are M_3 and N_5 [Birkhoff, 1967], both with 5 elements, ordered as shown in Figure 1.2. In M_3, join and meet do not distribute over one another:

$$(\ell_1 \sqcup \ell_2) \sqcap (\ell_3 \sqcup \ell_2) = \top \neq \ell_2 = (\ell_1 \sqcap \ell_3) \sqcup \ell_2$$
$$(\ell_1 \sqcap \ell_2) \sqcup (\ell_3 \sqcap \ell_2) = \bot \neq \ell_2 = (\ell_1 \sqcup \ell_3) \sqcap \ell_2.$$

The same is true of N_5:

$$(\ell_2 \sqcup \ell_1) \sqcap (\ell_3 \sqcup \ell_1) = \ell_3 \neq \ell_1 = (\ell_2 \sqcap \ell_3) \sqcup \ell_1$$
$$(\ell_1 \sqcap \ell_3) \sqcup (\ell_2 \sqcap \ell_3) = \ell_1 \neq \ell_3 = (\ell_1 \sqcup \ell_2) \sqcap \ell_3.$$

Thus, an arbitrary lattice can not be viewed as a preordered semiring. However, distributive lattices, i.e., lattices where join and meet distribute over one another, may be viewed as preordered semirings where multiplication, addition and order are given by join, meet and lattice order respectively. But we see no principled justification in restricting dependency lattices to distributive ones only.

2. Another crucial distinction is that while lattice operations (join and meet) are idempotent, semiring operations (addition and multiplication) need not be so. As a matter of fact, in the preordered semirings of our interest, like the semiring of natural numbers with discrete/total order, the operations are not idempotent. So these semirings can not be viewed as lattices.

From the above discussion, we see that preordered semirings and lattices have irreconcilable differences between them. Therefore, to unify linearity and dependency analyses, we cannot use either of the algebraic structures as the sole parameter to the type system because then, we would miss out something on the other side. A way out of this impasse comes from the observation that even though preordered semirings and lattices have irreconcilably different sets of axioms, they are still algebraic structures with two binary operators and a binary order relation. Therefore, one can define the cartesian product of these two structures. Concretely, given a preordered semiring $\mathcal{Q} = (Q, +, \cdot, 0, 1, <:)$ and a lattice $\mathcal{L} = (L, \sqcap, \sqcup, \top, \bot, \sqsubseteq)$, one can define $\mathcal{Q} \times \mathcal{L}$, the cartesian product of $\mathcal{Q}$ and $\mathcal{L}$, as the set $Q \times L$, together with:

- two constants, $(0, \top)$ and $(1, \bot)$;
- a binary operator, $+$, defined as: $(q_1, \ell_1) + (q_2, \ell_2) \triangleq (q_1 + q_2, \ell_1 \sqcap \ell_2)$;

- another binary operator, $\cdot$, defined as: $(q_1, \ell_1) \cdot (q_2, \ell_2) \triangleq (q_1 \cdot q_2, \ell_1 \sqcup \ell_2)$; and
- a binary order relation, $<:$, defined as: $(q_1, \ell_1) <: (q_2, \ell_2) \triangleq q_1 <: q_2$ and $\ell_1 \sqsubseteq \ell_2$.

We employ this construction to unify linearity and dependency analyses: see Chapter 5 for more details.

1.3.2 Comonadic vs. Monadic

In Section 1.1.7, we pointed out that linear type systems grade the exponential modality, !, of linear logic to analyze usage. In Section 1.2.3, we pointed out that dependency type systems (especially DCC) grade the monadic modality, T, of Moggi's computational metalanguage to analyze dependency. To unify linearity and dependency analyses, we need to understand how these graded modalities relate to one another. The exponential and monadic modality, while similar in some ways, are also different in other ways.

The exponential modality is comonadic whereas the monadic modality, as the name suggests, is monadic [MacLane, 1971]. Monads and comonads are standard category theoretic constructions that are dual to each other. A monad on a category $\mathbb{C}$ is an endofunctor, $T : \mathbb{C} \to \mathbb{C}$, along with natural transformations, $\eta : \mathbf{Id} \to T$ and $\mu : T \circ T \to T$, that obey standard identity and associativity axioms. Here, $\mathbf{Id}$ is the identity functor on $\mathbb{C}$. On the other hand, a comonad on a category $\mathbb{C}$ is an endofunctor, $D : \mathbb{C} \to \mathbb{C}$, along with natural transformations, $\epsilon : D \to \mathbf{Id}$ and $\delta : D \to D \circ D$, that obey standard identity and associativity axioms. Note the reversal of arrows in the definition of a comonad. In the context of programming languages, a monad T is a type constructor equipped with a return function, $\mathbf{return} : A \to T\ A$, and a join function, $\mathbf{join} : T\ T\ A \to T\ A$, that obey the equivalent axioms; and a comonad D is a type constructor equipped with an extract function, $\mathbf{extract} : D\ A \to A$, and a fork function, $\mathbf{fork} : D\ A \to D\ D\ A$, that obey the equivalent axioms.

While the exponential and monadic modalities in and of themselves behave like comonads and monads respectively, their graded versions, used in linear and dependency type systems, behave like graded comonads and graded monads respectively. Graded comonads and graded monads [Fujii, 2019] are basically comonads and monads graded over monoidal categories. Chapter 2 presents their formal category-theoretic definitions. Here, we consider simplified definitions of graded monads and comonads, in the context of programming languages. A monad T, graded over a monoid $\mathcal{M} = (M, \cdot, 1)$, is a type constructor, graded by $m \in M$, and equipped with a return function, $\mathbf{return} : A \to T_1\ A$, and a graded join function, $\mathbf{join}^{m_1, m_2} : T_{m_1}\ T_{m_2}\ A \to T_{m_1 \cdot m_2}\ A$, for all $m_1, m_2 \in M$, that obey the graded versions of the standard monad axioms. Similarly, a comonad D, graded over a monoid $\mathcal{M} = (M, \cdot, 1)$, is a type constructor, graded by $m \in M$, and equipped with an extract function, $\mathbf{extract} : D_1 A \to A$, and a graded fork function, $\mathbf{fork}^{m_1, m_2} : D_{m_1 \cdot m_2} A \to D_{m_1} D_{m_2} A$, for all $m_1, m_2 \in M$, that obey the graded versions of the standard comonad axioms.

Now, given a linear type system, parametrized over a preordered semiring $\mathcal{Q} = (Q, +, \cdot, 0, 1, <:)$, its graded exponential modality behaves like a graded comonad over monoid $(Q, \cdot, 1)$. Such a type system can derive a graded fork function: $\mathbf{fork}^{q_1, q_2} : !_{q_1 \cdot q_2} A \to !_{q_1} !_{q_2} A$, for all $q_1, q_2 \in Q$. However, it does not derive a graded join

function over $(Q, \cdot, 1)$. Next, given a dependency type system (especially DCC), parametrized over a lattice $\mathcal{L} = (L, \sqcap, \sqcup, \top, \bot, \sqsubseteq)$, its graded monadic modality behaves like a graded monad over monoid $(L, \sqcup, \bot)$. Such a type system can derive a graded join function: $\mathbf{join}^{\ell_1,\ell_2} : T_{\ell_1}\, T_{\ell_2}\, A \to T_{\ell_1 \sqcup \ell_2}\, A$, for all $\ell_1, \ell_2 \in L$. However, it does not derive a graded fork function over $(L, \sqcup, \bot)$. Note here that the graded fork function is essential for usage analysis and the graded join function is essential for dependency analysis. But neither the standard linear type systems nor the standard dependency type systems derive both these functions. As such, these linear type systems cannot carry out dependency analysis and these dependency type systems cannot carry out linearity analysis.

Our unified calculus, LDC, resolves this problem through a modality that is simultaneously graded comonadic and graded monadic. LDC can derive both graded join and graded fork functions for this modality, which helps unify linearity and dependency analyses. We present the details of this construction in Chapter 5.

We started this section with the goal of comparing linearity and dependency analyses. With this goal in mind, we went over some of the technical similarities and differences between the two analyses. We discussed how the similarities between the analyses can be utilized to reconcile the differences between them. This discussion gives us a perspective on the problem of unification of linearity and dependency analyses and acts as a background to our unified linearity and dependency calculus, LDC.

Over the last three sections, we have given an overview of our work on linearity and dependency analyses. We have also informally introduced our calculi for these analyses: GRAD, GMCC_e, DDC and LDC. We have gone over the challenges behind the design of these calculi and their novelties. At this point, we would like to draw the reader's attention to the sinew of our work, one that binds all our calculi together. That sinew is *grading.*

1.4 Grading in Programming Languages

This dissertation is a celebration of the concept of grading in programming languages. In this dissertation, we grade modalities, contexts, terms, types, and even typing judgments. All our key constructions employ grading in one form or the other. All our contributions result from ingenious uses of grading. Thus, this dissertation may be seen as an evidence of the power of grading in programming languages. Here, we want to point out that this dissertation explores the power of grading only with regard to linearity and dependency analyses. The concept of grading, however, is much more general and applicable to many other analyses. A complete discussion on grading and its applications in programming languages is out of scope of this dissertation. However, we shall discuss some facets of this topic in this section. We begin by dwelling upon the basic problem that grading addresses.

1.4.1 A Tale of Multiple Worlds

Recall the fundamental similarity that connects dependency and linearity analyses: both the analyses need to model multiple worlds with constrained mutual interactions among them. A typical dependency analysis may model low-security and high-security worlds with the constraint that information never leaks from the latter to the former. A typical linearity analysis may model nonlinear and linear worlds with the constraint that derivations in the former do not make use of assumptions from the latter. Now, let us consider these worlds closely. Since information cannot leak from high-security world to low-security world, an observer from the low-security world cannot distinguish between high-security data items; this privilege is restricted to observers from the high-security world. So an observer from the low-security world and an observer from the high-security world would have different perceptions of the world of data around them. In a similar vein, an observer from the nonlinear world cannot use, and is therefore unaware of, linear resources whereas an observer from the linear world can use, and is therefore aware of, linear resources. So in this case too, the two observers would have different perceptions of the world of resources around them. At this point, we may ask: Is grading related to difference in perceptions of observers? If so, how? We shall motivate the answer to these questions through examples.

Difference in perceptions of the surrounding world is a very common phenomenon which pervades academic disciplines and our daily lives alike. A very archetypal example of this phenomenon is presented in the novella *Flatland* [Abbott, 2008], where the protagonist beautifully sketches the radically different perceptions of the linelanders (1D people), the flatlanders (2D people) and the spacelanders (3D people) of the world around them. Another archetypal example of this phenomenon is presented by the age-old story of the blind men and the elephant, alluded to in the preface of the dissertation, where six blind men acquire radically different perceptions of the animal, based on the body parts they come in contact with. Yet another archetypal example of this phenomenon is presented by the *Kiṃśukopama Jātaka* [Cowell, 1907], the story regarding the *Kiṃśuka* tree alluded to in the preface of the dissertation, where four brothers acquire radically different perceptions of the *Kiṃśuka* tree, based upon the seasons they see the tree.

Now, when there are radical differences in perceptions of the surrounding world, how does one come to terms with the differing perceptions? The protagonist of the novella *Flatland* comes to terms with the differing perceptions of linelanders, flatlanders and spacelanders through an understanding of the multiplicity of dimensions. The six blind men come to terms with their differing perceptions of the elephant through an understanding of the multiplicity of body parts of the animal. The four brothers come to terms with their differing perceptions of the *Kiṃśuka* tree through an understanding of the multiplicity of seasons which affect the appearance of the tree. So, we see that one can come to terms with radically differing perceptions of the surrounding world by taking into consideration the *multiplicity* of the appropriate factor which germinates these differences. In the light of this principle, we can understand how grading is related to difference in perceptions of observers.

Grading shows a way to formalize multiplicities; upon formalization, the multiplicities act as grades. By formalizing multiplicities, grading helps us accommodate differing perceptions in the same system. This is true of both dependency and linearity analyses. To elaborate, by formalizing low and high security

through grades **L** and **H** (elements of lattice $\mathcal{L}_2$, see Section 1.2.3) respectively, a dependency calculus can accommodate, in the same system, the perceptions of observers from both low-security and high-security worlds. Again, by formalizing nonlinear and linear usage through grades ω and 1 (elements of preordered semiring $\mathcal{Q}_{\text{Lin}}$, see Section 1.1.7) respectively, a linearity calculus can accommodate, in the same system, the perceptions of observers from both nonlinear and linear worlds.

Dependency and linear type systems take advantage of grading through various means, especially through graded modalities and graded contexts, as we have already seen. Other means through which grading is employed by type systems include graded types, graded terms, graded equalities and even graded typing judgments. We shall refer to a type system that employs grading through any of the above means as a *graded type system*. By this definition, the dependency and linear type systems we have come across are all graded type systems. The literature contains a wealth of graded type systems used for a variety of applications. While we would not be able to review every graded type system appearing in literature, we would still like to give the reader a taste of some graded type systems that are used in applications other than dependency and linearity analyses.

1.4.2 Graded Type Systems

A graded type system is a type system that employs grading in some form, be it through grading modalities, grading contexts, grading types, grading terms, grading equalities, or even grading typing judgments. Graded type systems abound in literature; we have already come across several such systems in the last three sections. However, our focus has been restricted to graded type systems for dependency and linearity analyses. But as we pointed out earlier, graded type systems are also useful in other applications. Below, we consider two such applications:

1. One might want the same language to support both functional and imperative styles of programming. We know that each style has its own strengths and weaknesses. A language that supports both could offer the programmer the flexibility to exploit the strengths of both the styles without switching languages. It could also offer the opportunity of integrating functional and imperative features in the same program. With this goal in mind, Gifford and Lucassen [1986] introduced *Type and Effect Systems*. Type and effect systems grade typing judgments with effect annotations that delimit the side effects of the terms being type-checked. By grading typing judgments, such systems can accommodate both programs with and without side effects, i.e. both imperative and functional programs. We discuss the utility of grading in the context of type and effect systems in Section 1.4.3.

2. One might want a language to support multiple notions of equality for its terms, corresponding to multiple semantics for the language. Supporting multiple notions of equality is a particularly challenging problem in a dependently-typed language because the equality relation is a part and parcel of a dependent type system. We [Weirich et al., 2019] addressed an instance of this problem, in the context of *generative type abstraction*, by employing a graded equality relation. The graded equality relation

helped us formalize multiple semantics for the language in the same system. We discuss the utility of grading in the context of generative type abstraction in a dependently-typed language in Section 1.4.4.

1.4.3 Grading and Type and Effect Systems

Type and effect systems were introduced by Gifford and Lucassen [1986], whose goal was to support both functional and imperative styles of programming in the same language. Note here that Gifford and Lucassen [1986] used the nomenclature *fluent language* for their type and effect system. In fluent languages, terms have both types and effect classes associated with them. Effect classes are specifications of the side effects of terms. For example, pure functions do not have any side effects and as such, belong to the effect class empty, also written $\varnothing$. On the other hand, imperative procedures can allocate (`A`), read (`R`) and write (`W`) memory locations and as such, belong to the effect class `ARW`. A fluent language may be thought of as containing multiple sublanguages, one corresponding to each effect class. The language, as a whole, ensures that the sublanguages interact with one another honoring the constraints that bind the corresponding effect classes. For example, a pure function, belonging to the $\varnothing$ effect class, should not call an imperative procedure, belonging to the `ARW` effect class.

To enforce such constraints among effect classes, fluent languages employ a static type and effect checking discipline. The judgment used for this purpose is essentially a graded typing judgment, where the grade on the judgment denotes the effect class of the term being type-checked. For example, judgments type-checking pure functions would have grade $\varnothing$ whereas judgments type-checking imperative procedures would have grade `ARW`. The constraints among the effect classes are enforced by the inference rules for this graded typing judgment. Once the type system ensures that the effect classes interact following the set constraints, we can choose any sublanguage to write our programs. For example, if we wish to write functional programs, we can choose the sublanguage corresponding to the $\varnothing$ effect class and be assured that our programs won't have any side effects. Again, if we wish to write imperative programs, we can choose the sublanguage corresponding to the `ARW` effect class. In this way, we can choose between functional and imperative styles without switching languages.

Fluent languages also offer the opportunity of interweaving functional and imperative styles in the same program, which can aid optimization and parallel execution. To enable seamless integration of functional and imperative styles, fluent languages employ the idea of *subeffecting*. The idea of subeffecting comes from the fact that effect classes delimit or are upper bounds on the side-effects of terms. As such, a term belonging to a 'smaller' effect class (subeffect class) may also belong to a 'bigger' effect class. For example, a term belonging to the $\varnothing$ effect class can also belong to the `ARW` effect class. This subeffecting enables an embedding of the sublanguage corresponding to the $\varnothing$ effect class in the sublanguage corresponding to the `ARW` effect class. This embedding, in turn, enables integration of functional and imperative styles in the same program.

The ideas behind fluent languages or type and effect systems were further developed in Lucassen and Gifford [1988] and Talpin and Jouvelot [1994]. Initially, type and effect systems focused on tracking memory manipulations. However, over time, other effects were also brought under the ambit of these systems. For example,

type and effect systems were designed for checked exceptions, safe locking, etc. Many of these type and effect systems share the same basic structure: from this observation, Marino and Millstein [2009] designed a generic system that subsumed several existing type and effect systems. Their generic system, like that of Gifford and Lucassen [1986], employs the same basic principle of grading typing judgments, where grades on the judgments delimit the effects of the terms being type-checked. Their generic system also employs other gadgets, which make it more versatile but are not directly related to grading.

Before winding up our discussion on type and effect systems, we would like to draw the reader's attention to a pertinent observation. We pointed out that exploiting the inherent ordering among effect classes enables a seamless integration of functional and imperative styles in programs written in fluent languages. Ordering (among grades) plays an important role not only in fluent languages but also in other graded type systems. We have already seen the importance of an ordering relation in linearity and dependency analyses. The abstract algebraic structures employed for the analyses, i.e. preordered semiring and lattice, are both ordered structures. In Section 1.1.6, we saw how the ordering relation of the preordered semiring is employed to enable subusaging. In Section 1.2.2, we saw how dependency constraints themselves induce an ordering relation, leading to a lattice model.

The reason why ordering plays such an important role in graded type systems is that ordering provides a simple yet effective way to model one-way interaction between worlds (represented by grades). To elaborate, in the type and effect system discussed above, the ordering $\varnothing <: \texttt{ARW}$ helps model the one-way interaction between effect classes $\varnothing$ and `ARW`, whereby programs from the latter can call programs from the former but not vice-versa. Again, in linear type systems, the ordering $\omega <: 1$ helps model the one-way interaction between nonlinear and linear worlds, whereby programs from the latter can use resources from the former but not vice-versa. Similarly, in dependency type systems, the ordering $\mathbf{L} \sqsubseteq \mathbf{H}$ helps model the one-way interaction between low-security and high-security worlds, whereby programs from the latter can observe data from the former but not vice-versa.

1.4.4 Grading and Generative Type Abstraction

Generative type abstraction is an example of zero-cost abstraction. The philosophy behind zero-cost abstractions is that compile-time abstractions should not incur run-time cost. True to this philosophy, generative type abstraction allows programmers to create new types by abstracting over existing types and guarantees that the abstraction would not incur any run-time cost. To illustrate generative type abstraction, we use the `newtype` construct of Haskell. Below, `Age` is a new type formed by abstracting over `Int`. The type-checker treats `Age` and `Int` as distinct types but there is no run-time distinction between terms of these types. As such, one can define functions on `Age` using functions already defined on `Int`. For example, the function `ageNextYear`, defined on `Age`, is essentially the increment function on `Int`. But note the use of `coerce` in the definition of `ageNextYear`: it informs the type-checker to appropriately coerce the type of the increment function. At run time, however, `coerce` is completely ignored, thereby ensuring no run-time cost of `newtype` abstraction.

```
newtype Age = MkAge Int

ageNextYear :: Age -> Age
ageNextYear = coerce (+ 1)
```

Next, observe the dual nature of new types: a new type is distinct from its underlying type at compile time but not at run time. Thus, the equality relation among types at compile time and at run time are different. For example, types `Age` and `Int` are equal at run time but not at compile time. The type-checker needs to be aware of both the equality relations because `coerce` exploits the run-time equality relation. Accommodating two equality relations, unsurprisingly, requires a graded type system. As a matter of fact, without a graded type system, one easily runs into type unsoundness, owing to an unbridled `coerce` [Weirich et al., 2011]. Below, we explain how.

```
type family AgeToString (a :: Type) :: Type
type instance AgeToString Age = String
type instance AgeToString _ = Bool

wrong :: String
wrong = coerce True
```

The type family `AgeToString` is a type-level function that returns `String` on `Age` but `Bool` on all other types, including `Int`. This difference in treatment of `Age` and `Int` is exploited by `wrong` via the chain of equalities: `Bool` = `AgeToString Int` = `AgeToString Age` = `String`, resulting in type unsoundness. The problem in this chain lies in the equality: `AgeToString Int` = `AgeToString Age`, which does not hold (either at compile time or at run time), even though `Int` and `Age` are equal types at run time. Contrast it with the equality, `Int -> Int` = `Age -> Age`, which holds at run time.

We see that the arguments of type constructors, `AgeToString` and `->`, behave differently. Borrowing terminology from Quine [1960], the argument `a` in `AgeToString a` is *referentially opaque* whereas the arguments `a` and `b` in `a -> b` are *referentially transparent*. According to Quine [1960], a position is referentially transparent if only if it may be subjected to substitutivity of identity (i.e., replacement with equals). Put differently, in a sentence, substitution of an argument with a 'codesignative term (one referring to the same object)' should only be allowed in referentially transparent positions. This explains why `Int -> Int` = `Age -> Age` holds but `AgeToString Int` = `AgeToString Age` does not. Next, to accurately reason in the midst of referentially transparent and opaque arguments, Quine [1960] suggests a doctrine of 'indicating, selectively and changeably, just what positions in the contained sentence are to shine through as referential'. Implementing this doctrine inevitably requires two grades, one to indicate transparent positions and the other to indicate opaque positions. This approach to formalizing generative type abstraction by employing two grades to

indicate transparent and opaque positions was first developed in Weirich et al. [2011] and later followed up in Breitner et al. [2014].

Weirich et al. [2011] and Breitner et al. [2014] use *roles* (another nomenclature for grades) to distinguish between referentially transparent and opaque arguments. They mark transparent and opaque arguments with roles `Rep` (or representational) and `Nom` (or nominal) respectively. For example, the argument `a` of `AgeToString a` is marked with `Nom` whereas the arguments `a` and `b` of `a -> b` are both marked with `Rep`. Corresponding to two roles, there are two notions of equality among types, `Rep` equality (written as $=_{\texttt{Rep}}$) and `Nom` equality (written as $=_{\texttt{Nom}}$), denoting equality at run time and equality at compile time respectively. For example, `Age` $=_{\texttt{Rep}}$ `Int` but `Age` $\neq_{\texttt{Nom}}$ `Int`. This graded equality helps the type-checker fix the soundness problem pointed out above.

The type-checker can now reject `wrong` all the while accepting `ageNextYear`. We explain how. First, note that, `coerce : a` $=_{\texttt{Rep}}$ `b => a -> b`, meaning, `coerce` can only coerce to a `Rep`-equal type. Now, since the argument `a` of `AgeToString a` is marked with `Nom` and `Age` $\neq_{\texttt{Nom}}$ `Int`, one cannot derive the equality `AgeToString Age` $=_{\texttt{Rep}}$ `AgeToString Int`. So, one would not be able to `coerce` a boolean to a `String`, thereby rendering `wrong` ill-typed. However, `ageNextYear` would remain well-typed because one can `coerce` a function of type `Int -> Int` to `Age -> Age`, via the equality `Int -> Int` $=_{\texttt{Rep}}$ `Age -> Age`. This equality holds because the arguments of `->` have role `Rep` and `Age` $=_{\texttt{Rep}}$ `Int`. This way roles help accurately reason about equality at run time, a prerequisite for supporting zero-cost abstraction. Weirich et al. [2011] and Breitner et al. [2014] show how roles help polymorphic languages regain type soundness in the midst of generative type abstraction and type-level computation.

In Weirich et al. [2019], we extend the work of Breitner et al. [2014] to dependently-typed languages. Handling generative type abstraction in a dependently-typed language is a challenging problem, primarily owing to the dual nature of new types and the fact that in dependent systems, types also reduce. The key question to address, then, is how should new types reduce. For example, should the new type `Age` reduce to the underlying type `Int`? If `Age` unconditionally reduces to `Int`, then the type-checker cannot rule out `wrong`. On the other hand, if `Age` does not step at all, then it would be difficult to type-check `ageNextYear`. A way out of this impasse is shown by a graded operational semantics. In Weirich et al. [2019], we grade the value and small-step reduction judgments by the roles `Nom` and `Rep`. According to this graded semantics, `Age` is a value at `Nom` but reduces to `Int` at `Rep`. This graded semantics gets reflected into the type system through the graded equality relation: if a steps to b at role ρ, then $a =_\rho b$. Using this graded equality, the type system is able to rule out `wrong` while allowing `ageNextYear`. In this way, grades help us tackle the problem of generative type abstraction in dependently-typed languages.

By now, we have seen multiple areas of application of grading: linearity analysis, dependency analysis, type and effect systems, and generative type abstraction. There are other areas of application of grading in literature, which we do not consider here, owing to space constraints. In this regard, we would also like to point out that not all aspects of grading have been explored in literature. There is still a lot to discover in this area. No wonder, graded type systems are an active area of research in the programming languages community.

1.5 Practical Significance of Our Work

This dissertation presents calculi for dependency and linearity analyses in pure type systems. The focus is mainly on the metatheoretic properties of these calculi. As yet, none of them have been implemented in a practical programming language. Nevertheless, our work has practical significance. Our work can form the basis of linearity and dependency analyses in any practical programming language that is built upon a pure type system (PTS). Practical programming languages built upon PTSs abound:

- The Haskell language is built upon System FC [Sulzmann et al., 2007], which is essentially the PTS, System F, with type-equality coercions.
- The Twelf theorem prover is built upon the Edinburgh Logical Framework [Harper et al., 1993], which is essentially the PTS λP.
- The Coq proof assistant is built upon the Calculus of Inductive Constructions [Paulin-Mohring, 2015], which is essentially the PTS, CoC, with inductive types.

Our calculi for linearity and dependency analyses can be adapted for use in any of the above and other similar systems. In this regard, the minimalist design principle of our calculi may in fact turn out to be useful. We hope that our calculi will find their way into practical programming languages in near future.

1.6 Contributions

In this chapter, we have given an overview of the following: the problems we solve, why these problems are important, what makes them challenging, how we address these challenges while building upon existing work and some of the insights we gain in the process. Below, we summarize our contributions:

- We [Choudhury, 2022a,b] present GMCC_e, a calculus for dependency analysis in simple type systems. GMCC_e addresses several limitations of DCC [Abadi et al., 1999], the standard calculus on the subject. GMCC_e also provides insight into nuances of dependency analysis.
- We [Choudhury et al., 2022a,b] present DDC, a calculus for dependency analysis in pure type systems. DDC can analyze fine-grained notions of irrelevance that are out of reach of existing calculi.
- We [Choudhury et al., 2020a, 2021] present GRAD, a calculus for linearity analysis in pure type systems. GRAD has a simple, uniform and minimalist design, and addresses some shortcomings of QTT [Atkey, 2018], the first-of-its-kind calculus on the subject.
- We [Choudhury, 2023] present LDC, a calculus for combined linearity and dependency analysis in pure type systems. To the best of our knowledge, LDC is the first calculus that combines linearity analysis and general dependency analysis in pure type systems.

The rest of the dissertation is organized as follows. Chapters 2, 3, 4 and 5 present calculi $\mathrm{GMCC_e}$, DDC, GRAD and LDC respectively. Chapter 6 concludes the dissertation.

Chapter 2

Dependency Analysis in Simple Type Systems

In this chapter, we design calculi for dependency analysis in simple type systems. We know that Abadi et al. [1999] presented a general calculus for dependency analysis in simple type systems. Over the years, their calculus has become the standard on the subject. Their calculus, DCC, is a simple extension of Moggi's monadic metalanguage [Moggi, 1991]. The monadic metalanguage is a general calculus for analyzing computational effects. Moggi showed that computational effects in programming languages can be understood in terms of monads from category theory [MacLane, 1971]. At first sight, computational effects seem to be quite different from dependencies. So, it comes as a surprise (pointed out by Abadi et al. [1999] themselves) that with just a simple extension, a calculus for analyzing computational effects can also analyze dependencies.

In this regard, Abadi et al. [1999] point out a common feature underlying monads and security levels: just as there is no general way of projecting out of a 'monad world', there is also no non-trivial way of projecting out of a 'high-security world'. Concretely, just as there is no general function of type $TA \to A$ for a monad T and a type A, there is also no non-trivial function from high-security data to low-security variables. This analogy explains the monadic aspect of dependency analysis.

However, the monadic aspect of dependency analysis is just half of the story. Everyday experience shows us that security constraints can be enforced not only by restricting outflow but also by restricting inflow. For example, in a world with two security levels, $\mathbf{L} \sqsubseteq \mathbf{H}$, security can be enforced not only by restricting projection out of level $\mathbf{H}$, but also by restricting injection into level $\mathbf{L}$. These ways are dual to one another. The way of restricting projection, employed in DCC, goes via monads. On the other hand, the way of restricting injection goes via comonads. Concretely, similar to a monad T which restricts by disallowing a general function of type $TA \to A$, a comonad D restricts by disallowing a general function of type $A \to DA$. While the monadic aspect of dependency analysis has received considerable attention in literature, the comonadic aspect has received less so.

In this chapter, we show that just like the monadic aspect, the comonadic aspect of dependency analysis also has much to offer. Going further, we show that the monadic and the comonadic aspects interact nicely with one another. We design a language that integrates these two aspects into a single system. This integration helps us unify DCC, a monadic dependency calculus and λ° [Davies, 2017], a comonadic dependency calculus (that is known to be outside the reach of DCC [Abadi et al., 1999]). It also leads to a novel technique for proving correctness of dependency analysis. Above all, it shines light on nuances of dependency analysis.

In short, we make the following contributions in this chapter:

- We present a Graded Monadic Comonadic Calculus, GMCC, and its extension, $\mathrm{GMCC_e}$, and provide meaning-preserving translations from both DCC and λ° to $\mathrm{GMCC_e}$.
- We show that the protection judgment of DCC, when appropriately modified, enables comonadic reasoning in the language. Further, we show that under certain restrictions, DCC, with this modification, is equivalent to GMCC. This equivalence helps explain the nonstandard bind-rule of DCC in terms of standard categorical concepts.
- GMCC and $\mathrm{GMCC_e}$ are general calculi that are sound with respect to a class of categorical models. These categorical models motivate a novel technique for proving correctness of dependency analyses. We use this technique to provide simple proofs of correctness for both DCC and λ°.

In the next section, we consider some examples that show the utility of two prominent dependency analyses, i.e., information flow analysis and binding-time analysis. This section is meant to provide motivation to readers who are not very familiar with practical applications of dependency analysis.

2.1 Dependency Analysis in Action

2.1.1 Information Flow Analysis

Consider a database `Db` containing demographic information of a city. For the sake of simplicity, let's say the database is represented as a list of tuples with elements of the list corresponding to residents of the city and elements of the tuples corresponding to their demographic information. Further, let's assume that each tuple has only 4 elements for recording name, age, ethnicity and monthly income of a resident (in that order). According to the policies of the city council, name, age and ethnicities of the residents are non-sensitive information that may be shared without any constraint, whereas monthly income of the residents is sensitive information that may be shared only with people who are allowed to handle such information.

Now, consider the following queries:

1. What fraction of the elderly city residents (age $\geq$ 65 years) are ethnically Caucasian?

2. What is the average monthly income of ethnically Asian residents of the city?

With the above queries in mind, a programmer seeks the outputs of the following programs, written in `Haskell`-like syntax:

1.
```
lstEld = filter (\ x -> second x >= 65) Db
lstEldC = filter (\ x -> third x == "Caucasian") lstEld
fracEldC = (length lstEldC) / (length lstEld)
print fracEldC
```

2.
```
lstAsn = filter (\ x -> third x == "Asian") Db
totIncA = foldr (\ x y -> fourth x + y) 0 lstAsn
avgIncA = totIncA / (length lstAsn)
print avgIncA
```

Note that functions `second`, `third` and `fourth` access the second, third and fourth elements of a tuple respectively; the function `filter` filters out the elements of the input list that do not satisfy the input condition; the function `length` returns the length of the input list; the function `foldr` folds over the input list from right to left, using the input binary function and the input initial value. The question we need to address now is whether the programmer may be allowed to see the outputs of the above programs. Let's suppose this programmer does not have the permission to handle sensitive information. Then, the output of the second program should not be shared with this programmer because it reveals (at least partially) monthly income data. For example, if there's just a single ethnically Asian resident in the city, then the output of the second program gives away the monthly income of that resident. The output of the first program may, however, be shared freely with this programmer because it does not reveal any sensitive information.

Now, how do we reach this conclusion regarding the sharing of outputs by *just* analyzing the respective programs? In other words, given a program, how do we decide whether or not its output reveals any sensitive information? To answer this question, we need to perform an information flow analysis: If the output of a program *depends upon* any information deemed sensitive, then the output too needs to be treated as sensitive. Conversely, if the output *does not depend upon* any sensitive information, then the output too is not sensitive. In the second program above, the output, `avgIncA`, is sensitive because it *depends upon* `totIncA`, which in turn *depends upon* `fourth x`, a sensitive piece of information. On the other hand, in the first program, all the data used to compute the output are non-sensitive, rendering the output itself non-sensitive.

To analyze dependency of output upon sensitive information, information flow calculi typically incorporate sensitivity of information in the types themselves. For example, the type of `Db` would be `[(T L String, T L Int, T L String, T H Int)]`, where `T l String` and `T l Int` are the types of `l`-security strings and

integers respectively, with `l` being `L` or `H`, corresponding to low-security and high-security data respectively. In information flow calculi, functions, too, have types that take sensitivity of information into account. For example, the type of '+' would be `T l Int -> T l Int -> T l Int`, where `l` may be `L` or `H`. Now, if we write the above programs in such a calculus, we would see that `fracEldC` has type `T L Double` whereas `avgIncA` has type `T H Double`. Any user may access terms of type `T L Double` but terms of type `T H Double` may only be accessed by users with high-security clearance: this restriction would typically be enforced within the calculus through the noninterference property. In this way, information flow calculi ensure secure flow of information in security-critical applications.

2.1.2 Binding-Time Analysis

Our next example takes up another form of dependency analysis: binding-time analysis. Binding-time analysis helps in staged execution of programs. Staged execution comes in handy when programs have inputs that are statically known in addition to inputs that are known only at run-time. Computations that depend only upon static inputs may be carried out statically, thereby producing residual programs that can be executed faster at run-time. To find out which computations depend only upon static inputs, we perform a dependency analysis, called binding-time analysis.

Next, we shall use the same database example to show binding-time analysis in action. However, instead of sensitivity of information, here we focus on availability of information. From the given demographic parameters, name and ethnicity remain constant over time whereas monthly income varies. To account for this fact, the city council mandates that every resident update their monthly income on the last day of each month. Now, to get an accurate answer to the second query, one needs to run the second program on the last day of each month. But if the database contains millions of entries, running this program may burden the computing system on such days. However, observe that some or perhaps most of the work done by this program need not wait for the update in monthly incomes. For example, `lstAsn` can be computed statically beforehand because this computation depends only upon static information. Binding-time analysis identifies such computations, thereby enabling faster execution of programs.

Binding-time calculi, similar to information flow calculi, incorporate information about binding-time in the types themselves. For example, the type of `Db` would be `[(T Early String, T Early Int, T Early String, T Later Int)]`, where `T l String` and `T l Int` are respectively the types of strings and integers available at time `l`, with `l` being `Early` or `Later`, corresponding to early and late availability of data respectively. In binding-time calculi, similar to information flow calculi, functions too have types that take binding-time information into account. For example, the type of '+' would be `T l Int -> T l Int -> T l Int`, where `l` may be `Early` or `Later`. Now, if we write the second program in such a calculus, we would see that the computation of `lstAsn` depends upon early data only (To compute `totIncA`, however, we would need data that is available later, i.e., `fourth x`). So we could compute `lstAsn` beforehand and thereafter run *only the residual program* after the stipulated update, thereby reducing the burden on the computing system on the last day of each month. To give an estimate, if ethnically Asian residents constitute 5% of the total population, then, compared to the original program, the residual program may run 20x faster.

Both the examples discussed above show how dependency analysis addresses practical problems. Observe the similarity in the two problems we take up in this section. Though the problems have their origins in very different areas, one in information security and the other in program execution, nevertheless, they share the same basic structure. Both of them need to model two worlds, with just one-way interaction between the worlds: this is a classic dependency problem. When viewed as such, both the original problems yield to a dependency analysis in very much the same way. The dependency analysis in both cases is carried out with respect to the two-element abstract lattice, $\mathcal{L}_2$, with elements ℓ_1 and ℓ_2, where $\ell_1 \sqsubseteq \ell_2$. In the first case, $\ell_1 = \mathbf{L}$ and $\ell_2 = \mathbf{H}$; in the second case, $\ell_1 = \mathbf{Early}$ and $\ell_2 = \mathbf{Later}$. Note here how parametrization by an abstract lattice makes dependency analysis both powerful and versatile. With this motivation, we work towards building our dependency calculi, which are not specialized to any particular dependency analysis but are designed to analyze dependencies over abstract algebraic structures. These calculi would help us understand the abstract nature of dependency analysis. We present two key dependency calculi in this chapter: GMCC, and its extension, GMCC_e. The calculus GMCC is built up from a Graded Monadic Calculus, GMC, and a Graded Comonadic Calculus, GCC. In the next section, we present the Graded Monadic Calculus (GMC).

2.2 Graded Monadic Calculus

Moggi [1991] showed that computational effects can be understood in terms of monads. On the other hand, Gifford and Lucassen [1986] showed that side-effects can be tracked using effect classes (see Section 1.4.3). Wadler and Thiemann [2003] later adapted effect classes to monads but left open the question of a general theory of effects and monads. Eventually, Katsumata [2014] presented such a general theory in terms of graded monads. In this section, we adapt Katsumata's Explicit Subeffecting Calculus to present a simply-typed Graded Monadic Calculus (GMC).

GMC is an extension of the simply-typed λ-calculus with a grade-annotated monadic type constructor, T_m. The grades, m, are drawn from an arbitrary preordered monoid $\mathcal{M} = (M, \cdot, 1, <:)$. Recall that a preordered monoid $\mathcal{M}$ is a monoid $(M, \cdot, 1)$ along with a preorder $<:$ such that the binary operator is monotonic with respect to the order relation, meaning, if $m_1 <: m_1'$ and $m_2 <: m_2'$, then $m_1 \cdot m_2 <: m_1' \cdot m_2'$. Now, GMC is a type system parametrized by preordered monoids; so, whenever we need to be precise, we use GMC($\mathcal{M}$) to refer to GMC parametrized by preordered monoid $\mathcal{M}$. Throughout this dissertation, we follow this convention for any calculus parametrized by algebraic structures.
Next, we present the calculus formally.

2.2.1 Grammar and Type System

The grammar of the calculus appears in Figure 2.1. In addition to the types and terms of standard λ-calculus, we have a graded monadic type, $T_m\ A$, and terms related to it. The typing rules of the calculus appear in Figure 2.2. We omit the typing rules of standard λ-calculus and consider only the ones related to the graded

monadic type. Note that in this dissertation, we implicitly assume that the domains of contexts of all typing judgments are free from duplicates.

$$
\begin{aligned}
\text{types}, A, B &::= \mathbf{Unit} \mid \mathbf{Void} \mid A \to B \mid A \times B \mid A + B \mid T_m\ A \\
\text{terms}, a, b, f, g &::= x \mid \lambda x : A.b \mid b\ a \\
&\mid (a_1, a_2) \mid \mathbf{proj}_1\ a \mid \mathbf{proj}_2\ a \mid \mathbf{unit} \\
&\mid \mathbf{inj}_1\ a_1 \mid \mathbf{inj}_2\ a_2 \mid \mathbf{case}\ a\ \mathbf{of}\ b_1\ ;\ b_2 \mid \mathbf{abort}\ a \\
&\mid \mathbf{ret}\ a \mid \mathbf{lift}^m f \mid \mathbf{join}^{m_1,m_2}\, a \mid \mathbf{up}^{m_1,m_2}\, a \\
\text{contexts}, \Gamma &::= \varnothing \mid \Gamma, x : A
\end{aligned}
$$

FIGURE 2.1: Grammar of GMC

Now, we look at the typing rules. The rules M-RETURN, M-FMAP, and M-JOIN are generalizations of the corresponding rules for the ungraded monadic type. Observe that the rule M-JOIN 'joins' the grades using the binary operator of the monoid. The rule M-UP relaxes the grade on the monadic type. If $\mathcal{M}$ is the trivial preordered monoid, then the above rules degenerate to the standard typing rules for monads.
Next, we look at the equational theory of the calculus.

2.2.2 Equational Theory

Equality over terms of the graded monadic calculus is a congruent equivalence relation generated by the standard $\beta\eta$-equality rules over λ-terms and the additional rules which appear in Figure 2.3. For presenting these rules, we use the shorthand notation: $a\ {}^{m_1}\!\gg\!=^{m_2} f \triangleq \mathbf{join}^{m_1,m_2}((\mathbf{lift}^{m_1} f)\ a)$, where $a : T_{m_1}\ A$ and $f : A \to T_{m_2}\ B$. Note that ${}^{m_1}\!\gg\!=^{m_2}$ is a graded **bind**-operator. Also, note here that throughout this dissertation, we take the liberty of omitting the domain types of λs, whenever such omission is innocuous, as is the case with the rules in Figure 2.3.

Next, we motivate the rules. The first two rules correspond to preservation of identity function and composition of functions by **lift**. The next two rules correspond to reflexivity and transitivity of the order relation.

$\boxed{\Gamma \vdash a : A}$ *(Typing rules)*

$$
\text{M-RETURN}\ \frac{\Gamma \vdash a : A}{\Gamma \vdash \mathbf{ret}\ a : T_1\ A}
\qquad
\text{M-FMAP}\ \frac{\Gamma \vdash f : A \to B}{\Gamma \vdash \mathbf{lift}^m f : T_m\ A \to T_m\ B}
\qquad
\text{M-JOIN}\ \frac{\Gamma \vdash a : T_{m_1}\ T_{m_2}\ A}{\Gamma \vdash \mathbf{join}^{m_1,m_2}\, a : T_{m_1 \cdot m_2}\ A}
$$

$$
\text{M-UP}\ \frac{\Gamma \vdash a : T_{m_1}\ A \qquad m_1 <: m_2}{\Gamma \vdash \mathbf{up}^{m_1,m_2}\, a : T_{m_2}\ A}
$$

FIGURE 2.2: Typing rules of GMC (Excerpt)

$$\mathbf{lift}^m (\lambda x.x) \equiv \lambda x.x \tag{2.1}$$
$$\mathbf{lift}^m (\lambda x.g\,(f\,x)) \equiv \lambda x.(\mathbf{lift}^m g)\,((\mathbf{lift}^m f)\,x) \tag{2.2}$$
$$\mathbf{up}^{m_1,m_1} a \equiv a \tag{2.3}$$
$$\mathbf{up}^{m_2,m_3} (\mathbf{up}^{m_1,m_2} a) \equiv \mathbf{up}^{m_1,m_3} a \tag{2.4}$$
$$(\mathbf{up}^{m_1,m_1'} a)\ {}^{m_1'}{\gg\!=}^{m_2}\ f \equiv \mathbf{up}^{m_1\cdot m_2, m_1'\cdot m_2} (a\ {}^{m_1}{\gg\!=}^{m_2}\ f) \tag{2.5}$$
$$a\ {}^{m_1}{\gg\!=}^{m_2'}\ (\lambda x.\mathbf{up}^{m_2,m_2'} b) \equiv \mathbf{up}^{m_1\cdot m_2, m_1\cdot m_2'} (a\ {}^{m_1}{\gg\!=}^{m_2}\ \lambda x.b) \tag{2.6}$$
$$(\mathbf{ret}\ a)\ {}^{1}{\gg\!=}^{m}\ f \equiv f\ a \tag{2.7}$$
$$a\ {}^{m_1}{\gg\!=}^{1}\ (\lambda x.\mathbf{ret}\ x) \equiv a \tag{2.8}$$
$$(a\ {}^{m_1}{\gg\!=}^{m_2}\ f)\ {}^{m_1\cdot m_2}{\gg\!=}^{m_3}\ g \equiv a\ {}^{m_1}{\gg\!=}^{m_2\cdot m_3}\ (\lambda x.(f\ x\ {}^{m_2}{\gg\!=}^{m_3}\ g)) \tag{2.9}$$

FIGURE 2.3: Equality rules of GMC (Excerpt)

The two rules after that correspond to commutativity of **bind** with **up**. The two subsequent rules correspond to **ret** being the left and the right identity of **bind**. The last rule corresponds to associativity of **bind**.

Now, we want to interpret this calculus in a suitable category. The types of standard λ-calculus can be interpreted in any bicartesian closed category. To interpret the graded monadic type, we need a graded monad. A graded monad is a standard category-theoretic construction. An extensive treatise on graded monads and their duals, graded comonads, can be found in Fujii [2019]. For the sake of self-containment, we shall briefly review the theory behind graded monads in the following subsection.

2.2.3 Graded Monads

A graded monad is a lax monoidal functor [MacLane, 1971] of a certain kind (which we shall soon make precise). A lax monoidal functor from a monoidal category $(M, \otimes_M, 1_M)$ to a monoidal category $(N, \otimes_N, 1_N)$ is a 3-tuple (F, F_2, F_0) where,

- F is a functor from M to N
- For $X, Y \in \text{Obj}(M)$, morphisms
  $F_2(X,Y) : F(X) \otimes_N F(Y) \to F(X \otimes_M Y)$ are natural in X and Y
- $F_0 : 1_N \to F(1_M)$

such that the diagrams in Figure 2.4 commute.

Note that here we assume M and N to be strict monoidal categories, i.e. $1_M \otimes_M X = X = X \otimes_M 1_M$ and $(X \otimes_M Y) \otimes_M Z = X \otimes_M (Y \otimes_M Z)$ for any $X, Y, Z \in \text{Obj}(M)$, and similarly for N.

An example of a strict monoidal category is a preordered monoid $\mathcal{M}$. Any preorder $(M, <:)$ may be seen as a category, $\mathbf{C}((M, <:))$, which has M as its set of objects and for any $m_1, m_2 \in M$, a unique morphism

$$
\begin{array}{ccccc}
F(1_M) \otimes_N F(X) & \xleftarrow{F_0 \otimes_N \mathrm{id}} & F(X) & \xrightarrow{\mathrm{id} \otimes_N F_0} & F(X) \otimes_N F(1_M) \\
& \searrow^{F_2(1_M, X)} & \downarrow{\scriptstyle \mathrm{id}} & \swarrow_{F_2(X, 1_M)} & \\
& & F(X) & &
\end{array}
$$

$$
\begin{array}{ccc}
F(X) \otimes_N F(Y) \otimes_N F(Z) & \xrightarrow{\mathrm{id} \otimes_N F_2(Y,Z)} & F(X) \otimes_N F(Y \otimes_M Z) \\
{\scriptstyle F_2(X,Y) \otimes_N \mathrm{id}} \downarrow & & \downarrow {\scriptstyle F_2(X, Y \otimes_M Z)} \\
F(X \otimes_M Y) \otimes_N F(Z) & \xrightarrow{F_2(X \otimes_M Y, Z)} & F(X \otimes_M Y \otimes_M Z)
\end{array}
$$

Figure 2.4: Commutative diagrams for lax monoidal functor

from m_1 to m_2 if and only if $m_1 <: m_2$. Identity morphisms and composition of morphisms are given by reflexivity and transitivity of the order relation. Utilizing this categorification of preorders, a preordered monoid $\mathcal{M} = (M, \cdot, 1, <:)$ may be categorified to a strict monoidal category as: $\mathbf{C}(\mathcal{M}) = (\mathbf{C}((M, <:)), \cdot, 1)$.

Another example of a strict monoidal category is the category of endofunctors. Given any category $\mathbb{C}$, the endofunctors of $\mathbb{C}$ form a strict monoidal category, $\mathbf{End}_{\mathbb{C}}$, with the monoidal product given by composition of functors and the identity object given by the identity functor.

Now, we can make the definition of a graded monad precise. Given a strict monoidal category $\mathbb{M}$, an $\mathbb{M}$-graded monad over $\mathbb{C}$ is a lax monoidal functor from $\mathbb{M}$ to $\mathbf{End}_{\mathbb{C}}$. Our interest lies in $\mathbf{C}(\mathcal{M})$-graded monads, where $\mathcal{M}$ is a preordered monoid. We wish to use $\mathbf{C}(\mathcal{M})$-graded monads to interpret GMC($\mathcal{M}$). However, such graded monads, by themselves, do not stand up to the task. To see why, try interpreting rule M-Fmap! This shortcoming should not come as a surprise because we know that monads, in and of themselves, cannot model the monadic type constructor of Moggi's computational metalanguage [Moggi, 1991]. In order to do that, monads need to be accompanied with tensorial strengths. Here too, we need to add tensorial strengths to graded monads in order to interpret the graded monadic type constructor. Now, one can add tensorial strength separately after having defined a graded monad. But alternatively, one can also define a strong graded monad in one go using the category of strong endofunctors and strong natural transformations.

An endofunctor F on a monoidal category $(M, \otimes, 1, \alpha, \lambda, \rho)$ is said to be strong [Eilenberg and Kelly, 1966, Kock, 1970, 1972] if there exists morphisms $t_{X,Y} : X \otimes F(Y) \to F(X \otimes Y)$, natural in X and Y, for $X, Y \in \mathrm{Obj}(M)$, such that the diagrams in Figure 2.5 commute.

Given strong endofunctors (F, t^F) and (G, t^G) on M, a natural transformation $\alpha : F \to G$ is said to be strong, if for any $X, Y \in \mathrm{Obj}(M)$, the diagram in Figure 2.6 commutes. Strong endofunctors can be defined over arbitrary monoidal categories; however, we just need the ones over cartesian monoidal categories. Given any cartesian category $\mathbb{C}$, let $\mathbf{End}^s_{\mathbb{C}}$ denote the category whose objects are strong endofunctors over $(\mathbb{C}, \times, \top)$ (where $\top$ is the terminal object) and whose morphisms are strong natural transformations between them. Like category $\mathbf{End}_{\mathbb{C}}$, category $\mathbf{End}^s_{\mathbb{C}}$ too is strict monoidal with the monoidal product and the identity object defined in the same way.

$$
\begin{array}{ccccc}
(X \otimes Y) \otimes FZ & & \xrightarrow{\quad t_{X \otimes Y, Z} \quad} & & F((X \otimes Y) \otimes Z) \\
\downarrow{\scriptstyle \alpha^{-1}_{X,Y,FZ}} & & & & \downarrow{\scriptstyle F\alpha^{-1}_{X,Y,Z}} \\
X \otimes (Y \otimes FZ) & \xrightarrow{\mathrm{id} \otimes t_{Y,Z}} & X \otimes F(Y \otimes Z) & \xrightarrow{t_{X, Y \otimes Z}} & F(X \otimes (Y \otimes Z))
\end{array}
$$

$$
\begin{array}{ccc}
1 \otimes FX & \xrightarrow{t_{1,X}} & F(1 \otimes X) \\
& {\scriptstyle \lambda_{FX}} \searrow & \downarrow{\scriptstyle F\lambda_X} \\
& & FX
\end{array}
$$

FIGURE 2.5: Commutative diagrams for strong endofunctor

$$
\begin{array}{ccc}
X \otimes FY & \xrightarrow{\mathrm{id} \otimes \alpha_Y} & X \otimes GY \\
\downarrow{\scriptstyle t^F_{X,Y}} & & \downarrow{\scriptstyle t^G_{X,Y}} \\
F(X \otimes Y) & \xrightarrow{\alpha_{X \otimes Y}} & G(X \otimes Y)
\end{array}
$$

FIGURE 2.6: Commutative diagram for strong natural transformation

Finally, we have the definition of a strong graded monad. Given a strict monoidal category $\mathbb{M}$ and a cartesian category $\mathbb{C}$, a strong $\mathbb{M}$-graded monad over $\mathbb{C}$ is a lax monoidal functor from $\mathbb{M}$ to $\mathbf{End}^s_{\mathbb{C}}$. Now, using strong $\mathbf{C}(\mathcal{M})$-graded monads, we can build categorical models for the graded monadic calculus parametrized by $\mathcal{M}$.

2.2.4 Categorical Model

Let $\mathbb{C}$ be any bicartesian closed category. Let $(\mathbf{T}, \mu, \eta)$ be a strong $\mathbf{C}(\mathcal{M})$-graded monad over $\mathbb{C}$. Then, the interpretation, $[\![_]\!]$, of types and terms of GMC($\mathcal{M}$) is as follows. The types and terms of standard λ-calculus are interpreted in the usual way. The graded monadic type and terms related to it are interpreted as in Figure 2.7.

$$
\begin{aligned}
[\![T_m\ A]\!] &= \mathbf{T}_m [\![A]\!] \\
[\![\mathbf{ret}\ a]\!] &= [\![\Gamma]\!] \xrightarrow{[\![a]\!]} [\![A]\!] \xrightarrow{\eta_{[\![A]\!]}} \mathbf{T}_1 [\![A]\!] \\
[\![\mathbf{lift}^m f]\!] &= \Lambda\Big([\![\Gamma]\!] \times \mathbf{T}_m [\![A]\!] \xrightarrow{t^{\mathbf{T}_m}_{[\![\Gamma]\!], [\![A]\!]}} \mathbf{T}_m([\![\Gamma]\!] \times [\![A]\!]) \xrightarrow{\mathbf{T}_m(\Lambda^{-1} [\![f]\!])} \mathbf{T}_m [\![B]\!] \Big) \\
[\![\mathbf{join}^{m_1, m_2} a]\!] &= [\![\Gamma]\!] \xrightarrow{[\![a]\!]} \mathbf{T}_{m_1} \mathbf{T}_{m_2} [\![A]\!] \xrightarrow{\mu^{m_1, m_2}_{[\![A]\!]}} \mathbf{T}_{m_1 \cdot m_2} [\![A]\!] \\
[\![\mathbf{up}^{m_1, m_2} a]\!] &= [\![\Gamma]\!] \xrightarrow{[\![a]\!]} \mathbf{T}_{m_1} [\![A]\!] \xrightarrow{\mathbf{T}^{m_1 <: m_2}_{[\![A]\!]}} \mathbf{T}_{m_2} [\![A]\!]
\end{aligned}
$$

FIGURE 2.7: Interpretation of GMC (Excerpt)

There are a few things to note here:

- $\mathbf{T}(m)$, written as $\mathbf{T}_m$, is a functor
- $\mathbf{T}(m_1 <: m_2)$, written as $\mathbf{T}^{m_1 <: m_2}$, is a natural transformation
- η is a natural transformation from $\mathbf{Id}$ to $\mathbf{T}_1$
- μ^{m_1,m_2} are morphisms from $\mathbf{T}_{m_1} \circ \mathbf{T}_{m_2}$ to $\mathbf{T}_{m_1 \cdot m_2}$, and are natural in both m_1 and m_2
- $t^{\mathbf{T}_m}$ denotes the strength of $\mathbf{T}_m$
- Λ and Λ^{-1} denote currying and uncurrying respectively

Let us now see why this interpretation satisfies the equational theory of the calculus. Equations (2.1) and (2.2) follow because $\mathbf{T}_m$ is a functor, for any $m \in M$. Equations (2.3) and (2.4) follow because $\mathbf{T}$ is a functor and as such, preserves identity morphisms and composition of morphisms. Equations (2.5) and (2.6) follow because μ is natural in its first component and its second component respectively. Equations (2.7) and (2.8) follow respectively from the left and right unit laws for graded monad, laws that correspond to the commutative triangles in Figure 2.4. Equation (2.9) follows from the associative law for graded monad, the law that corresponds to the commutative square in Figure 2.4. Note that the soundness of this interpretation also relies upon the axioms regarding strength, shown in Figures 2.5 and 2.6.

Thus, given a preordered monoid $\mathcal{M}$, a bicartesian closed category, $\mathbb{C}$, along with a strong $\mathbf{C}(\mathcal{M})$-graded monad over $\mathbb{C}$, gives us a sound model of GMC($\mathcal{M}$).

Theorem 2.1 If $\Gamma \vdash a : A$ in GMC, then $[\![a]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], [\![A]\!])$. Further, if $\Gamma \vdash a_1 : A$ and $\Gamma \vdash a_2 : A$ such that $a_1 \equiv a_2$ in GMC, then $[\![a_1]\!] = [\![a_2]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], [\![A]\!])$.

Note that proofs of stated lemmas and theorems do not appear in the main body of the dissertation but may be found in the corresponding appendices.

2.3 DCC and GMC

In this section, we look at the relation between DCC and GMC. We have already touched upon DCC in Section 1.2.3. In this section, we review the calculus in more detail. For simplicity, we first focus on the terminating fragment of the calculus and consider non-termination later in this chapter.

Recall that the Dependency Core Calculus (DCC) is simply-typed λ-calculus, extended with an indexed type constructor, T_ℓ, which help analyze dependencies. The indices, ℓ, are elements of an abstract lattice, $\mathcal{L} = (L, \sqcup, \sqcap, \bot, \top)$. The lattice structure for the calculus is motivated by the lattice model of secure information flow [Denning, 1976]. As we discussed in Section 1.2.2, the lattice model provides an abstract structure to reason about dependency constraints. The lattice elements correspond to dependency levels, with $\ell_1 \sqsubseteq \ell_2$ meaning ℓ_2 may depend upon ℓ_1, and $\neg(\ell_1 \sqsubseteq \ell_2)$ meaning ℓ_2 should not depend upon ℓ_1. (Here, $\sqsubseteq$ is the implied order of the lattice.) Next, we look at the type system and equational theory of DCC.

$\boxed{\ell \sqsubseteq A}$ *(DCC Protect)*

$$\frac{\ell \sqsubseteq A \qquad \ell \sqsubseteq B}{\ell \sqsubseteq A \times B}\ \text{Prot-Prod} \qquad \frac{\ell \sqsubseteq B}{\ell \sqsubseteq A \to B}\ \text{Prot-Fun} \qquad \frac{\ell_1 \sqsubseteq \ell_2}{\ell_1 \sqsubseteq T_{\ell_2}\, A}\ \text{Prot-Monad} \qquad \frac{\ell \sqsubseteq A}{\ell \sqsubseteq T_{\ell'}\, A}\ \text{Prot-Already}$$

FIGURE 2.8: Protection rules for DCC

2.3.1 Type System and Equational Theory of DCC

DCC uses an auxiliary protection judgment to analyze dependency. The protection judgment, written $\ell \sqsubseteq A$, and presented in Figure 2.8, can be read as: the terms of type A may depend upon level ℓ. With this reading of the protection judgment, the calculus may be said to be correct when it satisfies the following condition: if $\ell \sqsubseteq A$ and $\neg(\ell \sqsubseteq \ell')$, then the terms of type A 'should not be visible' at ℓ'. We shall formalize this correctness condition in Section 2.8.2, once we have made clear the semantic motivation behind the protection judgment. As alluded to in Section 1.2.3, Abadi et al. [1999] do not present a semantic motivation for this judgment. In this chapter, we shall address this shortcoming by motivating the judgment from standard categorical concepts. For now, we look at how the type system employs this judgment.

The typing rules of DCC consist of the ones for standard λ-calculus, along with the introduction and elimination rules for T_ℓ, shown below.

$\boxed{\Gamma \vdash a : A}$ *(DCC Typing (Excerpt))*

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash \mathbf{eta}^{\ell}\ a : T_\ell\ A}\ \text{DCC-Eta} \qquad \frac{\Gamma \vdash a : T_\ell\ A \qquad \Gamma, x : A \vdash b : B \qquad \ell \sqsubseteq B}{\Gamma \vdash \mathbf{bind}^{\ell}\ x = a\ \mathbf{in}\ b : B}\ \text{DCC-Bind}$$

Observe how the protection judgment in rule DCC-BIND ensures that a is visible to B only if B has the necessary permission. Further, observe that rule DCC-BIND, unlike a standard monadic bind rule, does not have the return type, B, wrapped with the constructor T_ℓ. This difference is what makes the rule nonstandard, as pointed out in Section 1.2.3. We shall explain the motivation behind this deviation in Section 2.8.1.

Now, we consider the equational theory of DCC. Abadi et al. [1999] do not explicitly provide an equational theory for DCC. However, they provide an operational semantics for DCC. We describe the equational theory corresponding to the operational semantics they provide. The terms of DCC can be seen as λ-terms annotated with security labels. If we erase the annotations, we are left with plain λ-terms. Plain λ-terms already have an equational theory: the one generated by the standard $\beta\eta$-rules. Using this theory, we define the equational theory of DCC as follows: two DCC-terms are equal, if and only if, after erasure, they

are equal as λ-terms, i.e., $a_1 \simeq a_2 \triangleq \lfloor a_1 \rfloor \equiv \lfloor a_2 \rfloor$, where $\simeq$ is the equality relation on DCC-terms, $\equiv$ is the standard $\beta\eta$-equality on λ-terms, and $\lfloor a \rfloor$ denotes the plain λ-term corresponding to a DCC-term a. This erasure function on DCC-terms is defined by a straightforward recursion in all cases other than the ones that introduce and eliminate the graded modal type. In these two cases, the function is defined as: $\lfloor \textbf{eta}^{\ell}\, a \rfloor = \lfloor a \rfloor$ and $\lfloor \textbf{bind}^{\ell}\, x = a \textbf{ in } b \rfloor = \lfloor b \rfloor \{\lfloor a \rfloor / x\}$.

Now that we have seen the type system and equational theory of DCC, we explore how DCC relates to GMC. There are two specific questions that we can ask in this regard.

- Is DCC a graded monadic calculus? In other words, with appropriate restrictions, can we translate GMC to DCC while preserving meaning?
- Is DCC just a graded monadic calculus? In other words, with appropriate restrictions, can we translate DCC to GMC while preserving meaning?

We shall see that the answer to the first question is yes, while the answer to the second one is no.

2.3.2 Is DCC a Graded Monadic Calculus?

Both DCC and GMC are calculi parametrized by algebraic structures. So, to compare the two calculi, we first need to relate the parametrizing structures. DCC is parametrized by an arbitrary lattice $\mathcal{L}$ whereas GMC is parametrized by an arbitrary preordered monoid $\mathcal{M}$. A preordered monoid is a more general structure because any bounded semilattice may be seen as a preordered monoid (but not vice-versa). For example, a bounded join-semilattice is a preordered monoid with multiplication, unit and the preorder given by join, $\bot$ and the semilattice order respectively. A point to note here is that in the original formulation of Denning [1976], the semantics of secure information flow just constrains the model to be a bounded join-semilattice. However, under the practical assumption of finiteness, the model collapses to a lattice. Here, we shall compare DCC and GMC over the class of bounded join-semilattices. (A technical point to note is that a finite bounded join-semilattice is also a lattice but an infinite bounded join-semilattice may not be so. Now, DCC, as presented by Abadi et al. [1999], is parametrized by lattices; but DCC can as well be parametrized by the broader class of bounded join-semilattices because the calculus does not make use of the meet operation in its syntax/semantics.)

Now, let $\mathcal{L} = (L, \sqcup, \bot)$ be a bounded join-semilattice. Then, we can translate the types and terms of GMC($\mathcal{L}$) to their counterparts in DCC($\mathcal{L}$) as shown in Figure 2.9. This translation preserves typing and meaning. Below, $\overline{\Gamma}$ denotes Γ with the types translated.

Theorem 2.2 Let $\mathcal{L}$ be a bounded join-semilattice. If $\Gamma \vdash a : A$ in GMC($\mathcal{L}$), then $\overline{\Gamma} \vdash \overline{a} : \overline{A}$ in DCC($\mathcal{L}$). Further, if $\Gamma \vdash a_1 : A$ and $\Gamma \vdash a_2 : A$ such that $a_1 \equiv a_2$ in GMC($\mathcal{L}$), then $\overline{a_1} \simeq \overline{a_2}$ in DCC($\mathcal{L}$).

From the above theorem, we see that DCC is indeed a graded monadic calculus.

$$
\begin{aligned}
\overline{T_\ell\ A} &= T_\ell\ \overline{A} \\
\overline{\mathbf{ret}\ a} &= \mathbf{eta}^\perp\ \overline{a} \\
\overline{\mathbf{lift}^\ell f} &= \lambda x : T_\ell\ \overline{A}.\mathbf{bind}^\ell\ y = x\ \mathbf{in}\ \mathbf{eta}^\ell\ (\overline{f}\ y)\ \text{[Here, } f : A \to B\text{]} \\
\overline{\mathbf{join}^{\ell_1,\ell_2} a} &= \mathbf{bind}^{\ell_1}\ x = \overline{a}\ \mathbf{in}\ \mathbf{bind}^{\ell_2}\ y = x\ \mathbf{in}\ \mathbf{eta}^{\ell_1 \sqcup \ell_2}\ y \\
\overline{\mathbf{up}^{\ell_1,\ell_2} a} &= \mathbf{bind}^{\ell_1}\ x = \overline{a}\ \mathbf{in}\ \mathbf{eta}^{\ell_2}\ x
\end{aligned}
$$

FIGURE 2.9: Translation function from GMC to DCC (Excerpt)

2.3.3 Is DCC Just a Graded Monadic Calculus?

Now that (over the class of bounded join-semilattices) GMC can be translated into DCC, can we go the other way around? Let's explore. To translate DCC to GMC, we would need to translate the **bind** construct. We may attempt a translation for **bind** of DCC using **bind** of GMC. However, note that the signature of **bind** in GMC is $T_{\ell_1}\ A \to (A \to T_{\ell_2}\ B) \to T_{\ell_1 \sqcup \ell_2}\ B$ whereas that of **bind** in DCC is $T_{\ell_1}\ A \to (A \to B) \to \{\ell_1 \sqsubseteq B\} \to B$. For a successful translation, one needs to show that, if $\ell_1 \sqsubseteq B$, then there exists a function j of type $(T_{\ell_1}\ \underline{B} \to \underline{B})$ in GMC. In case such a function exists, for $a : T_{\ell_1}\ A$ and $f : A \to B$, one can get, $(j\ ((\mathbf{lift}^{\ell_1} \underline{f})\ \underline{a})) : \underline{B}$. Here, $\underline{\ }$ denotes a possible translation from DCC to GMC.

We attempt to define $j : T_{\ell_1}\ \underline{B} \to \underline{B}$ by structural recursion on the judgment, $\ell_1 \sqsubseteq B$. The interesting cases are rules PROT-MONAD and PROT-ALREADY.

- Rule PROT-MONAD. Here, we have $\ell \sqsubseteq T_{\ell'}\ B$ where $\ell \sqsubseteq \ell'$. Need to define $j : T_\ell\ T_{\ell'}\ \underline{B} \to T_{\ell'}\ \underline{B}$. But, $x : T_\ell\ T_{\ell'}\ \underline{B} \vdash \mathbf{join}^{\ell,\ell'} x : T_{\ell'}\ \underline{B}$ because $\ell \sqcup \ell' = \ell'$. So, $j = \lambda x.\mathbf{join}^{\ell,\ell'} x$.
- Rule PROT-ALREADY. Here, we have $\ell \sqsubseteq T_{\ell'}\ B$ where $\ell \sqsubseteq B$. Need to define $j : T_\ell\ T_{\ell'}\ \underline{B} \to T_{\ell'}\ \underline{B}$. Since $\ell \sqsubseteq B$, the hypothesis gives us a function $j_0 : T_\ell\ \underline{B} \to \underline{B}$. But, now we are stuck! Lifting this function can only give us: $\mathbf{lift}^{\ell'} j_0 : T_{\ell'}\ T_\ell\ \underline{B} \to T_{\ell'}\ \underline{B}$, not exactly what we need.

  Here, we could, for instance, add a non-standard flip-rule to GMC like: "from $\Gamma \vdash a : T_{\ell_1}\ T_{\ell_2}\ A$, derive $\Gamma \vdash \mathbf{flip}^{\ell_1,\ell_2} a : T_{\ell_2}\ T_{\ell_1}\ A$" and thereafter translate DCC into it. But such an exercise would defeat our whole purpose because then, GMC would no longer be a graded monadic calculus. Note that Algehed [2018] includes such a rule in his language SDCC, which is shown to be equivalent to (the terminating fragment of) DCC.

So we see that DCC is not just a graded monadic calculus. The rule PROT-ALREADY makes it something more than that. This rule enables one to flip the modal type constructors. In DCC, from $\Gamma \vdash a : T_{\ell_1}\ T_{\ell_2}\ A$, one can derive $\Gamma \vdash \mathbf{bind}^{\ell_1}\ x = a\ \mathbf{in}\ \mathbf{bind}^{\ell_2}\ y = x\ \mathbf{in}\ \mathbf{eta}^{\ell_2}\ \mathbf{eta}^{\ell_1}\ y : T_{\ell_2}\ T_{\ell_1}\ A$, using rule PROT-ALREADY. Such a derivation is not possible in a general monadic calculus.

However, if the calculus is also comonadic in addition to being monadic, such a derivation is possible. From $T_{\ell_1}\ T_{\ell_2}\ A$, using monadic join, we can get $T_{\ell_1 \sqcup \ell_2}\ A$, which is same as $T_{\ell_2 \sqcup \ell_1}\ A$, from which we can get

$T_{\ell_2}\, T_{\ell_1}\, A$, using comonadic fork. So it seems that DCC has some comonadic character to it. Then, is DCC a graded comonadic calculus? In order to address this question, we first need to build the theory of a graded comonadic calculus.

2.4 Graded Comonadic Calculus

Soon after Moggi [1991] showed that computational effects can be understood in terms of monads, Brookes and Geva [1992] showed that intensional behavior of programs, for example, the number of steps necessary for reduction, can be understood in terms of comonads. While monads model how programs affect the environment, comonads model how the environment affects programs. Comonads, with necessary extra structure, have been used by Uustalu and Vene [2008] and Petricek et al. [2013], among others, to model various notions of environment-dependent computation, like resource usage in programs, computation on streams, etc. We have already come across some of the calculi developed [Brunel et al., 2014, Ghica and Smith, 2014, Petricek et al., 2014] to provide a general account of environment-dependent computation, though our focus has been on the utility of these calculi in the narrower domain of resource usage analysis. Nevertheless, we saw how these calculi are parametrized by semiring-like structures and how their type systems make use of the semiring operations. On the semantic side, these calculi are modeled using semiring-graded comonads, possibly including additional structure [Katsumata, 2018]. But in this section, since our focus is on dependency analysis, so we shall design a graded comonadic calculus that is the dual of the graded monadic calculus presented in Section 2.2. This means that our graded comonadic calculus is parametrized over preordered monoids and therefore, as we shall see, modeled by preordered-monoid-graded comonads.

Our Graded Comonadic Calculus (GCC) is very similar to GMC. Like GMC, GCC includes the types and terms of standard λ-calculus. Akin to GMC, GCC has a graded modal type and terms related to it. The graded modal type of GCC and related terms are as follows.

$$\text{types}, A, B ::= \ldots \mid D_m\, A$$
$$\text{terms}, a, b, f, g ::= \ldots \mid \mathbf{extr}\ a \mid \mathbf{lift}^m f \mid \mathbf{fork}^{m_1,m_2}\ a \mid \mathbf{up}^{m_1,m_2}\ a$$

Now, we look at the typing rules and equational theory of GCC.

2.4.1 Type System and Equational Theory

The typing rules for terms of λ-calculus are standard. The typing rules for terms related to the graded comonadic type are shown in Figure 2.10.

The rules C-Extract, C-Fmap, and C-Fork are generalizations of the corresponding rules for the ungraded comonadic type. The rule C-Up, like rule M-Up, relaxes the grade on the modal type. If $\mathcal{M}$ is the trivial preordered monoid, then the above rules degenerate to the standard typing rules for comonads. Note that

$\boxed{\Gamma \vdash a : A}$ *(Typing rules)*

$$\frac{\Gamma \vdash a : D_1\ A}{\Gamma \vdash \mathbf{extr}\ a : A}\ \textsc{C-Extract} \qquad \frac{\Gamma \vdash f : A \to B}{\Gamma \vdash \mathbf{lift}^m f : D_m\ A \to D_m\ B}\ \textsc{C-Fmap} \qquad \frac{\Gamma \vdash a : D_{m_1 \cdot m_2}\ A}{\Gamma \vdash \mathbf{fork}^{m_1,m_2} a : D_{m_1}\ D_{m_2}\ A}\ \textsc{C-Fork}$$

$$\frac{\Gamma \vdash a : D_{m_1}\ A \qquad m_1 <: m_2}{\Gamma \vdash \mathbf{up}^{m_1,m_2} a : D_{m_2}\ A}\ \textsc{C-Up}$$

Figure 2.10: Typing rules of GCC (Excerpt)

$$\mathbf{lift}^m(\lambda x.x) \equiv \lambda x.x \tag{2.10}$$
$$\mathbf{lift}^m(\lambda x.g\ (f\ x)) \equiv \lambda x.(\mathbf{lift}^m g)\ ((\mathbf{lift}^m f)\ x) \tag{2.11}$$
$$\mathbf{up}^{m_1,m_1} a \equiv a \tag{2.12}$$
$$\mathbf{up}^{m_2,m_3}(\mathbf{up}^{m_1,m_2} a) \equiv \mathbf{up}^{m_1,m_3} a \tag{2.13}$$
$$f\ {}^{m_2}{\lll}^{m_1'}\ (\mathbf{up}^{m_1 \cdot m_2, m_1' \cdot m_2} a) \equiv \mathbf{up}^{m_1,m_1'}(f\ {}^{m_2}{\lll}^{m_1}\ a) \tag{2.14}$$
$$f\ {}^{m_2'}{\lll}^{m_1}\ (\mathbf{up}^{m_1 \cdot m_2, m_1 \cdot m_2'} a) \equiv (\lambda x.f\ (\mathbf{up}^{m_2,m_2'} x))\ {}^{m_2}{\lll}^{m_1}\ a \tag{2.15}$$
$$\mathbf{extr}\ (f\ {}^{m_2}{\lll}^{1}\ a) \equiv f\ a \tag{2.16}$$
$$(\lambda x.\mathbf{extr}\ x)\ {}^{1}{\lll}^{m_2}\ a \equiv a \tag{2.17}$$
$$g\ {}^{m_2}{\lll}^{m_1}\ (f\ {}^{m_3}{\lll}^{m_1 \cdot m_2}\ a) \equiv (\lambda x.g\ (f\ {}^{m_3}{\lll}^{m_2}\ x))\ {}^{m_2 \cdot m_3}{\lll}^{m_1}\ a \tag{2.18}$$

Figure 2.11: Equality rules of GCC (Excerpt)

rules C-Fmap and C-Up are essentially the same as rules M-Fmap and M-Up respectively whereas rules C-Extract and C-Fork are 'converses' of rules M-Return and M-Join respectively.

The rules that generate the equational theory of GCC appear in Figure 2.11. (We omit the $\beta\eta$-rules of standard λ-calculus.) To present the rules, we use the shorthand notation: $f\ {}^{m_2}{\lll}^{m_1}\ a \triangleq (\mathbf{lift}^{m_1} f)(\mathbf{fork}^{m_1,m_2} a)$, where $a : D_{m_1 \cdot m_2}\ A$ and $f : D_{m_2}\ A \to B$. Note that ${}^{m_2}{\lll}^{m_1}$ is a graded-**extend** operator.

The first four rules are same as their counterparts in GMC. The next two rules correspond to commutativity of **extend** with **up**. The two rules after that correspond to **extr** being the left and right identity of **extend**. The last rule corresponds to associativity of **extend**.

Next, we want to interpret the calculus in a suitable category. Similar to GMC, the types of standard λ-calculus can be interpreted in any bicartesian closed category. To interpret the graded comonadic type, we need a strong graded comonad, the dual of a strong graded monad, which we saw earlier. Next, we briefly go over the definition of a strong graded comonad and present the categorical model thereafter.

2.4.2 Graded Comonad and Categorical Model

While a graded monad is a kind of lax monoidal functor, a graded comonad is a kind of oplax monoidal functor. An oplax monoidal functor from a monoidal category $(M, \otimes_M, 1_M)$ to a monoidal category $(N, \otimes_N, 1_N)$ is nothing but a lax monoidal functor from $(M^{\mathrm{op}}, \otimes_M, 1_M)$ to $(N^{\mathrm{op}}, \otimes_N, 1_N)$. Now, given a strict monoidal category $\mathbb{M}$, an $\mathbb{M}$-graded comonad over a category $\mathbb{C}$ is an oplax monoidal functor from $\mathbb{M}$ to $\mathbf{End}_{\mathbb{C}}$. As with GMC, to interpret GCC, we need to add strength to graded comonads. We follow the same strategy as before and define a strong $\mathbb{M}$-graded comonad over a cartesian category $\mathbb{C}$ as an oplax monoidal functor from $\mathbb{M}$ to $\mathbf{End}^s_{\mathbb{C}}$. We use strong $\mathbf{C}(\mathcal{M})$-graded comonads to build categorical models of GCC($\mathcal{M}$).

Let $\mathcal{M} = (M, \cdot, 1, <:)$ be the preordered monoid parametrizing the calculus. Let $\mathbb{C}$ be any bicartesian closed category. Let $(\mathbf{D}, \delta, \epsilon)$ be a strong $\mathbf{C}(\mathcal{M})$-graded comonad over $\mathbb{C}$. Then, the interpretation, $[\![_]\!]$, of types and terms of GCC is as given in Figure 2.12.

$$
\begin{aligned}
[\![D_m\ A]\!] &= \mathbf{D}_m[\![A]\!] \\
[\![\mathbf{extr}\ a]\!] &= [\![\Gamma]\!] \xrightarrow{[\![a]\!]} \mathbf{D}_1[\![A]\!] \xrightarrow{\epsilon_{[\![A]\!]}} [\![A]\!] \\
[\![\mathbf{lift}^m f]\!] &= \Lambda\Big([\![\Gamma]\!] \times \mathbf{D}_m[\![A]\!] \xrightarrow{t^{\mathbb{D}_m}_{[\![\Gamma]\!],[\![A]\!]}} \mathbf{D}_m([\![\Gamma]\!] \times [\![A]\!]) \xrightarrow{\mathbb{D}_m(\Lambda^{-1}[\![f]\!])} \mathbf{D}_m[\![B]\!]\Big) \\
[\![\mathbf{fork}^{m_1,m_2}\, a]\!] &= [\![\Gamma]\!] \xrightarrow{[\![a]\!]} \mathbf{D}_{m_1\cdot m_2}[\![A]\!] \xrightarrow{\delta^{m_1,m_2}_{[\![A]\!]}} \mathbf{D}_{m_1}\mathbf{D}_{m_2}[\![A]\!] \\
[\![\mathbf{up}^{m_1,m_2}\, a]\!] &= [\![\Gamma]\!] \xrightarrow{[\![a]\!]} \mathbf{D}_{m_1}[\![A]\!] \xrightarrow{\mathbf{D}^{m_1 <: m_2}_{[\![A]\!]}} \mathbf{D}_{m_2}[\![A]\!]
\end{aligned}
$$

Figure 2.12: Interpretation of GCC (Excerpt)

Note that ϵ is a natural transformation from $\mathbf{D}_1$ to $\mathbf{Id}$ and δ^{m_1,m_2} are morphisms from $\mathbf{D}_{m_1\cdot m_2}$ to $\mathbf{D}_{m_1} \circ \mathbf{D}_{m_2}$, natural in both m_1 and m_2. By reasoning along the lines of Theorem 2.1, we can show that the above interpretation provides a sound model of GCC.

Theorem 2.3 If $\Gamma \vdash a : A$ in GCC, then $[\![a]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], [\![A]\!])$. Further, if $\Gamma \vdash a_1 : A$ and $\Gamma \vdash a_2 : A$ such that $a_1 \equiv a_2$ in GCC, then $[\![a_1]\!] = [\![a_2]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], [\![A]\!])$.

Now that we have a graded comonadic calculus with us, we put the comonadic character of DCC to the test. In particular, we test: with appropriate restrictions, can one translate GCC into DCC?

2.5 DCC and GCC

In Section 2.3.3, we saw that DCC is not just a graded monadic calculus. The rule Prot-Already lends a comonadic character to it. But does it make DCC a graded comonadic calculus? In other words, over the class of bounded join-semilattices, can we translate GCC into DCC? We can translate $D_\ell\ A$ to $T_\ell\ A$. The

constructs **lift** and **up** can be translated as in Figure 2.9. But to translate **extr** and **fork**, we need to revisit the protection judgment.

2.5.1 Protection Judgment, Revisited

To translate **extr** and **fork**, we need to be able to construct functions having types $T_\bot\, A \to A$ and $T_{\ell_1 \sqcup \ell_2}\, A \to T_{\ell_1}\, T_{\ell_2}\, A$ respectively, for an arbitrary type A. However, given the formulation of DCC by Abadi et al. [1999], such a construction is not possible. This is so because in order to construct a function having type $T_\bot\, A \to A$, we need to show that: $\bot \sqsubseteq A$, for an arbitrary A. The protection rules do not allow such a derivation. Similarly, in order to construct a function having type $T_{\ell_1 \sqcup \ell_2}\, A \to T_{\ell_1}\, T_{\ell_2}\, A$, we need to show that: $\ell_1 \sqcup \ell_2 \sqsubseteq T_{\ell_1}\, T_{\ell_2}\, A$, for an arbitrary A. Again, such a derivation is not allowed by the protection rules.

However, from a dependency perspective, $\bot \sqsubseteq A$ and $\ell_1 \sqcup \ell_2 \sqsubseteq T_{\ell_1}\, T_{\ell_2}\, A$ are sound judgments. The judgment $\bot \sqsubseteq A$ is sound because: $\bot$ is the lowest security level and as such, the terms of any type are at least as secure as $\bot$. The judgment $\ell_1 \sqcup \ell_2 \sqsubseteq T_{\ell_1}\, T_{\ell_2}\, A$ is sound because: $T_{\ell_1}\, T_{\ell_2}\, A$ is at least as secure as ℓ_1 and $T_{\ell_1}\, T_{\ell_2}\, A$ is also at least as secure as ℓ_2, so it must be at least as secure as $\ell_1 \sqcup \ell_2$. Note that this reasoning about security of levels is also supported by the lattice model of Denning [1976].

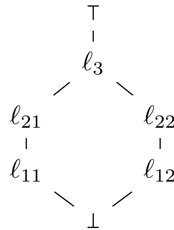
Now, the above judgments are not only sound, but also desirable. Compared to A, the type $T_\bot\, A$ offers no extra protection. So, it makes sense to allow programs like the one shown below.

$$x : T_\bot\, A \vdash \mathbf{bind}^\bot\ y = x\ \mathbf{in}\ y : A$$

Next, the type $T_{\ell_1}\, T_{\ell_2}\, A$ offers no less protection than the type $T_{\ell_1 \sqcup \ell_2}\, A$. So, programs like the one below should also be allowed.

$$x : T_{\ell_1 \sqcup \ell_2}\, A \vdash \mathbf{bind}^{\ell_1 \sqcup \ell_2}\ y = x\ \mathbf{in}\ \mathbf{eta}^{\ell_1}\ \mathbf{eta}^{\ell_2}\ y : T_{\ell_1}\, T_{\ell_2}\, A$$

Allowing programs like the one above has some interesting consequences. For example, consider the lattice shown below.



In this lattice, $\ell_{11} \sqcup \ell_{12} = \ell_3$. Now, putting $\ell_1 := \ell_{11}$ and $\ell_2 := \ell_{12}$ in the above program, we have:

$$x : T_{\ell_3}\, A \vdash \mathbf{bind}^{\ell_3}\ y = x\ \mathbf{in}\ \mathbf{eta}^{\ell_{11}}\ \mathbf{eta}^{\ell_{12}}\ y : T_{\ell_{11}}\, T_{\ell_{12}}\, A.$$

This program shows that we can observe ℓ_3-level values in an environment simultaneously protected by ℓ_{11} and ℓ_{12}. Two points are worth noting here.

- First, ℓ_{11} and ℓ_{12} together offer much more protection than either of them individually. Observe that neither ℓ_{11} nor ℓ_{12} individually offer as much protection as either ℓ_{21} or ℓ_{22}. But, ℓ_{11} and ℓ_{12} together offer more protection than both ℓ_{21} and ℓ_{22}, considered individually. Behind this observation, lies a fundamental security principle, the principle that forms the basis of applications like two-factor authentication, two-man rule, etc. It may be phrased in terms of the age-old proverb: the whole is more than just the sum of its parts.
- Second, ℓ_3 is compromised if ℓ_{11} and ℓ_{12} are simultaneously compromised. This is so because with a simultaneous access to ℓ_{11} and ℓ_{12}, one has access to ℓ_3 and all levels below it, even ℓ_{21} and ℓ_{22}. It may look counter-intuitive but that just shows the power of simultaneous access.

Now, to enable such reasoning within DCC, we add the following two rules to the protection judgment of the calculus.

$\boxed{\ell \sqsubseteq A}$ *(Extended protection rules)*

$$\textsc{Prot-Minimum}\quad \frac{}{\bot \sqsubseteq A} \qquad\qquad \textsc{Prot-Combine}\quad \frac{\ell_1 \sqsubseteq A \qquad \ell_2 \sqsubseteq A \qquad \ell \sqsubseteq \ell_1 \sqcup \ell_2}{\ell \sqsubseteq A}$$

These extra rules enable us to construct functions of type $T_\bot\, A \to A$ and $T_{\ell_1 \sqcup \ell_2}\, A \to T_{\ell_1}\, T_{\ell_2}\, A$, for an arbitrary A. As a side note, we want to point out that in the categorical model of DCC given by Abadi et al. [1999], for any A, the interpretations of $T_\bot\ A$ and $T_{\ell_1 \sqcup \ell_2}\ A$ are the same as those of A and $T_{\ell_1}\, T_{\ell_2}\ A$ respectively. As such, one can show that DCC extended with the above rules enjoys the same categorical model given by Abadi et al. [1999].

For the sake of precision, we shall refer to DCC extended with these rules as DCC_e. The equational theory of DCC_e is defined in the same way as that of DCC. Then, DCC is a proper sub-language of DCC_e. Now, since DCC is a graded monadic calculus, so is DCC_e. However, owing to the reasons described above, DCC is not a graded comonadic calculus. But DCC_e is a graded comonadic calculus, as we see next.

2.5.2 DCC_e is a Graded Comonadic Calculus

Over the class of bounded join-semilattices, we can translate GCC into DCC_e. Let $\mathcal{L} = (L, \sqcup, \bot)$ be a bounded join-semilattice. Then, we can translate the types and terms of $\text{GCC}(\mathcal{L})$ to their counterparts in $\text{DCC}(\mathcal{L})$ as shown in Figure 2.13. Note the role played by rules Prot-Minimum and Prot-Combine in the translation

$$\overline{D_\ell\, A} = T_\ell\, \overline{A}$$
$$\overline{\textbf{extr}\ a} = \textbf{bind}^{\bot}\ x = \overline{a}\ \textbf{in}\ x$$
$$\overline{\textbf{lift}^{\ell} f} = \lambda x : T_\ell\, \overline{A}.\textbf{bind}^{\ell}\ y = x\ \textbf{in}\ \textbf{eta}^{\ell}\ (\overline{f}\ y)\ [\text{Here, } f : A \to B]$$
$$\overline{\textbf{fork}^{\ell_1,\ell_2}\, a} = \textbf{bind}^{\ell_1 \sqcup \ell_2}\ x = \overline{a}\ \textbf{in}\ \textbf{eta}^{\ell_1}\ \textbf{eta}^{\ell_2}\ x$$
$$\overline{\textbf{up}^{\ell_1,\ell_2}\, a} = \textbf{bind}^{\ell_1}\ x = \overline{a}\ \textbf{in}\ \textbf{eta}^{\ell_2}\ x$$

FIGURE 2.13: Translation function from GCC to DCC_e (Excerpt)

of **extr** a and $\textbf{fork}^{\ell_1,\ell_2}\, a$ respectively. The theorem below shows that this translation preserves typing and meaning.

Theorem 2.4 If $\Gamma \vdash a : A$ in GCC($\mathcal{L}$), then $\overline{\Gamma} \vdash \overline{a} : \overline{A}$ in $\text{DCC}_e(\mathcal{L})$. Further, if $\Gamma \vdash a_1 : A$ and $\Gamma \vdash a_2 : A$ such that $a_1 \equiv a_2$ in GCC($\mathcal{L}$), then $\overline{a_1} \simeq \overline{a_2}$ in $\text{DCC}_e(\mathcal{L})$.

To summarize, earlier we showed that DCC is a graded monadic calculus by translating GMC into it. However, we couldn't translate DCC into GMC because DCC has some comonadic character to it. Thereafter, we designed a graded comonadic calculus, GCC. Using GCC, we put the comonadic character of DCC to the test. We could not translate GCC to DCC because DCC is not fully comonadic. Next, we found that such a translation is not possible only because DCC rules out certain derivations that are both sound and desirable. We extended DCC to DCC_e to allow these derivations and found that we can translate GCC into DCC_e. So, both GMC and GCC can be translated into DCC_e. Now, can we go the other way around and translate DCC_e into a calculus built up using just GMC and GCC? To answer the question, we first need to flesh out this calculus.

2.6 Graded Monadic Comonadic Calculus

2.6.1 The Calculus

The Graded Monadic Comonadic Calculus (GMCC) combines the Graded Monadic Calculus (GMC) and the Graded Comonadic Calculus (GCC) into a single system. We can view it as an extension of the standard simply-typed λ-calculus with a graded modal type constructor S_m, which behaves both like a graded monadic type constructor, T_m, and a graded comonadic type constructor, D_m. The calculus has as terms the union of those of GMC and GCC. The typing rules of the calculus include the rules of GMC and GCC (shown in Figures 2.2 and 2.10 respectively), with S_m replacing T_m and D_m.

The equational theory of the calculus is generated by the equational theories of GMC and GCC (presented in Figures 2.3 and 2.11 respectively) along with the additional rules shown in Figure 2.14. These additional rules ensure that the monadic and the comonadic fragments of the calculus behave well with respect to each other. Next, we look at the categorical models of GMCC.

$$\mathbf{extr}\,(\mathbf{ret}\;a) \equiv a \tag{2.19}$$

$$\mathbf{ret}\,(\mathbf{extr}\;a) \equiv a \tag{2.20}$$

$$\mathbf{fork}^{m_1,m_2}(\mathbf{join}^{m_1,m_2}\,a) \equiv a \tag{2.21}$$

$$\mathbf{join}^{m_1,m_2}(\mathbf{fork}^{m_1,m_2}\,a) \equiv a \tag{2.22}$$

FIGURE 2.14: Additional equality rules of GMCC

2.6.2 Categorical Model

We interpret the graded modal type constructor of GMCC as a kind of strong monoidal functor [MacLane, 1971]. A strong monoidal functor from a monoidal category $(M, \otimes_M, 1_M)$ to a monoidal category $(N, \otimes_N, 1_N)$ is a lax monoidal functor $(F, F_2, F_0) : (M, \otimes_M, 1_M) \to (N, \otimes_N, 1_N)$ where F_0 and $F_2(X,Y)$ are invertible, for all $X, Y \in \mathrm{Obj}(M)$. From this definition, one can show that a strong monoidal functor is not only lax but also oplax. Now, for a strong monoidal functor $S : (M, \otimes_M, 1_M) \to (N, \otimes_N, 1_N)$, we have: $S(1_M) \cong 1_N$ and $S(X \otimes_M Y) \cong S(X) \otimes_N S(Y)$, for all $X, Y \in \mathrm{Obj}(M)$. If these isomorphisms are identities, then the functor is said to be strict. We shall use strict monoidal functors extensively later in this chapter. Before moving further, to avoid potential confusion, note that the word 'strong' in 'strong monoidal functor' and in 'strong endofunctor' refers to different properties.

Let $\mathcal{M} = (M, \cdot, 1, <:)$ be the preordered monoid parametrizing the calculus. Let $\mathbb{C}$ be any bicartesian closed category. Let $\mathbf{S}$ be a strong monoidal functor from $\mathbf{C}(\mathcal{M})$ to $\mathbf{End}^s_{\mathbb{C}}$. Then, $\mathbf{S}$ is both a strong $\mathbf{C}(\mathcal{M})$-graded monad over $\mathbb{C}$ and a strong $\mathbf{C}(\mathcal{M})$-graded comonad over $\mathbb{C}$. With respect to $\mathbf{S}$, let μ, η, δ and ϵ denote the corresponding natural transformations. Then,

$$\epsilon \circ \eta = \mathrm{id} \qquad \eta \circ \epsilon = \mathrm{id}$$

$$\delta \circ \mu = \mathrm{id} \qquad \mu \circ \delta = \mathrm{id}$$

Now, the interpretation of GMCC using category $\mathbb{C}$ and functor $\mathbf{S}$, denoted $[\![_]\!]$, or more precisely $[\![_]\!]_{(\mathbb{C},\mathbf{S})}$, is as follows. The type $S_m\,A$ is interpreted as: $[\![S_m\,A]\!] = \mathbf{S}_m[\![A]\!]$. The terms are interpreted as in Figures 2.7 and 2.12.

This interpretation of the calculus is sound, as we see below.

Theorem 2.5 If $\Gamma \vdash a : A$ in GMCC, then $[\![a]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], [\![A]\!])$. Further, if $\Gamma \vdash a_1 : A$ and $\Gamma \vdash a_2 : A$ such that $a_1 \equiv a_2$ in GMCC, then $[\![a_1]\!] = [\![a_2]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], [\![A]\!])$.

The theorem above shows that given a preordered monoid $\mathcal{M}$, any bicartesian closed category $\mathbb{C}$ together with a strong monoidal functor $\mathbf{S}$ from $\mathbf{C}(\mathcal{M})$ to $\mathbf{End}^s_{\mathbb{C}}$ provides a sound model of GMCC($\mathcal{M}$). Now, in addition to soundness, GMCC($\mathcal{M}$) also enjoys completeness with respect to its class of categorical models. Formally, we can show:

Theorem 2.6 Given any preordered monoid $\mathcal{M}$, for typing derivations $\Gamma \vdash a_1 : A$ and $\Gamma \vdash a_2 : A$ in GMCC($\mathcal{M}$), if $[\![a_1]\!] = [\![a_2]\!]$ in all models of GMCC($\mathcal{M}$), then $a_1 \equiv a_2$ is derivable in GMCC($\mathcal{M}$).

We use entirely standard term-model techniques [Jacobs, 1999] to prove this completeness theorem. Briefly put, we first construct the classifying category and thereafter, the generic model in the classifying category. The generic model equates only the terms that are equal in the calculus. So, if the interpretations of two GMCC($\mathcal{M}$)-terms are equal in all models, and therefore in the generic model too, then these terms are equal in the calculus as well. In addition to proving completeness, we show that:

Theorem 2.7 The generic model satisfies the universal property.

The above theorem implies that any model of GMCC($\mathcal{M}$) can be factored through the generic model. This is a nice result but we don't explore its consequences here, we leave it for future work.

In this section, we have seen that GMCC is sound and complete with respect to its class of categorical models. In the next section, we shall explore the relation between GMCC and DCC$_e$.

2.7 GMCC and DCC$_e$

We saw that over the class of bounded join-semilattices, we can translate both GMC and GCC into DCC$_e$. In fact, over the same class of structures, we can go further and translate the combined calculus GMCC into DCC$_e$, following the translations presented in Figures 2.9 and 2.13.

Theorem 2.8 If $\Gamma \vdash a : A$ in GMCC($\mathcal{L}$), then $\overline{\Gamma} \vdash \overline{a} : \overline{A}$ in DCC$_e$($\mathcal{L}$). Further, if $\Gamma \vdash a_1 : A$ and $\Gamma \vdash a_2 : A$ such that $a_1 \equiv a_2$ in GMCC($\mathcal{L}$), then $\overline{a_1} \simeq \overline{a_2}$ in DCC$_e$($\mathcal{L}$).

We now go the other way around and translate DCC$_e$ into GMCC, over the class of bounded join-semilattices.

Note here that DCC$_e$ has a very liberal definition of equality, inherited from DCC. Two DCC$_e$-terms are equal if, after erasure, they are equal as λ-terms. So, $\mathbf{eta}^{\ell}\, a \simeq a$ for any DCC$_e$-term a. The same is not true in general in GMCC. For example, $\mathbf{up}^{\bot,\ell}(\mathbf{ret}\, a)$ may not be equal to a. To capture the notion of DCC$_e$-equality in GMCC, we need to define a similar relation between GMCC terms. This relation is defined as follows. For GMCC terms a_1 and a_2, we say $a_1 \simeq a_2$ if and only if $\lfloor a_1 \rfloor$ and $\lfloor a_2 \rfloor$, the plain λ-term counterparts of a_1 and a_2 respectively, are equal as λ-terms. The erasure operation $\lfloor\ \rfloor$ strips away the constructors **ret**, **join**, **extr**, **fork**, **lift** and **up** along with the grade annotations. For example, $\lfloor \mathbf{join}^{\ell_1,\ell_2}\, a \rfloor = \lfloor a \rfloor$.

Coming back to the translation from DCC$_e$ to GMCC, note that it is straightforward for types. The modal type $T_\ell\, B$ gets translated to $S_\ell\, \underline{B}$, where $\underline{\ }$ is the translation function on types and terms. For translating terms, we use the following lemma. Note that in Section 2.3.3, while trying to translate DCC to GMC, we could not prove a lemma like this one.

Lemma 2.9 If $\ell \sqsubseteq B$ in DCC$_e$, then there exists a term $\varnothing \vdash j : S_\ell\, \underline{B} \to \underline{B}$ in GMCC such that $\lfloor j \rfloor \equiv \lambda x.x$.

Now, with the above lemma, we can translate DCC_e terms to GMCC terms, as shown below. The function j used in the translation is as given by Lemma 2.9.

$$\underline{\mathbf{eta}^\ell\ a} = \mathbf{up}^{\perp,\ell}(\mathbf{ret}\ \underline{a}) \qquad \underline{\mathbf{bind}^\ell\ x = a\ \mathbf{in}\ b} = j\ ((\mathbf{lift}^\ell(\lambda x.\underline{b}))\ \underline{a})$$

This translation preserves typing and meaning. Below, $\underline{\Gamma}$ denotes Γ with the types translated.

Theorem 2.10 If $\Gamma \vdash a : A$ in $\text{DCC}_e(\mathcal{L})$, then $\underline{\Gamma} \vdash \underline{a} : \underline{A}$ in GMCC($\mathcal{L}$). Further, if $\Gamma \vdash a_1 : A$ and $\Gamma \vdash a_2 : A$ such that $a_1 \simeq a_2$ in $\text{DCC}_e(\mathcal{L})$, then $\underline{a_1} \simeq \underline{a_2}$ in GMCC($\mathcal{L}$).

Theorems 2.10 and 2.8 together show that over the class of bounded join-semilattices, GMCC and DCC_e, seen as dependency calculi, are equivalent. Therefore, like DCC_e, GMCC is also a generalization of the Dependency Core Calculus, DCC. Hence, dependency analysis, at least to the extent DCC is capable of, can be done using just a graded monadic comonadic calculus. This connection between dependency analysis and GMCC is important because:

- Dependency analysis can now benefit from a wider variety of categorical models. Some of these models provide simpler proofs of correctness of dependency analysis. For example, employing the categorical models we develop in Sections 2.8.1 and 2.9.2, we shall show, in a straightforward manner, that dependency analyses in DCC_e and λ° are correct.
- More dependency calculi can now be unified under a common framework. As a proof of concept, we show that the binding-time calculus, λ° of Davies [2017], can be encoded into GMCC_e, an extension of GMCC. Note that λ° can not be translated into DCC [Abadi et al., 1999].
- The non-standard **bind**-rule of DCC can be replaced with standard monadic and comonadic typing rules. This finding provides insight into the categorical basis of the **bind**-rule of DCC. (See Theorem 2.17)
- GMCC is formed by combining GMC and its dual, GCC. GMC may be seen as a restriction of the Explicit Subeffecting Calculus of Katsumata [2014]. The Explicit Subeffecting Calculus is a general system for analyzing effects. This clean connection between dependency analysis and effect analysis promises to be a fertile ground for new ideas, especially in the intersection of dependency, effect and coeffect analyses.

Before closing this section, we want to add some remarks on the equivalence between GMCC and DCC_e. From theorems 2.10 and 2.8, we see that (over the class of bounded join-semilattices) GMCC and DCC_e are equivalent upto erasure. We may ask: can this equivalence be made stronger? The answer is yes. In the next section, we shall show that (over the class of bounded join-semilattices) GMCC and DCC_e are semantically equivalent too. Further, on the syntactic side, we can show something stronger as well:

Theorem 2.11 Let $\Gamma \vdash a : A$ be any derivation in GMCC($\mathcal{L}$). Then, $\overline{\underline{a}} \equiv a$.

The above theorem says that any GMCC($\mathcal{L}$)-term, after a round trip to $\text{DCC}_e(\mathcal{L})$, is equal (not just equal upto erasure) to itself. This result is interesting because it shows that GMCC($\mathcal{L}$) may be embedded into

$DCC_e(\mathcal{L})$. Going the other way around, we can prove the following weaker result: if $\Gamma \vdash a : A$ is a derivation in $DCC_e(\mathcal{L})$, then $\underline{\overline{a}} \simeq a$. We are forced to use equality upto erasure here because we don't have an alternative equational theory for DCC_e.

Next, we develop the categorical semantics for DCC_e from first principles.

2.8 DCC_e: Categorical Semantics

Abadi et al. [1999] present a categorical model for DCC and prove noninterference using that model. Several other authors [Algehed and Bernardy, 2019, Bowman and Ahmed, 2015, Shikuma and Igarashi, 2006, Tse and Zdancewic, 2004] have presented alternative proofs of noninterference for DCC using various techniques, including parametricity. In this section, we present a class of categorical models for DCC_e, in the style of GMCC, and show noninterference for the calculus using these models. We also establish semantic equivalence between DCC_e and GMCC and explain the non-standard **bind**-rule of DCC in terms of standard category-theoretic concepts.

2.8.1 Categorical Models for DCC_e

Given that GMCC and DCC_e, considered as dependency calculi, are equivalent, we can simply use models of GMCC to interpret DCC_e. However, in this section, we shall build models for DCC_e from first principles and show them to be computationally adequate with respect to a call-by-value semantics for the calculus. The original operational semantics of DCC, presented by Abadi et al. [1999], is somewhat ad hoc from a categorical perspective because according to this semantics, $\mathbf{eta}^\ell\ a$ reduces to a. If we interpret **eta** as the unit of a monad, this reduction would require us to interpret that unit as the identity natural transformation, something that is not very general. On the other hand, a call-by-value semantics or a call-by-name semantics of the calculus [Tse and Zdancewic, 2004] does not impose such requirements and is quite general from a category-theoretic perspective. According to a call-by-value semantics, **bind**-expressions step as follows. Below, v denotes a value expression.

$\boxed{\vdash a \rightsquigarrow a'}$ *(Operational semantics (Excerpt))*

$$\textsc{CBV-BindLeft}\quad \frac{\vdash a \rightsquigarrow a'}{\vdash \mathbf{bind}^\ell\ x = a\ \mathbf{in}\ b \rightsquigarrow \mathbf{bind}^\ell\ x = a'\ \mathbf{in}\ b}$$

$$\textsc{CBV-BindBeta}\quad \frac{}{\vdash \mathbf{bind}^\ell\ x = \mathbf{eta}^\ell\ v\ \mathbf{in}\ b \rightsquigarrow b\{v/x\}}$$

Given $\rightsquigarrow$, we can define the multistep reduction relation, $\rightsquigarrow^*$, in the usual way. Note here that though we use a call-by-value semantics for DCC_e, we could have used a call-by-name semantics as well. DCC_e being a terminating calculus, the choice of one evaluation strategy over the other does not lead to significant

differences in metatheory. We chose call-by-value over call-by-name because the former allows reductions underneath the **eta**s, thereby making it easier to state the noninterference theorem (see Theorem 2.20, second bullet).

Given a bounded join-semilattice, $\mathcal{L} = (L, \sqcup, \bot)$, and any bicartesian closed category $\mathbb{C}$, we interpret the graded modal type constructor T of $\text{DCC}_e(\mathcal{L})$ as a strong monoidal functor $\mathbf{S}$ from $\mathbf{C}(\mathcal{L})$ to $\mathbf{End}^{\mathrm{s}}_{\mathbb{C}}$. Formally, $[\![T_\ell\ A]\!] = \mathbf{S}_\ell[\![A]\!]$. Interestingly, since $\sqcup$ is idempotent, the triple $(\mathbf{S}_\ell, \mathbf{S}^{\bot\sqsubseteq\ell} \circ \eta, \mu^{\ell,\ell})$ is a monad for any $\ell \in L$. Further, since $\mu^{\ell,\ell}$ is invertible, such a monad is idempotent [Johnstone, 2002].

For interpreting terms of $\text{DCC}_e(\mathcal{L})$, we need the following lemma.

Lemma 2.12 If $\ell \sqsubseteq A$, then $\exists$ an isomorphism $k^\ell_{[\![A]\!]} : \mathbf{S}_\ell[\![A]\!] \to [\![A]\!]$ such that $(k^\ell_{[\![A]\!]})^{-1} = \mathbf{S}^{\bot\sqsubseteq\ell}_{[\![A]\!]} \circ \eta_{[\![A]\!]}$.

Using the above lemma, we interpret terms as:

$$[\![\mathbf{eta}^\ell\ a]\!] = [\![\Gamma]\!] \xrightarrow{[\![a]\!]} [\![A]\!] \xrightarrow{\eta_{[\![A]\!]}} \mathbf{S}_\bot[\![A]\!] \xrightarrow{\mathbf{S}^{\bot\sqsubseteq\ell}_{[\![A]\!]}} \mathbf{S}_\ell[\![A]\!]$$

$$[\![\mathbf{bind}^\ell\ x = a\ \mathbf{in}\ b]\!] = [\![\Gamma]\!] \xrightarrow{\langle \mathrm{id}_{[\![\Gamma]\!]}, [\![a]\!]\rangle} [\![\Gamma]\!] \times \mathbf{S}_\ell[\![A]\!] \xrightarrow{t^{\mathbf{S}_\ell}_{[\![\Gamma]\!],[\![A]\!]}} \mathbf{S}_\ell([\![\Gamma]\!] \times [\![A]\!]) \xrightarrow{\mathbf{S}_\ell[\![b]\!]} \mathbf{S}_\ell[\![B]\!] \xrightarrow{k^\ell_{[\![B]\!]}} [\![B]\!]$$

For later reference, note that whenever we need to be precise, we use $[\![_]\!]_{(\mathbb{C},\mathbf{S})}$ to refer to the interpretation of $\text{DCC}_e(\mathcal{L})$ in category $\mathbb{C}$ using the strong monoidal functor, $\mathbf{S}$. The above interpretation of $\text{DCC}_e(\mathcal{L})$ is sound, as we see next.

Theorem 2.13 If $\Gamma \vdash a : A$ in DCC_e, then $[\![a]\!] \in \text{Hom}_{\mathbb{C}}([\![\Gamma]\!], [\![A]\!])$.

Theorem 2.14 If $\Gamma \vdash a : A$ in DCC_e and $\vdash a \leadsto a'$, then $[\![a]\!] = [\![a']\!]$.

Computational adequacy with respect to call-by-value semantics follows as a corollary of the above theorems. In the computational adequacy theorem presented below, we assume that the categorical interpretation is injective for ground types. In particular, $[\![\mathbf{true}]\!] \neq [\![\mathbf{false}]\!]$, where $\mathbf{true} \triangleq \mathbf{inj}_1\ \mathbf{unit} : \mathbf{Bool}$ and $\mathbf{false} \triangleq \mathbf{inj}_2\ \mathbf{unit} : \mathbf{Bool}$, and $\mathbf{Bool} \triangleq \mathbf{Unit} + \mathbf{Unit}$.

Theorem 2.15 Let the interpretation $[\![_]\!]_{(\mathbb{C},\mathbf{S})}$ be injective for ground types.

- Let $\Gamma \vdash b : \mathbf{Bool}$ and v be a value of type $\mathbf{Bool}$. If $[\![b]\!]_{(\mathbb{C},\mathbf{S})} = [\![v]\!]_{(\mathbb{C},\mathbf{S})}$, then $\vdash b \leadsto^* v$.
- Fix some $\ell \in L$. Let $\Gamma \vdash b : T_\ell\ \mathbf{Bool}$ and v be a value of type $T_\ell\ \mathbf{Bool}$. Suppose, the morphisms $\overline{\eta}_X \triangleq \mathbf{S}^{\bot\sqsubseteq\ell}_X \circ \eta_X$ are mono for any $X \in \text{Obj}(\mathbb{C})$. Now, if $[\![b]\!]_{(\mathbb{C},\mathbf{S})} = [\![v]\!]_{(\mathbb{C},\mathbf{S})}$, then $\vdash b \leadsto^* v$.

The mono requirement in the second part of the above theorem ensures that $[\![\mathbf{eta}^\ell\ \mathbf{true}]\!] \neq [\![\mathbf{eta}^\ell\ \mathbf{false}]\!]$, which is an injectivity condition necessary for the proof. The above theorem, in conjunction with theorem 2.5, shows that $\text{GMCC}(\mathcal{L})$ and $\text{DCC}_e(\mathcal{L})$ enjoy the same class of models. In fact, we can show that the two calculi are exactly equivalent over this class of models.

Theorem 2.16 Let $\mathcal{L}$ be any bounded join-semilattice, $\mathbb{C}$ be any bicartesian closed category and $\mathbf{S}$ be any strong monoidal functor from $\mathbf{C}(\mathcal{L})$ to $\mathbf{End}^{\mathrm{s}}_{\mathbb{C}}$. Now,

- If $\Gamma \vdash a : A$ in GMCC($\mathcal{L}$), then $[\![\overline{a}]\!]_{(\mathbb{C},\mathbf{S})} = [\![a]\!]_{(\mathbb{C},\mathbf{S})}$.
- If $\Gamma \vdash a : A$ in DCC$_e$($\mathcal{L}$), then $[\![\underline{a}]\!]_{(\mathbb{C},\mathbf{S})} = [\![a]\!]_{(\mathbb{C},\mathbf{S})}$.

The categorical semantics of DCC$_e$ helps in understanding the non-standard **bind**-rule of DCC. Recall that for any $\ell \in L$, the triple $(\mathbf{S}_\ell, \mathbf{S}^{\bot \sqsubseteq \ell} \circ \eta, \mu^{\ell,\ell})$ is an idempotent monad. Now, by lemma 2.12, if $\ell \sqsubseteq A$, then $\mathbf{S}_\ell[\![A]\!] \cong [\![A]\!]$, as witnessed by $k^\ell_{[\![A]\!]} : \mathbf{S}_\ell[\![A]\!] \to [\![A]\!]$ and $\mathbf{S}^{\bot \sqsubseteq \ell}_{[\![A]\!]} \circ \eta_{[\![A]\!]} : [\![A]\!] \to \mathbf{S}_\ell[\![A]\!]$. Using this lemma, we can show that if $\ell \sqsubseteq A$, then $[\![A]\!]$ is the carrier of an $\mathbf{S}_\ell$-algebra. What this means is that the protection judgment is, in essence, a syntactic mechanism for picking out the carriers of the monad algebras. This insight explains the signature of rule DCC-Bind: $T_\ell\ A \to (A \to B) \to \{\ell \sqsubseteq B\} \to B$. If $\ell \sqsubseteq B$, then $[\![B]\!]$ is the carrier of an $\mathbf{S}_\ell$-algebra and further, $[\![B]\!] \cong [\![T_\ell\ B]\!]$. As such, the return type of the rule can be B, in lieu of $T_\ell\ B$. The following theorem characterizes the protection judgment in terms of monad algebras.

Theorem 2.17 If $\ell \sqsubseteq A$ in DCC$_e$, then $([\![A]\!], k^\ell_{[\![A]\!]})$ is an $\mathbf{S}_\ell$-algebra.
Further, if $\ell \sqsubseteq A$ and $\ell \sqsubseteq B$, then for any $f \in \mathrm{Hom}_{\mathbb{C}}([\![A]\!], [\![B]\!])$, f is an $\mathbf{S}_\ell$-algebra morphism.
Hence, the full subcategory of $\mathbb{C}$ with Obj $:= \{[\![A]\!] \mid \ell \sqsubseteq A\}$ is also a full subcategory of the Eilenberg-Moore category, $\mathbb{C}^{\mathbf{S}_\ell}$.

Next, we use these models to prove noninterference for DCC$_e$.

2.8.2 Proof of Noninterference

Two functors in $\mathbf{End}^{\mathrm{s}}_{\mathbb{C}}$ are crucial to our proof of noninterference. One of them is the identity functor, **Id**. The other is the terminal functor, denoted by $*$, which is the functor that maps all objects to $\top$, the terminal object of the category, and all morphisms to $\langle\rangle$. Now, for every $\ell \in L$, we define a strong monoidal functor $\mathbf{S}^\ell$ from $\mathbf{C}(\mathcal{L})$ to $\mathbf{End}^{\mathrm{s}}_{\mathbb{C}}$ as follows.

$$\mathbf{S}^\ell(\ell') = \begin{cases} \mathbf{Id}, & \text{if } \ell' \sqsubseteq \ell \\ *, & \text{otherwise} \end{cases} \qquad \mathbf{S}^\ell(\ell_1 \sqsubseteq \ell_2) = \begin{cases} \mathrm{id}, & \text{if } \ell_2 \sqsubseteq \ell \\ \langle\rangle, & \text{otherwise} \end{cases}$$

Note that:

- $\mathbf{S}^\ell(\bot) = \mathbf{Id}$, for any $\ell \in L$. Then, for every $\mathbf{S}^\ell$, $\eta = \epsilon = \mathrm{id}$.
- Next, fix some $\ell_0 \in L$. Now, for any $\ell_1, \ell_2 \in L$, we have, $\mathbf{S}^{\ell_0}(\ell_1) \circ \mathbf{S}^{\ell_0}(\ell_2) = \mathbf{S}^{\ell_0}(\ell_1 \sqcup \ell_2)$. Then, $\mu^{\ell_1,\ell_2} = \delta^{\ell_1,\ell_2} = \mathrm{id}$. Therefore, the $\mathbf{S}^\ell$s are all strict monoidal functors.

Now, for any bicartesian closed category $\mathbb{C}$, any strong monoidal functor from $\mathbf{C}(\mathcal{L})$ to $\mathbf{End}^{\mathrm{s}}_{\mathbb{C}}$ provides a computationally adequate interpretation of DCC$_e$($\mathcal{L}$), provided the interpretation for ground types is injective. We know that **Set**, the category of sets and functions, is a bicartesian closed category. Then, we can show,

Theorem 2.18 $[\![_]\!]_{(\mathbf{Set},\mathbf{S}^\ell)}$, for any $\ell \in L$, is a computationally adequate interpretation of DCC$_e$($\mathcal{L}$).

Next, we explain the intuition behind these strong monoidal functors. $\mathbf{S}^\ell$ keeps untouched all information at levels ℓ' where $\ell' \sqsubseteq \ell$ but blacks out all information at every other level. So, $\mathbf{S}^\ell$ corresponds to the view of an observer at level ℓ. We can formalize this intuition. Suppose $\neg(\ell'' \sqsubseteq \ell)$, i.e. ℓ should not depend upon ℓ''. Then, if $\ell'' \sqsubseteq A$, the terms of type A should not be visible to an observer at level ℓ. In other words, if $\ell'' \sqsubseteq A$, $\mathbf{S}^\ell$ should black out all information from type A. This is indeed the case, as we see next.

Lemma 2.19 If $\ell'' \sqsubseteq A$ and $\neg(\ell'' \sqsubseteq \ell)$, then $[\![A]\!]_{(\mathbb{C},\mathbf{S}^\ell)} \cong \top$.

The above lemma takes us to our noninterference theorem. Recall the test of correctness for DCC from Section 2.3.1: if $\ell \sqsubseteq A$ and $\neg(\ell \sqsubseteq \ell')$, then the terms of type A should not be visible at ℓ'. We prove correctness for DCC$_e$ by formulating this test as follows.

Theorem 2.20 Let $\mathcal{L} = (L, \sqcup, \bot)$ be the parametrizing semilattice.

- Suppose $\ell \in L$ such that $\neg(\ell \sqsubseteq \bot)$. Let $\ell \sqsubseteq A$. Let $\varnothing \vdash f : A \to \textbf{Bool}$ and $\varnothing \vdash a_1 : A$ and $\varnothing \vdash a_2 : A$. Then, $\vdash f\ a_1 \leadsto^* v$ if and only if $\vdash f\ a_2 \leadsto^* v$, where v is a value of type **Bool**.

- Suppose $\ell, \ell' \in L$ such that $\neg(\ell \sqsubseteq \ell')$. Let $\ell \sqsubseteq A$. Let $\varnothing \vdash f : A \to T_{\ell'}\,\textbf{Bool}$ and $\varnothing \vdash a_1 : A$ and $\varnothing \vdash a_2 : A$. Then, $\vdash f\ a_1 \leadsto^* v$ if and only if $\vdash f\ a_2 \leadsto^* v$, where v is a value of type $T_{\ell'}\,\textbf{Bool}$.

As corollary of the above theorem, we can show that $\varnothing \vdash f : T_{\mathbf{H}} A_1 \to \textbf{Bool}$ and $\varnothing \vdash f' : (A_1 \to T_{\mathbf{H}} A_2) \to \textbf{Bool}$, for all types A_1 and A_2, are constant functions. We can also show that $\varnothing \vdash f'' : T_{\ell_1}\ A \to T_{\ell_2}\ \textbf{Bool}$, for any type A, is a constant function, whenever $\neg(\ell_1 \sqsubseteq \ell_2)$.

We use the $\mathbf{S}^\ell$s to prove the noninterference theorem above. This technique relies on the observation that for two entities E_1 and E_2, if E_1 can be present when E_2 is absent, then E_1 *does not depend* upon E_2. We call this technique the *presence-absence test*. In the next section, we shall use the same technique to prove correctness of binding-time analysis in λ°.

2.9 Binding-Time Calculus, λ°

λ° [Davies, 2017] is a foundational calculus for binding-time analysis. It lies at the heart of state-of-the-art metaprogramming languages like MetaOCaml [Calcagno et al., 2003]. λ° is essentially a dependency calculus that ensures early stage computations do not depend upon later stage ones. One might expect that DCC, being a core calculus of dependency, would subsume the dependency calculus λ°. However, Abadi et al. [1999] noted that λ° cannot be translated into DCC. One reason behind this shortcoming is that DCC does not fully utilize the power of comonadic aspect of dependency analysis, as discussed in Section 2.5.1. We extended DCC to DCC$_e$ to incorporate the comonadic aspect of dependency analysis into the calculus. This extension opens up the possibility of λ° being translated to DCC$_e$. In this section, we explore this possibility. We first review the calculus λ°, thereafter present a categorical model leading to a proof of correctness for the calculus and finally show how we can translate λ° into our graded monadic comonadic system.

Types, $A, B ::= A \to B \mid \bigcirc A \mid \mathbf{Unit} \mid \mathbf{Void} \mid A_1 + A_2$
Terms, $a, b ::= x \mid \lambda x : A.b \mid b\ a \mid \mathbf{next}\ a \mid \mathbf{prev}\ a \mid \mathbf{unit} \mid \mathbf{abort}\ a \mid \mathbf{inj}_1\ a_1 \mid \mathbf{inj}_2\ a_2 \mid \mathbf{case}\ a\ \mathbf{of}\ b_1\ ;\ b_2$
Contexts, $\Gamma ::= \varnothing \mid \Gamma, x :^n A$

FIGURE 2.15: Grammar of λ°

$\boxed{\Gamma \vdash a :^n A}$ *(Typing)*

$$\text{LC-Var}\quad \frac{}{\Gamma_1, x :^n A, \Gamma_2 \vdash x :^n A}$$

$$\text{LC-Lam}\quad \frac{\Gamma, x :^n A \vdash b :^n B}{\Gamma \vdash \lambda x : A.b :^n A \to B}$$

$$\text{LC-App}\quad \frac{\Gamma \vdash b :^n A \to B \qquad \Gamma \vdash a :^n A}{\Gamma \vdash b\ a :^n B}$$

$$\text{LC-Inj1}\quad \frac{\Gamma \vdash a_1 :^n A_1}{\Gamma \vdash \mathbf{inj}_1\ a_1 :^n A_1 + A_2}$$

$$\text{LC-Case}\quad \frac{\Gamma \vdash a :^n A_1 + A_2 \quad \Gamma \vdash b_1 :^n A_1 \to B \quad \Gamma \vdash b_2 :^n A_2 \to B}{\Gamma \vdash \mathbf{case}\ a\ \mathbf{of}\ b_1\ ;\ b_2 :^n B}$$

$$\text{LC-Next}\quad \frac{\Gamma \vdash a :^{n + succ\, 0} A}{\Gamma \vdash \mathbf{next}\ a :^n \bigcirc A}$$

$$\text{LC-Prev}\quad \frac{\Gamma \vdash a :^n \bigcirc A}{\Gamma \vdash \mathbf{prev}\ a :^{n + succ\, 0} A}$$

FIGURE 2.16: Typing rules of λ° (Excerpt)

2.9.1 The Calculus λ°

λ° is simply-typed λ-calculus extended with a 'next time' type constructor, $\bigcirc$. Intuitively, $\bigcirc A$ is the type of terms to be computed upon the 'next time'. The calculus models staged computation, with an earlier stage manipulating programs from later stage as data. For a time-ordered normalization, the calculus needs to ensure that computation from an earlier stage *does not depend* upon computation from a later stage. To model such a notion of independence of the past from the future, Davies [2017] uses temporal logic. In λ°, time is discretized as instants or moments, represented by natural numbers. For example, 0 denotes the present moment, $succ\, 0$ denotes the next moment and so on. (Here, $succ\, 0$ is used in lieu of 1 to avoid confusion with the identity element of a monoid.) We now look at the calculus formally, as presented by Davies [2017].

The grammar of λ° appears in Figure 2.15, typing rules in Figure 2.16 and the equational theory in Figure 2.17. Note the use of graded contexts and graded typing judgments in λ°. The graded typing judgment, $\Gamma \vdash a :^n A$, intuitively means that a is available at time instant n, provided the variables in Γ are available at their respective time instants. Note that λ° of Davies [2017] does not have sum types; we include them here for the sake of having non-trivial ground types.

With this background on λ°, let us now build categorical models for the calculus.

$$(\lambda x : A.b)\ a \equiv b\{a/x\} \qquad b \equiv \lambda x : A.b\ x$$

$$\mathbf{prev}\ (\mathbf{next}\ a) \equiv a \qquad a \equiv \mathbf{next}\ (\mathbf{prev}\ a)$$

$$\mathbf{case}\ (\mathbf{inj}_i\ a_i)\ \mathbf{of}\ b_1\ ;\ b_2 \equiv b_i\ a_i\ \ [\text{Here, } i = 1,2] \qquad b\ a \equiv \mathbf{case}\ a\ \mathbf{of}\ \lambda x_1.b\ (\mathbf{inj}_1\ x_1)\ ;\ \lambda x_2.b\ (\mathbf{inj}_2\ x_2)$$

FIGURE 2.17: Equality rules of λ° (Excerpt)

2.9.2 Categorical Models for λ°

The motivation for categorical models of λ°, in the style of GMCC, comes from the observation that rules LC-NEXT and LC-PREV are like rules C-FORK and M-JOIN respectively. Here, we can think of $\mathcal{N} = (\mathbb{N}, +, 0)$ with discrete ordering to be the parametrizing preordered monoid. Then, $\bigcirc$ corresponds to $S_{succ\,0}$, where S is the graded modal type constructor of GMCC. We see that λ° and GMCC($\mathcal{N}$) share several similarities; however, there is a crucial difference between the two calculi. In λ°, the types $\bigcirc^n (A \to B)$ and $\bigcirc^n A \to \bigcirc^n B$ (where $\bigcirc^n$ is the operator $\bigcirc$ applied n times) are isomorphic whereas in GMCC($\mathcal{N}$), the types $S_n\ (A \to B)$ and $S_n\ A \to S_n\ B$ (where S_n is the operator S applied n times) are not necessarily isomorphic. Owing to this difference, we need to modify our models in order to interpret λ°. More precisely, unlike S, we cannot model $\bigcirc$ using any strong endofunctor, but would require cartesian closed endofunctors. So next, we define the category of cartesian closed endofunctors.

Let $\mathbb{C}$ be a cartesian closed category. An endofunctor $F : \mathbb{C} \to \mathbb{C}$ is said to be finite-product-preserving if and only if the morphisms $p_\top \triangleq \langle\rangle : F(\top) \to \top$ and $p_{X,Y} \triangleq \langle F\pi_1, F\pi_2\rangle : F(X \times Y) \to FX \times FY$, for $X, Y \in \text{Obj}(\mathbb{C})$, have inverses. A finite-product-preserving endofunctor $F : \mathbb{C} \to \mathbb{C}$ is said to be cartesian closed if and only if the morphisms $q_{X,Y} \triangleq \Lambda(F(\text{app}) \circ p^{-1}_{Y^X,X}) : F(Y^X) \to (FY)^{(FX)}$, for $X, Y \in \text{Obj}(\mathbb{C})$, have inverses. Now, the cartesian closed endofunctors of $\mathbb{C}$, with natural transformations as morphisms, form a category, $\mathbf{End}^{\text{cc}}_{\mathbb{C}}$. Like $\mathbf{End}^{\text{s}}_{\mathbb{C}}$, $\mathbf{End}^{\text{cc}}_{\mathbb{C}}$ is also a strict monoidal category with the monoidal product and the identity object defined in the same way. We shall use $\mathbf{End}^{\text{cc}}_{\mathbb{C}}$ to build models for λ°.

Let $\mathbb{C}$ be any bicartesian closed category. Let $\mathbf{S}$ be a strong monoidal functor from $\mathbf{C}(\mathcal{N})$ to $\mathbf{End}^{\text{cc}}_{\mathbb{C}}$. Then, the interpretation, $[\![_]\!]$, or more precisely $[\![_]\!]_{(\mathbb{C},\mathbf{S})}$, of types and terms is as follows. The modal operator and contexts are interpreted as:

$$[\![\bigcirc A]\!] = \mathbf{S}_{succ\,0}[\![A]\!] \qquad [\![\varnothing]\!] = \top \qquad [\![\Gamma, x :^n A]\!] = [\![\Gamma]\!] \times \mathbf{S}_n[\![A]\!]$$

Terms are interpreted as:

$$
\begin{aligned}
[\![x]\!] &= [\![\Gamma_1]\!] \times \mathbf{S}_n[\![A]\!] \times [\![\Gamma_2]\!] \xrightarrow{\pi_i} \mathbf{S}_n[\![A]\!] \\
[\![\lambda x : A.b]\!] &= [\![\Gamma]\!] \xrightarrow{\Lambda[\![b]\!]} (\mathbf{S}_n[\![B]\!])^{(\mathbf{S}_n[\![A]\!])} \xrightarrow{q^{-1}} \mathbf{S}_n([\![B]\!]^{[\![A]\!]}) \\
[\![b\ a]\!] &= [\![\Gamma]\!] \xrightarrow{\langle q \circ [\![b]\!], [\![a]\!]\rangle} (\mathbf{S}_n[\![B]\!])^{(\mathbf{S}_n[\![A]\!])} \times \mathbf{S}_n[\![A]\!] \xrightarrow{\text{app}} \mathbf{S}_n[\![B]\!] \\
[\![\mathbf{next}\ a]\!] &= [\![\Gamma]\!] \xrightarrow{[\![a]\!]} \mathbf{S}_{n+succ\,0}[\![A]\!] \xrightarrow{\delta^{n,succ\,0}} \mathbf{S}_n\mathbf{S}_{succ\,0}[\![A]\!] \\
[\![\mathbf{prev}\ a]\!] &= [\![\Gamma]\!] \xrightarrow{[\![a]\!]} \mathbf{S}_n\mathbf{S}_{succ\,0}[\![A]\!] \xrightarrow{\mu^{n,succ\,0}} \mathbf{S}_{n+succ\,0}[\![A]\!] \\
[\![\mathbf{inj}_1\ a_1]\!] &= [\![\Gamma]\!] \xrightarrow{[\![a_1]\!]} \mathbf{S}_n[\![A_1]\!] \xrightarrow{\mathbf{S}_n i_1} \mathbf{S}_n([\![A_1]\!] + [\![A_2]\!]) \\
[\![\mathbf{case}\ a\ \mathbf{of}\ b_1\ ;\ b_2]\!] &= [\![\Gamma]\!] \xrightarrow{\langle\langle [\![b_1]\!], [\![b_2]\!]\rangle, [\![a]\!]\rangle} (\mathbf{S}_n[\![B]\!]^{[\![A_1]\!]} \times \mathbf{S}_n[\![B]\!]^{[\![A_2]\!]}) \times \mathbf{S}_n([\![A_1]\!] + [\![A_2]\!]) \\
&\xrightarrow{p^{-1} \times \text{id}} \mathbf{S}_n([\![B]\!]^{[\![A_1]\!]} \times [\![B]\!]^{[\![A_2]\!]}) \times \mathbf{S}_n([\![A_1]\!] + [\![A_2]\!]) \\
&\xrightarrow{p^{-1}} \mathbf{S}_n(([\![B]\!]^{[\![A_1]\!]} \times [\![B]\!]^{[\![A_2]\!]}) \times ([\![A_1]\!] + [\![A_2]\!])) \\
&\cong \mathbf{S}_n([\![B]\!]^{[\![A_1]\!] + [\![A_2]\!]} \times ([\![A_1]\!] + [\![A_2]\!])) \xrightarrow{\mathbf{S}_n \text{app}} \mathbf{S}_n[\![B]\!]
\end{aligned}
$$

This gives us a sound interpretation of λ°.

Theorem 2.21 If $\Gamma \vdash a :^n A$ in λ°, then $[\![a]\!] \in \text{Hom}_{\mathbb{C}}([\![\Gamma]\!], \mathbf{S}_n[\![A]\!])$. Further, if $\Gamma \vdash a_1 :^n A$ and $\Gamma \vdash a_2 :^n A$ such that $a_1 \equiv a_2$ in λ°, then $[\![a_1]\!] = [\![a_2]\!] \in \text{Hom}_{\mathbb{C}}([\![\Gamma]\!], \mathbf{S}_n[\![A]\!])$.

Such an interpretation is also computationally adequate, provided it is injective for ground types. The operational semantics for the calculus is assumed to be the call-by-value semantics induced by the β-rules in Figure 2.17. We denote the multi-step reduction relation corresponding to this operational semantics by $\longmapsto^*$. Formally, we can state adequacy as:

Theorem 2.22 Let the interpretation $[\![_]\!]_{(\mathbb{C},\mathbf{S})}$ be injective for ground types. Let $\Gamma \vdash b :^0 \mathbf{Bool}$ and v be a value of type **Bool**. If $[\![b]\!]_{(\mathbb{C},\mathbf{S})} = [\![v]\!]_{(\mathbb{C},\mathbf{S})}$ then $\vdash b \longmapsto^* v$.

Next, we use these categorical models to show correctness of binding-time analysis in λ°.

2.9.3 Correctness of Binding-Time Analysis in λ°

A binding-time analysis is correct if computation from an earlier stage *does not depend* upon computation from a later stage. Here, using the categorical model, we shall show that λ° satisfies this condition. We shall use the same *presence-absence test* technique that we used to prove noninterference for DCC_e. The goal is to show that computations at a given stage can proceed when computations from all later stages are blacked out. Towards this end, we present a strong monoidal functor, $\mathbf{S}^0$, which keeps untouched all computations

$\boxed{\Gamma \vdash a :^n A}$ *(Typing)*

$$\frac{\text{LCK-VAR}}{\Gamma_1, x :^n A, \Gamma_2 \vdash x :^n A} \qquad \frac{\Gamma, x :^n A \vdash b :^n B}{\Gamma \vdash \lambda x : A.b :^n A \to B}\text{LCK-LAM} \qquad \frac{\Gamma \vdash b :^n A \to B \qquad \Gamma \vdash a :^n A}{\Gamma \vdash b\ a :^n B}\text{LCK-APP}$$

$$\frac{\Gamma \vdash a :^{n+succ\,k} A}{\Gamma \vdash \textbf{next}\ a :^{n+k} \bigcirc A}\text{LCK-NEXT} \qquad \frac{\Gamma \vdash a :^{n+k} \bigcirc A}{\Gamma \vdash \textbf{prev}\ a :^{n+succ\,k} A}\text{LCK-PREV} \qquad \frac{\Gamma \vdash a :^{n_1} A \qquad n_1 \leq k \qquad n_2 \leq k}{\Gamma \vdash a :^{n_2} A}\text{LCK-UPTOK}$$

FIGURE 2.18: Typing rules of $\lambda^\circ(k)$ (Excerpt)

at time instant 0 but blacks out all computations from every time instant later than 0.

$$\mathbf{S}^0(n) = \begin{cases} \mathbf{Id} & \text{if } n = 0 \\ * & \text{otherwise} \end{cases}$$

Here, $\mathbf{S}^0$ is a strict monoidal functor from $\mathbf{C}(\mathcal{N})$ to $\mathbf{End}^{cc}_{\mathbb{C}}$ (where $\mathbb{C}$ is a bicartesian closed category) with $\eta = \epsilon = \text{id}$ and $\delta = \mu = \text{id}$. Now, for any bicartesian closed category $\mathbb{C}$, any strong monoidal functor from $\mathbf{C}(\mathcal{N})$ to $\mathbf{End}^{cc}_{\mathbb{C}}$ provides a computationally adequate interpretation of λ°, given the interpretation for ground types is injective. Therefore,

Theorem 2.23 $[\![-]\!]_{(\mathbf{Set},\mathbf{S}^0)}$ is a computationally adequate interpretation of λ°.

Using adequacy, we can prove the following noninterference theorem.

Theorem 2.24 Let $\varnothing \vdash f :^0 \bigcirc A \to \mathbf{Bool}$ and $\varnothing \vdash b_1 :^0 \bigcirc A$ and $\varnothing \vdash b_2 :^0 \bigcirc A$. Then, $\vdash f\ b_1 \longmapsto^* v$ if and only if $\vdash f\ b_2 \longmapsto^* v$, where v is a value of type $\mathbf{Bool}$.

The above theorem shows that computations from time instant 0 can proceed independently of computations from all later time instants. Now, once the computations from time instant 0 are done, we can move on to computations from time instant $succ\,0$, which would act as the new 0. Then, using the same argument, we can show that computations from the new 0 can proceed independently of computations from all new later time instants. Repeating this argument over and over again, we see that we can normalize λ°-expressions in a time-ordered manner. This is an informal argument that shows binding-time analysis in λ° is correct. Next, we formalize this argument.

Our claim is that for any $n \in \mathbb{N}$, computations at time instant n can proceed independently of computations from all later time instants. Theorem 2.24 proves the claim for $n = 0$. Now, for any $n = k$, numerically greater than 0, we need to show that computations from time instant k can proceed independently of computations from time instants $succ\,k$ and later. To prove this statement, we update the type system of λ° to $\lambda^\circ(k)$, which is similar to λ° but does not distinguish among computations from time instants n_1 and n_2, if both n_1 and n_2 are numerically less than or equal to k. We present $\lambda^\circ(k)$ in Figure 2.18.

The variable rule and the rules introducing and eliminating function and sum types in $\lambda^\circ(k)$ are the same as their counterparts in λ°. However, the rules LCk-Next and LCk-Prev differ from rules LC-Next and LC-Prev because they use k, in lieu of 0, as the offset. The rule LCk-UptoK is new and is added to eliminate distinctions among computations from time instants n_1 and n_2, where both n_1 and n_2 are numerically less than or equal to k. Note here that λ° is just $\lambda^\circ(0)$.

Next, we present a meaning-preserving translation from $\lambda^\circ(k)$ to λ°. Define a function $f : \mathbb{N} \to \mathbb{N}$ such that $f(n) = 0$ if $n \leq k$, and $(n - k)$ otherwise. Now, given a judgment $x_1 :^{n_1} A_1, x_2 :^{n_2} A_2, \ldots, x_m :^{n_m} A_m \vdash b :^n B$ in $\lambda^\circ(k)$, we can show that $x_1 :^{f(n_1)} A_1, x_2 :^{f(n_2)} A_2, \ldots, x_m :^{f(n_m)} A_m \vdash b :^{f(n)} B$ in λ°. The intended translation function is essentially identity on types and terms. So, the translation also preserves meaning. Observe that this translations maps: computations from time instants numerically less than or equal to k to computations at time instant 0; and computations from time instants $succ\, k$ and later to computations at time instants $succ\, 0$ and later.

Now, for λ°, we know that computations from time instant 0 can proceed independently of computations from all later time instants. Therefore, by virtue of the above translation, we can deduce that in $\lambda^\circ(k)$, computations from time instants k and earlier can proceed independently of computations from time instants $succ k$ and later. Since $\lambda^\circ(k)$ is just λ° that treats time instants less than or equal to k the same, we conclude that in λ°, computations from time instant k do not depend upon computations from time instants $succ\, k$ or later. This completes the proof of our claim and shows that binding-time analysis in λ° is correct.

Observe how the presence-absence test aids this correctness proof, especially with regard to Theorems 2.23 and 2.24. The presence-absence test is simple but effective in showing correctness of dependency analyses. We used it to show correctness of dependency analyses in DCC_e and λ° in a very straightforward manner. To put it in perspective, Davies [2017] requires 10 journal pages to establish correctness of λ° using syntactic methods whereas our proof of correctness for λ° follows almost immediately from the soundness theorem (Theorem 2.21). This shows that the presence-absence test could be a useful tool for establishing correctness of dependency calculi.

Next, we show how to translate λ° into a graded monadic comonadic framework.

2.9.4 Can We Translate λ° to GMCC?

Here, we consider how we might translate λ° to GMCC. We can instantiate the parametrizing preordered monoid $\mathcal{M}$ of GMCC to $\mathcal{N} = (\mathbb{N}, +, 0)$ with discrete ordering; then, we may translate $\bigcirc$ as: $\widehat{\bigcirc A} = S_{succ\, 0}\, \widehat{A}$. We can translate contexts as:

$$\widehat{\varnothing} = \varnothing \qquad \widehat{\Gamma, x :^n A} = \widehat{\Gamma}, x : S_n\, \widehat{A}$$

A typing judgment $\Gamma \vdash a :^n A$ can then be translated as: $\widehat{\Gamma} \vdash \widehat{a} : S_n\, \widehat{A}$. For the modal terms, we have:

$$\widehat{\textbf{next}\; a} = \textbf{fork}^{n, succ\, 0} \widehat{a} \qquad \widehat{\textbf{prev}\; a} = \textbf{join}^{n, succ\, 0} \widehat{a}$$

This translation works well for the modal constructs; however, we run into a problem when dealing with functions and applications. The problem is that with the above translation, it is not possible to show typing is preserved in case of functions and applications. The reason behind this problem is the difference between the calculi λ° and GMCC we referred to earlier: the types $\bigcirc^{n}(A \to B)$ and $\bigcirc^{n} A \to \bigcirc^{n} B$ are isomorphic in λ° but not necessarily in GMCC.

This difference arises from the fact that in λ°, grades pervade all the typing rules, including the ones for functions and applications, while in GMCC, they are restricted to the monadic and comonadic typing rules. We could have designed GMCC by permeating the grades along all the typing rules in lieu of restricting them to a fragment of the calculus. We avoided such a design for the sake of simplicity. However, now that we understand the calculus, we can consider the implications of such a design choice. In the next section, we explore this design choice and present GMCC_e, which is GMCC extended with graded contexts and graded typing judgments.

2.10 GMCC_e

2.10.1 The Calculus

Like GMCC, GMCC_e is parametrized by an arbitrary preordered monoid, $\mathcal{M}$. The types of the calculus are the same as those of GMCC. With respect to terms, GMCC_e differs from GMCC in having only the following two term-level constructs (in lieu of **ret**, **extr**, etc.) for introducing and eliminating the modal type.

$$\text{terms}, a, b ::= \ldots (\lambda\text{-calculus terms}) \mid \mathbf{split}^{m}\, a \mid \mathbf{merge}^{m}\, a$$

GMCC_e also differs from GMCC with regard to contexts and typing judgments, both of which are graded in GMCC_e, as in λ°. The typing judgment of GMCC_e, $\Gamma \vdash a :^{m} A$, intuitively means that a can be observed at m, provided the variables in Γ are observable at their respective grades. The typing rules of the calculus appear in Figure 2.19. The rules pertaining to standard λ-calculus terms are as expected. (Note that the two rules for introducing the sum type are combined into a single rule E-Inj, with $i = 1, 2$. This concise presentation is also used in other type systems appearing later in the dissertation.) The rules E-Split and E-Merge introduce and eliminate the modal type. These rules are similar to rule C-Fork and rule M-Join respectively. The rule E-Up, akin to rules M-Up and C-Up, implicitly relaxes the grade at which a term is observed.

The equational theory of the calculus is induced by the $\beta\eta$-rules of standard λ-calculus along with the following $\beta\eta$-rule for modal terms.

$$\mathbf{merge}^{m}(\mathbf{split}^{m}\, a) \equiv a \qquad a \equiv \mathbf{split}^{m}(\mathbf{merge}^{m}\, a)$$

$\boxed{\Gamma \vdash a :^m A}$ *(Typing)*

$$\text{E-VAR}\quad \frac{}{\Gamma_1, x :^m A, \Gamma_2 \vdash x :^m A}$$

$$\text{E-LAM}\quad \frac{\Gamma, x :^m A \vdash b :^m B}{\Gamma \vdash \lambda x : A.b :^m A \to B}$$

$$\text{E-APP}\quad \frac{\Gamma \vdash b :^m A \to B \qquad \Gamma \vdash a :^m A}{\Gamma \vdash b\ a :^m B}$$

$$\text{E-PAIR}\quad \frac{\Gamma \vdash a_1 :^m A_1 \qquad \Gamma \vdash a_2 :^m A_2}{\Gamma \vdash (a_1, a_2) :^m A_1 \times A_2}$$

$$\text{E-PROJ}\quad \frac{\Gamma \vdash a :^m A_1 \times A_2}{\Gamma \vdash \mathbf{proj}_i\ a :^m A_i}$$

$$\text{E-INJ}\quad \frac{\Gamma \vdash a_i :^m A_i}{\Gamma \vdash \mathbf{inj}_i\ a_i :^m A_1 + A_2}$$

$$\text{E-CASE}\quad \frac{\begin{array}{c}\Gamma \vdash a :^m A_1 + A_2 \\ \Gamma \vdash b_1 :^m A_1 \to B \\ \Gamma \vdash b_2 :^m A_2 \to B\end{array}}{\Gamma \vdash \mathbf{case}\ a\ \mathbf{of}\ b_1\ ;\ b_2 :^m B}$$

$$\text{E-SPLIT}\quad \frac{\Gamma \vdash a :^{m_1 \cdot m_2} A}{\Gamma \vdash \mathbf{split}^{m_2}\ a :^{m_1} S_{m_2}\ A}$$

$$\text{E-MERGE}\quad \frac{\Gamma \vdash a :^{m_1} S_{m_2}\ A}{\Gamma \vdash \mathbf{merge}^{m_2}\ a :^{m_1 \cdot m_2} A}$$

$$\text{E-UNIT}\quad \frac{}{\Gamma \vdash \mathbf{unit} :^m \mathbf{Unit}}$$

$$\text{E-VOID}\quad \frac{\Gamma \vdash a :^m \mathbf{Void}}{\Gamma \vdash \mathbf{abort}\ a :^m A}$$

$$\text{E-UP}\quad \frac{\Gamma \vdash a :^{m_1} A \qquad m_1 <: m_2}{\Gamma \vdash a :^{m_2} A}$$

FIGURE 2.19: Typing rules of GMCC_e

A graded presentation of the calculus, where both contexts and typing judgments are graded, has some interesting consequences. Unlike GMCC, GMCC_e enjoys the following properties.

Lemma 2.25 Let $\mathcal{M}$ be any preordered monoid. Then, in $\text{GMCC}_\text{e}(\mathcal{M})$,

- The types S_m **Unit** and **Unit** are isomorphic.
- The types $S_m\ (A_1 \times A_2)$ and $S_m\ A_1 \times S_m\ A_2$, for all types A_1 and A_2, are isomorphic.
- The types $S_m\ (A \to B)$ and $S_m\ A \to S_m\ B$, for all types A and B, are isomorphic.

The third property above reminds us of the isomorphism between types $\bigcirc^n(A \to B)$ and $\bigcirc^n A \to \bigcirc^n B$ in λ°. Recall that we could not translate λ° into GMCC because such an isomorphism does not hold in general in GMCC. We shall see later that GMCC_e, which satisfies these isomorphisms, can readily capture λ°.

GMCC_e also satisfies a multiplication lemma shown below. Here, the graded context $m' \cdot \Gamma$ is obtained by pre-multiplying every grade in Γ by m'.

Lemma 2.26 Let $\Gamma \vdash a :^m A$ in $\text{GMCC}_\text{e}(\mathcal{M})$. Then, $m' \cdot \Gamma \vdash a :^{m' \cdot m} A$, for any $m' \in \mathcal{M}$.

In the following chapters, we shall find that several of the graded type systems we develop satisfy similar multiplication lemmas.

Now, we build categorical models for GMCC_e. The models are similar to those of GMCC; however, as in the case of λ°, we need to use cartesian closed endofunctors in lieu of just strong endofunctors. Let $\mathcal{M}$ be the parametrizing preordered monoid. Let $\mathbb{C}$ be any bicartesian closed category. Next, let $\mathbf{S}$ be a strong

monoidal functor from $\mathbf{C}(\mathcal{M})$ to $\mathbf{End}^{cc}_{\mathbb{C}}$. Then, the interpretation $[\![_]\!]$ of types, contexts and terms is as follows.

$$[\![S_m\ A]\!] = \mathbf{S}_m[\![A]\!] \qquad [\![\varnothing]\!] = \top \qquad [\![\Gamma, x :^m A]\!] = [\![\Gamma]\!] \times \mathbf{S}_m[\![A]\!]$$

$$[\![\mathbf{split}^{m_2}\ a]\!] = [\![\Gamma]\!] \xrightarrow{[\![a]\!]} \mathbf{S}_{m_1 \cdot m_2}[\![A]\!] \xrightarrow{\delta^{m_1,m_2}_{[\![A]\!]}} \mathbf{S}_{m_1}\mathbf{S}_{m_2}[\![A]\!]$$
$$[\![\mathbf{merge}^{m_2}\ a]\!] = [\![\Gamma]\!] \xrightarrow{[\![a]\!]} \mathbf{S}_{m_1}\mathbf{S}_{m_2}[\![A]\!] \xrightarrow{\mu^{m_1,m_2}_{[\![A]\!]}} \mathbf{S}_{m_1 \cdot m_2}[\![A]\!]$$

Note that the types and terms of the standard λ-calculus are interpreted as in the case of λ°.
This gives us a sound interpretation of $\text{GMCC}_\text{e}(\mathcal{M})$.

Theorem 2.27 If $\Gamma \vdash a :^m A$ in GMCC_e, then $[\![a]\!] \in \text{Hom}_{\mathbb{C}}([\![\Gamma]\!], \mathbf{S}_m[\![A]\!])$. Further, if $\Gamma \vdash a_1 :^m A$ and $\Gamma \vdash a_2 :^m A$ such that $a_1 \equiv a_2$ in GMCC_e, then $[\![a_1]\!] = [\![a_2]\!] \in \text{Hom}_{\mathbb{C}}([\![\Gamma]\!], \mathbf{S}_m[\![A]\!])$.

Next, we show that both DCC_e and λ° can be translated to GMCC_e.

2.10.2 Relating DCC_e to GMCC_e

Here, we translate DCC_e to GMCC_e. We know that, over the class of bounded join-semilattices, DCC_e is equivalent to GMCC. Given that GMCC is quite close to GMCC_e, we shall use GMCC as the source language for our translation. Let $\mathcal{L} = (L, \sqcup, \bot)$ be an arbitrary join-semilattice. Then, the translation $\widetilde{\ }$ from $\text{GMCC}(\mathcal{L})$ to $\text{GMCC}_\text{e}(\mathcal{L})$ is as follows. For types, $\widetilde{A} = A$. For terms,

$$\widetilde{\mathbf{ret}\ a} = \mathbf{split}^{\bot}\widetilde{a} \qquad \widetilde{\mathbf{extr}\ a} = \mathbf{merge}^{\bot}\widetilde{a}$$
$$\widetilde{\mathbf{join}^{\ell_1,\ell_2}\ a} = \mathbf{split}^{\ell_1 \sqcup \ell_2}(\mathbf{merge}^{\ell_2}(\mathbf{merge}^{\ell_1}\widetilde{a})) \qquad \widetilde{\mathbf{fork}^{\ell_1,\ell_2}\ a} = \mathbf{split}^{\ell_1}(\mathbf{split}^{\ell_2}(\mathbf{merge}^{\ell_1 \sqcup \ell_2}\widetilde{a}))$$
$$\widetilde{\mathbf{lift}^{\ell} f} = \lambda x.\mathbf{split}^{\ell}(\widetilde{f}\ (\mathbf{merge}^{\ell} x)) \qquad \widetilde{\mathbf{up}^{\ell_1,\ell_2}\ a} = \mathbf{split}^{\ell_2}(\mathbf{merge}^{\ell_1}\widetilde{a})$$

The standard λ-calculus terms are translated as expected. This translation is sound, as we see next. Below, we use Γ^m to denote the graded-context that has Γ as the underlying context and marks every assumption with grade m.

Theorem 2.28 If $\Gamma \vdash a : A$ in $\text{GMCC}(\mathcal{L})$, then $\Gamma^{\bot} \vdash \widetilde{a} :^{\bot} A$ in $\text{GMCC}_\text{e}(\mathcal{L})$. Further, if $\Gamma \vdash a_1 : A$ and $\Gamma \vdash a_2 : A$ such that $a_1 \equiv a_2$ in $\text{GMCC}(\mathcal{L})$, then $\widetilde{a_1} \equiv \widetilde{a_2}$ in $\text{GMCC}_\text{e}(\mathcal{L})$.

One might wonder here whether we can go the other way around and translate $\text{GMCC}_\text{e}(\mathcal{L})$ to $\text{GMCC}(\mathcal{L})$. Though the two calculi are very similar, such a translation is not possible, owing to Proposition 2.25. More concretely, observe that the type $S_{\mathbf{H}}\ (S_{\mathbf{H}}\ \mathbf{Bool} \to \mathbf{Bool})$ has four distinct terms in $\text{GMCC}_\text{e}(\mathcal{L}_2)$, derivable

at $\mathbf{L}$. The terms are:

$$\mathbf{split}^{\mathbf{H}}(\lambda x.\mathbf{true}) \qquad \mathbf{split}^{\mathbf{H}}(\lambda x.\mathbf{false})$$
$$\mathbf{split}^{\mathbf{H}}(\lambda x.\mathbf{merge}^{\mathbf{H}} x) \qquad \mathbf{split}^{\mathbf{H}}(\lambda x.\mathbf{not}\,(\mathbf{merge}^{\mathbf{H}} x))$$

However, the same type, $S_{\mathbf{H}}\,(S_{\mathbf{H}}\,\mathbf{Bool} \to \mathbf{Bool})$, has only two distinct terms in GMCC($\mathcal{L}_2$):

$$\mathbf{up}^{\mathbf{L},\mathbf{H}}\mathbf{ret}\,(\lambda x.\mathbf{true}) \qquad \mathbf{up}^{\mathbf{L},\mathbf{H}}\mathbf{ret}\,(\lambda x.\mathbf{false})$$

Here, $\mathcal{L}_2 \triangleq \mathbf{L} \sqsubseteq \mathbf{H}$ and $\mathbf{Bool} \triangleq \mathbf{Unit} + \mathbf{Unit}$ and $\mathbf{true} \triangleq \mathbf{inj}_1\,\mathbf{unit}$ and $\mathbf{false} \triangleq \mathbf{inj}_2\,\mathbf{unit}$ and $\mathbf{not} \triangleq \lambda z.\mathbf{case}\,z\,\mathbf{of}\,(\lambda y.\mathbf{false})\,;\,(\lambda y.\mathbf{true})$.

The above distinction between GMCC$_\mathrm{e}$($\mathcal{L}_2$) and GMCC($\mathcal{L}_2$) is important because in GMCC$_\mathrm{e}$($\mathcal{L}_2$), the function $S_{\mathbf{H}}\,\mathbf{Bool} \to \mathbf{Bool}$, if wrapped under $S_{\mathbf{H}}$, may return a high-security output whereas in GMCC($\mathcal{L}_2$), it must always return a constant output. From the perspective of dependency analysis, it is harmless, and in fact desirable, to allow any function to return a high-security output, provided the function itself is wrapped under the high-security label, $\mathbf{H}$. Note that this distinction between GMCC$_\mathrm{e}$ and GMCC is not specific to the parameter, $\mathcal{L}_2$, but can be seen for any arbitrary parameter, $\mathcal{L}$. To elaborate, if $\neg(\ell_1 \sqsubseteq \ell_2)$ in $\mathcal{L}$, then the type $S_{\ell_1}\,(S_{\ell_1}\,\mathbf{Bool} \to S_{\ell_2}\,\mathbf{Bool})$ has only two distinct terms in GMCC($\mathcal{L}$) but four distinct terms in GMCC$_\mathrm{e}$($\mathcal{L}$). Therefore, we see that GMCC($\mathcal{L}$) misses out some aspects of dependency analysis that can be carried out in GMCC$_\mathrm{e}$($\mathcal{L}$). This difference between GMCC($\mathcal{L}$) and GMCC$_\mathrm{e}$($\mathcal{L}$) stems from the fact that unlike the former, the latter employs graded contexts and graded typing judgments. In GMCC$_\mathrm{e}$($\mathcal{L}$), the grade on the typing judgment of a term helps pass on the information about the observer level to all its subterms, thereby allowing derivations like the ones referred to above.

Note here that as a dependency calculus, GMCC$_\mathrm{e}$($\mathcal{L}$) enjoys noninterference. Noninterference for GMCC$_\mathrm{e}$($\mathcal{L}$) may be shown using the same presence-absence test technique we employed to show noninterference for DCC$_e$($\mathcal{L}$) in Section 2.8.2. We leave the details as an exercise. Now, as a dependency calculus, GMCC$_\mathrm{e}$($\mathcal{L}$) has several advantages over GMCC($\mathcal{L}$) and DCC$_e$($\mathcal{L}$):

- From the above discussion, we see that GMCC$_\mathrm{e}$($\mathcal{L}$) subsumes GMCC($\mathcal{L}$) and DCC$_e$($\mathcal{L}$) and can also capture aspects of dependency analysis that are outside the scope of these two calculi. As such, GMCC$_\mathrm{e}$($\mathcal{L}$) is better at analyzing dependencies, compared to these two calculi.
- The use of graded contexts and graded typing judgments provides us the necessary flexibility to extend GMCC$_\mathrm{e}$($\mathcal{L}$) to dependent type systems. In dependent type systems, where both a term and its type may depend upon the same contextual assumption, it is grading (of contexts and typing judgments) that makes it possible to impose different constraints on the use of an assumption in a term and in its type.
- GMCC$_\mathrm{e}$ has clear introduction and elimination rules for the graded modal type, viz., rules E-Split and E-Merge. On the other hand, GMCC has rules C-Fork and M-Join, which are neither introduction nor elimination rules. DCC$_e$, though, has clear introduction and elimination rules for the graded

modal type, viz., rules DCC-Eta and DCC-Bind. However, DCC_e requires an auxiliary protection judgment in the elimination rule DCC-Bind, a requirement which adds some complexity to the type system. In this regard, observe how GMCC_e elegantly introduces and eliminates the graded modal type, without requiring any auxiliary judgments.

Owing to these advantages, we shall use $\text{GMCC}_\text{e}(\mathcal{L})$ as the basis for designing our dependency calculi for pure type systems in the following chapter.

2.10.3 Relating λ° to GMCC_e

Here we return to our incomplete translation of λ° from Section 2.9.4. Though we couldn't translate λ° to $\text{GMCC}(\mathcal{N})$, we can now easily translate it to $\text{GMCC}_\text{e}(\mathcal{N})$. The translation for the modal type and terms is as follows:

$$\widehat{\bigcirc A} = S_{succ\,0}\,\widehat{A} \qquad \widehat{\textbf{next}\ a} = \textbf{split}^{succ\,0}\widehat{a} \qquad \widehat{\textbf{prev}\ a} = \textbf{merge}^{succ\,0}\widehat{a}$$

This translation is sound, as we see next. Below, $\widehat{\Gamma}$ denotes Γ with the types of the assumptions translated.

Theorem 2.29 If $\Gamma \vdash a :^n A$ in λ°, then $\widehat{\Gamma} \vdash \widehat{a} :^n \widehat{A}$ in $\text{GMCC}_\text{e}(\mathcal{N})$. Further, if $\Gamma \vdash a_1 :^n A$ and $\Gamma \vdash a_2 :^n A$ such that $a_1 \equiv a_2$ in λ°, then $\widehat{a_1} \equiv \widehat{a_2}$ in $\text{GMCC}_\text{e}(\mathcal{N})$.

Note that we can also go the other way around and translate $\text{GMCC}_\text{e}(\mathcal{N})$ to λ°, extended with products. The types and terms of standard λ-calculus are translated as expected; the modal types and terms are translated as follows. Below, $\textbf{next}^n\ a$ denotes the **next** operator applied n times on a, and likewise for $\textbf{prev}^n\ a$.

$$\underbrace{S_n\ A} = \bigcirc^n \underbrace{A} \qquad \underbrace{\textbf{split}^n a} = \textbf{next}^n\ \underbrace{a} \qquad \underbrace{\textbf{merge}^n a} = \textbf{prev}^n\ \underbrace{a}$$

This translation is sound, as we see next. Below, $\underbrace{\Gamma}$ denotes Γ with the types of the assumptions translated.

Theorem 2.30 If $\Gamma \vdash a :^n A$ in $\text{GMCC}_\text{e}(\mathcal{N})$, then $\underbrace{\Gamma} \vdash \underbrace{a} :^n \underbrace{A}$ in λ°, extended with products. Further, if $\Gamma \vdash a_1 :^n A$ and $\Gamma \vdash a_2 :^n A$ such that $a_1 \equiv a_2$ in $\text{GMCC}_\text{e}(\mathcal{N})$, then $\underbrace{a_1} \equiv \underbrace{a_2}$ in λ°, extended with products.

These translations enable back-and-forth movement between $\text{GMCC}_\text{e}(\mathcal{N})$ and λ°, extended with products. We can show that a λ°-term, after a round trip to $\text{GMCC}_\text{e}(\mathcal{N})$, is syntactically equal to itself.

Theorem 2.31 If $\Gamma \vdash a :^n A$ in λ°, then $\underbrace{\hat{a}} = a$.

However, note that the same is not true of a $\text{GMCC}_\text{e}(\mathcal{N})$-term, which may not be equal to itself after a round trip to λ° because the trip may change its type. For example, if $\Gamma \vdash a :^n S_m\ A$ in $\text{GMCC}_\text{e}(\mathcal{N})$, where m is numerically greater than $succ 0$, then the round trip would map $S_m\ A$ to the type formed m applications of $S_{succ\,0}$ on $\underbrace{\hat{A}}$, which is syntactically different from $S_m\ A$, even if $\underbrace{\hat{A}} = A$. The reason behind this asymmetry between $\text{GMCC}_\text{e}(\mathcal{N})$ and λ° is that while the type constructor S of $\text{GMCC}_\text{e}(\mathcal{N})$ is graded by $n \in \mathbb{N}$, the type constructor $\bigcirc$ of λ° is ungraded and can encode a graded modal type constructor only via repeated

applications.

To sum up this discussion, we translated both DCC_e and λ° to GMCC_e. We found that $\text{GMCC}_\text{e}(\mathcal{L})$ subsumes $\text{DCC}_e(\mathcal{L})$ whereas $\text{GMCC}_\text{e}(\mathcal{N})$ subsumes λ°. Now, we know that (the terminating fragment of) DCC is a proper sublanguage of DCC_e. We also know that DCC cannot capture λ°. So, we conclude that GMCC_e is a more powerful dependency calculus than DCC.

Reviewing the literature, we find that GMCC_e is related to another dependency calculus, the sealing calculus of Shikuma and Igarashi [2006]. The sealing calculus also subsumes the terminating fragment of DCC. On comparing the sealing calculus with GMCC_e, we find that similar to **split** and **merge** of GMCC_e, the sealing calculus has **seal** and **unseal** to introduce and eliminate the graded modal type. However, the sealing calculus is less general than GMCC_e because the former does not subsume λ°. Additionally, the sealing calculus works for lattices only whereas GMCC_e works for any preordered monoid. In this chapter, we shall not go any further in our comparison of GMCC_e and the sealing calculus. But in the next chapter (see Section 3.2.4), we shall present an embedding of the sealing calculus into SDC, an alternative presentation of $\text{GMCC}_\text{e}(\mathcal{L})$ we develop there.

2.11 Discussions and Related Work

2.11.1 Nontermination

Till now, we did not include nontermination in any of our calculi. DCC, as presented by Abadi et al. [1999], includes nonterminating computations. So here we discuss how we may add nontermination to GMCC and GMCC_e. One of the features of these calculi is that they can be decomposed into a λ-calculus fragment and a monadic/comonadic fragment. The categorical models for the calculi also reflect the separation between the two fragments. In $\text{GMCC}(\mathcal{M})$, the λ-calculus fragment is modeled by a bicartesian closed category, $\mathbb{C}$, whereas the monadic/comonadic fragment is modeled using strong monoidal functors from $\mathbf{C}(\mathcal{M})$ to $\mathbf{End}^{\text{s}}_{\mathbb{C}}$. In $\text{GMCC}_\text{e}(\mathcal{M})$, the λ-calculus fragment is modeled by a bicartesian closed category, $\mathbb{C}$ (with the modal type constructors being modeled by cartesian closed endofunctors over $\mathbb{C}$), whereas the monadic/comonadic fragment is modeled using strong monoidal functors from $\mathbf{C}(\mathcal{M})$ to $\mathbf{End}^{\text{cc}}_{\mathbb{C}}$.

The separation between the λ-calculus and monadic/comonadic fragments in GMCC and GMCC_e makes it easier to add nontermination to these calculi. To include nontermination in these calculi, we just need to add the monadic/comonadic fragment on top of an already nonterminating calculus, for example, λ-calculus with general recursion. The nonterminating calculus can then be modeled by an appropriate category $\mathbb{D}$, such as, **Cpo** (the category of complete partial orders and continuous functions), and the monadic/comonadic fragment by strong monoidal functors from the parametrizing monoidal category to the category of strong or cartesian closed endofunctors over $\mathbb{D}$. Owing to the separation between the two fragments, we conjecture that our proofs, with straightforward modifications, can be carried over to the new setting. Though we don't

work out the details of adding nontermination to GMCC or GMCC_e, we do allow nontermination in the dependency calculi we design in the next chapter. In particular, we extend SDC, an alternative presentation of $\text{GMCC}_\text{e}(\mathcal{L})$, to pure type systems, which include type systems that allow nonterminating computations.

2.11.2 Algehed's SDCC

Algehed [2018] is similar in spirit to our work. Algehed [2018] designs a calculus, SDCC, that is equivalent to (the terminating fragment of) DCC. SDCC has the same types as DCC. However, it replaces the **bind** construct of DCC with four new constructs: μ, *map*, *up* and *com*. The constructs μ, *map* and *up* serve the same purpose as **join**, **lift** and **up** respectively. The construct *com* is the **flip** operator we discuss in Section 2.3.3. This construct is what separates SDCC from our work. While Algehed [2018] went with *com* and designed a calculus equivalent to DCC, we went with **extr** and **fork** and designed a calculus that is equivalent to an extension of DCC. This choice helped us realize the power of comonadic aspect of dependency analysis.

2.11.3 Relational Semantics of DCC, Revisited

Abadi et al. [1999] present a relational categorical model for $\text{DCC}(\mathcal{L})$. They interpret each of the T_ℓs as a separate monad on a base category, $\mathcal{DC}$. However, their interpretation can also be phrased in terms of a strong monoidal functor from $\mathbf{C}(\mathcal{L})$ to $\mathbf{End}^s_{\mathcal{DC}}$. In other words, the model given by Abadi et al. [1999] is an instance of the general class of models admitted by DCC_e, and hence DCC.

In this regard, we want to point out a technical problem with the category $\mathcal{DC}$. Since $\mathcal{DC}$ models simply-typed λ-calculus, it should be cartesian closed. Abadi et al. [1999] claim that it is so. However, contrary to their claim, the category $\mathcal{DC}$ is not cartesian closed. The exponential object in $\mathcal{DC}$ does not satisfy the universal property, unless the relations in the definition of $\text{Obj}(\mathcal{DC})$ are restricted to reflexive ones only. Appendix A.10 presents a counter-example that shows $\mathcal{DC}$, as presented by Abadi et al. [1999], is not cartesian closed. Appendix A.10 also shows that $\mathcal{DC}$, when restricted as mentioned, becomes cartesian closed. Note that with this restriction, the objects of $\mathcal{DC}$ are nothing but classified sets which Kavvos [2019] uses to build categorical models of DCC based on cohesion. How the cohesion-based models of Kavvos [2019] relate to our graded models of DCC is something that could be an interesting future work.

2.12 Conclusion

Dependency analysis is as much an analysis of dependence as of independence. By controlling the flow of information, it aims to ensure that certain entities are independent of certain other entities while being dependent upon yet other entities. To control flow, it needs to make use of unidirectional devices, devices that allow flow along one direction but block along the other. The two simple yet robust unidirectional devices in programming languages are monads and comonads. Monads allow inflow but block outflow

whereas comonads allow outflow but block inflow. When used together, they ensure that flow respects the constraints imposed upon it by a predetermined structure, like a preordered monoid. Such a controlled flow can be used for a variety of purposes like enforcing security constraints, staging computations, etc.

In this chapter, we went through the nuances of dependency analysis in simple type systems. This chapter acts as the basis for the next one where we shall extend dependency analysis from simple type systems to pure type systems.

Chapter 3

Dependency Analysis in Pure Type Systems

Several simple/polymorphic type systems in literature include dependency analysis in some form. We saw some of these type systems in the last chapter. In the context of dependent type systems, however, dependency analysis has received relatively less attention. In particular, there is no dependently-typed analogue of DCC in literature. This state of affairs is unfortunate because, as we show in this chapter, dependency analysis can provide an elegant foundation for understanding compile-time and run-time irrelevance, two important issues in the design of dependently-typed languages. Recall that compile-time irrelevance identifies sub-expressions that are not needed for type-checking while run-time irrelevance identifies sub-expressions that do not affect the result of evaluation. By ignoring or erasing such sub-expressions, compilers for dependently-typed languages increase the expressiveness of the type system, improve on compilation time and produce more efficient executables.

In this chapter, we address the question of dependency analysis in dependent type systems. We design a language that augments pure type systems with a *primitive* notion of dependency analysis and use it to track compile-time and run-time irrelevance. We call the language DDC, for Dependent Dependency Calculus, in homage to DCC. Although the dependency analyses in DDC are structured differently, we show that DDC can faithfully embed the terminating fragment of DCC and support its many well-known applications, in addition to our novel applications of tracking compile-time and run-time irrelevance.

DDC is an extension of a simply-typed dependency calculus, SDC, which we develop in this chapter. SDC may be seen as an alternative presentation of $\text{GMCC}_{\text{e}}(\mathcal{L})$, where $\mathcal{L}$ is a bounded join-semilattice. We develop this alternative presentation owing to two main reasons. First, this presentation extends smoothly to dependent type systems. Second, it facilitates a straightforward syntactic proof of noninterference that easily generalizes to a dependently-typed setting. On our way to DDC from SDC, we develop an intermediate dependency calculus, called $\text{DDC}^{\top}$, which can analyze run-time irrelevance in dependent type systems. What

makes DDC more advanced than $\text{DDC}^{\top}$ is that DDC also internalizes dependency analysis in typing itself, thereby allowing the calculus to exploit compile-time irrelevance while type-checking.

The main idea behind the design of all the dependency calculi in this chapter may be captured by a slogan: 'Nondependence is unobservability'. To elaborate, every piece of information in these calculi is tagged with a dependency level (i.e., an element of a dependency lattice) that marks its nature. The calculi, then, ensure that such tagged information may only be observed at permissible dependency levels. This simple method is very effective in analyzing dependencies not just in a simply-typed setting but also in a dependently-typed one. This chapter shows the efficacy of the method in analyzing compile-time and run-time irrelevance in dependently-typed languages. Further, the chapter also employs this method to analyze strong Σ-types with erasable first components, a thorny problem that had eluded researchers in this area [Abel and Scherer, 2012, Mishra-Linger, 2008].

One of the key contributions of this chapter is the connection it establishes between compile-time and run-time irrelevance analyses and a general dependency analysis. Though analysis of compile-time and run-time irrelevance in dependently-typed languages is a well-studied topic, to the best of our knowledge, no prior work explicitly draws the connection between these analyses and a general dependency analysis. This connection is significant because it helps us see these analyses in a new light, as standard dependency analysis problems, where relevant expressions may not depend upon irrelevant ones. Soundness of these analyses, then, are just corollaries of the noninterference theorem for general dependency analysis. This statement is true of run-time and compile-irrelevance analyses in DDC.

This chapter focuses on building a general dependent dependency calculus, which is then applied to analyze run-time and compile-time irrelevance in dependent type systems. Owing to its general nature, the calculus may also be applied to analyze dependencies in other areas, like security and metaprogramming. However, we don't explore the details of such applications in this chapter and leave it for future work.

In sum, we make the following contributions in this chapter:

- We design a language SDC, for Simple Dependency Calculus, that can analyze dependencies in a simply-typed language. SDC is equivalent to GMCC_{e}, when parametrized over bounded join-semilattices. The structure of dependency analysis in SDC enables smooth extension to dependent type systems and facilitates a syntactic proof of noninterference that easily generalizes to dependent types.
- We extend SDC to a dependently-typed calculus, $\text{DDC}^{\top}$. We analyze run-time irrelevance using this calculus and show the analysis is sound using the noninterference theorem. $\text{DDC}^{\top}$ contains SDC as a sub-language. As such, it can be used to analyze dependencies other than run-time irrelevance as well.
- We generalize $\text{DDC}^{\top}$ to DDC. We analyze both run-time and compile-time irrelevance using this calculus and show that the analyses are sound. To the best of our knowledge, DDC is the only system that can simultaneously analyze run-time and compile-time irrelevance, while in the presence of strong Σ-types with erasable first components.

3.1 Irrelevance Analysis as a Dependency Analysis

In this section, we show that run-time and compile-time irrelevance analyses in dependent type systems are essentially dependency analyses. Through examples, we bring out the nuances of these analyses and determine the lattice structures necessary for carrying them out. We shall use these lattice structures to parametrize our dependent dependency calculi in the following sections.

3.1.1 Run-Time Irrelevance

Parts of a program that are not required during run time are said to be run-time irrelevant. Our goal is to identify such parts of programs. Let's consider some examples. We shall mark variables and arguments with $\top$ if they can be erased prior to execution and leave them unmarked if they should be preserved. With this convention, the polymorphic identity function can be marked as:

```
id : Π x:⊤Type. x → x
id = λ⊤x. λy. y
```

The first parameter, x, of the identity function is only needed during type-checking; it can be erased before execution. The second parameter, y, though, is required during run time. When we apply this function to arguments, as in `(id Bool⊤ True)`, we can erase the first argument, `Bool`, but the second one, `True`, must be retained.

Indexed data structures provide another example of run-time irrelevance. Consider the `Vec` datatype of length-indexed lists, as it might look in a core language inspired by GHC [Sulzmann et al., 2007, Weirich et al., 2017]. The `Vec` datatype has two parameters, `n` and `a`, that also appear in the types of the data constructors, `Nil` and `Cons`. The parameters `n` and `a` are relevant to `Vec`, but are irrelevant to the data constructors. (Note that in the types of the data constructors, the equality constraints `(n ~ Zero)` and `(n ~ Succ n0)` force `n` to be equal to the length of the vector. Going forward, we shall elide these constraints because they are not central to our discussion on irrelevance.)

```
Vec  : Nat → Type → Type
Nil  : Π n:⊤Nat. Π a:⊤Type.(n ~ Zero)⇒ Vec n a
Cons : Π n:⊤Nat. Π a:⊤Type. Π n0:⊤Nat.(n ~ Succ n0)⇒ a → Vec n0 a → Vec n a
```

Now consider a higher-order function, `vmap`, that maps a given function over a given vector. The length of the vector and the type arguments are not necessary for running `vmap`; they are all erasable. So we mark them with $\top$.

```
vmap : Π n:ᵀNat.Π a b:ᵀType. (a → b) → Vec n a → Vec n b
vmap = λᵀ n a b. λ f xs.
          case xs of
            Nil → Nil
            Cons n0ᵀ x xs → Cons n0ᵀ (f x) (vmap n0ᵀ aᵀ bᵀ f xs)
```

Note that the ⊤-marked variables $\texttt{n}_0$, `a` and `b` appear in the definition of `vmap`, but only in ⊤ contexts. Further, note that even though these arguments are marked with ⊤, which describes their use in the *definition* of `vmap`, this marking does not reflect their use in the *type* of `vmap`. In particular, the variables `a` and `b` appear in relevant positions in the type of `vmap`. To paraphrase this point, in this chapter, the marking on an argument of a Π-type denotes how that argument is used in the body of a function having that type and not how that argument is used in the body of the type itself. We do not track how an argument of a Π-type is used in its body. What's more, we do not even put any constraints as to how an argument of a Π-type may be used in its body.

Now we argue why run-time irrelevance analysis is a dependency analysis. By definition, an expression is run-time irrelevant if it is not required during run time, or in other words, if it does not affect run-time evaluation in any way whatsoever. Therefore, expressions that are run-time relevant cannot depend upon run-time irrelevant expressions. Phrasing this condition in terms of our convention, we may say, 'unmarked' terms *should not depend* upon terms marked with ⊤. This is a standard dependency constraint, modeled by the lattice, $\mathcal{L}_2 = \bot \sqsubseteq \top$ (where ⊥ represents 'unmarked'). In Section 3.3.5, we shall use this lattice to analyze run-time irrelevance in dependent type systems.

3.1.2 Compile-Time Irrelevance

One of the quintessential features of dependent type systems is the conversion rule: if $a : A$ and $A \equiv B$, then $a : B$. Such a rule is not required in standard simple/polymorphic type systems, where the equality relation on types is identity. In dependent type systems, however, such a rule is necessary because here types, like terms, may reduce, giving rise to a nontrivial equality relation. The relation, $A \equiv B$, is usually taken to be the β-equivalence relation on types. Now, with the conversion rule in place, one may need to find out, while type-checking, if $A \equiv B$ holds. One way to find that out is to normalize the types A and B and check for syntactic equality. This procedure involves computations on types. As such, an irrelevance analysis, whereby expressions unnecessary to these computations are erased, could help the procedure. Such expressions, which are irrelevant while types are checked for equality, are called compile-time irrelevant expressions, and are identified by compile-time irrelevance analysis.

Let's consider some examples of compile-time irrelevant expressions. We shall follow the same convention and mark variables and arguments with ⊤ if they can be erased prior to type-checking and leave them unmarked if they should be preserved. With this convention, the `phantom` function, which ignores its argument and just returns the type `Bool`, can be marked as follows. Following standard practice, we abbreviate $\Pi\texttt{x}{:}^{\top}\texttt{A.B}$ to $^{\top}\texttt{A} \to \texttt{B}$, whenever `x` does not appear free in `B`.

```
phantom : ⊤Nat → Type
phantom = λ⊤ x.  Bool
```

Now, to type-check `idp` below, we need to show that `phantom 0` equals `phantom 1`. If we don't take compile-time irrelevance into account, we need to β-reduce the types of both the input and the output of `idp` and show that they are equal. At times, such β-reductions may be costly, as we have already seen in an example in Section 1.2.5. On the other hand, compile-time irrelevance analysis immediately informs us that `phantom 0` $\equiv$ `phantom 1`.

```
idp : phantom 0⊤ → phantom 1⊤
idp = λ x. x
```

Compile-time irrelevance analysis also enables us to use the dependency information contained in the type of a function to reason about it abstractly. For example, in the term `ida` below, the type of the argument `f` informs us that `f` ignores its input; as such, we may equate types, `f 0`$^{\top}$ and `f 1`$^{\top}$, even though we are not aware of the definition of `f`. Note that in the absence of compile-time irrelevance analysis, we cannot type-check `ida`. Therefore, compile-time irrelevance analysis makes type-checking more flexible.

```
ida : Π f :⊤(⊤Nat → Type).  f 0⊤ → f 1⊤
ida = λ⊤ f. λ x. x
```

Compile-time irrelevance analysis also makes type-checking faster, as we have already discussed. Below, we present another example that illustrates this point. Consider the following program that type-checks without compile-time irrelevance analysis but requires the type-checker to show that the 28th Fibonacci number, `fib 28`, computes to `317811`: a computation which may consume considerable time and space.

```
idn : Π f :⊤(⊤Nat → Type). f (fib 28)⊤ → f 317811⊤
idn = λ⊤ f. λ x. x
```

Before moving further, observe that we have used just two annotations on variables and terms, $\top$ and 'unmarked', for analyzing both run-time and compile-time irrelevance. In both the cases, we have used $\top$ to mark irrelevant expressions, while leaving the relevant ones 'unmarked'. Upon considering run-time and compile-time irrelevance together, we see that both arguments that can be erased at run time and arguments that can be safely ignored by the type-checker are marked with $\top$. Such a marking scheme seems to suggest that we can analyze both run-time and compile-time irrelevance using just two dependency levels, $\bot$ and $\top$. However, it turns out that two dependency levels are not enough for the purpose, especially in the presence of strong Σ-types with erasable first components, also known as strong irrelevant Σ-types. Next, we explain the challenge posed by such Σ-types to irrelevance analysis and the way we overcome the challenge.

3.1.3 Strong Irrelevant Σ-types

We begin by exploring what happens when $\top$ and 'unmarked' are used to denote irrelevance and relevance respectively, in the context of simultaneous run-time and compile-time analyses. Consider the type $\Sigma\texttt{m:}^{\top}\texttt{Nat. Vec m a}$, which contains pairs whose first components are marked as irrelevant. This type might be useful, say, for the output of a `filter` function for vectors, where the length of the output vector cannot be calculated statically. If we never need to use this length at run time, then it would be good to mark it with $\top$ so that it is not stored. Note that even if `m` is marked with $\top$, it is safe for `m` to be used in a relevant position in the body of the Σ-type, i.e., in `Vec m a`. The marking on `m` indicates how the first component of a pair having the Σ-type is used, not how `m` itself is used in the body of the type. Though this marking on `m` is fine as far as Σ-formation and Σ-introduction rules are concerned, it poses problem with regard to strong Σ-elimination rules, i.e., projection rules. Below, we explain why.

Given the way we have tracked run-time and compile-time irrelevance till now, marking `m` with $\top$ means that the first component of a pair having the type, $\Sigma\texttt{m:}^{\top}\texttt{Nat. Vec m a}$, is not only run-time irrelevant but also compile-time irrelevant. This results in a significant limitation vis-à-vis strong Σ-types because we would not be able to project the second component of a pair having this type. To elaborate, say we have, $\texttt{ys} : \Sigma\texttt{m} :^{\top}\texttt{Nat. Vec m a}$ and we wish to type-check the second projection, ($\mathbf{proj}_2$ `ys`). The type of ($\mathbf{proj}_2$ `ys`) has to be `Vec (`$\mathbf{proj}_1$ `ys) a`. But the type, `Vec (`$\mathbf{proj}_1$ `ys) a`, is ill-formed because a compile-time irrelevant expression, `(`$\mathbf{proj}_1$ `ys)`, appears in a compile-time relevant position. The term ($\mathbf{proj}_1$ `ys`) is compile-time irrelevant by definition (since $\texttt{ys} : \Sigma\texttt{m:}^{\top}\texttt{Nat. Vec m a}$). And the argument `m` in `Vec m a` is compile-time relevant because otherwise the type-checker would equate `Vec 0 a` with `Vec 1 a`, rendering the length index meaningless.

Therefore, we don't want to mark the first component of a pair having type $\Sigma\texttt{m:Nat. Vec m a}$ with $\top$. However, if we leave it unmarked, we cannot erase it at run time, something we might want to do. A way out of this quandary comes by observing that to erase the first component at run time, we just need to declare it run-time irrelevant, not necessarily compile-time irrelevant too. In other words, the first component of a pair having type $\Sigma\texttt{m:Nat. Vec m a}$ may be compile-time relevant, while being run-time irrelevant. We need a new annotation to mark such terms, which are neither completely irrelevant nor completely relevant. Let's use $\mathcal{C}$ as that new annotation. Terms marked with $\mathcal{C}$ are compile-time relevant but may be run-time irrelevant. Therefore, such terms should not depend upon terms marked with $\top$, which may be both compile-time and run-time irrelevant. Further, 'unmarked' terms, which are both run-time and compile-time relevant, should not depend upon terms marked with $\mathcal{C}$ (which may be run-time irrelevant). These constraints among the three annotations, $\top$, $\mathcal{C}$ and 'unmarked', can be modeled by a three level dependency lattice: $\bot \sqsubseteq \mathcal{C} \sqsubseteq \top$ (where $\bot$ represents 'unmarked'). In Section 3.4, we shall use this lattice, also denoted as $\mathcal{L}_3$, to simultaneously analyze run-time and compile-time irrelevance, in the presence of strong irrelevant Σ-types. Here, we use $\mathcal{L}_3$ to type-check the following `filter` function on vectors that employs strong irrelevant Σ-types.

```
filter : Πn:^⊤Nat.Πa:^⊤Type.(a → Bool) → Vec n a → Σm:^𝒞Nat. Vec m a
filter = λ^⊤ n a. λ f vec.
          case vec of
            Nil → (Zero^𝒞, Nil)
            Cons n_0^⊤ x xs
              | f x  → ((Succ (proj_1 ys))^𝒞, Cons (proj_1 ys)^⊤ x (proj_2 ys))
                          where
                            ys = filter n_0^⊤ a^⊤ f xs
              | _    → filter n_0^⊤ a^⊤ f xs
```

The `filter` function above, written in Haskell-like syntax, filters out the elements of the given vector that do not satisfy the given condition. Eisenberg et al. [2021] observe that, in Haskell, it is important to use projection functions to access the components of the pair that results from the recursive call (i.e., $\mathbf{proj}_1$ `ys` and $\mathbf{proj}_2$ `ys` above) to ensure that the `filter` function is not excessively strict. To elaborate, if `filter` instead used pattern matching to eliminate the pair returned by the recursive call, it would have needed to filter the entire input vector in order to return the first element of the output vector. However, using projection functions, `filter` can return the first element of the output vector by scanning just up to the index where that element appears in the input vector, thereby reaping the benefits of laziness. So, in a lazy language like Haskell, we would want the output type of `filter` to be a strong Σ-type rather than a weak Σ-type.

Next, note that the goal of the `filter` function is to output the filtered vector only and not necessarily its length along with it. However, since the vector type requires length as an index and given that the length of the output vector is not known at compile time, the `filter` function needs an existentially quantified vector type as the type of its output. The output itself, then, is a pair, whose first component is an unintended byproduct. So, we would want that component to be marked as erasable. This makes the output type a strong irrelevant Σ-type. This `filter` function demonstrates the practical utility of strong irrelevant Σ-types because it supports the same run-time behavior of the standard filter function on lists but with more expressive data structures, i.e., vectors.

Now, as motivated in this section, one needs the new annotation, $\mathcal{C}$, to mark strong irrelevant Σ-types, which is what we see in the definition of the `filter` function above. Other than $\mathcal{C}$, the definition also uses the other two annotations, i.e., $\top$ and 'unmarked'. The annotation $\top$ marks arguments that are both run-time and compile-time irrelevant, such as, `n`, `a` and $\mathtt{n}_0$ whereas arguments that are both compile-time and run-time relevant, like `f` and `vec`, are left unmarked. Note that the use of $\top$ and 'unmarked' annotations in the `filter` function resemble their use in the `vmap` function from Section 3.1.1.

Before moving forward, we would like to draw the reader's attention to two important points in the definition of `filter`. First, observe how a $\mathcal{C}$-marked term, $\mathbf{proj}_1$ `ys`, is used in a $\top$ context in `Cons` $(\mathbf{proj}_1\ \mathtt{ys})^\top$ `x` $(\mathbf{proj}_2\ \mathtt{ys})$. Such use is permitted by the dependency structure of lattice $\mathcal{L}_3$. More generally, $\bot$-marked terms can be used in any context and $\mathcal{C}$-marked terms can be used in $\mathcal{C}$ or $\top$ contexts. Second, observe that $\top$-marked arguments, `n` and `a`, appear in relevant positions in the type of `filter`, for example, in `Vec n a`.

These appearances are harmless and do not violate the imposed dependency structure because as we pointed out before, the marking on an argument in a type shows how that argument is used in terms having that type and not how it is used in the body of the type.

In this section, we have shown that run-time and compile-time irrelevance analyses in dependent type systems are essentially dependency analyses. In this section, we have also determined the lattice structures necessary for these dependency analyses. Next, we design our dependency calculi for these analyses. But we shall start with a simply-typed system.

3.2 A Simple Dependency Calculus

We already have a calculus, GMCC_e, for dependency analysis in simple type systems. However, it is hard to see how one might extend this calculus, as presented in Section 2.10.1, to dependent types. So, in this section, we develop an alternative presentation of GMCC_e, over the class of bounded join-semilattices, such that the new presentation smoothly extends to dependent types. We call this new presentation Simple Dependency Calculus or SDC. SDC is parameterized by an arbitrary dependency lattice, following the lattice model of Denning [1976]. Technically, we can parametrize SDC by an arbitrary bounded join-semilattice; however, if we limit ourselves to finite structures, this distinction does not really matter.

Now, SDC has the same types as GMCC_e, i.e., the types of standard λ-calculus and modal types, graded by elements of the parametrizing structure. (We don't have a **Void** type in SDC, but it can be easily added to the calculus.) The terms of SDC are also similar to those of GMCC_e. Like GMCC_e, SDC contains the terms of standard λ-calculus along with constructs for introducing and eliminating the graded modal type. The constructs **lock** and **unlock** of SDC, used to introduce and eliminate the modal type, correspond to **split** and **merge** of GMCC_e, used for the same purpose. However, unlike **merge**, **unlock** is a **let**-binder which is there to help us smoothly extend SDC to dependent types. Next, we present SDC formally.

3.2.1 The Calculus

The grammar and type system of SDC appear in Figure 3.1. As discussed, SDC is an extension of the simply-typed λ-calculus with graded modal types, $S_\ell\, A$, where the grades, denoted by ℓ and m, are elements of the parametrizing lattice, $\mathcal{L}$. The graded modal type $S_\ell\, A$ may be thought of as putting the terms of A in ℓ-secure boxes. This type is introduced using **lock** and eliminated using **unlock**. Further, SDC also employs graded contexts and graded typing judgments, similar to GMCC_e.

The typing judgment of SDC has the form $\Omega \vdash a :^{\ell} A$, which means that "$\ell$ is allowed to observe a" or that "a is visible at ℓ", assuming the variables in Ω are visible at the respective levels they are marked with. The typing rules for SDC appear in Figure 3.1. Most rules are similar to their counterparts in GMCC_e and simply propagate the level of the sub-terms to the derived term.

(Grammar)

$$
\begin{array}{llcl}
\textit{types} & A, B, C & ::= & \textbf{Unit} \mid A \to B \mid A_1 \times A_2 \mid A_1 + A_2 \mid S_\ell\, A \\
\textit{terms} & a, b, c & ::= & x \mid \lambda x{:}A.b \mid b\ a \mid (a, b) \mid \textbf{let}\ (x, y) = a\ \textbf{in}\ b \mid \textbf{unit} \mid \textbf{let unit} = a\ \textbf{in}\ b \mid \\
 & & & \textbf{inj}_1\ a \mid \textbf{inj}_2\ a \mid \textbf{case}\ a\ \textbf{of}\ x_1.b_1; x_2.b_2 \mid \textbf{lock}^\ell\ a \mid \textbf{unlock}^\ell\ x = a\ \textbf{in}\ b \\
\textit{contexts} & \Omega & ::= & \varnothing \mid \Omega, x{:}^\ell A
\end{array}
$$

$\boxed{\Omega \vdash a :^\ell A}$ *(Typing rules)*

$$
\frac{x{:}^{\ell_0} A \in \Omega \qquad \ell_0 \sqsubseteq \ell}{\Omega \vdash x :^\ell A}\ \text{SDC-Var}
\qquad
\frac{\Omega, x{:}^\ell A \vdash b :^\ell B}{\Omega \vdash \lambda x{:}A.b :^\ell A \to B}\ \text{SDC-Lam}
\qquad
\frac{\Omega \vdash b :^\ell A \to B \qquad \Omega \vdash a :^\ell A}{\Omega \vdash b\ a :^\ell B}\ \text{SDC-App}
$$

$$
\frac{}{\Omega \vdash \textbf{unit} :^\ell \textbf{Unit}}\ \text{SDC-Unit}
\qquad
\frac{\Omega \vdash a :^\ell \textbf{Unit} \qquad \Omega \vdash b :^\ell B}{\Omega \vdash \textbf{let unit} = a\ \textbf{in}\ b :^\ell B}\ \text{SDC-LetUnit}
\qquad
\frac{\Omega \vdash a :^\ell A \qquad \Omega \vdash b :^\ell B}{\Omega \vdash (a, b) :^\ell A \times B}\ \text{SDC-Pair}
$$

$$
\frac{\Omega \vdash a :^\ell A \times B \qquad \Omega, x{:}^\ell A, y{:}^\ell B \vdash c :^\ell C}{\Omega \vdash \textbf{let}\ (x, y) = a\ \textbf{in}\ c :^\ell C}\ \text{SDC-LetPair}
\qquad
\frac{\Omega \vdash a_i :^\ell A_i}{\Omega \vdash \textbf{inj}_i\ a_i :^\ell A_1 + A_2}\ \text{SDC-Inj}
$$

$$
\frac{\Omega \vdash a :^\ell A_1 + A_2 \qquad \Omega, x_1{:}^\ell A_1 \vdash b_1 :^\ell B \qquad \Omega, x_2{:}^\ell A_2 \vdash b_2 :^\ell B}{\Omega \vdash \textbf{case}\ a\ \textbf{of}\ x_1.b_1; x_2.b_2 :^\ell B}\ \text{SDC-Case}
$$

$$
\frac{\Omega \vdash a :^{\ell \sqcup \ell_0} A}{\Omega \vdash \textbf{lock}^{\ell_0}\ a :^\ell S_{\ell_0} A}\ \text{SDC-Lock}
\qquad
\frac{\Omega \vdash a :^\ell S_{\ell_0} A \qquad \Omega, x{:}^{\ell \sqcup \ell_0} A \vdash b :^\ell B}{\Omega \vdash \textbf{unlock}^{\ell_0}\ x = a\ \textbf{in}\ b :^\ell B}\ \text{SDC-Unlock}
$$

FIGURE 3.1: Grammar and typing rules of Simple Dependency Calculus

The variable rule SDC-VAR is somewhat different from its GMCC$_\text{e}$-counterpart, rule E-VAR, in that it just requires $\ell_0 \sqsubseteq \ell$ and not necessarily $\ell_0 = \ell$. In some sense, rule SDC-VAR is a fusion of rules E-VAR and E-UP. We shall see that a rule like E-UP is derivable in SDC, thanks to rule SDC-VAR. The intuition behind rule SDC-VAR is that an observer at ℓ may observe any variable with clearance level less than or equal to ℓ. If the clearance level of the variable is higher than or incomparable to ℓ, then this rule does not apply, ensuring that information never flows to less secure levels.

The other rules, barring rules SDC-LOCK and SDC-UNLOCK, are unremarkable from the perspective of dependency analysis. Rule SDC-LOCK, like rule E-SPLIT, puts an $(\ell \sqcup \ell_0)$-secure term in an ℓ_0-secure box and releases it at level ℓ. An ℓ_0-secure boxed term can be unboxed only by someone who has security clearance for ℓ_0, as we see in rule SDC-UNLOCK. The join operation in this rule ensures that b can depend on a only if b itself is ℓ_0-secure or $\ell_0 \sqsubseteq \ell$. Before moving ahead, notice the similarity and difference between

rule SDC-Unlock and rule E-MergeL (rule E-Merge specialized to bounded-join semilattices):

$$\frac{\begin{array}{c}\Omega \vdash a :^{\ell} S_{\ell_0} A \\ \Omega, x :^{\ell \sqcup \ell_0} A \vdash b :^{\ell} B\end{array}}{\Omega \vdash \mathbf{unlock}^{\ell_0}\, x = a \,\mathbf{in}\, b :^{\ell} B}\ \text{SDC-Unlock} \qquad \frac{\Omega \vdash a :^{\ell} S_{\ell_0} A}{\Omega \vdash \mathbf{merge}^{\ell_0}\, a :^{\ell \sqcup \ell_0} A}\ \text{E-MergeL}$$

Both the rules eliminate the graded modal type but they do so in different ways: while rule E-MergeL manipulates the grades to the right of the turnstile, rule SDC-Unlock manipulates the grades to the left of the turnstile. This left manipulation of grades by rule SDC-Unlock provides us the flexibility needed to extend the calculus to a dependently-typed setting, as we elaborate in Section 3.3. This is the main reason why we use SDC, rather than $\mathrm{GMCC_e}$, as the basis for our dependent dependency calculi.

Now, we consider some example derivations in SDC. Given $\mathcal{L}_2 \triangleq \mathbf{L} \sqsubseteq \mathbf{H}$, we can derive the following terms at level $\mathbf{L}$ in $\mathrm{SDC}(\mathcal{L}_2)$:

$$\varnothing \vdash \lambda x.\mathbf{true} :^{\mathbf{L}} S_{\mathbf{H}}\, \mathbf{Bool} \to \mathbf{Bool} \qquad \varnothing \vdash \lambda x.\mathbf{false} :^{\mathbf{L}} S_{\mathbf{H}}\, \mathbf{Bool} \to \mathbf{Bool}$$

Here, $\mathbf{Bool} \triangleq \mathbf{Unit} + \mathbf{Unit}$ and $\mathbf{true} \triangleq \mathbf{inj}_1\, \mathbf{unit}$ and $\mathbf{false} \triangleq \mathbf{inj}_2\, \mathbf{unit}$. Note that a non-constant function from $S_{\mathbf{H}}\, \mathbf{Bool}$ to $\mathbf{Bool}$ at level $\mathbf{L}$ would violate noninterference. However, that is not so at level $\mathbf{H}$. In fact, we can derive the following non-constant functions at level $\mathbf{H}$ in $\mathrm{SDC}(\mathcal{L}_2)$:

$$\varnothing \vdash \lambda x.\mathbf{unlock}^{\mathbf{H}}\, y = x \,\mathbf{in}\, y :^{\mathbf{H}} S_{\mathbf{H}}\, \mathbf{Bool} \to \mathbf{Bool}$$
$$\varnothing \vdash \lambda x.\mathbf{unlock}^{\mathbf{H}}\, y = x \,\mathbf{in}\, \mathbf{not}\, y :^{\mathbf{H}} S_{\mathbf{H}}\, \mathbf{Bool} \to \mathbf{Bool}$$

Here, $\mathbf{not} \triangleq \lambda z.\mathbf{case}\, z \,\mathbf{of}\, z_1.\mathbf{false}; z_2.\mathbf{true}$.
From the above judgments, we can see that the type $S_{\mathbf{H}}\, (S_{\mathbf{H}}\, \mathbf{Bool} \to \mathbf{Bool})$ has four distinct terms at level $\mathbf{L}$ (and also at level $\mathbf{H}$) in $\mathrm{SDC}(\mathcal{L}_2)$, as in $\mathrm{GMCC_e}(\mathcal{L}_2)$.

Next, given any lattice $\mathcal{L}$ and $\ell_1, \ell_2 \in \mathcal{L}$, we can derive the following terms at level $\bot$ in $\mathrm{SDC}(\mathcal{L})$:

$$x :^{\bot} S_{\ell_1 \sqcup \ell_2} A \vdash \mathbf{lock}^{\ell_1}\, \mathbf{lock}^{\ell_2}\, \mathbf{unlock}^{\ell_1 \sqcup \ell_2}\, y = x \,\mathbf{in}\, y :^{\bot} S_{\ell_1} S_{\ell_2} A$$
$$x :^{\bot} S_{\ell_1} S_{\ell_2} A \vdash \mathbf{lock}^{\ell_1 \sqcup \ell_2}\, (\mathbf{unlock}^{\ell_2}\, y_2 = (\mathbf{unlock}^{\ell_1}\, y_1 = x \,\mathbf{in}\, y_1) \,\mathbf{in}\, y_2) :^{\bot} S_{\ell_1 \sqcup \ell_2} A$$

The above terms may be used to define **fork** and **join** operators respectively in $\mathrm{SDC}(\mathcal{L})$. Note that these terms correspond to $x :^{\bot} S_{\ell_1 \sqcup \ell_2} A \vdash \mathbf{split}^{\ell_1} \mathbf{split}^{\ell_2} \mathbf{merge}^{\ell_1 \sqcup \ell_2} x :^{\bot} S_{\ell_1} S_{\ell_2} A$ and $x :^{\bot} S_{\ell_1} S_{\ell_2} A \vdash \mathbf{split}^{\ell_1 \sqcup \ell_2} \mathbf{merge}^{\ell_2} \mathbf{merge}^{\ell_1} x :^{\bot} S_{\ell_1 \sqcup \ell_2} A$ respectively of $\mathrm{GMCC_e}(\mathcal{L})$ (where $\mathcal{L}$ is overloaded to represent the bounded join-semilattice corresponding to the lattice $\mathcal{L}$).

Next, we look at the metatheory of SDC.

3.2.2 Metatheoretic Properties

SDC satisfies several interesting properties with regard to grades. First, we can always make any contextual assumption in a typing judgment more lenient, meaning, if a term is derivable with a contextual assumption held at some grade, then that term is also derivable with that assumption held at any lower grade, as we see in the lemma below. Note that for any two contexts, Ω_1 and Ω_2, we say that $\Omega_1 \sqsubseteq \Omega_2$ if and only if $\lfloor \Omega_1 \rfloor = \lfloor \Omega_2 \rfloor$ and $\overline{\Omega_1} \sqsubseteq \overline{\Omega_2}$. We use $\lfloor \Omega \rfloor$ and $\overline{\Omega}$ to denote the underlying context and the vector of grades for a given graded-context Ω. Further, given two vectors (of grades) of equal length, $\mathbf{u}_1$ and $\mathbf{u}_2$, we say $\mathbf{u}_1 \sqsubseteq \mathbf{u}_2$ if and only if $\mathbf{u}_1(i) \sqsubseteq \mathbf{u}_2(i)$, for all indices i.

Lemma 3.1 (Narrowing) If $\Omega' \vdash a :^{\ell} A$ and $\Omega \sqsubseteq \Omega'$, then $\Omega \vdash a :^{\ell} A$.

The narrowing lemma above allows us to arbitrarily downgrade contextual assumptions. Inversely, we cannot arbitrarily upgrade contextual assumptions, but we can upgrade them to the level of the typing judgment, as shown below.

Lemma 3.2 (Restricted Upgrading) If $\Omega_1, z :^{m_0} C, \Omega_2 \vdash a :^{\ell} A$ and $\ell_0 \sqsubseteq \ell$, then $\Omega_1, z :^{\ell_0 \sqcup m_0} C, \Omega_2 \vdash a :^{\ell} A$.

The restricted upgrading lemma above is a corollary of the following multiplication lemma. Here, the graded-context $m \sqcup \Omega$ is obtained by taking the pointwise join of every grade in Ω with m. Note the similarity between this lemma and the multiplication lemma of GMCC_{e} (Lemma 2.26).

Lemma 3.3 (Multiplication) If $\Omega \vdash a :^{\ell} A$, then $m \sqcup \Omega \vdash a :^{m \sqcup \ell} A$.

Next, we show that SDC satisfies subsumption, meaning, if a term is visible at some grade, then it is also visible at all higher grades. Note that this lemma is akin to the typing rule E-Up of GMCC_{e}.

Lemma 3.4 (Subsumption) If $\Omega \vdash a :^{\ell} A$ and $\ell \sqsubseteq m$, then $\Omega \vdash a :^{m} A$.

The subsumption property is necessary to prove the substitution lemma for SDC. To prove the substitution lemma, we also need a weakening lemma, as shown below. Note that in the substitution lemma, we need to ensure that the level of the variable being substituted matches up with that of the substitute.

Lemma 3.5 (Weakening) If $\Omega_1, \Omega_2 \vdash a :^{\ell} A$, then $\Omega_1, z :^{m} C, \Omega_2 \vdash a :^{\ell} A$.

Lemma 3.6 (Substitution) If $\Omega_1, z :^{m} C, \Omega_2 \vdash a :^{\ell} A$ and $\Omega_1 \vdash c :^{m} C$, then $\Omega_1, \Omega_2 \vdash a\{c/z\} :^{\ell} A$.

Now, let us look at the operational semantics of SDC. We use a call-by-name small-step semantics. An excerpt of the semantics appears in Figure 3.2. The step rules for the terms of standard λ-calculus are standard and as such, are elided. Rules SDCStep-UnlockLeft and SDCStep-UnlockBeta show how the modal terms step. Observe that in rule SDCStep-UnlockBeta, the grades on the introduction form and the elimination form (**lock** and **unlock** respectively) match up. Note that we could have also used a call-by-value reduction strategy for SDC but we chose a call-by-name reduction strategy because it provides stronger guarantees with regard to noninterference (especially in the presence of nonterminating computations), as discussed in Section 3.2.3.

$\boxed{\vdash a \rightsquigarrow a'}$ *(Small step semantics)*

$$\frac{\vdash a \rightsquigarrow a'}{\vdash \mathbf{unlock}^{\ell}\ x = a\ \mathbf{in}\ b \rightsquigarrow \mathbf{unlock}^{\ell}\ x = a'\ \mathbf{in}\ b}\ \text{SDCStep-UnlockLeft}$$

$$\frac{}{\vdash \mathbf{unlock}^{\ell}\ x = \mathbf{lock}^{\ell}\ a\ \mathbf{in}\ b \rightsquigarrow b\{a/x\}}\ \text{SDCStep-UnlockBeta}$$

FIGURE 3.2: Operational semantics of SDC (Excerpt)

Given this operational semantics, SDC enjoys a standard preservation and progress based type soundness property shown below.

Theorem 3.7 (Preservation) If $\Omega \vdash a :^{\ell} A$ and $\vdash a \rightsquigarrow a'$, then $\Omega \vdash a' :^{\ell} A$.

Theorem 3.8 (Progress) If $\varnothing \vdash a :^{\ell} A$, then either a is a value or there exists a' such that $\vdash a \rightsquigarrow a'$.

Next, we show that SDC satisfies noninterference. Observe that the standard type soundness theorems above are not strong enough to show noninterference because the operational semantics we use does not really track dependencies. So to prove noninterference, we need to devise an alternative method. The method we devise, presented next, is essentially a syntactic formulation of the presence-absence test described in the last chapter.

3.2.3 A Syntactic Proof of Noninterference

Recall that when observers with low-security clearance are oblivious to high-security data, we say that the system enjoys *noninterference*. Noninterference is a consequence of observer-appropriate views of the world of data around them. For example, the terms $\mathbf{lock}^{\mathbf{H}}\ \mathbf{true}$ and $\mathbf{lock}^{\mathbf{H}}\ \mathbf{false}$ should appear the same to an observer at level $\mathbf{L}$ whereas an observer at level $\mathbf{H}$ should be able to differentiate between them. To capture this notion of observer-appropriate views, we design an indexed equivalence relation on open terms, called *indexed indistinguishability*, and denoted as $a \sim_{\ell} b$, with the intuitive meaning that a and b are indistinguishable to observers at level ℓ. An excerpt of the rules generating this relation appears in Figure 3.3. Note that to define this relation, we need to know the levels of the free variables appearing in the terms; however, we don't need to know the types of those variables. So, we define grade-only contexts, denoted by Φ, and defined as:

$$\Phi ::= \varnothing \mid \Phi, x : \ell$$

These contexts are like graded contexts, Ω, but without the type information on variables. We shall use $\lceil \Omega \rceil$ to denote the grade-only context corresponding to Ω. Next, we motivate the rules that generate the indexed indistinguishability relation.

Intuitively speaking, $\Phi \vdash a \sim_{\ell} b$ means that a and b appear the same to an observer at level ℓ. For example, $\mathbf{lock}^{\mathbf{H}}\ \mathbf{true} \sim_{\mathbf{L}} \mathbf{lock}^{\mathbf{H}}\ \mathbf{false}$ but $\neg(\mathbf{lock}^{\mathbf{H}}\ \mathbf{true} \sim_{\mathbf{H}} \mathbf{lock}^{\mathbf{H}}\ \mathbf{false})$. The relation $\sim_{\ell}$ is defined by structural induction on terms. We think of terms as abstract syntax trees, where certain nodes are annotated with labels that determine whether an observer from a given level is allowed to inspect the sub-trees rooted at those nodes. If the label on a node is ℓ_0 and the observer is from level ℓ, and we have $\ell_0 \sqsubseteq \ell$, then that

$\boxed{\Phi \vdash a \sim_\ell b}$ *(Indexed indistinguishability)*

$$\textsc{Ind-Var}\quad \frac{x : \ell_0 \in \Phi \qquad \ell_0 \sqsubseteq \ell}{\Phi \vdash x \sim_\ell x}$$

$$\textsc{Ind-Lam}\quad \frac{\Phi, x : \ell \vdash b_1 \sim_\ell b_2}{\Phi \vdash \lambda x : A.b_1 \sim_\ell \lambda x : A.b_2}$$

$$\textsc{Ind-App}\quad \frac{\Phi \vdash b_1 \sim_\ell b_2 \qquad \Phi \vdash a_1 \sim_\ell a_2}{\Phi \vdash b_1\ a_1 \sim_\ell b_2\ a_2}$$

$$\textsc{Ind-Pair}\quad \frac{\Phi \vdash a_1 \sim_\ell a_2 \qquad \Phi \vdash b_1 \sim_\ell b_2}{\Phi \vdash (a_1, b_1) \sim_\ell (a_2, b_2)}$$

$$\textsc{Ind-LetPair}\quad \frac{\Phi \vdash a_1 \sim_\ell a_2 \qquad \Phi, x : \ell, y : \ell \vdash c_1 \sim_\ell c_2}{\Phi \vdash \mathbf{let}\ (x, y)\ =\ a_1\ \mathbf{in}\ c_1 \sim_\ell \mathbf{let}\ (x, y)\ =\ a_2\ \mathbf{in}\ c_2}$$

$$\textsc{Ind-Lock}\quad \frac{\Phi \vdash_\ell^{\ell_0} a_1 \sim a_2}{\Phi \vdash \mathbf{lock}^{\ell_0}\ a_1 \sim_\ell \mathbf{lock}^{\ell_0}\ a_2}$$

$$\textsc{Ind-Unlock}\quad \frac{\Phi \vdash a_1 \sim_\ell a_2 \qquad \Phi, x : \ell_0 \sqcup \ell \vdash b_1 \sim_\ell b_2}{\Phi \vdash \mathbf{unlock}^{\ell_0}\ x = a_1\ \mathbf{in}\ b_1 \sim_\ell \mathbf{unlock}^{\ell_0}\ x = a_2\ \mathbf{in}\ b_2}$$

$\boxed{\Phi \vdash_\ell^{\ell_0} a_1 \sim a_2}$

$$\textsc{CInd-Leq}\quad \frac{\ell_0 \sqsubseteq \ell \qquad \Phi \vdash a_1 \sim_\ell a_2}{\Phi \vdash_\ell^{\ell_0} a_1 \sim a_2}$$

$$\textsc{CInd-Nleq}\quad \frac{\neg(\ell_0 \sqsubseteq \ell)}{\Phi \vdash_\ell^{\ell_0} a_1 \sim a_2}$$

FIGURE 3.3: Indexed indistiguishability relation for SDC (Excerpt)

observer may inspect the sub-tree rooted at that node; otherwise the entire sub-tree would appear as a blob to that observer. So, we can also read $\Phi \vdash a \sim_\ell b$ as: 'a and b are syntactically equal at all parts of the terms other than the ones marked with ℓ_0, where they may be arbitrarily different, provided $\neg(\ell_0 \sqsubseteq \ell)$.' The rule IND-LOCK in Figure 3.3 elucidates this point. If $\neg(\ell_0 \sqsubseteq \ell)$, then for any a_1 and a_2, the terms $\mathbf{lock}^{\ell_0}\ a_1$ and $\mathbf{lock}^{\ell_0}\ a_2$ would appear the same to an observer at level ℓ. Otherwise, they would appear the same only when a_1 and a_2 themselves appear the same. The auxiliary judgment, $\Phi \vdash_\ell^{\ell_0} a_1 \sim a_2$, also referred to as conditional indistinguishability relation, ensures that this is indeed the case.

Note the similarity between the indistinguishability relation at ℓ and the strong monoidal functor, $\mathbf{S}^\ell$, from Section 2.8.2. While the latter blacks out all information from levels ℓ_0, where $\neg(\ell_0 \sqsubseteq \ell)$, the former homogenizes all such information. But in effect, both $\sim_\ell$ and $\mathbf{S}^\ell$ model the same thing: the view of an observer at level ℓ. However, the benefit of using the indexed indistinguishability relation is that being of a syntactic nature, it smoothly generalizes to an arbitrary pure type system. On the other hand, designing equivalent strong monoidal functors for an arbitrary pure type system is challenging owing to the complexity of categorical models for dependent type systems.

Next, observe that though rule IND-LOCK needs to treat the cases $(\ell_0 \sqsubseteq \ell)$ and $\neg(\ell_0 \sqsubseteq \ell)$ differently, rule IND-UNLOCK does not need to make any such distinctions. This is so because rule IND-UNLOCK, similar to rule SDC-UNLOCK, realizes its objective by manipulating the grade to the left of the turnstile

in one of its premise judgments. Further, observe that barring rule Ind-Lock, all the rules for indexed indistinguishability relation simply mirror the corresponding typing rules. Indeed, we can show that:

Lemma 3.9 (Typing implies grading) If $\Omega \vdash a :^{\ell} A$ then $\lceil \Omega \rceil \vdash a \sim_{\ell} a$.

The judgment $\lceil \Omega \rceil \vdash a \sim_{\ell} a$ is sometimes abbreviated to $\lceil \Omega \rceil \vdash a : \ell$, and is referred to as a grading judgment. A term a for which $\lceil \Omega \rceil \vdash a : \ell$ holds is referred to as a well-graded term at level ℓ. Then, the above lemma may be restated as: well-typed terms are well-graded. The next lemma shows that indistinguishable terms themselves are also well-graded.

Lemma 3.10 (Indistinguishable terms are well-graded) If $\Phi \vdash a_1 \sim_{\ell} a_2$, then $\Phi \vdash a_1 : \ell$ and $\Phi \vdash a_2 : \ell$.

Now, we explore some properties of the indistinguishability relation. The following lemma shows that indistinguishability is an equivalence relation. Observe that at the highest element of the lattice, $\top$, this equivalence degenerates to the identity relation.

Lemma 3.11 (Equivalence) Indexed indistinguishability at ℓ is an equivalence relation on well-graded terms at ℓ.

The indistinguishability relation satisfies a substitution lemma like the one shown below.

Lemma 3.12 (Indistinguishability under substitution) If $\Phi_1, z : m, \Phi_2 \vdash a_1 \sim_{\ell} a_2$ and $\Phi_1 \vdash^{m}_{\ell} c_1 \sim c_2$, then $\Phi_1, \Phi_2 \vdash a_1\{c_1/z\} \sim_{\ell} a_2\{c_2/z\}$.

We explain the significance of this lemma through an example. Consider the statement of the lemma under the assumption that $\neg(m \sqsubseteq \ell)$. To be more specific, say $\ell = \mathbf{L}$ and $m = \mathbf{H}$. Then, for any two terms c_1 and c_2, by rule CInd-Nleq, we have: $\varnothing \vdash^{\mathbf{H}}_{\mathbf{L}} c_1 \sim c_2$. Now, suppose $z :^{\mathbf{H}} C \vdash b :^{\mathbf{L}} \mathbf{Bool}$. Then, by lemma 3.9, we have: $z : \mathbf{H} \vdash b \sim_{\mathbf{L}} b$. And by the above lemma, we have: $\varnothing \vdash b\{c_1/z\} \sim_{\mathbf{L}} b\{c_2/z\}$. This is almost noninterference in action. What's left to show is that the small-step semantics preserves the indistinguishability relation, as stated below.

Theorem 3.13 (Noninterference) If $\Phi \vdash a_1 \sim_{\ell} a_2$ and $\vdash a_1 \leadsto a_1'$, then there exists a_2' such that $\vdash a_2 \leadsto a_2'$ and $\Phi \vdash a_1' \sim_{\ell} a_2'$.

Further to the above statement, note that since the step relation is deterministic, there is exactly one such a_2' that a_2 steps to. In other words, by the above theorem, ℓ-indistinguishable terms remain ℓ-indistinguishable as they reduce. Applying this theorem to the last example, we see that $b\{c_1/z\}$ and $b\{c_2/z\}$ take steps in tandem and are $\mathbf{L}$-indistinguishable after each and every step. Now, given that SDC itself is a terminating calculus, both the terms reduce to boolean values, values that are themselves $\mathbf{L}$-indistinguishable as well. But the indistinguishability relation on boolean values is just identity. Hence, $b\{c_1/z\}$ and $b\{c_2/z\}$ reduce to the same value, thereby showing that high-security inputs do not interfere in low-security outputs.

Observe that the above theorem guarantees a very strong form of noninterference. The standard noninterference theorem just requires that high-security inputs do not interfere in low-security outputs, meaning, a program that outputs low-security values, when fed with two different high-security inputs, either diverges in both cases or produces the same value in both cases. The above theorem goes a step further and shows that

high-security inputs also do not affect the number of steps in reductions that produce low-security values, meaning, a program that outputs low-security values, when fed with two different high-security inputs, either diverges in both cases or produces the same value *in the same number of steps* in both cases.

In this regard, it's worth noting that this strong form of noninterference holds for call-by-name reduction but not for call-by-value reduction. To see why, consider the function, $\varnothing \vdash \lambda x.\mathbf{true} :^{\mathbf{L}} S_{\mathbf{H}}\,\mathbf{Bool} \to \mathbf{Bool}$, which outputs **true**, no matter what high-security argument it is applied to. In case of call-by-value reduction, however, the number of steps taken to produce the output depends upon the argument, for example, $(\lambda x.\mathbf{true})\ (\mathbf{lock}^{\mathbf{H}}\ \mathbf{true})$ outputs **true** in a single step but $(\lambda x.\mathbf{true})\ ((\lambda y.y)\ (\mathbf{lock}^{\mathbf{H}}\ \mathbf{true}))$ does so in two steps. Taking this argument forward, if the calculus concerned is not normalizing, the most one can guarantee in case of call-by-value reduction is termination-insensitive noninterference, i.e., noninterference ignoring termination behavior. But in case of call-by-name reduction, we can guarantee the above-mentioned strong form of noninterference even when the calculus concerned is not normalizing, as we see from Theorem 3.27.

Now that we have shown SDC enjoys noninterference, we compare SDC with standard dependency calculi from literature, DCC and sealing calculus. We know that the sealing calculus subsumes the terminating fragment of DCC. In the following section, we show that SDC is no less expressive than the sealing calculus, and therefore, the terminating fragment of DCC.

3.2.4 Relation with Sealing Calculus

SDC is very similar to the sealing calculus, $\lambda^{[]}$, of Shikuma and Igarashi [2006]. Like SDC, $\lambda^{[]}$ uses graded typing judgments. But unlike SDC, $\lambda^{[]}$ uses standard ungraded typing contexts, Γ. Both the calculi have the same types. As far as terms are concerned, there is only one difference. The sealing calculus uses a **merge**-like construct called **unseal** to eliminate the graded modal type, whereas SDC uses **unlock**. Below, we present the rules for sealing and unsealing terms in $\lambda^{[]}$. We take the liberty of making small cosmetic changes to the presentation of Shikuma and Igarashi [2006].

$$\textsc{SC-Seal}\ \frac{\Gamma \vdash a :^{\ell \sqcup \ell_0} A}{\Gamma \vdash \mathbf{seal}^{\ell_0}\ a :^{\ell} S_{\ell_0} A} \qquad \textsc{SC-Unseal}\ \frac{\Gamma \vdash a :^{\ell} S_{\ell_0} A \qquad \ell_0 \sqsubseteq \ell}{\Gamma \vdash \mathbf{unseal}^{\ell_0} a :^{\ell} A}$$

Now, we define a translation $\overline{}$, from $\lambda^{[]}$ to SDC. Most of the cases are handled inductively in a straightforward manner. For **seal** and **unseal**, we have, $\overline{\mathbf{seal}^{\ell_0}\ a} = \mathbf{lock}^{\ell_0}\ \overline{a}$ and $\overline{\mathbf{unseal}^{\ell_0} a} = \mathbf{unlock}^{\ell_0}\ x = \overline{a}\,\mathbf{in}\,x$.

With this translation, we can provide a forward and a backward simulation connecting the two languages. The reduction relation below, $\longmapsto$, is full reduction for both the languages, the reduction strategy used by Shikuma and Igarashi [2006] for $\lambda^{[]}$. Recall that full reduction is a non-deterministic reduction strategy whereby a β-redex in any sub-term may be reduced.

Theorem 3.14 (Forward Simulation) If $\vdash a \longmapsto a'$ in $\lambda^{[]}$, then $\vdash \overline{a} \longmapsto \overline{a'}$ in SDC.

Theorem 3.15 (Backward Simulation) For any term a in $\lambda^{[]}$, if $\vdash \overline{a} \longmapsto b$ in SDC, then there exists a' in $\lambda^{[]}$ such that $\vdash a \longmapsto a'$ and $\overline{a'} = b$.

The translation from $\lambda^{[]}$ to SDC also preserves typing. In fact, a source term and its target have the same type. Below, for an ordinary context Γ, the graded-context Γ^{ℓ} denotes Γ with the grade for every assumption set to ℓ.

Theorem 3.16 (Translation Preserves Typing) If $\Gamma \vdash a :^{\ell} A$, then $\Gamma^{\ell} \vdash \overline{a} :^{\ell} A$.

The above translation shows that the sealing calculus can be embedded into SDC. The straightforward nature of this translation shows the closeness of the two calculi. Now, given this closeness, the reader may wonder: why do we need a new dependency calculus at all? Could we not have just used the sealing calculus as the basis of our dependent dependency calculi? The answer to this question lies in the key difference between the sealing calculus and SDC: SDC, unlike sealing calculus, uses graded contexts. The use of graded contexts in SDC has several benefits:

- In Section 3.2.3, we saw how graded contexts help us prove noninterference. In the absence of graded contexts, a proof of noninterference, while possible, is substantially harder, as can be seen from the long proof of noninterference for sealing calculus in Shikuma and Igarashi [2006].
- In Section 3.4, we shall see how graded contexts help us impose different restrictions on the same assumption in different premise judgments of a typing rule. This flexibility offered by graded contexts is crucial to internalizing dependency analysis in the type system.
- In Chapter 5, we shall see that graded contexts help us unify dependency and linearity analyses. For linearity analysis, especially in dependent type systems, we need to use graded contexts, owing to the reasons pointed out in Section 1.1.8. Now, if we analyze dependencies using graded contexts as well, we are at an advantage when it comes to unifying the two analyses.

So, the use of graded contexts endows SDC with several desirable features missing in the sealing calculus. That is why we use SDC as our base as we extend dependency analyses to dependent type systems.

3.3 Dependent Dependency Calculi

In this section and the next one, we present calculi that extend dependency analysis à la SDC to dependent types. The first extension, called $\text{DDC}^{\top}$, is a straightforward integration of dependency analysis and dependent types. $\text{DDC}^{\top}$ can be used to analyze dependencies in general, including *run-time irrelevance*. Thereafter, in Section 3.4, we generalize $\text{DDC}^{\top}$ to DDC, which internalizes dependency analysis in typing. Internalized dependency analysis enables DDC to take advantage of *compile-time irrelevance* in the type system itself. Both $\text{DDC}^{\top}$ and DDC are dependent dependency calculi but owing to internalized dependency analysis, DDC is the more general dependency calculus. As such, we could have presented DDC without

Type System	$\mathcal{S}$	$\mathcal{A}$	$\mathcal{R}$
Simply-Typed λ-Calculus	$\{*, \Box\}$	$\{(*, \Box)\}$	$\{(*, *, *)\}$
Polymorphic λ-Calculus	$\{*, \Box\}$	$\{(*, \Box)\}$	$\{(*, *, *), (\Box, *, *)\}$
λP	$\{*, \Box\}$	$\{(*, \Box)\}$	$\{(*, *, *), (*, \Box, \Box)\}$
Calculus of Constructions	$\{*, \Box\}$	$\{(*, \Box)\}$	$\{(s_1, s_2, s_2) \mid s_1, s_2 \in \mathcal{S}\}$
Type-in-Type	$\{*\}$	$\{(*, *)\}$	$\{(*, *, *)\}$

Table 3.1: Well-known type systems seen through PTS formalism

$$
\begin{array}{llll}
a, A, b, B, c, C & ::= & s \mid x \mid \mathbf{Unit} \mid \mathbf{unit} \mid \mathbf{let\ unit} = a \mathbf{\ in\ } b \mid & \textit{sorts, variables and unit} \\
& & \Pi x :^{\ell} A.B \mid \lambda x :^{\ell} A.b \mid a\ b^{\ell} \mid & \textit{dependent functions} \\
& & \Sigma x{:}^{\ell} A.B \mid (a^{\ell}, b) \mid \mathbf{let}\ (x^{\ell}, y)\ =\ a \mathbf{\ in\ } b \mid & \textit{dependent pairs} \\
& & A + B \mid \mathbf{inj}_1\ a \mid \mathbf{inj}_2\ a \mid \mathbf{case}\ a \mathbf{\ of\ } x_1.b_1 ; x_2.b_2 & \textit{disjoint sums}
\end{array}
$$

Figure 3.4: Grammar of Dependent Dependency Calculus (Types and terms)

first presenting DDC$^{\top}$. However, we chose this style of presentation because DDC$^{\top}$ is an interesting calculus in its own right.

Both DDC$^{\top}$ and DDC are generic pure type systems, parametrized over an arbitrary lattice. Recall that a pure type system (PTS) [Barendregt, 1993] is a type system characterized by a triple $(\mathcal{S}, \mathcal{A}, \mathcal{R})$, where $s \in \mathcal{S}$ are sorts, $\mathcal{A} \subseteq \mathcal{S} \times \mathcal{S}$ is a set of axioms and $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S} \times \mathcal{S}$ is a set of rules. The triple $(\mathcal{S}, \mathcal{A}, \mathcal{R})$, when instantiated appropriately, results in well-known type systems, such as, Simply-Typed λ-Calculus, Polymorphic λ-Calculus, λP, Calculus of Constructions, Type-in-Type, etc. Table 3.1 shows the appropriate instantiations for the mentioned type systems. Note that a PTS need not be normalizing: among the PTSs displayed in Table 3.1, the Type-in-Type system is a PTS that allows nonterminating computations.

PTSs use the same syntactic category for both types and terms. As presented in Barendregt [1993], the grammar of a PTS contains sorts, variables, function types, λ-abstractions and applications. But in the PTSs we design in this dissertation, we extend the grammar with pair types, disjoint unions and a Unit type along with the introduction and elimination forms for these types. Furthermore, the types and terms in our PTSs are also graded. Next, we look at the graded types and terms of the two PTSs we design in this chapter, DDC$^{\top}$ and DDC.

3.3.1 DDC$^{\top}$ and DDC: Graded Types and Terms

Both DDC$^{\top}$ and DDC share the same syntax, shown in Figure 3.4. Some types are annotated with grades to facilitate dependency analysis. The grade ℓ on the dependent function type, $\Pi x :^{\ell} A.B$, denotes the minimum clearance necessary for observing the argument x in the body of a function having this type. Similarly, the grade ℓ on the dependent pair type, $\Sigma x{:}^{\ell} A.B$, denotes the minimum clearance necessary for observing the first component of a pair having this type. We can interpret these types as a fusion of the graded modality, $S_{\ell}\, A$,

$\boxed{\Omega \vdash a :^{\ell} A}$ *(Typing)*

$$\text{DCT-Var}\quad \frac{\Omega \vdash A :^{\top} s \qquad \ell_0 \sqsubseteq \ell}{\Omega, x :^{\ell_0} A \vdash x :^{\ell} A} \qquad \text{DCT-Weak}\quad \frac{\Omega \vdash a :^{\ell} A \qquad \Omega \vdash B :^{\top} s}{\Omega, y :^{\ell_0} B \vdash a :^{\ell} A} \qquad \text{DCT-Axiom}\quad \frac{(s_1, s_2) \in \mathcal{A}}{\varnothing \vdash s_1 :^{\ell} s_2}$$

$$\text{DCT-Pi}\quad \frac{\Omega \vdash A :^{\ell} s_1 \qquad \Omega, x :^{\ell} A \vdash B :^{\ell} s_2 \qquad (s_1, s_2, s_3) \in \mathcal{R}}{\Omega \vdash \Pi x :^{\ell_0} A.B :^{\ell} s_3} \qquad \text{DCT-Lam}\quad \frac{\Omega, x :^{\ell \sqcup \ell_0} A \vdash b :^{\ell} B \qquad \Omega \vdash (\Pi x :^{\ell_0} A.B) :^{\top} s}{\Omega \vdash \lambda x :^{\ell_0} A.b :^{\ell} \Pi x :^{\ell_0} A.B} \qquad \text{DCT-App}\quad \frac{\Omega \vdash b :^{\ell} \Pi x :^{\ell_0} A.B \qquad \Omega \vdash a :^{\ell \sqcup \ell_0} A}{\Omega \vdash b\ a^{\ell_0} :^{\ell} B\{a/x\}}$$

$$\text{DCT-Conv}\quad \frac{\Omega \vdash a :^{\ell} A \qquad \lceil \Omega \rceil \vdash A \equiv_{\top} B \qquad \Omega \vdash B :^{\top} s}{\Omega \vdash a :^{\ell} B}$$

FIGURE 3.5: DDC$^{\top}$ type system (Core rules)

and their standard, ungraded counterparts. More concretely, $\Pi x :^{\ell} A.B$ and $\Sigma x{:}^{\ell} A.B$ may be interpreted as $\Pi y : (S_{\ell}\, A).\mathbf{unlock}^{\ell}\, x = y\, \mathbf{in}\, B$ and $\Sigma y : (S_{\ell}\, A).\mathbf{unlock}^{\ell}\, x = y\, \mathbf{in}\, B$ respectively (assuming y is fresh in both cases). Because of this fusion, we do not need to add graded modal types separately to the language—we can encode types $S_{\ell}\, A$ as $\Sigma x{:}^{\ell} A.\mathbf{Unit}$ (assuming x is fresh).

The above discussion suggests that we could alternatively work with standard Π-types and Σ-types along with a graded modality. This is true in case of DDC$^{\top}$, where both the approaches lead to equivalent systems. However, in general, the alternative approach is less flexible than our approach of using graded Π-types and Σ-types. To see why, consider the types $\Pi x :^{\ell} A.B$ and $\Pi y : (S_{\ell}\, A).\mathbf{unlock}^{\ell}\, x = y\, \mathbf{in}\, B$: the grade ℓ in $\Pi x :^{\ell} A.B$ does not play any role in type-checking B but the grade ℓ in $\Pi y : (S_{\ell}\, A).\mathbf{unlock}^{\ell}\, x = y\, \mathbf{in}\, B$ might restrict the well-typedness of $\mathbf{unlock}^{\ell}\, x = y\, \mathbf{in}\, B$. In Section 3.4.2, we shall discuss this difference in more detail and explain why the flexibility offered by our approach is essential to designing DDC.

In line with the graded types, the terms introducing and eliminating these types are also graded. Grades on terms help us track dependency levels of arguments and variables. For example, the grade ℓ on $\lambda x :^{\ell} A.b$ indicates that the variable x may be used only in m contexts, where $\ell \sqsubseteq m$. Similarly, the grade ℓ on $a\ b^{\ell}$ indicates that the term b is visible only at levels m, where $\ell \sqsubseteq m$. The type system ensures that this is indeed the case.

Now, DDC$^{\top}$ and DDC share the same grammar but have different type systems. In this section, we present the type system of DDC$^{\top}$. We shall present the type system of DDC in the following section.

3.3.2 $\text{DDC}^{\top}$: Π-Types

Similar to SDC, the type system of $\text{DDC}^{\top}$ uses graded contexts and graded typing judgments. The core typing rules of $\text{DDC}^{\top}$ appear in Figure 3.5. Below, we motivate these rules. Rule DCT-Var is similar to its counterpart in SDC: the observer must have the necessary clearance to observe the data represented by the variable. Observe that in this rule, the type A is checked at $\top$. In $\text{DDC}^{\top}$, $\top$ corresponds to the compile-time level, where well-formed types reside. We shall see that this is not the case in DDC, where well-formed types reside at a level different from $\top$. This is the main difference between the two calculi and it motivates the superscript $\top$ in '$\text{DDC}^{\top}$'. Next, we have rule DCT-Weak for adding extra assumptions to the context. Note that rule DCT-Weak is necessary because rule DCT-Var allows extra assumptions only to the left of the variable that gets type-checked. This manner of presentation, whereby an explicit weakening rule complements the variable rule, is standard for pure type systems [Barendregt, 1993]. Next, rule DCT-Axiom allows all observers to make use of the axioms. Note that the set $\mathcal{A}$ in this rule varies depending upon the PTS under consideration, as can be seen from Table 3.1.

Coming to Π-types, rule DCT-Pi checks the domain, A, and the codomain, B, at the same level as the Π-type. Observe that the assumption, $x : A$, is also held at this level while checking B, thereby ensuring that B can use this assumption without any dependency constraints. Further, note the use of the set $\mathcal{R}$ in this rule, which varies depending upon the PTS under consideration. Finally, note that the type itself is annotated with an arbitrary label, ℓ_0: as discussed, the purpose of ℓ_0 is to denote the minimum clearance at which the argument x may be used in the body of a function having this type. In line with this setup, in rule DCT-Lam, the parameter of the function is introduced into the context at level $\ell_0 \sqcup \ell$, akin to rule SDC-Unlock. Conversely, in rule DCT-App, the argument to the function is checked at level $\ell_0 \sqcup \ell$, akin to rule SDC-Lock. Note here how rule DCT-Lam smoothly fuses rule SDC-Unlock with a standard abstraction rule. Such a smooth fusion would not have been possible if rule E-MergeL were used in place of rule SDC-Unlock. This is the reason why we prefer rule SDC-Unlock to rule E-MergeL, as indicated in Section 3.2.1.

Next, we look at the conversion rule. Rule DCT-Conv converts the type of an expression to an equivalent type. The judgment, $\lceil \Omega \rceil \vdash A \equiv_{\top} B$, used in this rule is a label-indexed definitional equality relation instantiated to $\top$. This relation is the closure of the indexed indistinguishability relation (presented for the simply-typed system in Section 3.2.3 and for the dependently-typed system in Section 3.3.5) under small-step call-by-name β-reduction. However, when instantiated to $\top$, the relation degenerates to β-equivalence (because the indexed indistinguishability relation at $\top$ is just identity). So, rule DCT-Conv is essentially casting a term to a β-equivalent type. In the next section, we shall utilize the flexibility of label-indexing to cast a term to a type that may not be β-equivalent.

Note here that we use a call-by-name reduction strategy for both $\text{DDC}^{\top}$ and DDC, as we do for SDC. The small-step reduction relation, $\rightsquigarrow$, for these calculi are standard: however, the β-rules need to ensure that the grades on the introduction form and the corresponding elimination form match up, as in rule SDCStep-UnlockBeta, shown in Figure 3.2. Next, we shall move ahead with the typing rules for Σ-types and sum types. But at this point, readers may want to convince themselves that the examples on run-time irrelevance

presented in Section 3.1.1 can be expressed in DDC$^{\top}$, extended with the necessary type constructors and data constructors.

3.3.3 DDC$^{\top}$: Σ-Types and Sum Types

DDC$^{\top}$ includes Σ-types, with typing rules as shown below.

DCT-Sigma

$$\frac{\Omega \vdash A :^{\ell} s_1 \qquad \Omega, x :^{\ell} A \vdash B :^{\ell} s_2 \qquad (s_1, s_2, s_3) \in \mathcal{R}}{\Omega \vdash \Sigma x{:}^{\ell_0} A.B :^{\ell} s_3}$$

DCT-Pair

$$\frac{\Omega \vdash a :^{\ell \sqcup \ell_0} A \qquad \Omega \vdash b :^{\ell} B\{a/x\} \qquad \Omega \vdash \Sigma x{:}^{\ell_0} A.B :^{\top} s}{\Omega \vdash (a^{\ell_0}, b) :^{\ell} \Sigma x{:}^{\ell_0} A.B}$$

Like Π-types, Σ-types include a grade that is not related to how the bound variable is used in the body of the type, as we see in rule DCT-Sigma. The grade annotating the Σ-type indicates the level at which the first component of a pair having that type may be used. In line with this setup, in rule DCT-Pair, the first component, a, is checked at a level raised by ℓ_0, the level annotating the type, akin to rule SDC-Lock.

Next, we look at the elimination rule for pairs. Rule DCT-LetPair, shown below, eliminates pairs via pattern-matching. The body, c, is checked after introducing the pattern variables, x and y, into the context. The level of the first pattern variable, x, is raised by ℓ_0, akin to rule SDC-Unlock. Observe that the result type, C, is refined by the pattern-match, informing the type system that the pattern (x^{ℓ_0}, y) is equal to the scrutinee, a.

DCT-LetPair

$$\frac{\Omega \vdash a :^{\ell} \Sigma x{:}^{\ell_0} A.B \qquad \Omega, x :^{\ell \sqcup \ell_0} A, y :^{\ell} B \vdash c :^{\ell} C\{(x^{\ell_0}, y)/z\} \qquad \Omega, z :^{\top} (\Sigma x{:}^{\ell_0} A.B) \vdash C :^{\top} s}{\Omega \vdash \mathbf{let}\ (x^{\ell_0}, y) = a\ \mathbf{in}\ c :^{\ell} C\{a/z\}}$$

Because of this refinement in the result type, we can define the projection operations using the above elimination form. In particular, the first projection, $\mathbf{proj}_1^{\ell_0}\ a \triangleq \mathbf{let}\ (x^{\ell_0}, y) = a\ \mathbf{in}\ x$, while the second projection, $\mathbf{proj}_2^{\ell_0}\ a \triangleq \mathbf{let}\ (x^{\ell_0}, y) = a\ \mathbf{in}\ y$. These projections can be type-checked according to the following derived rules:

DCT-Proj1

$$\frac{\Omega \vdash a :^{\ell} \Sigma x{:}^{\ell_0} A.B \qquad \ell_0 \sqsubseteq \ell}{\Omega \vdash \mathbf{proj}_1^{\ell_0}\ a :^{\ell} A}$$

DCT-Proj2

$$\frac{\Omega \vdash a :^{\ell} \Sigma x{:}^{\ell_0} A.B}{\Omega \vdash \mathbf{proj}_2^{\ell_0}\ a :^{\ell} B\{\mathbf{proj}_1^{\ell_0}\ a/x\}}$$

Observe that the derived rule DCT-Proj1 limits access to the first component through the side condition, $\ell_0 \sqsubseteq \ell$, akin to rule SC-Unseal. The side condition helps align the observability of the first component of

the pair with the label on the Σ-type, as intended. So, we see that pairs can be eliminated in DDC$^\top$ via both pattern-matching and derived projections.

Next, we present the typing rules for sums. The rules DCT-Inj and DCT-Case, shown below, do not add anything significant to their simply-typed counterparts, rules SDC-Inj and SDC-Case respectively, as far as the grades are concerned.

$$\textsc{DCT-Sum}\ \frac{\Omega \vdash A_1 :^{\ell} s \qquad \Omega \vdash A_2 :^{\ell} s}{\Omega \vdash A_1 + A_2 :^{\ell} s} \qquad \textsc{DCT-Inj}\ \frac{\Omega \vdash a_i :^{\ell} A_i \qquad \Omega \vdash A_1 + A_2 :^{\top} s}{\Omega \vdash \mathbf{inj}_i\, a_i :^{\ell} A_1 + A_2}$$

$$\textsc{DCT-Case}\ \frac{\begin{array}{c}\Omega, z :^{\top} A_1 + A_2 \vdash B :^{\top} s \\ \Omega \vdash a :^{\ell} A_1 + A_2 \\ \Omega, x_1 :^{\ell} A_1 \vdash b_1 :^{\ell} B\{\mathbf{inj}_1\, x_1/z\} \\ \Omega, x_2 :^{\ell} A_2 \vdash b_2 :^{\ell} B\{\mathbf{inj}_2\, x_2/z\}\end{array}}{\Omega \vdash \mathbf{case}\, a\, \mathbf{of}\, x_1.b_1; x_2.b_2 :^{\ell} B\{a/z\}}$$

Now that we have seen the type system of DDC$^\top$, we show that the calculus is indeed a conservative extension of SDC.

3.3.4 Embedding SDC into DDC$^\top$

We define a translation function, $\overline{\overline{\cdot}}$, that takes the types and terms of SDC to their counterparts in DDC$^\top$. For types, the translation is defined as: $\overline{\overline{A \to B}} := {}^{\bot}\overline{\overline{A}} \to \overline{\overline{B}}$ and $\overline{\overline{S_\ell\, A}} := \Sigma x{:}^{\ell}\overline{\overline{A}}.\mathbf{Unit}$. (Here, ${}^{\ell}A \to B$ stands for $\Pi x :^{\ell} A.B$, with x assumed to be fresh.) For terms, the translation is straightforward except for the following cases: $\overline{\overline{\mathbf{lock}^{\ell}\ a}} := (\overline{\overline{a}}^{\ell}, \mathbf{unit})$ and $\overline{\overline{\mathbf{unlock}^{\ell}\ x = a\, \mathbf{in}\, b}} := \mathbf{let}\ (x^{\ell}, y)\ =\ \overline{\overline{a}}\, \mathbf{in}\, \overline{\overline{b}}$, where y is assumed to be fresh. Note a technical point here: to ensure that the translation is injective on terms, we add an extra bottom element, $\bot_0$, to the lattice, below its original bottom element, $\bot$, and translate products and pairs as: $\overline{\overline{A \times B}} := {}^{\bot_0}\overline{\overline{A}} \times \overline{\overline{B}}$ and $\overline{\overline{(a, b)}} = (\overline{\overline{a}}^{\bot_0}, \overline{\overline{b}})$. Without this extra bottom element, we would have needed to map both $\mathbf{lock}^{\bot}\ a$ and $(a, \mathbf{unit})$ to the same term.

Now, we translate typing judgments. DDC$^\top$ is a generic pure type system whereas SDC is based upon the simply-typed λ-calculus. Now, simply-typed λ-calculus is essentially the PTS where $\mathcal{S} = \{*, \square\}$ and $\mathcal{A} = \{(*, \square)\}$ and $\mathcal{R} = \{(*, *, *)\}$, as shown in Table 3.1. So we translate SDC to DDC$^\top$, with $\mathcal{S}, \mathcal{A}$ and $\mathcal{R}$ instantiated as such. Next, any well-formed type in any PTS must have a sort, unless the type itself is a sort. But in the usual presentation of simply-typed calculi, including SDC, types are not associated with any sort. So given any typing judgment in SDC, we first need to assign sorts to the types appearing in the context and the type of the derived term. Since SDC is simply-typed, every type is assigned sort, $*$. With this assignment, we can show that the translation presented above preserves typing, as stated in the lemma below. The lemma below uses some new notation, whose meanings are as follows. Given Ω, a context of SDC, let $cod(\Omega)$ denote the set of types associated with the assumptions in Ω. Next, given a set of types of SDC, call it S, let $atm(S)$ denote the set of atomic types that build up the types of S. And finally, given a set of atomic types of SDC, call it T, let $T \mathrel{\overline{\overline{:}}}^{\ell} *$ denote the set of assumptions where each element

of T is assigned the sort, $*$, at level ℓ. Then, given $\Omega \vdash a :^{\ell} A$, a typing judgment of SDC, the notation $atm(cod(\Omega) \cup \{A\}) :^{\ell} *$, used in the lemma below, denotes the set of assumptions where every atomic type appearing in the judgment is assigned the sort, $*$. The other notation used below, $\overline{\overline{\Omega}}$, denotes the context Ω, with the types translated.

Lemma 3.17 (Typing Preservation) If $\Omega \vdash a :^{\ell} A$, then $atm(cod(\Omega) \cup \{A\}) :^{\top} *, \overline{\overline{\Omega}} \vdash \overline{\overline{a}} :^{\ell} \overline{\overline{A}}$.

Next, assuming a standard call-by-name small-step semantics for both the languages, we can provide a bisimulation between the two.

Lemma 3.18 (Forward Simulation) If $\vdash a \leadsto a'$ in SDC, then $\vdash \overline{\overline{a}} \leadsto \overline{\overline{a'}}$ in $\text{DDC}^{\top}$.

Lemma 3.19 (Backward Simulation) For any term a in SDC, if $\vdash \overline{\overline{a}} \leadsto b$ in $\text{DDC}^{\top}$, then there exists a' in SDC such that $\vdash a \leadsto a'$ and $\overline{\overline{a'}} = b$.

This bisimulation shows that SDC can be embedded into $\text{DDC}^{\top}$ in a meaning-preserving manner. Hence, like SDC, $\text{DDC}^{\top}$ too can analyze dependencies in general. Next, we use $\text{DDC}^{\top}$ to analyze a specific dependency, viz., run-time irrelevance in dependent type systems.

3.3.5 Analyzing Run-time Irrelevance

Following our discussion in Section 3.1.1, we use the two element lattice, $\mathcal{L}_2 \triangleq \bot \sqsubseteq \top$, to analyze run-time irrelevance. The levels of this lattice, $\bot$ and $\top$, correspond to run-time relevant and run-time irrelevant terms respectively. Our goal is to erase run-time irrelevant terms, i.e., terms marked with $\top$. Towards this end, given an arbitrary lattice $\mathcal{L}$, we define a general indexed erasure function on $\text{DDC}^{\top}(\mathcal{L})$ terms, denoted $\lfloor \cdot \rfloor_{\ell}$, which erases everything that an observer from level ℓ should not be able to observe. This function is defined by straightforward recursion in most cases. For example,

$$\lfloor x \rfloor_{\ell} = x \qquad \lfloor \Pi x :^{\ell_0} A.B \rfloor_{\ell} = \Pi x :^{\ell_0} \lfloor A \rfloor_{\ell}.\lfloor B \rfloor_{\ell}$$

For abstraction, application and pair, the definition becomes interesting:

$$\lfloor \lambda x :^{\ell_0} A.b \rfloor_{\ell} = \begin{cases} \lambda x :^{\ell_0} \lfloor A \rfloor_{\ell}.\lfloor b \rfloor_{\ell} & \text{if } \top \sqsubseteq \ell \\ \lambda x :^{\ell_0} \textbf{Unit}.\lfloor b \rfloor_{\ell} & \text{otherwise} \end{cases} \qquad \lfloor b\ a^{\ell_0} \rfloor_{\ell} = \begin{cases} \lfloor b \rfloor_{\ell}\ (\lfloor a \rfloor_{\ell})^{\ell_0} & \text{if } \ell_0 \sqsubseteq \ell \\ \lfloor b \rfloor_{\ell}\ \textbf{unit}^{\ell_0} & \text{otherwise} \end{cases}$$

$$\lfloor (a^{\ell_0}, b) \rfloor_{\ell} = \begin{cases} ((\lfloor a \rfloor_{\ell})^{\ell_0}, \lfloor b \rfloor_{\ell}) & \text{if } \ell_0 \sqsubseteq \ell \\ (\textbf{unit}^{\ell_0}, \lfloor b \rfloor_{\ell}) & \text{otherwise} \end{cases}$$

The case for abstraction is so defined because at levels other than $\top$, we don't want to retain information about the domain type; as such, it is replaced with **Unit**. The cases for application and pair are so defined because if $\neg(\ell_0 \sqsubseteq \ell)$, an observer from level ℓ should not be able to observe a; as such, a is replaced with **unit**.

$\boxed{\Phi \vdash a \sim_\ell b}$ *(Indexed indistinguishability)*

$$\textsc{IndD-Var}\quad \frac{\ell_0 \sqsubseteq \ell}{\Phi_1, x\!:\ell_0, \Phi_2 \vdash x \sim_\ell x} \qquad \textsc{IndD-Sort}\quad \frac{}{\Phi \vdash s \sim_\ell s} \qquad \textsc{IndD-Pi}\quad \frac{\Phi \vdash A_1 \sim_\ell A_2 \qquad \Phi, x\!:\ell \vdash B_1 \sim_\ell B_2}{\Phi \vdash \Pi x\!:^{\ell_0} A_1.B_1 \sim_\ell \Pi x\!:^{\ell_0} A_2.B_2}$$

$$\textsc{IndD-Lam}\quad \frac{\Phi, x\!:\ell_0 \sqcup \ell \vdash b_1 \sim_\ell b_2 \qquad \Phi \vdash^{\top}_{\ell} A_1 \sim A_2}{\Phi \vdash \lambda x\!:^{\ell_0} A_1.b_1 \sim_\ell \lambda x\!:^{\ell_0} A_2.b_2} \qquad \textsc{IndD-App}\quad \frac{\Phi \vdash b_1 \sim_\ell b_2 \qquad \Phi \vdash^{\ell_0}_{\ell} a_1 \sim a_2}{\Phi \vdash b_1\ {a_1}^{\ell_0} \sim_\ell b_2\ {a_2}^{\ell_0}} \qquad \textsc{IndD-Sigma}\quad \frac{\Phi \vdash A_1 \sim_\ell A_2 \qquad \Phi, x\!:\ell \vdash B_1 \sim_\ell B_2}{\Phi \vdash \Sigma x\!:^{\ell_0} A_1.B_1 \sim_\ell \Sigma x\!:^{\ell_0} A_2.B_2}$$

$$\textsc{IndD-Pair}\quad \frac{\Phi \vdash^{\ell_0}_{\ell} a_1 \sim a_2 \qquad \Phi \vdash b_1 \sim_\ell b_2}{\Phi \vdash ({a_1}^{\ell_0}, b_1) \sim_\ell ({a_2}^{\ell_0}, b_2)} \qquad \textsc{IndD-LetPair}\quad \frac{\Phi \vdash a_1 \sim_\ell a_2 \qquad \Phi, x\!:\ell_0 \sqcup \ell, y\!:\ell \vdash b_1 \sim_\ell b_2}{\Phi \vdash \textbf{let}\ (x^{\ell_0}, y)\ =\ a_1\ \textbf{in}\ b_1 \sim_\ell \textbf{let}\ (x^{\ell_0}, y)\ =\ a_2\ \textbf{in}\ b_2}$$

$\boxed{\Phi \vdash^{\ell_0}_{\ell} a_1 \sim a_2}$

$$\textsc{CIndD-Leq}\quad \frac{\ell_0 \sqsubseteq \ell \qquad \Phi \vdash a_1 \sim_\ell a_2}{\Phi \vdash^{\ell_0}_{\ell} a_1 \sim a_2} \qquad \textsc{CIndD-Nleq}\quad \frac{\neg(\ell_0 \sqsubseteq \ell)}{\Phi \vdash^{\ell_0}_{\ell} a_1 \sim a_2}$$

FIGURE 3.6: Indexed indistiguishability relation for DDC$^{\top}$ and DDC (Excerpt)

Observe that this erasure function is closely related to the indistinguishability relation, defined in Section 3.2.3. The relation there was defined on the terms of SDC. But we can extend this relation to the terms of DDC$^{\top}$ and DDC in a straightforward manner, as shown in Figure 3.6. In the dependent setting too, most of the rules for this relation simply mirror the corresponding typing rules. But for terms involving secure values, for example, application and pair, the rules get interesting. Observe how rules IndD-App and IndD-Pair fuse rule Ind-Lock with rules Ind-App and Ind-Pair respectively. This fusion makes sense because from a dependency perspective, the terms $b\ a^{\ell_0}$ and (a^{ℓ_0}, b) are same as the terms $b\ (\textbf{lock}^{\ell_0}\ a)$ and $(\textbf{lock}^{\ell_0}\ a, b)$ respectively. Next, consider rule IndD-Lam. In this rule, if $\ell = \top$, the types A_1 and A_2 must themselves be indistinguishable at $\top$ (and therefore be syntactically equal); otherwise, they may be arbitrarily different. This condition makes sense because type information in function abstractions matter only at the compile-time level, denoted by $\top$ in DDC$^{\top}$. Now coming back to the erasure function, we can show that it maps the equivalence classes formed by this indistinguishability relation to their respective canonical elements, as shown by the lemma below.

Lemma 3.20 (Canonical Element) If $\Phi \vdash a_1 \sim_\ell a_2$, then $\lfloor a_1 \rfloor_\ell = \lfloor a_2 \rfloor_\ell$.

In fact, we can go further and show that a well-graded term is indistinguishable from itself upon erasure. Recall that a term a is said to be well-graded at ℓ if and only if $\Phi \vdash a \sim_\ell a$, for some Φ.

Lemma 3.21 (Erasure Indistinguishability) If $\Phi \vdash a : \ell$, then $\Phi \vdash a \sim_\ell \lfloor a \rfloor_\ell$.

Now, the purpose of an erasure function is to erase parts of a program that do not play any role in reduction. Put differently, erasure should not affect reduction. This is true of our erasure function, which preserves the reduction behavior of a program, as we see in the lemma below. This lemma shows that erased terms simulate the reduction behavior of their unerased counterparts in a step-by-step manner.

Lemma 3.22 (Erasure Simulation) If $\Phi \vdash a : \ell$ and $\vdash a \leadsto b$, then $\vdash \lfloor a \rfloor_\ell \leadsto \lfloor b \rfloor_\ell$. Otherwise, if a is a value, then so is $\lfloor a \rfloor_\ell$.

The above lemma helps us show soundness of run-time irrelevance analysis in $\text{DDC}^\top(\mathcal{L}_2)$. Below, we explain how. Let $\Omega \vdash b :^\bot \textbf{Bool}$. By the above lemma, if b diverges, then so does $\lfloor b \rfloor_\bot$. Otherwise, b reduces to a boolean value, say v. Then, using the lemma again, $\lfloor b \rfloor_\bot$ reduces to $\lfloor v \rfloor_\bot$. But $\lfloor v \rfloor_\bot = v$, since the erasure function on boolean values is just identity. Therefore, erasure of $\top$-marked terms does not affect the run-time behavior of programs from level $\bot$. Thus, $\text{DDC}^\top(\mathcal{L}_2)$ analyzes run-time irrelevance soundly.

Now, observe that though $\text{DDC}^\top$ can analyze run-time irrelevance, it does not take advantage of this analysis in its type system. To put it more concretely, the term `ida =` $\lambda^\top$ `f.` λ `x. x :` Π `f:`$^\top$`(`$^\top$`Nat → Type). f 0`$^\top$ `→ f 1`$^\top$ from Section 3.1.2 wouldn't type-check in $\text{DDC}^\top(\mathcal{L}_2)$. To type-check this term, the conversion rule needs to be updated so that it makes use of irrelevance analysis. More succinctly, the type system needs to support compile-time irrelevance. But supporting compile-time irrelevance requires more changes to the type system than just an update to the conversion rule. Next, we motivate these changes as we generalize $\text{DDC}^\top$ to DDC.

3.4 DDC: Supporting Compile-Time Irrelevance

3.4.1 Towards Compile-Time Irrelevance

Recall that terms which may be safely ignored while checking for type equality are said to be compile-time irrelevant. $\text{DDC}^\top$ cannot support compile-time irrelevance because the conversion rule DCT-Conv checks for type equality at $\top$. To elaborate, the equality judgment used in this rule, $\Phi \vdash a \equiv_\top b$, is a special case of the more general judgment, $\Phi \vdash a \equiv_\ell b$, which is the closure of the indistinguishability relation at ℓ under β-equivalence. When ℓ is instantiated to $\top$, the indistinguishability relation degenerates to identity. As such, the equality relation at $\top$ is standard β-equivalence. So, rule DCT-Conv does not ignore any part of A or B while checking for their equality.

$$\frac{\Omega \vdash a :^\ell A \qquad \lceil \Omega \rceil \vdash A \equiv_\top B \qquad \Omega \vdash B :^\top s}{\Omega \vdash a :^\ell B} \ \text{DCT-Conv}$$

To support compile-time irrelevance then, we need the conversion rule to check for equality at a grade strictly less than $\top$ so that $\top$-marked terms may be ignored during the check. Now, recall from Section 3.1.3 that we

need a lattice with at least three elements to simultaneously analyze run-time and compile-time irrelevance. The simplest of such lattices is $\mathcal{L}_3 \triangleq \bot \sqsubseteq \mathcal{C} \sqsubseteq \top$. When $\mathcal{L}_3$ is the parametrizing latttice, we may check for type equality at $\mathcal{C}$ so that $\top$-marked terms may be ignored during the check. When the parameter is some other lattice, say $\mathcal{L}$, we can add two new elements, $\mathcal{C}$ and $\top$, such that for any $\ell \in \mathcal{L}$, we have, $\ell \sqsubseteq \mathcal{C} \sqsubseteq \top$, and subsequently use $\mathcal{C}$ for the same purpose. In this way, by checking for type equality at $\mathcal{C}$, we may support compile-time irrelevance for any given lattice.

Referring back to the term `idp` from Section 3.1.2, observe that for $\texttt{phantom : Nat}^\top \to \texttt{Type}$, we have, $\texttt{phantom 0}^\top \equiv_\mathcal{C} \texttt{phantom 1}^\top$. With this equality, we can type-check $\texttt{idp = } \lambda \texttt{ x. x : phantom 0}^\top \to \texttt{phantom 1}^\top$, without even knowing the definition of `phantom`. The other terms presented in that section, such as, `ida` and `idn`, can also be type-checked in a similar way.

Next, observe that in rule DCT-Conv, the new type B is also checked at $\top$. If we want to check for type equality at $\mathcal{C}$, we need to make sure that the types themselves are also checked at $\mathcal{C}$ and not at $\top$. However, checking types at $\mathcal{C}$ would rule out variables marked with $\top$ from appearing in them. This would restrict us from expressing many examples, including the polymorphic identity function, $\texttt{id =}\lambda^\top \texttt{x. } \lambda \texttt{y. y : } \Pi \texttt{ x:}^\top \texttt{Type. x} \to \texttt{x}$.

To move out of this impasse, we take inspiration from the 'context reset operation' employed in Erasure Pure Type System or EPTS [Mishra-Linger and Sheard, 2008, Mishra-Linger, 2008] to solve a similar problem. The context reset operation, when adapted to our setting, would entail using a judgment of the form $\mathcal{C} \sqcap \Omega \vdash A :^\mathcal{C} s$ to allow $\top$-marked assumptions in the context to be used in the type. The operation $\mathcal{C} \sqcap \Omega$ takes the point-wise meet of $\mathcal{C}$ with the labels on the assumptions in the context Ω, essentially bringing down all $\top$ labels to $\mathcal{C}$, thereby making $\top$-marked assumptions available for use in an expression type-checked at $\mathcal{C}$. We call this operation *truncation* since it truncates $\top$-marked assumptions to $\mathcal{C}$-marked ones. Other systems in literature also use similar mechanisms while analyzing irrelevance — for example, Pfenning [2001] and Abel and Scherer [2012] employ 'context resurrection' operation to make otherwise unusable proof variables and irrelevant variables in the context available for use, similar to how $C \sqcap \Omega$ makes otherwise unusable $\top$-marked variables in the context available for use (at $\mathcal{C}$).

Employing this idea of truncation, we design a general dependency calculus, DDC, which takes advantage of compile-time irrelevance in its type system. DDC is a generalization of $\text{DDC}^\top$ and $\text{EPTS}^\bullet$ [Mishra-Linger, 2008], a standard calculus for run-time and compile-time irrelevance analysis. When $\mathcal{C}$ equals $\top$, DDC degenerates to $\text{DDC}^\top$, which does not support compile-time irrelevance. When $\mathcal{C}$ equals $\bot$, DDC degenerates to $\text{EPTS}^\bullet$, which identifies compile-time and run-time irrelevance. A crucial distinction between $\text{EPTS}^\bullet$ and DDC, however, is that while the former is hardwired to use a two element lattice, the latter can use any lattice. Thus, not only can DDC distinguish between run-time and compile-time irrelevance, but also it can simultaneously track other dependencies.

$$\boxed{\Omega \vdash a :^{\ell} A}$$ *(DDC core typing rules)*

$$\textsc{Ddc-Var}\quad \frac{\Omega \Vdash A :^{\top} s \qquad \ell_0 \sqsubseteq \ell \qquad \ell \sqsubseteq \mathcal{C}}{\Omega, x :^{\ell_0} A \vdash x :^{\ell} A}$$

$$\textsc{Ddc-Weak}\quad \frac{\Omega \vdash a :^{\ell} A \qquad \Omega \Vdash B :^{\top} s}{\Omega, y :^{\ell_0} B \vdash a :^{\ell} A}$$

$$\textsc{Ddc-Pi}\quad \frac{\Omega \vdash A :^{\ell} s_1 \qquad \Omega, x :^{\ell} A \vdash B :^{\ell} s_2 \qquad (s_1, s_2, s_3) \in \mathcal{R}}{\Omega \vdash \Pi x :^{\ell_0} A.B :^{\ell} s_3}$$

$$\textsc{Ddc-Lam}\quad \frac{\Omega, x :^{\ell \sqcup \ell_0} A \vdash b :^{\ell} B \qquad \Omega \Vdash (\Pi x :^{\ell_0} A.B) :^{\top} s}{\Omega \vdash \lambda x :^{\ell_0} A.b :^{\ell} \Pi x :^{\ell_0} A.B}$$

$$\textsc{Ddc-App}\quad \frac{\Omega \vdash b :^{\ell} \Pi x :^{\ell_0} A.B \qquad \Omega \Vdash a :^{\ell \sqcup \ell_0} A}{\Omega \vdash b\ a^{\ell_0} :^{\ell} B\{a/x\}}$$

$$\textsc{Ddc-Conv}\quad \frac{\Omega \vdash a :^{\ell} A \qquad \lceil \mathcal{C} \sqcap \Omega \rceil \vdash A \equiv_{\mathcal{C}} B \qquad \Omega \Vdash B :^{\top} s}{\Omega \vdash a :^{\ell} B}$$

$$\boxed{\Omega \Vdash a :^{\ell} A}$$ *(Truncate to $\mathcal{C}$)*

$$\textsc{TC-Leq}\quad \frac{\Omega \vdash a :^{\ell} A \qquad \ell \sqsubseteq \mathcal{C}}{\Omega \Vdash a :^{\ell} A}$$

$$\textsc{TC-Top}\quad \frac{\mathcal{C} \sqcap \Omega \vdash a :^{\mathcal{C}} A}{\Omega \Vdash a :^{\top} A}$$

FIGURE 3.7: DDC type system (Core rules)

3.4.2 Core Type System

The core typing rules of DDC appear in Figure 3.7. To begin with, note that unlike DDC$^{\top}$, DDC maintains the invariant that $\ell \sqsubseteq \mathcal{C}$, for all typing judgments, $\Omega \vdash a :^{\ell} A$. To ensure that this is the case, rule DDC-VAR includes the precondition, $\ell \sqsubseteq \mathcal{C}$. This restriction implies that we cannot really derive any term at $\top$ in DDC; as such, we cannot use $\top$-marked assumptions of the context in any derivation. However, as discussed before, we would like to make use of such assumptions in types. Further, we also want to allow use of such assumptions in $\top$-marked terms, such as, a in $b\ a^{\top}$, because without allowing such usage, irrelevance analysis is pointless. DDC allows such usage via the truncation operation mentioned above, whereby $\top$-marked assumptions are truncated to C-marked ones and made available for use at $\mathcal{C}$. So, the judgment $\Omega \vdash a :^{\top} A$ can indeed be expressed in DDC, but in a roundabout way via the truncated judgment, $\mathcal{C} \sqcap \Omega \vdash a :^{\mathcal{C}} A$. Use of a truncated judgment is one of the two key technical differences between DDC and DDC$^{\top}$, the other one being equating types at $\mathcal{C}$ in the conversion rule DDC-CONV.

Going through the typing rules of DDC in Figure 3.7, we find that DDC uses truncation wherever DDC$^{\top}$ uses $\top$ as the observer level of a typing judgment. This is true of rules DDC-VAR, DDC-WEAK, DDC-LAM, and DDC-CONV, where the types are checked using truncated judgments in the premises. Next, observe that in rule DDC-APP, a different judgment, $\Omega \Vdash a :^{\ell \sqcup \ell_0} A$, is used to type-check the argument, a. This judgment makes sense here because if $\ell_0 = \top$, we need to allow $\top$-marked assumptions to be used in a, which is possible only via truncation. On the other hand, if $\ell_0 \neq \top$, the argument a may be checked using the normal typing judgment, $\Omega \vdash a :^{\ell \sqcup \ell_0} A$. The new judgment, $\Omega \Vdash a :^{\ell} A$, succinctly expresses the conditional proposition:

if $\ell = \top$, use the truncated typing judgment, $\mathcal{C} \sqcap \Omega \vdash a :^{\mathcal{C}} A$; otherwise use the normal typing judgment, $\Omega \vdash a :^{\ell} A$. With regard to rule TC-Leq for this judgment, note that as per our usage of the labels, $\top$ and $\mathcal{C}$, laid out in Section 3.4.1, the conditions $\ell \neq \top$ and $\ell \sqsubseteq \mathcal{C}$ are equivalent. In the typing rules of DDC, we highlight uses of the judgment, $\Omega \Vdash a :^{\ell} A$, in gray to emphasize the modification with respect to DDC$^{\top}$.

To see how irrelevance is analyzed in DDC, let's consider the polymorphic identity function, `id`, again. In DDC, this function is type-checked the same way as in DDC$^{\top}$, as shown below ($\bot$ grades are left unmarked, following our adopted style).

```
id : Π x:⊤Type. x → x
id = λ⊤x. λ y. y
```

The type of the function, however, is checked in a different way in DDC. In DDC$^{\top}$, the type Π `x:`$^{\top}$`Type . x → x` is checked at $\top$. But in DDC it is checked at $\mathcal{C}$, using the truncated judgment, `x:`$^{\mathcal{C}}$`Type` $\vdash$ `x → x` $:^{\mathcal{C}}$ `Type`. Observe here that if we had used standard Π-types along with a graded modality in lieu of our graded Π-types and had represented $\Pi x :^{\ell} A.B$ as $\Pi y : S_{\ell}\, A.\mathbf{unlock}^{\ell}\, x = y\, \mathbf{in}\, B$, we would have run into a problem because $\Pi y : S_{\top}$ `Type`.$\mathbf{unlock}^{\top}$ `x = y`$\,\mathbf{in}\,$`x → x` cannot be checked at $\mathcal{C}$. To elaborate, `y` $:^{\mathcal{C}} S_{\top}$ `Type` $\nvdash$ $\mathbf{unlock}^{\top}$ `x = y`$\,\mathbf{in}\,$`x → x` $:^{\mathcal{C}}$ `Type` because a term of type $S_{\top}$ `Type` cannot be unlocked and used at level $\mathcal{C}$. This is why we use graded Π-types in our dependency calculi. As this example shows, graded Π-types offer us the flexibility to use the same bound variable in the body of the function and in the body of the type under restrictions that are totally different from one another. This is the flexibility of our approach we alluded to in Section 3.3.1.

We see how use of a truncated judgment differentiates DDC from DDC$^{\top}$. As pointed out before, the truncated judgment is necessary because unlike DDC$^{\top}$, in DDC, types are equated at $\mathcal{C}$, and therefore also checked at $\mathcal{C}$. Contrast the conversion rule DCT-Conv with rule Ddc-Conv, which checks for equality of types at $\mathcal{C}$ instead of $\top$, thereby ignoring compile-time irrelevant parts of types during the check. This is how DDC supports compile-time irrelevance. Since the equality check at $\mathcal{C}$ is what distinguishes DDC from DDC$^{\top}$, we next consider the indexed equality judgment in more detail.

An excerpt of the indexed definitional equality relation appears in Figure 3.8. The judgment $\Phi \vdash a \equiv_{\ell} b$ may be read as: a is definitionally equal to b at level ℓ, assuming the variables in Φ are visible at the respective levels they are marked with. Going through the rules in Figure 3.8, we can see that the equality relation is essentially the closure of the indexed indistinguishability relation (presented in Section 3.3.5) under β-reduction. Now, the indistinguishability relation at ℓ does not care about parts of terms marked with ℓ_0, whenever $\neg(\ell_0 \sqsubseteq \ell)$. This feature is inherited by the equality relation. This feature enables the equality relation at $\mathcal{C}$, used in rule Ddc-Conv, to ignore parts of the involved types that are marked with $\top$.

3.4.3 Σ-types

Now we consider the typing rules for Σ-types. The introduction and elimination rules for Σ-types are shown below.

$\boxed{\Phi \vdash a \equiv_\ell b}$ *(Equality rules)*

$$\textsc{Eq-Refl}\ \frac{\Phi \vdash a : \ell}{\Phi \vdash a \equiv_\ell a} \qquad \textsc{Eq-Sym}\ \frac{\Phi \vdash a \equiv_\ell b}{\Phi \vdash b \equiv_\ell a} \qquad \textsc{Eq-Trans}\ \frac{\Phi \vdash a \equiv_\ell b \qquad \Phi \vdash b \equiv_\ell c}{\Phi \vdash a \equiv_\ell c} \qquad \textsc{Eq-Beta}\ \frac{\Phi \vdash a : \ell \qquad \vdash a \leadsto b}{\Phi \vdash a \equiv_\ell b}$$

$$\textsc{Eq-Pi}\ \frac{\Phi \vdash A_1 \equiv_\ell A_2 \qquad \Phi, x : \ell \vdash B_1 \equiv_\ell B_2}{\Phi \vdash \Pi x{:}^{\ell_0} A_1.B_1 \equiv_\ell \Pi x{:}^{\ell_0} A_2.B_2} \qquad \textsc{Eq-Lam}\ \frac{\Phi, x : \ell \sqcup \ell_0 \vdash b_1 \equiv_\ell b_2 \qquad \Phi \vdash^{\top} A_1 \equiv_\ell A_2}{\Phi \vdash \lambda x{:}^{\ell_0} A_1.b_1 \equiv_\ell \lambda x{:}^{\ell_0} A_2.b_2} \qquad \textsc{Eq-App}\ \frac{\Phi \vdash b_1 \equiv_\ell b_2 \qquad \Phi \vdash^{\ell_0} a_1 \equiv_\ell a_2}{\Phi \vdash b_1\ {a_1}^{\ell_0} \equiv_\ell b_2\ {a_2}^{\ell_0}}$$

$\boxed{\Phi \vdash^{\ell_0} a_1 \equiv_\ell a_2}$

$$\textsc{CEq-Leq}\ \frac{\ell_0 \sqsubseteq \ell \qquad \Phi \vdash a_1 \equiv_\ell a_2}{\Phi \vdash^{\ell_0} a_1 \equiv_\ell a_2} \qquad \textsc{CEq-Nleq}\ \frac{\neg(\ell_0 \sqsubseteq \ell)}{\Phi \vdash^{\ell_0} a_1 \equiv_\ell a_2}$$

FIGURE 3.8: Indexed definitional equality relation for DDC (Excerpt)

$$\textsc{Ddc-Pair}\ \frac{\Omega \Vdash a :^{\ell \sqcup \ell_0} A \qquad \Omega \vdash b :^{\ell} B\{a/x\} \qquad \Omega \Vdash \Sigma x{:}^{\ell_0} A.B :^{\top} s}{\Omega \vdash (a^{\ell_0}, b) :^{\ell} \Sigma x{:}^{\ell_0} A.B} \qquad \textsc{Ddc-LetPair}\ \frac{\Omega \vdash a :^{\ell} \Sigma x{:}^{\ell_0} A.B \qquad \Omega, x :^{\ell \sqcup \ell_0} A, y :^{\ell} B \vdash c :^{\ell} C\{(x^{\ell_0}, y)/z\} \qquad \Omega, z :^{\top} (\Sigma x{:}^{\ell_0} A.B) \Vdash C :^{\top} s}{\Omega \vdash \mathbf{let}\ (x^{\ell_0}, y) = a\ \mathbf{in}\ c :^{\ell} C\{a/z\}}$$

Similar to rule DDC-APP, rule DDC-PAIR uses the judgment, $\Omega \Vdash a :^{\ell \sqcup \ell_0} A$, to type-check a, the first component of the pair. The idea behind using this judgment in this rule is the same: if $\ell_0 = \top$, allow $\top$-marked assumptions to be used in a; otherwise, use the normal typing judgment, $\Omega \vdash a :^{\ell \sqcup \ell_0} A$.

Next, observe that rule DDC-LETPAIR is same as rule DCT-LETPAIR other than its use of truncated judgment to check the return type. We can use the elimination form type-checked by this rule to define the projection operations in DDC, as we did in DDC$^{\top}$. But note that in rule DDC-PAIR, if $\ell_0 = \top$, the first component a should be treated as compile-time irrelevant. In such a case, we cannot use projections to eliminate the pair because we would not be able to type-check the second projection, whose type depends upon the first projection. So the derived second-projection rule DDC-PROJ2, shown below, includes a side-condition, $\ell_0 \sqsubseteq \mathcal{C}$. Thus, pairs having type $\Sigma x :^{\top} A.B$, where B mentions x, can only be eliminated via pattern-matching in DDC. However, pairs having type $\Sigma x :^{\ell_0} A.B$, where $\ell_0 \neq \top$, can be eliminated via projections. See lemma B.52 for the derivation of the projection rules below.

$$\textsc{Ddc-Proj1}\ \frac{\Omega \vdash a :^{\ell} \Sigma x{:}^{\ell_0} A.B \qquad \ell_0 \sqsubseteq \ell}{\Omega \vdash \mathbf{proj}_1^{\ell_0}\ a :^{\ell} A} \qquad \textsc{Ddc-Proj2}\ \frac{\Omega \vdash a :^{\ell} \Sigma x{:}^{\ell_0} A.B \qquad \ell_0 \sqsubseteq \mathcal{C}}{\Omega \vdash \mathbf{proj}_2^{\ell_0}\ a :^{\ell} B\{\mathbf{proj}_1^{\ell_0}\ a/x\}}$$

Let us consider the `filter` function from Section 3.1.3 in light of the above discussion. The output of this function, call it `ys`, has type, $\Sigma \texttt{m} :^{\mathcal{C}} \texttt{Nat}.\texttt{Vec m a}$. We can eliminate this output via projections: π_1 `ys` $:^{\mathcal{C}}$ `Nat` and π_2 `ys : Vec (`π_1 `ys) a`. Observe that (π_1 `ys`) is used in the type of (π_2 `ys`). We can substitute (π_1 `ys` $:^{\mathcal{C}}$ `Nat`) for `m` in `Vec m a` because `Vec m a` is derived using the truncated judgment: `a` $:^{\mathcal{C}}$ `Type, m` $:^{\mathcal{C}}$`Nat` $\vdash$ `Vec m a` $:^{\mathcal{C}}$ `Type`. However, since π_1 `ys` $:^{\mathcal{C}}$ `Nat`, the term π_1 `ys` cannot be used at $\bot$, and as such, may be safely ignored at this level. This example shows how DDC analyzes strong irrelevant Σ-types.

Next, we look at the metatheoretic properties of DDC. But before moving ahead, note that DDC also includes sum types, whose typing rules are straightforward extensions of their counterparts in DDC$^{\top}$.

3.4.4 Noninterference

DDC satisfies a noninterference theorem analogous to the one for SDC. Recall that noninterference for SDC is shown via the indexed indistinguishability relation, $\Phi \vdash b_1 \sim_\ell b_2$. The indistinguishability relation at ℓ models the view of an observer from level ℓ, where information from levels ℓ_0, for which $\neg(\ell_0 \sqsubseteq \ell)$, does not matter. To prove noninterference, we showed that the operational semantics of SDC preserves the indistinguishability relation. This technique of proving noninterference smoothly extends to the dependently-typed setting of DDC. In fact, the statements of the lemmas that we need to prove noninterference for DDC are exactly the same as those of the ones for SDC, presented in Section 3.2.3. Below, we state these lemmas again, but here in the context of DDC.

Lemma 3.23 (Typing implies grading) If $\Omega \vdash a :^{\ell} A$ then $\lceil \Omega \rceil \vdash a : \ell$.

Lemma 3.24 (Indistinguishable terms are well-graded) If $\Phi \vdash a_1 \sim_\ell a_2$, then $\Phi \vdash a_1 : \ell$ and $\Phi \vdash a_2 : \ell$.

Lemma 3.25 (Equivalence) Indexed indistinguishability at ℓ is an equivalence relation on well-graded terms at ℓ.

Lemma 3.26 (Indistinguishability under substitution) If $\Phi_1, z : m, \Phi_2 \vdash a_1 \sim_\ell a_2$ and $\Phi_1 \vdash^m_\ell c_1 \sim c_2$, then $\Phi_1, \Phi_2 \vdash a_1\{c_1/z\} \sim_\ell a_2\{c_2/z\}$.

Theorem 3.27 (Noninterference for DDC) If $\Phi \vdash a_1 \sim_\ell a_2$ and $\vdash a_1 \leadsto a_1'$, then there exists a_2' such that $\vdash a_2 \leadsto a_2'$ and $\Phi \vdash a_1' \sim_\ell a_2'$.

The noninterference theorem above shows that DDC analyzes dependencies soundly. Now, DDC also internalizes dependency analysis in the type system itself via the conversion rule, which equates types ignoring compile-time irrelevant parts. As such, DDC relies upon noninterference to ensure that the type system does not equate types with different head forms, for example, **Bool** and **Unit** or $\Sigma x{:}^{\ell} A.B$ and $\Pi x{:}^{\ell} A.B$. If noninterference did not hold in DDC and $\top$-inputs could interfere in $\mathcal{C}$-outputs, then DDC would have derived inconsistent equalities at $\mathcal{C}$, such as, $\mathbf{Bool} \equiv_{\mathcal{C}} \mathbf{Unit}$. To see how, suppose $x{:}^{\top} A \vdash b :^{\mathcal{C}} \mathbf{Bool}$ and $a_1, a_2 : A$ such that the terms $b\{a_1/x\}$ and $b\{a_2/x\}$ reduce to **true** and **false** respectively. Now, by rule Eq-App, we have, $(\lambda^{\top} x.\mathbf{if}\, b\, \mathbf{then}\, \mathbf{Bool}\, \mathbf{else}\, \mathbf{Unit})\ {a_1}^{\top} \equiv_{\mathcal{C}} (\lambda^{\top} x.\mathbf{if}\, b\, \mathbf{then}\, \mathbf{Bool}\, \mathbf{else}\, \mathbf{Unit})\ {a_2}^{\top}$. But then, by β-equivalence, $\mathbf{Bool} \equiv_{\mathcal{C}} \mathbf{Unit}$. (Here, $\mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2 \triangleq \mathbf{case}\, b\, \mathbf{of}\, y_1.c_1; y_2.c_2$, where y_1 and y_2 are fresh.) Therefore, consistency of the equality relation of DDC, and by extension, soundness of the type system itself, hinges

upon noninterference. However, consistency does not follow immediately from noninterference. We need some additional machinery to prove it. Next, we describe this machinery and show that the equality relation of DDC is indeed consistent.

3.4.5 Consistency of Definitional Equality

The equality relation of a dependent type system is consistent if and only if there exists no derivation that equates two types having different *head forms*. To give an example, a consistent equality relation would not equate **Bool** with **Unit** or $\Sigma x{:}^{\ell} A.B$ with $\Pi x :^{\ell} A.B$ or s with $\Pi x :^{\ell} A.B$. Consistency of equality, though easy to state, is difficult to prove. It does not follow via a straightforward induction on the derivation of the equality judgment: an inductive argument fails in the case of the transitive rule, such as, rule Eq-Trans in the context of DDC. As such, we adopt an indirect approach to prove consistency, following Weirich et al. [2017].

The basic idea of our approach is to design an alternative relation, which can be easily shown to be consistent and which would also be equivalent to the equality relation. We design such an alternative relation called *joinability*, denoted $\Phi \vdash a \Leftrightarrow_{\ell} b$, which may be read as: a and b are *joinable* at ℓ, assuming Φ. We show that joinability implies consistency and also that two terms are joinable if and only if they are definitionally equal. Together, these two results imply that the definitional equality relation itself is consistent.

The joinability relation is defined using parallel reduction [Barendregt, 1984]. The concept of parallel reduction was introduced by Tait and Martin-Löf to construct an alternative (and elegant) proof of the Church-Rosser property for the untyped λ-calculus. The key distinction between ordinary reduction and parallel reduction is that while the former allows only a single sub-term to reduce at each step, the latter allows all sub-terms to reduce simultaneously at each step. Owing to this feature, one can prove confluence for the parallel reduction relation by a straightforward induction on its definition. Thereafter, one can show that the transitive closure of parallel reduction is the same as the transitive closure of ordinary reduction and thereby conclude that the transitive closure of ordinary reduction is also confluent, i.e., the Church-Rosser property. The parallel reduction relation is also useful in proving other properties of the λ-calculus [Takahashi, 1995]. Here, we shall use this relation to define joinability and prove consistency.

Our parallel reduction relation, denoted as $\Phi \vdash a \Rightarrow_{\ell} b$, is a fusion of standard parallel reduction and the indistinguishability relation. An excerpt of the rules for the relation appears in Figure 3.9. First, notice that this relation allows all the sub-terms of a term to reduce simultaneously, as we can see in rules Par-Pi, Par-Lam, Par-App, and Par-AppBeta. However, a sub-term may also reduce to itself via rule Par-Refl, rendering the relation nondeterministic. Next, notice that unobservable parts of terms may 'reduce' arbitrarily via rule CPar-Nleq. Such arbitrary 'reductions' are allowed because parts of terms that remain unobservable at a given level do not really matter at that level.

Now, the key property of parallel reduction that interests us is confluence. We can show that our parallel reduction relation and its transitive closure are confluent, as stated in the lemma below. Note that the

$\boxed{\Phi \vdash a \Rightarrow_\ell b}$ *(Parallel reduction rules)*

$$\textsc{Par-Refl}\quad \frac{\Phi \vdash a : \ell}{\Phi \vdash a \Rightarrow_\ell a}$$

$$\textsc{Par-Pi}\quad \frac{\Phi \vdash A_1 \Rightarrow_\ell A_2 \qquad \Phi, x\!:\ell \vdash B_1 \Rightarrow_\ell B_2}{\Phi \vdash \Pi x\!:^{\ell_0} A_1.B_1 \Rightarrow_\ell \Pi x\!:^{\ell_0} A_2.B_2}$$

$$\textsc{Par-Lam}\quad \frac{\Phi, x\!:\ell \sqcup \ell_0 \vdash b_1 \Rightarrow_\ell b_2 \qquad \Phi \vdash^{\top}_{\ell} A_1 \Rightarrow A_2}{\Phi \vdash \lambda x\!:^{\ell_0} A_1.b_1 \Rightarrow_\ell \lambda x\!:^{\ell_0} A_2.b_2}$$

$$\textsc{Par-App}\quad \frac{\Phi \vdash b_1 \Rightarrow_\ell b_2 \qquad \Phi \vdash^{\ell_0}_{\ell} a_1 \Rightarrow a_2}{\Phi \vdash b_1\ {a_1}^{\ell_0} \Rightarrow_\ell b_2\ {a_2}^{\ell_0}}$$

$$\textsc{Par-AppBeta}\quad \frac{\Phi \vdash b \Rightarrow_\ell (\lambda x\!:^{\ell_0} A.b') \qquad \Phi \vdash^{\ell_0}_{\ell} a \Rightarrow a'}{\Phi \vdash b\ a^{\ell_0} \Rightarrow_\ell b'\{a'/x\}}$$

$\boxed{\Phi \vdash^{\ell_0}_{\ell} a_1 \Rightarrow a_2}$

$$\textsc{CPar-Leq}\quad \frac{\ell_0 \sqsubseteq \ell \qquad \Phi \vdash a_1 \Rightarrow_\ell a_2}{\Phi \vdash^{\ell_0}_{\ell} a_1 \Rightarrow a_2}$$

$$\textsc{CPar-Nleq}\quad \frac{\neg(\ell_0 \sqsubseteq \ell)}{\Phi \vdash^{\ell_0}_{\ell} a_1 \Rightarrow a_2}$$

FIGURE 3.9: Parallel reduction relation (Excerpt)

$\boxed{\Phi \vdash a \Rightarrow^*_\ell b}$ *(MultiPar rules)*

$$\textsc{MultiPar-One}\quad \frac{\Phi \vdash a \Rightarrow_\ell b}{\Phi \vdash a \Rightarrow^*_\ell b}$$

$$\textsc{MultiPar-Many}\quad \frac{\Phi \vdash a \Rightarrow^*_\ell b \qquad \Phi \vdash b \Rightarrow^*_\ell c}{\Phi \vdash a \Rightarrow^*_\ell b}$$

FIGURE 3.10: Transitive closure of parallel reduction relation

transitive closure of $\Phi \vdash a \Rightarrow_\ell b$, denoted as $\Phi \vdash a \Rightarrow^*_\ell b$, has a standard inductive definition, as shown in Figure 3.10.

Lemma 3.28 (Confluence) If $\Phi \vdash a \Rightarrow_\ell a_1$ and $\Phi \vdash a \Rightarrow_\ell a_2$, then there exists b such that $\Phi \vdash a_1 \Rightarrow_\ell b$ and $\Phi \vdash a_2 \Rightarrow_\ell b$. Similarly, if $\Phi \vdash a \Rightarrow^*_\ell a_1$ and $\Phi \vdash a \Rightarrow^*_\ell a_2$, then there exists b such that $\Phi \vdash a_1 \Rightarrow^*_\ell b$ and $\Phi \vdash a_2 \Rightarrow^*_\ell b$.

Next, we define joinability using parallel reduction. Terms a_1 and a_2 are said to be joinable if and only if they can use one or more steps of parallel reduction to reach to two terms that are syntactically equal, as shown below.

$$\textsc{Join}\quad \frac{\Phi \vdash a_1 \Rightarrow^*_\ell b \qquad \Phi \vdash a_2 \Rightarrow^*_\ell b}{\Phi \vdash a_1 \Leftrightarrow_\ell a_2}$$

The joinability relation, though defined in a different way, is nevertheless equivalent to the definitional equality relation, as we see in the theorem below. Both the directions of this theorem are proved by induction on the respective premise judgments. It is worth remarking here that the proof of the 'if'-part of

$\boxed{\mathsf{Ct}\ A_1\ A_2}$ *(Consistency)*

$$\frac{}{\mathsf{Ct}\ s\ s}\ \textsc{Ct-Sort} \qquad \frac{}{\mathsf{Ct}\ \mathbf{Unit}\ \mathbf{Unit}}\ \textsc{Ct-Unit} \qquad \frac{}{\mathsf{Ct}\ (\Pi x{:}^{\ell} A_1.B_1)\ (\Pi x{:}^{\ell} A_2.B_2)}\ \textsc{Ct-Pi} \qquad \frac{}{\mathsf{Ct}\ (\Sigma x{:}^{\ell} A_1.B_1)\ (\Sigma x{:}^{\ell} A_2.B_2)}\ \textsc{Ct-Sigma}$$

$$\frac{}{\mathsf{Ct}\ (A_1 + B_1)\ (A_2 + B_2)}\ \textsc{Ct-Sum} \qquad \frac{\neg\mathsf{VType}\ A_1}{\mathsf{Ct}\ A_1\ A_2}\ \textsc{Ct-L} \qquad \frac{\neg\mathsf{VType}\ A_2}{\mathsf{Ct}\ A_1\ A_2}\ \textsc{Ct-R}$$

$\boxed{\mathsf{VType}\ A}$ *(Value Types)*

$$\frac{}{\mathsf{VType}\ s}\ \textsc{VTy-Sort} \qquad \frac{}{\mathsf{VType}\ \mathbf{Unit}}\ \textsc{VTy-Unit} \qquad \frac{}{\mathsf{VType}\ \Pi x{:}^{\ell} A.B}\ \textsc{VTy-Pi} \qquad \frac{}{\mathsf{VType}\ \Sigma x{:}^{\ell} A.B}\ \textsc{VTy-Sigma} \qquad \frac{}{\mathsf{VType}\ A + B}\ \textsc{VTy-Sum}$$

FIGURE 3.11: Consistency relation between types

this theorem relies upon confluence of parallel reduction (as stated in Lemma 3.28) for the EQ-TRANS case of the induction.

Theorem 3.29 (Joinability and Definitional Equality) $\Phi \vdash a \Leftrightarrow_{\ell} b$ if and only if $\Phi \vdash a \equiv_{\ell} b$.

Now, to show that definitional equality is consistent, we just need to show joinability is consistent. In the beginning of this subsection, we defined what it means for a relation to be consistent. We formalize that definition in terms of a judgment, $\mathsf{Ct}\ A_1\ A_2$, which is presented in Figure 3.11, and which may be read as, the types A_1 and A_2 have consistent head forms. Using this judgment, we can state consistency of joinability as:

Theorem 3.30 (Consistency) If $\Phi \vdash a \Leftrightarrow_{\ell} b$, then $\mathsf{Ct}\ a\ b$.

Together, theorems 3.29 and 3.30 imply consistency of the definitional equality relation of DDC. Now, consistency of equality is essential to guarantee type-soundness because an inconsistent equality relation could allow nonsensical type assignments, such as **true** : **Unit**. Next, using consistency, we show that DDC does not allow such nonsensical type assignments and is in fact, type-sound.

3.4.6 Type-Soundness

We show DDC is type-sound via standard preservation and progress theorems. In order to prove these theorems, we need several lemmas, similar to the ones we needed to show type-soundness of SDC in Section 3.2.2. The statements of these lemmas are very similar to their SDC counterparts. So below, we shall state them without additional explanation. Note that we prove these lemmas for DDC but the lemmas also hold for $\text{DDC}^{\top}$, since $\text{DDC}^{\top}$ is just DDC where $\mathcal{C} = \top$.

Lemma 3.31 (Narrowing) If $\Omega' \vdash a :^{\ell} A$ and $\Omega \sqsubseteq \Omega'$, then $\Omega \vdash a :^{\ell} A$

Lemma 3.32 (Restricted upgrading) If $\Omega_1, y :^{m_0} B, \Omega_2 \vdash a :^{\ell} A$ and $\ell_0 \sqsubseteq \ell$, then $\Omega_1, y :^{\ell_0 \sqcup m_0} B, \Omega_2 \vdash a :^{\ell} A$.

In the multiplication and subsumption lemmas below, note the additional condition, $m \sqsubseteq \mathcal{C}$. This condition is necessary because DDC maintains the invariant: if $\Omega \vdash a :^{\ell} A$, then $\ell \sqsubseteq \mathcal{C}$. This invariant implies that there is no typing judgment at level $\top$ in DDC.

Lemma 3.33 (Multiplication) If $\Omega \vdash a :^{\ell} A$ and $m \sqsubseteq \mathcal{C}$, then $m \sqcup \Omega \vdash a :^{m \sqcup \ell} A$.

Lemma 3.34 (Subsumption) If $\Omega \vdash a :^{\ell} A$ and $\ell \sqsubseteq m$ and $m \sqsubseteq \mathcal{C}$, then $\Omega \vdash a :^{m} A$

Lemma 3.35 (Weakening) If $\Omega_1, \Omega_2 \vdash a :^{\ell} A$ and $\Omega_1 \Vdash B :^{\top} s$, then $\Omega_1, y :^{m} B, \Omega_2 \vdash a :^{\ell} A$.

The substitution lemma for DDC, shown below, is interesting. Since DDC does not directly allow any typing derivation at level $\top$, we cannot substitute a $\top$-marked variable with a term derived via any normal typing judgment of DDC. However, such a variable may be substituted with a term derived via the truncated judgment, which as we know, can indirectly derive terms at level $\top$.

Lemma 3.36 (Substitution) If $\Omega_1, y :^{m} B, \Omega_2 \vdash a :^{\ell} A$ and $\Omega_1 \Vdash b :^{m} B$, then $\Omega_1, \Omega_2\{b/y\} \vdash a\{b/y\} :^{\ell} A\{b/y\}$

Finally, we state the preservation and progress theorems, which establish type soundness of DDC.

Theorem 3.37 (Preservation) If $\Omega \vdash a :^{\ell} A$ and $\vdash a \rightsquigarrow a'$, then $\Omega \vdash a' :^{\ell} A$.

Theorem 3.38 (Progress) If $\varnothing \vdash a :^{\ell} A$, then either a is a value or there exists some a' such that $\vdash a \rightsquigarrow a'$.

Next, we consider type-checking in DDC. DDC is a generic pure type system, parametrized by an arbitrary lattice. When the parameterizing pure type system is strongly normalizing, for example, the Calculus of Constructions, type-checking in DDC is decidable. We leave the detailed design of a decidable type-checking algorithm for future work but provide some outlines of such an algorithm next.

3.4.7 Type-Checking in DDC$^{\top}$ and DDC

On account of being a generic PTS, not all instances of DDC$^{\top}$ and DDC admit decidable type-checking. When the parametrizing PTS itself is not normalizing, type-checking becomes undecidable because the equality relation between types is undecidable. However, if the parametrizing PTS is strongly normalizing, we can design a decidable type-checking algorithm for DDC$^{\top}$ and DDC. Such an algorithm is standard, but relies upon a decision procedure for the equality relation between types. The most common of such decision procedures is normalize-and-compare [Crary, 2004], whereby types are normalized and compared for syntactic equality. But this procedure rests upon the strong normalization property of the calculus.

Proving from first principles that DDC$^{\top}$ and DDC are strongly normalizing, whenever the underlying PTS is so, is highly nontrivial and requires considerable extra machinery. But in case of DDC$^{\top}$, we can show strong normalization just by a translation to the underlying PTS. For DDC however, such a translation is not possible because DDC supports compile-time irrelevance, which allows it to type-check more terms than the underlying PTS. For example, the term, $\texttt{ida = } \lambda^{\top}\texttt{ f. } \lambda\texttt{ x. x : } \Pi\texttt{ f :}^{\top}\texttt{(}^{\top}\texttt{Nat} \to \texttt{Type). f 0}^{\top} \to \texttt{f 1}^{\top}$, from Section 3.1.2, type-checks in DDC (extended with `Nat`) but would not type-check in the underlying PTS

because if the labels are removed, the term is indeed ill-typed. However, we may show strong normalization for DDC via translation to calculi that support compile-time irrelevance and that have already been shown to be strongly normalizing. One such calculus is ICC* [Barras and Bernardo, 2008, Miquel, 2001]. ICC* extends the Calculus of Constructions (a PTS) with support for compile-time irrelevance and has been shown to be strongly normalizing. So, via a translation to ICC*, we can show that DDC is strongly normalizing, when the underlying PTS is the Calculus of Constructions. A proof of strong normalization for DDC, with an arbitrary strongly normalizing PTS as the underlying system, is left for future work.

We pointed out in the paragraph above that DDC$^\top$ is strongly normalizing, whenever the underlying PTS is so. Here, we elaborate why. First, note that the equality relation used in the conversion rule of DDC$^\top$ is just standard β-equivalence. As such, DDC$^\top$ (unlike DDC) does not type-check more terms over and above the underlying PTS. In fact, at levels other than $\top$, the calculus type-checks less terms. For example: $x :^\top A \vdash x :^\bot A$ does not hold in DDC$^\top$ but $x : A \vdash x : A$ holds in the underlying PTS. Next, observe that for any typing derivation of DDC$^\top$, if we strip away all the labels from types, terms, contexts and typing judgments, we would be left with a derivation that is valid in the underlying PTS. And finally, note that when all labels are stripped away, the operational semantics of DDC$^\top$ becomes identical to that of the underlying PTS. Together, these observations imply that an infinite sequence of reductions of a well-typed term in DDC$^\top$ would correspond to the same in the underlying PTS, which would result in a contradiction, given that the underlying PTS is strongly normalizing.

Next, as pointed out above, DDC is strongly normalizing, when the Calculus of Constructions is the underlying PTS. We show this by a translation of DDC to ICC*.

ICC* extends the Calculus of Constructions with an implicit Π-type. The implicit Π-type is akin to the Π-type of DDC, with the label on the type set to $\top$. This implicit Π-type, denoted as $\Pi[x\!:\!A].B$, is introduced and eliminated by implicit abstraction and application, denoted as $\lambda[x\!:\!A].b$ and $b\ [a]$ respectively. ICC* also has the standard (and explicit) Π-type, denoted as $\Pi(x\!:\!A).B$, which is introduced and eliminated by standard abstraction and application, denoted as $\lambda(x\!:\!A).b$ and $b\ a$ respectively. Now, the implicit Π-type is the type of functions that do not use their arguments. The type system of the calculus ensures that this is indeed the case by means of an extraction function. The extraction function is essentially an erasure function that strips away implicit lambda-binders and implicit arguments of applications. The function is defined recursively; the interesting cases of the definition are shown below.

$$(\lambda(x\!:\!A).b)^* = \lambda x.b^* \qquad (\lambda[x\!:\!A].b)^* = b^*$$
$$(b\ a)^* = b^*\ a^* \qquad (b\ [a])^* = b^*$$

Next, we look at how the extraction function is employed by the type system to ensure the expected behavior of implicit abstractions. The key typing rules of ICC* appear in Figure 3.12. Observe the rule for implicit abstractions: rule ICCS-ILam. This rule ensures that the argument x is not used explicitly in the body of the function. Now, even though an implicit abstraction cannot use the argument explicitly in its body, an implicit Π can do so, as we see in rule ICCS-IPi. This differential treatment of a bound variable in a type and in a term having that type is also followed in DDC. Note that rules ICCS-EPi, ICCS-ELam,

$\boxed{\Gamma \vdash a : A}$ *(Typing rules)*

$$\textsc{ICCS-IPi}\quad \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathcal{R}}{\Gamma \vdash \Pi[x{:}A].B : s_3}$$

$$\textsc{ICCS-ILam}\quad \frac{\Gamma, x : A \vdash b : B \qquad \Gamma \vdash \Pi[x{:}A].B : s \qquad x \notin \mathsf{fv}\, b^*}{\Gamma \vdash \lambda[x{:}A].b : \Pi[x{:}A].B}$$

$$\textsc{ICCS-IApp}\quad \frac{\Gamma \vdash b : \Pi[x{:}A].B \qquad \Gamma \vdash a : A}{\Gamma \vdash b\ [a] : B\{a/x\}}$$

$$\textsc{ICCS-EPi}\quad \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathcal{R}}{\Gamma \vdash \Pi(x{:}A).B : s_3}$$

$$\textsc{ICCS-ELam}\quad \frac{\Gamma, x : A \vdash b : B \qquad \Gamma \vdash \Pi(x{:}A).B : s}{\Gamma \vdash \lambda(x{:}A).b : \Pi(x{:}A).B}$$

$$\textsc{ICCS-EApp}\quad \frac{\Gamma \vdash b : \Pi(x{:}A).B \qquad \Gamma \vdash a : A}{\Gamma \vdash b\ a : B\{a/x\}}$$

$$\textsc{ICCS-Conv}\quad \frac{\Gamma \vdash a : A \qquad \Gamma \vdash B : s \qquad A^* \cong_{\beta\eta} B^*}{\Gamma \vdash a : B}$$

$\boxed{\vdash a \rightsquigarrow a'}$ *(Step rules)*

$$\textsc{ICCSStep-EBeta}\quad \frac{}{\vdash (\lambda(x{:}A).b)\ a \rightsquigarrow b\{a/x\}}$$

$$\textsc{ICCSStep-IBeta}\quad \frac{}{\vdash (\lambda[x{:}A].b)\ [a] \rightsquigarrow b\{a/x\}}$$

FIGURE 3.12: Key typing rules and step rules of ICC*

and ICCS-EApp, i.e., the formation, introduction and elimination rules for the explicit Π-type are standard PTS rules. Next, observe how ICC* supports compile-time irrelevance: the conversion rule ICCS-Conv checks for equality of types after stripping away the implicit λ-binders and implicit arguments. This rule is similar to the conversion rule Ddc-Conv, which checks for equality of types ignoring $\top$-marked variables and arguments. However, a technical difference between the two conversion rules is that unlike rule ICCS-Conv, rule Ddc-Conv does not check for η-equality. Finally, observe the β-rules of ICC*, also presented in Figure 3.12: these rules require that implicit and explicit functions be applied with implicit and explicit arguments respectively. They remind us of our β-rules, where we also require that the grades on the introduction form and the corresponding elimination form match up.

With this introduction to ICC*, we can now present the translation function from DDC (with Calculus of Constructions as the underlying PTS) to ICC*. Note here that ICC* is a PTS à la Barendregt [1993] and does not include Σ-types and sum types. So we shall translate just the core fragment of DDC to ICC*. We present the translation function, $\widetilde{\ }$, in Figure 3.13. The key point to note is that this function maps arguments labeled $\mathcal{C}$ and below to explicit arguments, and arguments labeled greater than $\mathcal{C}$ (i.e., $\top$) to implicit arguments.

We can show that this translation preserves operational semantics, definitional equality and typing, as stated in the lemma below. Here, $\widetilde{\Omega}$ denotes Ω with the labels on the assumptions omitted and the types translated.

Lemma 3.39 The translation function, $\widetilde{\ }$, preserves meaning and typing:

$$\widetilde{x} = x \qquad \widetilde{s} = s \qquad \widetilde{\Pi x :^{\ell} A. B} = \begin{cases} \Pi(x : \widetilde{A}).\widetilde{B} & \text{if } \ell \sqsubseteq \mathcal{C} \\ \Pi[x : \widetilde{A}].\widetilde{B} & \text{otherwise} \end{cases}$$

$$\widetilde{\lambda x :^{\ell} A. b} = \begin{cases} \lambda(x : \widetilde{A}).\widetilde{b} & \text{if } \ell \sqsubseteq \mathcal{C} \\ \lambda[x : \widetilde{A}].\widetilde{b} & \text{otherwise} \end{cases} \qquad \widetilde{b\ a^{\ell}} = \begin{cases} \widetilde{b}\ \widetilde{a} & \text{if } \ell \sqsubseteq \mathcal{C} \\ \widetilde{b}\ [\widetilde{a}] & \text{otherwise} \end{cases}$$

FIGURE 3.13: Translation from a sublanguage of DDC to ICC*

- If $\vdash a \rightsquigarrow a'$, then $\vdash \widetilde{a} \rightsquigarrow \widetilde{a'}$.
- If $\Phi \vdash A \equiv_{\mathcal{C}} B$, then $\widetilde{A}^* \cong_{\beta\eta} \widetilde{B}^*$.
- If $\Omega \vdash a :^{\ell} A$, then $\widetilde{\Omega} \vdash \widetilde{a} : \widetilde{A}$.

Now, since ICC* is a strongly normalizing calculus, by the above lemma, we can conclude the same for the specified instance of DDC as well. Again, since this instance of DDC is strongly normalizing, we can design a decision procedure for the equality relation and thereby ensure decidable type-checking.

We pointed out that if the parametrizing PTS is not normalizing, type-checking in DDC would be undecidable. However in such cases, there exists an alternative approach to ensure decidable type-checking by designing an equivalent annotated calculus, as demonstrated in Weirich et al. [2017]. The key idea is to design an annotated version of the original calculus, one that would book-keep additional information from typing and equality derivations. In the annotated calculus, the conversion rule would include an explicit proof that witnesses the equality between the concerned types. Therefore, while checking for equality between types, one can just verify the witness provided (decidable in nature) in lieu of using a normalize-and-compare strategy. Following this approach, one can design a type-checking algorithm, even for non-terminating instances of DDC.

3.5 Discussions and Related Work

3.5.1 Irrelevance Analysis in Dependent Type Theories

Analysis of compile-time and run-time irrelevance in dependent type theories is a well-studied topic [Abel and Scherer, 2012, Atkey, 2018, Barras and Bernardo, 2008, Brady, 2021, McBride, 2016, Miquel, 2001, Mishra-Linger and Sheard, 2008, Mishra-Linger, 2008, Moon et al., 2021, Nuyts and Devriese, 2018, Pfenning, 2001, Tejiščák, 2020, Weirich et al., 2017, etc]. When compared with existing literature, our $\text{DDC}^{\top}$ is closest to EPTS of Mishra-Linger [2008] and the core fragment of TT_* of Tejiščák [2020]. However, $\text{DDC}^{\top}$ can be parametrized by an arbitrary lattice whereas both EPTS and TT_* are hardwired to work with a two-element lattice only. On the other hand, DDC is the only system that we are aware of that can analyze run-time and compile-time irrelevance separately and makes use of the latter in the conversion rule. Further, DDC

analyzes these irrelevances in the presence of strong irrelevant Σ-types, something which, to the best of our knowledge, no prior work has been able to. In this regard, note that the system of Moon et al. [2021] can also analyze run-time and compile-time irrelevance separately but this system does not make use of compile-time irrelevance in the conversion rule and cannot support strong irrelevant Σ-types.

Prior work has identified the difficulty in handling strong irrelevant Σ-types in a setting that supports compile-time irrelevance. Abel and Scherer [2012] point out that such Σ-types make their theory inconsistent. Similarly, EPTS$^\bullet$ of Mishra-Linger [2008] cannot define the projections for pairs having such Σ-types. The reason behind this failure is that EPTS$^\bullet$ is hardwired to work with a two-element lattice, which leads to an identification of compile-time and run-time irrelevance. As such, projections from pairs having strong irrelevant Σ-types result in type unsoundness. For example, considering the first components to be run-time irrelevant, the pairs $(\mathbf{Int}, \mathbf{unit})$ and $(\mathbf{Bool}, \mathbf{unit})$ are run-time equivalent. Since EPTS$^\bullet$ identifies run-time and compile-time irrelevance, these pairs are also compile-time equivalent. Then, taking the first projections of these pairs, one ends up with **Int** and **Bool** being compile-time equivalent, which is unsound. We resolve this problem by distinguishing between run-time and compile-time irrelevance, using a lattice with three elements.

Next, we compare our work with existing literature with regard to the equality relation. DDC supports compile-time irrelevance, whereby the equality relation can ignore irrelevant sub-terms. However, since our equality relation is untyped, we cannot include type-dependent rules in our system, such as η-equivalence rule for the **Unit** type. Several prior works on irrelevance [Barras and Bernardo, 2008, Miquel, 2001, Mishra-Linger, 2008, Tejiščák, 2020] use an untyped equality relation. However, some prior work, such as Abel and Scherer [2012], Pfenning [2001], do consider compile-time irrelevance in the context of a type-directed equality relation. Such systems, unlike DDC, can allow type-dependent equality rules. But their drawback is that they require bound variables to be treated the same way in the body of a Π-type and in the body of a function having that type, thereby ruling out several use cases of irrelevance analysis, including the polymorphic identity function.

3.5.2 Graded-Context Type Systems

The type systems presented in this chapter are closely related to the graded-context type systems introduced in Section 1.1 [Atkey, 2018, Brunel et al., 2014, Ghica and Smith, 2014, McBride, 2016, Moon et al., 2021, Petricek et al., 2014]. Recall that those type systems are parametrized by preordered semirings and they employ graded contexts to provide a fine-grained accounting of resource usage. The type systems presented in this chapter also employ graded contexts, but for this discussion and the dissertation in general, we shall reserve the term 'graded-context type systems' to refer exclusively to the graded-context type systems parametrized by preordered semirings, and geared for usage accounting.

Now, a typical judgment from a graded-context type system looks like:

$$x :^1 \mathbf{Bool}, y :^1 \mathbf{Int}, z :^0 \mathbf{Bool} \vdash \mathbf{if}\, x\, \mathbf{then}\, y + 1\, \mathbf{else}\, y - 1 \;:^1\; \mathbf{Int}$$

The above judgment type-checks one copy of $\textbf{if}\, x\, \textbf{then}\, y + 1\, \textbf{else}\, y - 1$, where the variable x is used once (in the condition), the variable y is used once (in each of the branches) and the variable z remains unused. As such, the variables are marked with the corresponding grades in the context.

This form of judgment is very similar to the typing judgments of the systems presented in this chapter, with $q \in \mathcal{Q}$ appearing in place of $\ell \in \mathcal{L}$. However, there is a crucial difference between the two: in the typing judgments of systems presented in this chapter, any grade, $\ell \in \mathcal{L}$, may appear to the right of the turnstile, whereas in typing judgments of graded-context systems, the grade to the right of the turnstile is fixed at $1 \in \mathcal{Q}$. A graded-context type system that allows an arbitrary grade to the right of the turnstile is not closed under substitution [Atkey, 2018, McBride, 2016]. As such, these systems cannot vary the grade on typing judgments. This limitation does not pose any problem in usage analysis, which is the primary goal of graded-context type systems. However, this limitation poses problems in dependency analysis, as we shall see in Section 5.1.1. The key problem is that graded-context type systems cannot derive a join operator, like the one we derived for SDC in Section 3.2.1. Since such a join operator is essential for dependency analysis, graded-context type systems are not well-suited for it. On the other hand, dependency type systems, including the ones presented in this chapter, cannot analyze usage. The reason behind this shortcoming is that dependency type systems are parametrized by lattices, which do not really model usage.

On a close examination, we find that the mentioned difference between graded-context type systems and the dependency type systems presented in this chapter stem from the difference in their primary goals. The primary goal of graded-context type systems is usage analysis, which requires counting. On the other hand, the primary goal of the systems presented in this chapter is dependency analysis, which requires comparison. Now, one can count using a fixed reference, as one does for a physical quantity using a standard unit of measurement. Comparison, however, by its very definition, is between two entities and does not need a third fixed reference. This motivates why graded-context type systems have a fixed reference level whereas dependency type systems have a variable one. This basic difference between graded-context type systems and dependency type systems may presently seem irreconcilable. However, in Chapter 5, we shall show that this difference can indeed be reconciled, and that reconciliation will pave the way for a unified type system for usage and dependency analysis.

3.5.3 Dependency Analysis and Dependent Type Systems

Dependency analysis and dependent type systems have come together in some existing works.

Prost [2000] extends the λ-cube so that it may track dependencies, similar to what we do for pure type systems. However, unlike our approach, Prost [2000] uses sorts to track dependencies. This approach is inspired by the distinction between different sorts in the Calculus of Constructions, where computationally relevant and irrelevant terms live in sorts `Set` and `Prop` respectively. The problem with such an approach is that it ties up two distinct language features, sorts and dependency analysis, which can be treated in a more orthogonal manner, as we show in this chapter. The technical downsides of employing sorts for dependency analysis are described in detail in Mishra-Linger [2008].

The system CCCC of Bernardy and Guilhem [2013] is very related to the calculi presented in this chapter. Bernardy and Guilhem [2013] use colors to erase terms while we use grades. Colors and grades both form a lattice structure and their usage in the respective type systems are quite similar. However, Bernardy and Guilhem [2013] use internalized parametricity to reason about erasure; so it is important that their type system is logically consistent and hence, normalizing in nature. Our work, on the other hand, does not rely on the normalizing nature of the underlying type system; we take a more direct route to analyzing erasure. As such, our work can also applied to type systems that allow nonterminating computations, such as, the Type-in-Type calculus.

Lourenço and Caires [2015] design a calculus for tracking information flow in a dependent type system, similar to the calculi presented in this chapter. But Lourenço and Caires [2015] focus on more imperative features, like modeling of state, whereas we focus on irrelevance analysis. A distinguishing feature of their system though is that they allow security labels to depend upon run-time values, something that we don't attempt here. Adding value-dependent security labels to DDC and more generally, treating labels as first-class citizens, just like types and terms, seems to be an interesting direction for future research.

3.6 Conclusion

We started this chapter with the aim of designing a dependently-typed calculus that can analyze dependencies in general, and run-time and compile-time irrelevance in particular. Towards this end, we first designed a simple dependency calculus, SDC, and thereafter extended it to two dependently-typed calculi, $\text{DDC}^{\top}$ and DDC. $\text{DDC}^{\top}$ can analyze run-time irrelevance while DDC can analyze both run-time and compile-time irrelevance, in addition to other dependencies. A novelty of DDC is that it can analyze strong irrelevant Σ-types, which are outside the scope of existing systems analyzing irrelevance. Another novelty of DDC lies in its use of graded contexts and graded typing judgments, which enable a straightforward syntactic proof of noninterference, both in simply-typed and dependently-typed settings. With these useful features, DDC can serve as a foundation for future dependency calculi for dependent type systems.

Chapter 4

Linearity Analysis in Pure Type Systems

Recall from Section 1.1 that the graded-context type systems based on bounded linear logic [Brunel et al., 2014, Ghica and Smith, 2014, Petricek et al., 2014] can analyze a wide variety of usages: linear usage, no usage, bounded usage, unrestricted usage, etc. However, these systems are restricted to simple/polymorphic types. In this chapter, we extend them to dependent types.

In particular, this chapter makes the following contributions:

- Section 4.2 presents GLC, a graded-context simple type system, based on bounded linear logic, with standard algebraic types and a graded modal type. This system is not novel; instead, it establishes a foundation for the dependent system. However, even at this stage, we identify subtleties in the design space.
- Section 4.3 presents a heap-based operational semantics for GLC, inspired by Turner and Wadler [1999]. Using this heap-based semantics, we show that usage accounting in GLC is sound. A heap-based semantics is necessary to show soundness of usage accounting because standard operational semantics does not track resource usage.
- Section 4.4 presents interesting properties that follow from soundness of usage accounting, such as, the *single pointer property* for linear resources [Turner and Wadler, 1999]. The single pointer property says that a linear resource is referenced by precisely one pointer at run time. This property is interesting because it ensures safety of in-place update of linear resources.
- Sections 4.6 and 4.7 present the key contribution of this chapter: the language, GRAD, which extends usage analysis from simple to dependent types. We describe the design of the type system of GRAD in Section 4.6 and show the type system is sound with respect to heap-based semantics in

(A) Ordering in $\mathcal{Q}_{\text{Lin}}$ (B) Ordering in $\mathcal{Q}_{\text{Aff}}$

FIGURE 4.1: Ordering for tracking linear usage and affine usage

Section 4.7. From this soundness result, we derive several interesting corollaries, including well-known 'free theorems'.

Both GLC and GRAD are graded-context type systems, parametrized over an arbitrary preordered semiring. In Section 1.1.6, we discussed why a preordered semiring is an appropriate algebraic structure to model resource usage in type systems. In that section, we also saw some examples of preordered semirings that are employed to model resource usage. In the next section, we shall review these examples and add new ones. We shall also point out subtle distinctions among the different algebraic structures that parametrize graded-context type systems and aid in usage accounting.

4.1 Modeling Usage: Preordered Semirings and Variants

We first look at a few preordered semirings that are interesting from the perspective of usage analysis:

- The *boolean semiring*, $\mathbb{B}$, has two elements, 0 and 1, with the property that $1 + 1 = 1$. This semiring admits two distinct preorders that make sense: $\mathbb{B}_{=}$, the semiring with discrete order and $\mathbb{B}_{\geq}$, the semiring with order $1 <: 0$. When parametrized over $\mathbb{B}_{=}$, a type system precisely tracks run-time relevance: if a variable is marked with 0 in the context, then that variable is not used at run time, otherwise it is used at run time. When parametrized over $\mathbb{B}_{\geq}$, a type system can only track run-time irrelevance: if a variable is marked with 0 in the context, then that variable is not used at run time. A type system parametrized by the boolean semiring with the preorder $0 <: 1$ does not make sense from the perspective of usage analysis because it allows derivations like $x :^0 A \vdash x : A$, where a resource is used but its usage is marked as 0.

- The *linearity* and *affinity* preordered semirings, denoted $\mathcal{Q}_{\text{Lin}}$ and $\mathcal{Q}_{\text{Aff}}$ respectively, have three elements, 0, 1 and ω. Addition and multiplication in these semirings are defined in the usual way after interpreting ω as 'greater than 1'. So then, $1 + 1 = \omega + 1 = 1 + \omega = \omega + \omega = \omega$ and $\omega \cdot \omega = \omega$. However, $\mathcal{Q}_{\text{Lin}}$ and $\mathcal{Q}_{\text{Aff}}$ are ordered differently, as shown in Figure 4.1. Owing to this difference in ordering, 1 signifies linear usage in $\mathcal{Q}_{\text{Lin}}$ but affine usage in $\mathcal{Q}_{\text{Aff}}$. However, in both $\mathcal{Q}_{\text{Lin}}$ and $\mathcal{Q}_{\text{Aff}}$, 0 signifies no usage while ω signifies unrestricted usage.

- The set of natural numbers, $\mathbb{N}$, with operations defined as usual, form a semiring. There are several preorders that make sense for the semiring of natural numbers. Two of them are particularly interesting:

(Grammar)

grades	$q, r \in Q$		
types	A, B, C	$::=$	$\mathbf{Unit} \mid {}^{q}A \to B \mid \Box^{q}\, A \mid {}^{q}A \times B \mid A + B$
terms	a, b, c	$::=$	$x \mid \lambda^{q} x{:}A.a \mid a\ b^{q}$
		$\mid$	$\mathbf{unit} \mid \mathbf{let\,unit} = a\ \mathbf{in}\ b \mid \mathbf{box}_q\ a \mid \mathbf{let\,box}_q\ x = a\ \mathbf{in}\ b$
		$\mid$	$(a^{q}, b) \mid \mathbf{let}\,({x_1}^{q}, x_2) = a\ \mathbf{in}\ b$
		$\mid$	$\mathbf{inj}_1\ a \mid \mathbf{inj}_2\ a \mid \mathbf{case}_q\ a\,\mathbf{of}\ b_1; b_2$
graded contexts	Γ	$::=$	$\varnothing \mid \Gamma, x{:}^{q} A$
typing judgment			$\boxed{\Gamma \vdash a : A}$

Figure 4.2: Grammar of GLC

$\mathbb{N}_{=}$, the semiring with discrete order and $\mathbb{N}_{\geq}$, the semiring with order $\ldots <: 4 <: 3 <: 2 <: 1 <: 0$. Type systems parametrized over $\mathbb{N}_{=}$ and $\mathbb{N}_{\geq}$ track exact and bounded usage respectively. To elaborate, if a variable in the context is marked with $n \in \mathbb{N}$ in these type systems, then it is used exactly n times and at most n times respectively at run time.

Algebraic structures close to preordered semirings have been used to track resource usage in many type systems [Abel and Bernardy, 2020, Atkey, 2018, Brunel et al., 2014, Gaboardi et al., 2016, Ghica and Smith, 2014, McBride, 2016, Orchard et al., 2019, Petricek et al., 2014]. However, there are some variations with respect to the formal requirements of such algebraic structures. For example: Brunel et al. [2014] require the algebraic structure to be a semiring with an order relation that forms a bounded sup-semilattice. Abel and Bernardy [2020] require the algebraic structure to be a semiring with an additional meet operator. McBride [2016] require the algebraic structure to be a hemiring [Golan, 1999], which is essentially a semiring without multiplicative identity. Atkey [2018] require the algebraic structure to be a semiring that satisfies additional restrictions. Our calculi, however, are parametrized over an arbitrary preordered semiring, exactly as defined in Section 1.1.6. We add additional constraints, as required, only while deriving specific properties in Sections 4.4.1, 4.4.2 and 4.7.2.

Next, we present GLC, our graded-context simple type system for usage accounting.

4.2 A Graded-Context Simple Type System

4.2.1 The Basics

Our goal is to design a dependently-typed language that analyzes resource usage. In Section 1.1.8, we discussed how graded-context type systems provide us the flexibility that is necessary to combine usage analysis with dependent types. So, in this chapter, to analyze usage in dependently-typed languages, we design a graded-context dependent type system. Many of the ideas pertaining to usage analysis, however,

are best understood in a simply-typed setting. So, we start with a graded-context simply-typed language, which is similar to the one by Petricek et al. [2014]. We call the language GLC, for Graded λ-calculus.

GLC is parametrized over an arbitrary preordered semiring, $\mathcal{Q} = (Q, 1, \cdot, 0, +, <:)$. The grammar for GLC($\mathcal{Q}$) appears in Figure 4.2. The typing judgment for GLC($\mathbf{u}$) has the form: $\Gamma \vdash a : A$, where Γ is a graded context. We explain the intuition behind this judgment below. The meaning of the judgment, $x_1 :^{q_1} A_1, x_2 :^{q_2} A_2, \ldots, x_n :^{q_n} A_n \vdash b : B$, is: at run time, to evaluate one copy of b, we use q_i copies of x_i, where i ranges from 1 to n.

A natural question to ask here is: what does using q_i copies of x_i mean? More generally, what does variable 'use' mean? If we think of variables as pointers to data in memory, a 'use' can be seen as a memory look-up of data during program evaluation. So, using q_i copies of x_i may be understood as q_i look-ups of the data pointed to by x_i. Now, the number of memory look-ups during program evaluation depends on the reduction strategy employed. So, when we say that q_i copies of x_i are used in evaluating b, we are surely assuming some reduction strategy. Indeed, this is the case.

The type system of GLC accounts usage, implicitly assuming a call-by-name reduction strategy. Had we assumed other reduction strategies, for example, call-by-value or call-by-need, our usage accounting would have been quite different. To see why, let us consider some examples. While evaluating the term, $(\lambda x.42)\, z$, using a call-by-name reduction strategy, we do not look-up z, but we do so while using a call-by-value reduction strategy. Again, while evaluating the term, $\mathbf{let}\, x = 42\ \mathbf{in}\ \mathbf{let}\, y = x\ \mathbf{in}\ y + y$, using a call-by-name reduction strategy, we look-up x twice, but while using a call-by-need reduction strategy, we look-up x only once. So then, the meaning of the judgment, $x_1 :^{q_1} A_1, x_2 :^{q_2} A_2, \ldots, x_n :^{q_n} A_n \vdash b : B$, may be refined as: to evaluate b using a call-by-name reduction strategy, we look-up q_i times the data pointed to by x_i, where i ranges from 1 to n. Informally, we say that b uses x_i for q_i times or that one copy of b needs q_i copies of x_i.

Next, GLC is a graded-context type system. In Section 1.1.7, we saw how graded contexts aid usage accounting via context operations. Now, context operations endow graded contexts with an algebraic structure. Below, we explore this algebraic structure. A graded-context, Γ, has two components: $\lfloor \Gamma \rfloor$, the underlying standard context and $\overline{\Gamma}$, the vector of grades in Γ. Now, by lifting the operations and relation defined on grades, i.e., $+$, $\cdot$ and $<:$, we can also define them on graded contexts. Given graded-contexts Γ_1 and Γ_2 such that $\lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor$, we define $\Gamma_1 + \Gamma_2$ as the graded-context Γ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$ and $\overline{\Gamma} = \overline{\Gamma_1} + \overline{\Gamma_2}$ (point-wise addition). Similarly, given a graded-context Γ, for any $q \in Q$, we define $q \cdot \Gamma$ as the graded context obtained by pre-multiplying every grade in Γ by q. Further, given graded-contexts Γ_1 and Γ_2 such that $\lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor$, we say $\Gamma_1 <: \Gamma_2$ if and only if $\overline{\Gamma_1} <: \overline{\Gamma_2}$ (pointwise order). With these operations and relation thus defined, graded contexts form a nice algebraic structure: for any standard context Δ, graded-contexts Γ that satisfy $\lfloor \Gamma \rfloor = \Delta$ form a preordered left $\mathcal{Q}$-semimodule [Golan, 1999]. (A proof of this proposition appears in Appendix C.1.) This algebraic structure provides an abstract characterization of graded contexts. We don't explore the implications of this abstract characterization, but leave it for future work.

4.2.2 Type System

We now look at the type system of GLC. The typing rules appear inline below.

$$\text{GLC-VAR}\quad \frac{}{0\cdot\Gamma, x :^1 A \vdash x : A}
\qquad
\text{GLC-WEAK}\quad \frac{\Gamma \vdash a : A}{\Gamma, y :^0 B \vdash a : A}$$

$$\text{GLC-UNIT}\quad \frac{}{\varnothing \vdash \mathbf{unit} : \mathbf{Unit}}
\qquad
\text{GLC-LETUNIT}\quad \frac{\Gamma_1 \vdash a : \mathbf{Unit} \qquad \Gamma_2 \vdash b : B \qquad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor}{\Gamma_1 + \Gamma_2 \vdash \mathbf{let\,unit} = a \ \mathbf{in}\ b : B}$$

In rule GLC-VAR, x just needs one copy of x and no copy of any other variable; so in the context, x appears at grade 1 and all other variables at grade 0. Note that $x :^1 A$ occurs as the last assumption in the context. As such, an explicit weakening rule GLC-WEAK is necessary. Next, constants, like **unit**, do not need any resource, so the context in rule GLC-UNIT is empty. To eliminate a term of type **Unit**, we match it with **unit** in rule GLC-LETUNIT. Since the elimination form requires resources used by both the terms, we add the two contexts in the conclusion.

$$\text{GLC-LAM}\quad \frac{\Gamma, x :^q A \vdash b : B}{\Gamma \vdash \lambda^q x : A . b : ({}^q A \to B)}
\qquad
\text{GLC-APP}\quad \frac{\Gamma_1 \vdash b : {}^q A \to B \qquad \Gamma_2 \vdash a : A \qquad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor}{\Gamma_1 + q \cdot \Gamma_2 \vdash b\ a^q : B}$$

Coming to function types, ${}^q A \to B$ is the type of functions that use their argument q times. In the introduction rule GLC-LAM, $\lambda^q x : A . b$ needs Γ and q copies of its argument; so b would need Γ and q copies of x. In the elimination rule GLC-APP, b needs Γ_1 and q copies of its argument. But each copy of a, to-be argument of b, needs Γ_2. Then, q copies of a would need $q \cdot \Gamma_2$. Therefore, $b\ a^q$ would need $\Gamma_1 + q \cdot \Gamma_2$.

$$\text{GLC-BOX}\quad \frac{\Gamma \vdash a : A}{q \cdot \Gamma \vdash \mathbf{box}_q\ a : \Box^q A}
\qquad
\text{GLC-LETBOX}\quad \frac{\Gamma_1 \vdash a : \Box^q A \qquad \Gamma_2, x :^q A \vdash b : B \qquad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor}{\Gamma_1 + \Gamma_2 \vdash \mathbf{let\,box}_q\ x = a \ \mathbf{in}\ b : B}$$

Next, the graded modal type, $\Box^q A$, is introduced by the construct $\mathbf{box}_q\ a$, which uses the expression q times to build the box. This box can then be passed around as an entity in itself. When unboxed through rule GLC-LETBOX, the continuation would have access to q copies of the content inside the box.

Note the comonadic nature of the modality here. Over the preordered monoid $(Q, \cdot, 1, <:)$, one can derive the standard extract and fork functions for this modality:

$$\mathbf{extract} := \lambda x . \mathbf{let\,box}_1\ y = x\ \mathbf{in}\ y\ :\ \Box^1 A \to A$$

$$\mathbf{fork}^{q_1, q_2} := \lambda x . \mathbf{let\,box}_{q_1 \cdot q_2}\ y = x\ \mathbf{in}\ \mathbf{box}_{q_1}\ \mathbf{box}_{q_2}\ y\ :\ \Box^{q_1 \cdot q_2} A \to \Box^{q_1}\ \Box^{q_2}\ A$$

Here, A is an arbitrary type and $q_1, q_2 \in Q$ (missing grades are assumed to be 1). The comonadic nature of this modality is essential to supporting usage analysis. To give an example, for supporting promotion, i.e., $!A \to !!A$, or equivalently $\Box^\omega A \to \Box^\omega \Box^\omega A$, one needs $\mathbf{fork}^{\omega,\omega}$.

$$\text{GLC-Pair}\quad \frac{\Gamma_1 \vdash a_1 : A_1 \qquad \Gamma_2 \vdash a_2 : A_2 \qquad \lfloor\Gamma_1\rfloor = \lfloor\Gamma_2\rfloor}{q \cdot \Gamma_1 + \Gamma_2 \vdash (a_1{}^q, a_2) : {}^qA_1 \times A_2}$$

$$\text{GLC-LetPair}\quad \frac{\Gamma_1 \vdash a : {}^qA_1 \times A_2 \qquad \Gamma_2, x_1 :^q A_1, x_2 :^1 A_2 \vdash b : B \qquad \lfloor\Gamma_1\rfloor = \lfloor\Gamma_2\rfloor}{\Gamma_1 + \Gamma_2 \vdash \mathbf{let}\,(x_1{}^q, x_2) = a \;\mathbf{in}\; b : B}$$

Next, ${}^qA \times B$ is the type of pairs that contain q copies of first component and 1 copy of second component. The components of such pairs do not share variable usages. Therefore, the introduction rule adds the resource requirements. Note that such pairs must be eliminated via pattern-matching because both the components should be used in the continuation. An elimination form that projects only one component of the pair cannot, in general, honor the usage constraints vis-à-vis the other component.

$$\text{GLC-Inj1}\quad \frac{\Gamma \vdash a : A_1}{\Gamma \vdash \mathbf{inj}_1\, a : A_1 + A_2}$$

$$\text{GLC-Case}\quad \frac{q <: 1 \qquad \Gamma_1 \vdash a : A_1 + A_2 \qquad \Gamma_2, x_1 :^q A_1 \vdash b_1 : B \qquad \Gamma_2, x_2 :^q A_2 \vdash b_2 : B \qquad \lfloor\Gamma_1\rfloor = \lfloor\Gamma_2\rfloor}{q \cdot \Gamma_1 + \Gamma_2 \vdash \mathbf{case}_q\, a\, \mathbf{of}\, x_1.b_1; x_2.b_2 : B}$$

Finally, we look at the rules for sum types. The introduction rule is as expected. In the elimination rule GLC-Case, the scrutinee and the branches may have different resource requirements. Further, the branches may need multiple copies of the scrutinee. So, in the conclusion judgment, the resource requirement of the scrutinee is first multiplied by the appropriate grade and then added to the resource requirement of the branches. Note the side condition, $q <: 1$, in this rule. This condition reveals an interesting aspect of the reduction strategy implicit in our accounting. In a call-by-name reduction of $\mathbf{case}_q\, a\, \mathbf{of}\, x_1.b_1; x_2.b_2$, the scrutinee a will first be reduced to a value, irrespective of whether the branches require it or not. Now, suppose the branches don't require it, which would mean $q = 0$ in the above rule. But then, we have already 'used up' resources in reducing a; we cannot go back and undo that usage. Therefore, we require that the branches 'use' a at least once.

In GLC, we also have a sub-usaging rule that allows us to provide more resources than is necessary. Note here that the relation $\Gamma_2 <: \Gamma_1$ means Γ_2 has more resources than Γ_1.

$$\text{GLC-Sub}\quad \frac{\Gamma_1 \vdash a : A \qquad \Gamma_2 <: \Gamma_1}{\Gamma_2 \vdash a : A}$$

$\boxed{\vdash a \leadsto a'}$ *(Small-step reduction)*

S-AppBeta
$$\frac{}{\vdash (\lambda^q x{:}A.b)\ a^q \leadsto b\{a/x\}}$$

S-LetBoxBeta
$$\frac{}{\vdash \textbf{let}\,\textbf{box}_q\ x = \textbf{box}_q\ a\ \textbf{in}\ b \leadsto b\{a/x\}}$$

S-LetPairBeta
$$\frac{}{\vdash \textbf{let}\,({x_1}^q, x_2) = ({a_1}^q, a_2)\ \textbf{in}\ b \leadsto b\{a_1/x_1\}\{a_2/x_2\}}$$

Figure 4.3: Small-step call-by-name reduction (Excerpt)

Now, let us consider a few examples of terms that can be derived and terms that cannot be derived in GLC($\mathcal{Q}_{\text{Lin}}$):

$$\varnothing \vdash \lambda^1 x.x : {}^1A \to A \qquad \varnothing \nvdash \lambda^0 x.x : {}^0A \to A$$
$$\varnothing \vdash \lambda^\omega x.(x^1, x) : {}^\omega A \to {}^1A \times A \qquad \varnothing \nvdash \lambda^1 x.(x^1, x) : {}^1A \to {}^1A \times A$$
$$\varnothing \vdash \lambda^\omega x.\textbf{unit} : {}^\omega A \to \textbf{Unit} \qquad \varnothing \nvdash \lambda^1 x.\textbf{unit} : {}^1A \to \textbf{Unit}$$

Note that the term $\lambda^0 x.x$ uses a resource that should not be used whereas the term $\lambda^1 x.(x^1, x)$ copies a linear resource and the term $\lambda^1 x.\textbf{unit}$ discards a linear resource. As such, these terms are not derivable in GLC($\mathcal{Q}_{\text{Lin}}$). On the other hand, the term $\lambda^1 x.x$ uses a linear resource once whereas the term $\lambda^\omega x.(x^1, x)$ copies an unrestricted resource and the term $\lambda^\omega x.\textbf{unit}$ discards an unrestricted resource. As such, these terms are derivable in GLC($\mathcal{Q}_{\text{Lin}}$).

4.2.3 Type Soundness

Next, we look at the metatheoretic properties of GLC. The weakening and substitution lemmas for GLC are presented below. The weakening lemma adds an extra assumption to the context, but at grade 0, because that assumption is not used by the derived term. The substitution lemma is quite interesting because it accounts the resources required by the term after substitution: if the variable being substituted is used r times, then the context of the substitute is multiplied by r and added to the context of the original term.

Lemma 4.1 (Weakening) If $\Gamma_1, \Gamma_2 \vdash a : A$, then $\Gamma_1, z :^0 C, \Gamma_2 \vdash a : A$.

Lemma 4.2 (Substitution) If $\Gamma_1, z :^r C, \Gamma_2 \vdash a : A$ and $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$, then $\Gamma_1 + r \cdot \Gamma, \Gamma_2 \vdash a\{c/z\} : A$.

Now, we consider the operational semantics of the language. Since the type system accounts usage with respect to a call-by-name semantics, we employ a call-by-name small-step reduction strategy for the language. All the step rules are standard other than the β-rules that appear in Figure 4.3. These rules ensure that the grade in the introduction form matches with that in the elimination form. With respect to this operational semantics, we can show that GLC is type-sound, via the standard preservation and progress theorems.

Theorem 4.3 (Preservation) If $\Gamma \vdash a : A$ and $\vdash a \rightsquigarrow a'$, then $\Gamma \vdash a' : A$.

Theorem 4.4 (Progress) If $\varnothing \vdash a : A$, then either a is a value or there exists a' such that $\vdash a \rightsquigarrow a'$.

The standard type-soundness theorem, however, is not very informative with regard to usage analysis because standard operational semantics does not really track usage. In order to address this issue, we need to devise a semantics that can track resource usage during computation. To devise such a semantics, we consult existing literature. Turner and Wadler [1999] develop a heap-based semantics to track linear and nonlinear usage of resources. They use this semantics to prove soundness of their linear simple type system and from soundness, they derive useful properties, like the single pointer property for linear resources. In the following section, we shall extend their heap-based semantics so that it can track not only linear and nonlinear usages but also usages over an arbitrary preordered semiring. This extension will help us reason about resource usage in GLC and also help us show that usage accounting in GLC is indeed sound.

4.3 Heap Semantics for GLC

The heap semantics presented here shows how a term evaluates when the free variables of the term are assigned other terms. These assignments are stored in a heap, represented here as an ordered list. To each assignment in the heap, we associate an *allowed usage*, represented by a grade. We change the allowed usage as reduction consumes resources. For example, a typical reduction goes like this:

$$\begin{array}{ll}
[x \overset{3}{\mapsto} 20, y \overset{1}{\mapsto} x + x](x + y) & \textit{look up value of } x\textit{, decrement its usage} \\
\Rightarrow [x \overset{2}{\mapsto} 20, y \overset{1}{\mapsto} x + x]20 + y & \textit{look up value of } y\textit{, decrement its usage} \\
\Rightarrow [x \overset{2}{\mapsto} 20, y \overset{0}{\mapsto} x + x]20 + (x + x) & \textit{look up value of } x\textit{, decrement its usage} \\
\Rightarrow [x \overset{1}{\mapsto} 20, y \overset{0}{\mapsto} x + x]20 + (20 + x) & \textit{look up value of } x\textit{, decrement its usage} \\
\Rightarrow [x \overset{0}{\mapsto} 20, y \overset{0}{\mapsto} x + x]20 + (20 + 20) & \textit{addition step} \\
\Rightarrow [x \overset{0}{\mapsto} 20, y \overset{0}{\mapsto} x + x]60 &
\end{array}$$

There are a few points to note. First, we assume that heaps do not define any variable more than once. Second, we also assume that heaps do not contain any cyclical reference, i.e. the definition of a variable does not refer to itself or to any other variable appearing subsequently in the heap. Third, the grades in the above example and other similar examples that appear later in this chapter are drawn from $\mathbb{N}_=$, the semiring of natural numbers with discrete order. Fourth, we don't have an **Int** type and a + function (over **Int**) in our language, but we use them in our examples.

Now, the example above expresses reduction informally as a sequence of heap-expression pairs. Next, we formalize the reduction in terms of a step judgment.

4.3.1 The Step Judgment

The step judgment looks like:

$$[H]\, a \Rightarrow^{r}_{S} [H' \,;\, \mathbf{u'} \,;\, \Gamma']\, a'$$

The meaning of this judgment is that r copies of the term a use the resources of heap H and step to r copies of the term a', with H' being the new heap. The judgment is sometimes written without r, in which case it is assumed to be 1. This judgment also maintains additional information, which we shall explain one by one below.

First, a heap is an ordered list of assignments. The assignments are of the form $x \overset{q}{\mapsto} (\Gamma|a|A)$, which associates an *assignee variable*, x, with its *allowed usage*, q, and the assigned term, a. An assignment also includes some additional information, viz., an *embedded context*, Γ, and an *embedded type*, A, which are not important as far as reductions are concerned but are present to help in the proof of the soundness theorem. Note that similar to a context Γ, a heap H has two components: $\lfloor H \rfloor$, the underlying heap without the allowed usages and $\overline{H}$, the vector of allowed usages of the assignments in H. Further, note that when we need to refer to the list of assignments in H, without the allowed usages, embedded contexts and types, we shall use $\llbracket H \rrbracket$.

Now, we explain the role of S. Since we use a call-by-name reduction, we don't evaluate the terms in the heap; we just modify the grades associated with the terms as they are retrieved. Therefore, after any step, H' will contain all the assignments of H, possibly at different allowed usages. But we may add new assignments to the (tail-end of the) heap during reduction. For example, $[\varnothing]\, ((\lambda^1 x.x)\, 10) + x \Rightarrow [y \overset{1}{\mapsto} 10]\, y + x$. Note the renaming of variable in this example. Such renaming is necessary to avoid variable capture. To allocate new variable names appropriately, we need to keep track of the support set of variables [Pitts, 2013], that must be avoided while choosing fresh names. That support set is denoted by S.

Next, we explain the role of Γ'. Γ', also referred to as the *added context*, records the new names added to the heap, along with their allowed usages and types. To give an example, the added context corresponding to the new assignment, $x \overset{q}{\mapsto} (\Gamma|a|A)$, is $x :^q A$. Note that with this setup, we have, $\mathtt{dom}\ H' = \mathtt{dom}\ H \cup \mathtt{dom}\ \Gamma'$, where $\mathtt{dom}$ denotes the domain function.

Finally, we explain the role of $\mathbf{u'}$. Because we work with an arbitrary preordered semiring, possibly without subtraction, our heap-based reduction relation is non-deterministic. We explain why with the help of an example. Consider the step: $[x \overset{q}{\mapsto} a]x \Rightarrow [x \overset{q'}{\mapsto} a]a$, where $q = q' + 1$. Here, we are using x once, so we need to reduce its allowed usage by 1. But in some preordered semirings, there may exist multiple grades, $q'' \neq q'$, such that $q = q' + 1 = q'' + 1$. For example, in the linearity semiring $\mathcal{Q}_{\text{Lin}}$, we have, $\omega = 1 + 1 = \omega + 1$. So, in $\mathcal{Q}_{\text{Lin}}$, $[x \overset{\omega}{\mapsto} a]x$ steps in two different ways: $[x \overset{\omega}{\mapsto} a]x \Rightarrow [x \overset{1}{\mapsto} a]a$ and $[x \overset{\omega}{\mapsto} a]x \Rightarrow [x \overset{\omega}{\mapsto} a]a$. This nondeterministic nature of reduction implies that given an initial heap and a final heap, we really don't know how much resource has been consumed by the computation. The only way to know this is to keep track of resources while they are being consumed. And this is where $\mathbf{u'}$ comes in! The amount of resources consumed is expressed using $\mathbf{u'}$, also referred to as the *consumption vector*, with its components showing usage of the corresponding variables in H in any given step. (For any newly-added definition, we set the corresponding component of $\mathbf{u'}$ to 0.)

With this understanding of the step judgment, let us now look at some of the step rules.

4.3.2 Step Rules

First, we look at the variable rule.

$$\textsc{Heap-Var}\quad \frac{r <: 1}{[H_1, x \overset{(q+r)}{\mapsto} (\Gamma|a|A), H_2]\, x \Rightarrow^r_S [H_1, x \overset{q}{\mapsto} (\Gamma|a|A), H_2\, ;\, \mathbf{0}^{|H_1|} \diamond r \diamond \mathbf{0}^{|H_2|}\, ;\, \varnothing]\, a}$$

Here, $\diamond$ is the concatenation function; $|_|$ is the length function; and $\mathbf{0}^n$ denotes a vector of 0's of length n. We may write $\mathbf{0}$ for $\mathbf{0}^n$, if n can be inferred. There are a few points to note in this rule. First, we look up r copies of x and get r copies of a. So, the allowed usage of x changes from $(q + r)$ to q. Second, the consumption vector records the information that r copies of x have been used. Third, because we are interested only in reducing one or more copies of x, we add the precondition, $r <: 1$, to the rule. Fourth, since no new definitions are added to the heap, the added context is empty.

Next, we look at the rules for function application.

$$\textsc{Heap-AppL}\quad \frac{[H]\, b \Rightarrow^r_{S \cup \mathrm{fv}\, a} [H'\, ;\, \mathbf{u'}\, ;\, \Gamma]\, b'}{[H]\, b\ a^q \Rightarrow^r_S [H'\, ;\, \mathbf{u'}\, ;\, \Gamma]\, b'\ a^q}$$

$$\textsc{Heap-AppBeta}\quad \frac{y\ \mathit{fresh} \qquad b' = b\{y/x\}}{[H]\, (\lambda^q x{:}A_0.b)\ a^q \Rightarrow^r_S [H, y \overset{r\cdot q}{\mapsto} (\Gamma|a|A)\, ;\, \mathbf{0}\, ;\, y :^{r\cdot q} A]\, b'}$$

The rule Heap-AppL is straightforward. Note, however, the change in support set. In rule Heap-AppBeta, there are a few things to note. First, the rule loads the substitution into the heap instead of carrying it out. By allowing substitutions via the heap only, the semantics makes sure that substitutions honor usage constraints. Second, since the rule reduces r copies of the application, where each copy uses its argument q times, the new assignment in the heap has allowed usage $r \cdot q$. Third, the rule renames the bound variable x with a fresh name y. Fourth, in the new assignment, the embedded context Γ and the embedded type A are arbitrary because the step relation (unlike the soundness theorem) is not concerned with appropriateness of typing. Fifth, information about the new assignment is recorded in the added context, $y :^{r\cdot q} A$.

Next, we look at the rules for modal terms and pairs. The left step rules are as expected; the β-rules are shown below.

$$\textsc{Heap-LetBoxBeta}\quad \frac{y\ \mathit{fresh} \qquad b' = b\{y/x\}}{[H]\, \mathbf{let}\, \mathbf{box}_q\, x = \mathbf{box}_q\ a\ \mathbf{in}\ b \Rightarrow^r_S [H, x \overset{r\cdot q}{\mapsto} (\Gamma|a|A)\, ;\, \mathbf{0}\, ;\, x :^{r\cdot q} A]\, b'}$$

HEAP-LETPAIRBETA

$$\frac{\begin{array}{c} y_1, y_2 \textit{ fresh} \\ b' = b\{y_1/x_1\}\{y_2/x_2\} \end{array}}{[H]\,\mathbf{let}\,(x_1{}^q, x_2) = (a_1{}^q, a_2)\ \mathbf{in}\ b \Rightarrow^r_S [H, y_1 \overset{r\cdot q}{\mapsto} (\Gamma_1|a_1|A_1), y_2 \overset{r}{\mapsto} (\Gamma_2|a_2|A_2)\,;\mathbf{0}\,;\,y_1 :^{r\cdot q} A_1, y_2 :^r A_2]\,b'}$$

Note the similarity between these rules and rule HEAP-APPBETA. Like rule HEAP-APPBETA, in rule HEAP-LETBOXBETA, the new assignment has allowed usage $r \cdot q$ since we are reducing r copies, where each copy uses the term inside the box q times. Similarly, in rule HEAP-LETPAIRBETA, the new assignments have allowed usages $r \cdot q$ and r respectively because again we are reducing r copies and each copy uses the first and second components of the pair q times and 1 time respectively.

Next, we look at the rules for sums.

HEAP-CASEL

$$\frac{[H]\,a \Rightarrow^r_{S\,\cup(\,\mathrm{fv}\,b_1 - \{x_1\})\,\cup(\,\mathrm{fv}\,b_2 - \{x_2\})} [H'\,;\,\mathbf{u}'\,;\,\Gamma]\,a'}{[H]\,\mathbf{case}_q\,a\,\mathbf{of}\,x_1.b_1; x_2.b_2 \Rightarrow^r_S [H'\,;\,\mathbf{u}'\,;\,\Gamma]\,\mathbf{case}_q\,a'\,\mathbf{of}\,x_1.b_1; x_2.b_2}$$

HEAP-CASEBETA1

$$\frac{y_1 \textit{ fresh} \qquad b_1' = b_1\{y_1/x_1\}}{[H]\,\mathbf{case}_q\,(\mathbf{inj}_1\,a_1)\,\mathbf{of}\,x_1.b_1; x_2.b_2 \Rightarrow^r_S [H, y_1 \overset{r\cdot q}{\mapsto} (\Gamma_1|a_1|A_1)\,;\mathbf{0}\,;\,y_1 :^{r\cdot q} A_1]\,b_1'}$$

Rule HEAP-CASEL is as expected. Note, however, the change in support set. In rule HEAP-CASEBETA1, similar to the other β-rules, we add a new assignment to the heap with allowed usage $r \cdot q$ because we are reducing r copies of a **case**-expression, whose branches use the respective pattern variables q times.

Finally, we look at the rules for subusage.

HEAP-SUBL

$$\frac{\begin{array}{c}[H_1]\,a \Rightarrow^r_S [H'\,;\,\mathbf{u}'\,;\,\Gamma]\,a' \\ H_2 <: H_1\end{array}}{[H_2]\,a \Rightarrow^r_S [H'\,;\,\mathbf{u}'\,;\,\Gamma]\,a'}$$

HEAP-SUBR

$$\frac{\begin{array}{c}[H]\,a \Rightarrow^{r_1}_S [H'\,;\,\mathbf{u}'\,;\,\Gamma]\,a' \\ r_1 <: r_2\end{array}}{[H]\,a \Rightarrow^{r_2}_S [H'\,;\,\mathbf{u}'\,;\,\Gamma]\,a'}$$

The order relation on heaps, used in rule HEAP-SUBL, is similar to that on graded contexts: for heaps H_1 and H_2 where $\lfloor H_1 \rfloor = \lfloor H_2 \rfloor$, we say $H_2 <: H_1$ if and only if $\overline{H_2} <: \overline{H_1}$ (pointwise order). The subusage rules above are meant to allow for wastage of resources, as and when permitted by the parametrizing preordered semiring. Rule HEAP-SUBL allows wastage by potentially discarding some resources from the initial heap. Rule HEAP-SUBR allows wastage by potentially discarding some copies of the reduct.

Now that we have seen the step rules, let us look at the multi-step relation. The multi-step relation is the transitive closure of the (single) step relation. The following rules build this relation:

Multi-One

$$\frac{[H]\, a \Rightarrow^{r}_{S} [H' \,;\, \mathbf{u}' \,;\, \Gamma]\, b}{[H]\, a \Rrightarrow^{r}_{S} [H' \,;\, \mathbf{u}' \,;\, \Gamma]\, b}$$

Multi-Many

$$\frac{[H]\, a \Rrightarrow^{r}_{S} [H' \,;\, \mathbf{u}' \,;\, \Gamma']\, b_1 \qquad [H']\, b_1 \Rrightarrow^{r}_{S} [H'' \,;\, \mathbf{u}'' \,;\, \Gamma'']\, b}{[H]\, a \Rrightarrow^{r}_{S} [H'' \,;\, (\mathbf{u}' \diamond \mathbf{0}) + \mathbf{u}'' \,;\, \Gamma', \Gamma'']\, b}$$

Rule Multi-One is straightforward. In rule Multi-Many, there are three points to note. First, we use the same grade r in all the three judgments: r copies of a reduce to r copies of b_1, which then reduce to r copies of b. Second, total resources consumed is the sum of the resources consumed during the two constituent multi-steps. Third, the set of new names added to the heap is the union of the sets of new names added during the two constituent multi-steps.

4.3.3 Heap Semantics vs. Standard Operational Semantics

Now that we have two semantics for the same language, let us find out how they compare with one another. The reader may have already noticed the differences between the two. Below, we shall make precise the differences along with the similarities:

1. The primary difference between the two semantics is that while heap semantics is resource-conscious, standard operational semantics is resource-agnostic. This distinction is significant and is the motivating factor behind designing heap semantics. We shall see the implications of this distinction throughout this chapter, especially in Section 4.3.4.

2. Another difference between the two semantics is that heap semantics, unlike standard operational semantics, is nondeterministic, as we saw in Section 4.3.1. However, the nondeterminism in heap semantics is limited to resource usage. Put in other words, if a term steps in two different ways, when provided with a heap, then the resulting terms are the same, even though the resulting heaps may have different allowed usage vectors. In this regard, note that if the concerned step rule adds a new assignment to the heap, like rules Heap-AppBeta and Heap-LetBoxBeta, then the resulting terms may have different sets of free variables. However, viewed as closures, the terms are still α-equivalent. Given heap term pairs (H_1, a_1) and (H_2, a_2), we say that they are α-equivalent, written $(H_1, a_1) \sim_\alpha (H_2, a_2)$, if the closures $(\llbracket H_1 \rrbracket, a_1)$ and $(\llbracket H_2 \rrbracket, a_2)$ are identical upto systematic renaming of variables. So like standard operational semantics, in heap semantics too, a term does not step to multiple reducts. Formally:

   Lemma 4.5 (Determinism) If $[H_1]\, a_1 \Rightarrow^{r_1}_{S_1} [H'_1 \,;\, \mathbf{u}'_1 \,;\, \Gamma'_1]\, a'_1$ and $[H_2]\, a_2 \Rightarrow^{r_2}_{S_2} [H'_2 \,;\, \mathbf{u}'_2 \,;\, \Gamma'_2]\, a'_2$ such that $(H_1, a_1) \sim_\alpha (H_2, a_2)$, then $(H'_1, a'_1) \sim_\alpha (H'_2, a'_2)$.

3. Yet another important difference between the two semantics is that heap semantics, unlike standard operational semantics, has a variable look-up rule, rule Heap-Var. This rule lies at the essence of heap semantics and is responsible for ensuring fairness of usage via serialization of substitutions. To elaborate, the β-rules of standard operational semantics substitute all occurrences of the bound

variable in one go whereas the β-rules of heap semantics load the potential substitutions into the heap and thereafter carry them out one by one via the variable look-up rule (till resources last). As such, reductions using the two semantics may proceed differently from each other. However, in spite of this difference, we can show that the two semantics are similar, as stated in the lemma below.

Lemma 4.6 (Similarity) Let $[H]\, a \Rightarrow^{r}_{S} [H' ; \mathbf{u}' ; \Gamma']\, a'$. If $|H| = |H'|$, then $a\{H\} = a'\{H'\}$, otherwise $\vdash a\{H\} \rightsquigarrow a'\{H'\}$.

Here $a\{H\}$ denotes the term obtained by substituting in a, in reverse order, the definitions in H. The lemma above shows that heap-based reduction follows standard substitution-based reduction, possibly at a slower pace because of serialization of substitution: the steps that just look up variables in heap-based reduction have no corresponding steps in standard substitution-based reduction.

Now that we see how the two semantics compare with one another, let us step back and consider the reason behind designing an alternative semantics. We designed a heap semantics for the language so that we can keep an account of resource usage during reduction. But how do we find out whether this heap semantics accounts usage correctly? A basic correctness principle for any accounting is the principle of conservation: that whatever we are accounting for is conserved; that it does not come out of or vanish into thin air; that we can trace its progression. So, we define correct usage as usage that honors the principle of conservation. Next, we show that our heap semantics honors this principle.

4.3.4 Accounting of Resources

The step relation enforces correct usage of resources:

Lemma 4.7 (Conservation) If $[H]\, a \Rightarrow\!\!\!\!\Rightarrow^{r}_{S} [H' ; \mathbf{u}' ; \Gamma']\, a'$, then $\overline{H} \diamond \overline{\Gamma'} <: \overline{H'} + \mathbf{u}'$.

Here, $\overline{H}$ represents the initial resources and $\overline{\Gamma'}$ the newly added resources whereas $\overline{H'}$ represents the resources left and $\mathbf{u}'$ the resources that were consumed. The above lemma, then, says that the initial resources concatenated with those that are added during reduction, are equal to or more than the sum of resources that remain and resources that are used up. Note that if we didn't allow sub-usaging or if the preorder is discrete, the above lemma guarantees an exact equality. In that case, the reduction relation enforces strict conservation of resources. In general, the conservation lemma ensures that reductions don't consume more resources than what they are entitled to.

From an accounting perspective, the left-hand side of the above inequality, $\overline{H} \diamond \overline{\Gamma'}$, denotes the 'credit' (the resources a reduction is entitled to), $\mathbf{u}'$ denotes the 'debit' (the resources consumed by the reduction) and $\overline{H'}$ denotes the 'available balance' (the resources that may be used by future reductions). Then, the above lemma may be read as: 'available balance of no resource should drop below zero'. Now, if the available balance of any resource touches 0, then that resource can no longer be used in reduction. So in heap-based semantics, unlike standard substitution-based semantics, terms can 'get stuck' due to unavailability of resources. The

following example illustrates this point:

$$
\begin{aligned}
& [x \overset{2}{\mapsto} 20, y \overset{1}{\mapsto} x + x](x + y) && \textit{look up value of } x\textit{, decrement its usage} \\
\Rightarrow\, & [x \overset{1}{\mapsto} 20, y \overset{1}{\mapsto} x + x]20 + y && \textit{look up value of } y\textit{, decrement its usage} \\
\Rightarrow\, & [x \overset{1}{\mapsto} 20, y \overset{0}{\mapsto} x + x]20 + (x + x) && \textit{look up value of } x\textit{, decrement its usage} \\
\Rightarrow\, & [x \overset{0}{\mapsto} 20, y \overset{0}{\mapsto} x + x]20 + (20 + x) && \textit{look up value of } x\textit{, stuck!}
\end{aligned}
$$

On a closer look, we find that the reduction above gets stuck because the starting heap does not contain enough resources for the evaluation of the term: the term needs to use x thrice but the initial heap contains only two copies of x.

Owing to the conservation lemma, heap-based reductions can get stuck if the starting heap does not contain enough resources. But even if the starting heap contains enough resources, reductions may still get stuck because of 'unwise usage'. For example, over the linearity semiring $\mathcal{Q}_{\text{Lin}}$, the reduction: $[x \overset{\omega}{\mapsto} 5]x + (x + x) \Rightarrow [x \overset{1}{\mapsto} 5]5 + (x + x) \Rightarrow [x \overset{0}{\mapsto} 5]5 + (5 + x)$ gets stuck because in the first step, ω is 'unwisely' split as $1 + 1$ in lieu of $\omega + 1$.

In sum, heap-based reductions can get stuck either due to insufficient resources in the starting heap or due to unwise usage of resources during the course of reduction. So then, if the starting heap contains enough resources and unwise usage is avoided, reductions should not get stuck. This idea forms the basis of the soundness theorem. The soundness theorem shows, given a heap that contains enough resources, a well-typed term that is not a value, can always take a step such that the resulting heap contains enough resources for the evaluation of the resulting term. To state this theorem, however, we first need to formalize what it means for a heap to contain enough resources to evaluate a term. We do this next.

4.3.5 Heap Compatibility

The key idea behind the languages we design in this chapter is that, if the resources contained in a heap are judged to be 'right' for a term by the type system, then the evaluation of that term in that heap does not get stuck. With the heap-based reduction rules enforcing correctness of usage, this property would imply that the type system accounts usage correctly.

The compatibility relation, $H \Vdash \Gamma$, presented below, formalizes the judgment that the heap H contains enough resources to evaluate any term that type-checks in the context Γ. The relation $H \Vdash \Gamma$ is read as: heap H is compatible with context Γ. Heaps that are compatible with some context are referred to as *well-formed* heaps.

$\boxed{H \Vdash \Gamma}$ *(Heap compatibility)*

$$\dfrac{}{\varnothing \Vdash \varnothing}\ \textsc{Compat-Empty} \qquad \dfrac{H \Vdash \Gamma_1 + (q \cdot \Gamma_2) \qquad \Gamma_2 \vdash a : A}{H, x \overset{q}{\mapsto} (\Gamma_2 | a | A) \Vdash \Gamma_1, x :^q A}\ \textsc{Compat-Cons}$$

Now, consider the rule Compat-Cons for well-formed heaps. This rule is like a converse of the substitution lemma. It loads q potential substitutions into the heap and lets the context use the variable q times. The following lemma makes precise the relationship between compatibility and substitution:

Lemma 4.8 (Multi-substitution) If $H \Vdash \Gamma$ and $\Gamma \vdash a : A$, then $\varnothing \vdash a\{H\} : A$.

Next, observe how rule Compat-Cons records the context and type of a as embedded context and embedded type in the assignment, $x \overset{q}{\mapsto} (\Gamma_2|a|A)$. This extra information recorded through embedded context and embedded type will be crucial to proving several useful properties in Sections 4.3.6 and 4.4.2.

Now, we look at an example derivation of a compatible heap context pair. Here, we simplify the judgments by omitting the embedded contexts and types.

Example 4.1.

$$\dfrac{\dfrac{\dfrac{\varnothing \Vdash \varnothing \qquad \varnothing \vdash 20 : \mathbf{Int}}{x_1 \overset{7}{\mapsto} 20 \Vdash x_1 :^7 \mathbf{Int}} \qquad x_1 :^2 \mathbf{Int} \vdash x_1 + x_1 : \mathbf{Int}}{x_1 \overset{7}{\mapsto} 20, x_2 \overset{3}{\mapsto} x_1 + x_1 \Vdash x_1 :^1 \mathbf{Int}, x_2 :^3 \mathbf{Int}} \qquad x_1 :^1 \mathbf{Int}, x_2 :^2 \mathbf{Int} \vdash x_1 + (x_2 + x_2) : \mathbf{Int}}{x_1 \overset{7}{\mapsto} 20, x_2 \overset{3}{\mapsto} x_1 + x_1, x_3 \overset{1}{\mapsto} x_1 + (x_2 + x_2) \Vdash x_1 :^0 \mathbf{Int}, x_2 :^1 \mathbf{Int}, x_3 :^1 \mathbf{Int}}$$

In the above derivation, the context $x_1 :^7 \mathbf{Int}$ gets split as $x_1 :^1 \mathbf{Int} + 3 \cdot x_1 :^2 \mathbf{Int}$ and the context $x_1 :^1 \mathbf{Int}, x_2 :^3 \mathbf{Int}$ gets split as $(x_1 :^0 \mathbf{Int}, x_2 :^1 \mathbf{Int}) + 1 \cdot (x_1 :^1 \mathbf{Int}, x_2 :^2 \mathbf{Int})$. So, effectively the resources in the definition, $x_1 \overset{7}{\mapsto} 20$, are divided between three copies of $x_2 \mapsto x_1 + x_1$, and one copy of $x_3 \mapsto x_1 + (x_2 + x_2)$. Observe that the heap keeps a record, in the form of allowed usages, of how the contexts get split.

The compatibility relation is crucial to our soundness theorem. So, we shall explore it in more detail below.

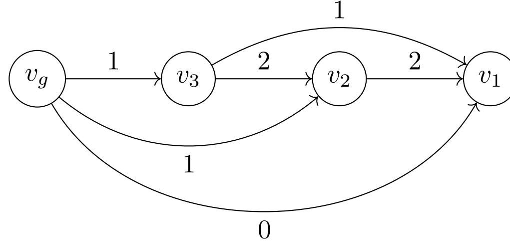
4.3.6 Graphical and Algebraic Views of Well-Formed Heaps

A heap can be viewed as a memory graph, where an assignee variable corresponds to a memory location and the assigned term to data stored in that location. The allowed usage then, is the number of references to the location. Such a graphical view of a heap is not interesting in general; however, if the heap is well-formed, its graphical view gets quite interesting and informative.

A well-formed heap H, where $H \Vdash \Gamma$, can be viewed as a weighted directed acyclic graph $G_{H,\Gamma}$, described as follows. Let H contain n assignments with the jth one being $x_j \overset{q_j}{\mapsto} (\Gamma_j|a_j|A_j)$. Then, $G_{H,\Gamma}$ is a directed

acyclic graph with $(n+1)$ nodes, where n nodes correspond to the n variables defined in H and one extra node, referred to as the *ground node*, corresponds to Γ. Let v_j be the node corresponding to the variable x_j and v_g be the ground node. Now, for $x_i :^{q_{ji}} A_i \in \Gamma_j$, we add an edge from v_j to v_i with weight $w(v_j, v_i) := q_{ji}$. We do this for all nodes, including v_g. This gives us a directed acyclic graph (the graph is acyclic because the heap does not contain cyclical references). This directed acyclic graph has a unique topological ordering: $v_g, v_n, v_{n-1}, \ldots, v_2, v_1$.

For example 4.1, we have the following directed acyclic graph. Note the topological ordering of the graph: v_g, v_3, v_2, v_1.



Now given a well-formed heap, we can express the allowed usages of the assignee variables in terms of the edge weights of the memory graph: Let us define the length of a path in a memory graph to be the product of the weights along the path. Then, the allowed usage of a variable is the sum of the lengths of all paths from the source node to the node corresponding to that variable. Observe that this is so for the example graph.

A path p from v_g to v_j represents a chain of references, with the last one pointed at v_j. The length of p shows how many times this path is used to reference v_j. The sum of the lengths of all the paths from v_g to v_j, then, gives a (static) count of the total number of times location v_j is referenced. And this sum is equal to q_j, the allowed usage of the assignment for x_j in the heap. Put succinctly, the allowed usage of an assignment for a variable is equal to the (static) count of the number of times the location corresponding to that variable is referenced. We call this property *count balance*. This is an important and interesting property. So next, we present an algebraic formalization of this property.

For a well-formed heap, H, containing n assignments of the form $x_j \overset{q_j}{\mapsto} (\Gamma_j | a_j | A_j)$, we write $\langle H \rangle$ to denote the $n \times n$ matrix whose j^{th} row is $\overline{\Gamma_j} \diamond \mathbf{0}$. We call $\langle H \rangle$ the *transformation matrix* corresponding to H. The transformation matrix for example 4.1 is:

$$\begin{pmatrix} 0 & 0 & 0 \\ 2 & 0 & 0 \\ 1 & 2 & 0 \end{pmatrix}$$

Note that the transformation matrix of any well-formed heap is strictly lower triangular. The strict lower triangular property of the matrix corresponds to the acyclicity of the graph. Further, note that the transformation matrix of a well-formed heap is also the adjacency matrix of the corresponding memory graph,

excluding the ground node. Now, with the matrix operations over a semiring defined in the usual way, we can state the graphical count balance property as follows.

Lemma 4.9 (Count Balance) If $H \Vdash \Gamma$, then $\overline{H} = \overline{H} \times \langle H \rangle + \overline{\Gamma}$.

For example 4.1, check that $\overline{H} = \begin{pmatrix} 7 & 3 & 1 \end{pmatrix}$ satisfies the above equation. Below, we repeat the graphical intuition behind this lemma. By this lemma, for any node v_j in $G_{H,\Gamma}$, we have, $q_j = \Sigma_k\, q_k w(v_k, v_j) + w(v_g, v_j)$. The right-hand side of this equation gives a static estimate of demand, the amount of resources we shall need, whereas the left-hand side gives a static estimate of supply, the amount of resources we shall have. So, $H \Vdash \Gamma$ is a static guarantee that the heap H shall supply the resource demands of the context Γ.

Therefore, if $H \Vdash \Gamma$ and $\Gamma \vdash a : A$, we should be able to evaluate a in heap H without running out of resources. This is the gist of the soundness theorem.

4.3.7 Soundness Theorem

Theorem 4.10 (Soundness) If $H \Vdash \Gamma$ and $\Gamma \vdash a : A$, then either a is a value or there exists H', $\mathbf{u'}$, Γ'_0 and Γ' such that:

- $[H]\, a \Rightarrow^1_S [H' \,;\, \mathbf{u'} \,;\, \Gamma'_0]\, a'$
- $\Gamma' \vdash a' : A$
- $H' \Vdash \Gamma'$
- $\overline{\Gamma} \diamond \mathbf{0} \;+\; \mathbf{u'} \times \langle H' \rangle \;+\; \mathbf{0} \diamond \overline{\Gamma'_0} \;\;<:\;\; \overline{\Gamma'} \;+\; \mathbf{u'} \;+\; (\mathbf{0} \diamond \overline{\Gamma'_0}) \times \langle H' \rangle$

The soundness theorem states that computations can proceed without ever getting stuck, provided the starting heap contains enough resources. Note that in the above theorem, as the term a steps to a', the typing context changes from Γ to Γ'. This change is to be expected because during the step, resources from the heap may have been consumed or new resources may have been added. For example, the step $[x \overset{1}{\mapsto} \mathbf{unit}]x \Rightarrow^1_S [x \overset{0}{\mapsto} \mathbf{unit}]\mathbf{unit}$ consumes resource, and as such, the redex and reduct are typed as $x :^1 \mathbf{Unit} \vdash x : \mathbf{Unit}$ and $x :^0 \mathbf{Unit} \vdash \mathbf{unit} : \mathbf{Unit}$ respectively. On the other hand, the step $[\varnothing](\lambda^1 x.x)\;\mathbf{unit}^1 \Rightarrow^1_S [y \overset{1}{\mapsto} \mathbf{unit}]\,y$ adds resource, and as such, the redex and reduct are typed as $\varnothing \vdash (\lambda^1 x.x)\;\mathbf{unit}^1 : \mathbf{Unit}$ and $y :^1 \mathbf{Unit} \vdash y : \mathbf{Unit}$ respectively. Though the typing context may change after reduction, the new context, which type-checks the reduct, is compatible with the new heap. As such, we can apply the soundness theorem again and again until we reach a value. Graphically, as evaluation progresses, some of the weights in the memory graph may change or new nodes may be added to the graph but the count balance property is always maintained. So, we see that the dynamics of the language aligns perfectly with its statics.

Further, observe that the contexts type-checking the redex and the reduct are related by the fourth clause of the soundness theorem. We explain the intuitive meaning of this clause below. For the moment being,

assume that the preorder (of the parametrizing preordered semiring) is discrete. Then, the clause simplifies to:

$$\begin{aligned} & \overline{\Gamma} \diamond \mathbf{0} \quad + \quad \mathbf{u'} \times \langle H' \rangle \quad + \quad \mathbf{0} \diamond \overline{\Gamma'_0} \\ = \; & \overline{\Gamma'} \quad + \quad \mathbf{u'} \quad + \quad (\mathbf{0} \diamond \overline{\Gamma'_0}) \times \langle H' \rangle \end{aligned}$$

This equation can be understood as a transaction between contexts and heaps. The context pays the heap $\mathbf{0} \diamond \overline{\Gamma'_0}$ resources and gets $(\mathbf{0} \diamond \overline{\Gamma'_0}) \times \langle H' \rangle$ resources in return. The heap pays the context $\mathbf{u'}$ and gets $\mathbf{u'} \times \langle H' \rangle$ resources in return. The equation, then, is the 'balance sheet' of this transaction. For an arbitrary preorder, the transaction gets skewed in favor of the heap, meaning, the context gets less from the heap for what it pays. This is so because the heap contains more resources than is necessary for reduction; so it may discard extra resources.

The soundness theorem above shows that GLC accounts resource usage correctly. This theorem subsumes and is more informative than the standard type soundness theorem, presented in Section 4.2.3. Next, we employ this theorem to reason about resource usage in GLC.

4.4 Applications

4.4.1 No Usage

Till now, we have developed the theory of GLC over an arbitrary preordered semiring. But an arbitrary semiring is too general a structure to reason about usage. For instance, the set $\{0, 1\}$ with $1 + 1 = 0$ and all other operations defined in the usual way is also a semiring. But such a semiring does not capture our notion of usage because grades 0 and 1 are supposed to represent no usage and some usage respectively.

For grade 0 to represent no usage in a preordered semiring $\mathcal{Q}$, the inequation $0 <: q + 1$ must have no solution in $\mathcal{Q}$. Now, in some preordered semirings, there may be grades q_0, other than 0, for which the inequation $q_0 <: q + 1$ has no solution; in such cases, those grades would also represent no usage. To simplify matters, we shall restrict ourselves to preordered semirings where 0 is maximal among all grades q_0 for which $q_0 <: q + 1$ has no solution. Such preordered semirings are referred to as zero-unusable. An alternative definition of a zero-unusable semiring is a preordered semiring in which 0 is a maximal element and the inequation $0 <: q + 1$ has no solution. Note that all the preordered semirings introduced in Section 4.1 satisfy this condition and as such, are zero-unusable.

Next, we relate grade 0 with no usage in GLC parametrized over zero-unusable semirings.

Lemma 4.11 Let $\mathcal{Q}$ be a zero-unusable semiring. In GLC($\mathcal{Q}$), if $[H]\, a \Rightarrow^1_S [H' ; \mathbf{u'} ; \Gamma'_0]\, a'$ and $x_i \overset{0}{\mapsto} (\Gamma_i | a_i | A_i) \in H$, then $\mathbf{u'}(x_i) = 0$ and $x_i \overset{0}{\mapsto} (\Gamma_i | a_i | A_i) \in H'$.

Here, $\mathbf{u'}(x_i)$ denotes the component of $\mathbf{u'}$ corresponding to x_i. The above lemma shows that variables with allowed usage 0 cannot be referenced during computation. Further, the allowed usages of such variables

always remain at grade 0. Now, since such variables cannot be referenced, their definitions should not affect results of computations. In other words, when a term is reduced in two heap configurations that differ only in the assignments of 0-graded variables, the resulting reducts should be identical. The following lemma shows that this is indeed the case.

Lemma 4.12 Let $\mathcal{Q}$ be a zero-unusable semiring. Next, let $H_i = x \overset{0}{\mapsto} (\Gamma_i | a_i | A_i)$ and $H_j = x \overset{0}{\mapsto} (\Gamma_j | a_j | A_j)$. Then, in GLC($\mathcal{Q}$), if $[H_1, H_i, H_2]\, b \Rightarrow^1_{S \,\cup\, \mathrm{fv}\, a_j} [H_1', H_i, H_2' \,;\, \mathbf{u'} \,;\, \Gamma_0']\, b'$,
then $[H_1, H_j, H_2]\, b \Rightarrow^1_{S \,\cup\, \mathrm{fv}\, a_i} [H_1', H_j, H_2' \,;\, \mathbf{u'} \,;\, \Gamma_0']\, b'$.

In conjunction with the soundness theorem, the above lemmas show that in GLC parametrized over zero-unusable semirings, the grade 0 indeed represents no usage. An interesting corollary of this result is that functions in GLC (thus parametrized) cannot make use of 0-marked arguments. More precisely,

Lemma 4.13 Let $\mathcal{Q}$ be a zero-unusable semiring. Let $\varnothing \vdash f : {}^0A \to B$ and $\varnothing \vdash a_1 : A$ and $\varnothing \vdash a_2 : A$ in GLC($\mathcal{Q}$). Then, $f\ {a_1}^0$ and $f\ {a_2}^0$ have the same operational behavior.

With regard to GLC (which is strongly normalizing), having the same operational behavior implies reduction to the same value. From the above lemma, we see that any function of type ${}^0A \to \mathbf{Bool}$ is a constant function. This property is uncannily similar to a noninterference theorem, with 0 interpreted as a high-security label. In Section 4.5, we shall find that this similarity is not a mere coincidence but an outcome of the close connection between usage and dependency analyses.

4.4.2 Linear Usage

Next, we relate grade 1 and linear usage. Just as grade 0 does not represent no usage for an arbitrary preordered semiring, grade 1 too does not represent linear usage for an arbitrary preordered semiring. To give an example, in the boolean semiring $\mathbb{B}_{\geq}$, grade 1 represents unrestricted usage because $1 + 1 = 1$. To ensure that 1 represents linear usage, we need to restrict our class of preordered semirings. We do so by imposing the following constraints on preordered semirings.

- Zerosumfree [Golan, 1999]: For $q_1, q_2 \in Q$, the equation $q_1 + q_2 = 0$ implies $q_1 = q_2 = 0$.
- Entire [Golan, 1999]: For $q_1, q_2 \in Q$, the equation $q_1 \cdot q_2 = 0$ implies $q_1 = 0$ or $q_2 = 0$.
- For $q_1, q_2 \in Q$, the equation $q_1 + q_2 = 1$ implies $q_1 = 1$ and $q_2 = 0$ or vice-versa.
- For $q_1, q_2 \in Q$, the equation $q_1 \cdot q_2 = 1$ implies $q_1 = q_2 = 1$.
- 0 and 1, where $0 \neq 1$, are maximal elements with respect to <: relation.

We shall refer to preordered semirings that satisfy the above constraints as one-linear. The linearity semiring, $\mathcal{Q}_{\mathrm{Lin}}$, is a one-linear semiring but the affinity semiring, $\mathcal{Q}_{\mathrm{Aff}}$, is not (because in $\mathcal{Q}_{\mathrm{Aff}}$, 1 is not a maximal element). Observe here that one-linear semirings are also zero-unusable.

Now, when GLC is parametrized over one-linear semirings, the soundness theorem guarantees that 1-marked arguments are used exactly once. As a matter of fact, there is exactly one way to reference a 1-marked argument at run time. Formally, we can show:

Lemma 4.14 Let $\mathcal{Q}$ be a one-linear semiring. In GLC($\mathcal{Q}$), if $H \Vdash \Gamma$ and $x_i \overset{1}{\mapsto} (\Gamma_i|a_i|A_i) \in H$, then in the graph $G_{H,\Gamma}$, there is exactly one path from the ground node to v_i and further, all the weights on that path are 1.

The above lemma (in conjunction with the soundness theorem) may be seen as a generalization of the single pointer property [Turner and Wadler, 1999], which guarantees that linear resources have exactly one pointer to them. The single pointer property is interesting because it ensures safety of in-place update of linear resources: if there is only one pointer to a memory location, there is no need to copy the content of that location in order to update it, it can be updated in-place.

This completes our discussion on the syntax and semantics of GLC. Now, GLC is a simply-typed calculus for linearity analysis, or more generally, usage analysis. In Chapter 3, we designed SDC, another simply-typed calculus but one that is meant for dependency analysis. In Section 1.3, we abstractly described how linearity and dependency analyses compare against one another. Now that we have concrete calculi for both the analyses, we can present a more detailed and nuanced comparison. So in the next section, we compare usage and dependency analyses by comparing our simply-typed calculi for the two analyses, GLC and SDC.

4.5 GLC and SDC: A Comparison

GLC is a general calculus for usage analysis whereas SDC is a general calculus for dependency analysis. Recall the key difference between usage analysis and dependency analysis pointed out in Section 3.5.2: usage analysis requires counting whereas dependency analysis requires comparison. Since counting does not need a variable reference, calculi for usage analysis, including GLC, employ a fixed reference in their type systems. On the other hand, calculi for dependency analysis, including SDC, allow their type systems to vary the reference level. More concretely, typing judgments have a fixed grade, i.e., 1, to the right of the turnstile in a calculus for usage analysis but a variable grade in a calculus for dependency analysis. So, we may say that calculi for usage analysis formalize the perspective of an observer from '1-world' whereas calculi for dependency analysis formalize the perspective of observers from all worlds. Though this is a point of discord between the two kinds of calculi, nevertheless, it places them on different ends of the same platform. Being on the same platform makes them similar in certain ways, which we sketch out below.

We know that the calculi for usage analysis are parametrized by preordered semirings. Now, given a preordered semiring $\mathcal{Q}$, data from a world $q \in \mathcal{Q}$, where $\neg(\exists q_0, q <: q_0 + 1)$, should not be usable in 1-world. For example, in $\mathbb{B}_{\geq}$ and $\mathcal{Q}_{\text{Lin}}$, data from world 0 should not be usable in 1-world because $\neg(\exists q_0, 0 <: q_0 + 1)$. Such constraints are similar to dependency constraints: data from a world ℓ_1 should not be observable in world ℓ_2 if $\neg(\ell_1 \sqsubseteq \ell_2)$. This similarity suggests that calculi for usage analysis and calculi for dependency analysis might, as well, fill in each other's roles. While not entirely true, this observation indeed has some

$$\overline{\mathbf{Unit}} = \mathbf{Unit} \qquad \overline{{}^{q}A \to B} = S_q\,\overline{A} \to \overline{B} \qquad \overline{\Box^{q} A} = S_q\,\overline{A} \qquad \overline{{}^{q}A \times B} = S_q\,\overline{A} \times \overline{B} \qquad \overline{A + B} = \overline{A} + \overline{B}$$

$$\begin{aligned}
\overline{x} &= x \\
\overline{\mathbf{unit}} &= \mathbf{unit} \\
\overline{\mathbf{let\,unit} = a\ \mathbf{in}\ b} &= \mathbf{let\,unit} = \overline{a}\ \mathbf{in}\ \overline{b} \\
\overline{\lambda^{q} x{:}A.b} &= \lambda y{:}S_q\,\overline{A}.\mathbf{unlock}^{q}\ x = y\,\mathbf{in}\,\overline{b} \quad (y \text{ fresh}) \\
\overline{b\ a^{q}} &= \overline{b}\,(\mathbf{lock}^{q}\ \overline{a}) \\
\overline{\mathbf{box}_q\ a} &= \mathbf{lock}^{q}\ \overline{a} \\
\overline{\mathbf{let\,box}_q\ x = a\ \mathbf{in}\ b} &= \mathbf{unlock}^{q}\ x = \overline{a}\,\mathbf{in}\,\overline{b} \\
\overline{(a_1{}^{q}, a_2)} &= (\mathbf{lock}^{q}\ \overline{a_1}, \overline{a_2}) \\
\overline{\mathbf{let}\,(x_1{}^{q}, x_2) = a\ \mathbf{in}\ b} &= \mathbf{let}\,(x_1', x_2) = \overline{a}\ \mathbf{in}\ \mathbf{unlock}^{q}\ x_1 = x_1'\,\mathbf{in}\,\overline{b} \quad (x_1' \text{ fresh}) \\
\overline{\mathbf{inj}_1\ a_1} &= \mathbf{inj}_1\ \overline{a_1} \\
\overline{\mathbf{case}_q\ a\,\mathbf{of}\ x_1.b_1; x_2.b_2} &= \mathbf{case}\,\overline{a}\,\mathbf{of}\ x_1.\overline{b_1}; x_2.\overline{b_2}
\end{aligned}$$

FIGURE 4.4: Type and term translation from GLC($\mathbb{B}_{\geq}$) to SDC($\mathbb{B}_{\geq}$)

value: in certain cases, a calculus designed for dependency analysis may be employed for usage analysis and vice-versa. To give an example, our calculus for dependency analysis, SDC, can analyze some usages and our calculus for usage analysis, GLC, can analyze some dependencies. Next, we show how.

4.5.1 Usage Analysis in SDC

When parametrized over the two-element lattice, $\mathcal{L}_2 = \mathbf{L} \sqsubseteq \mathbf{H}$, SDC ensures that data from world $\mathbf{H}$ is not observable in world $\mathbf{L}$. On the other hand, GLC, when parametrized over the boolean semiring, $\mathbb{B}_{\geq} = 1 <: 0$, ensures that data from world 0 is unusable in world 1. Now, $\mathbb{B}_{\geq}$ is also a lattice with join and meet defined as multiplication and addition respectively, making it isomorphic to $\mathcal{L}_2$. So, we may parametrize SDC over $\mathbb{B}_{\geq}$. Now, SDC, thus parametrized, can in fact analyze no usage by treating 0 and 1 as high-security and low-security labels respectively. Moreover, SDC, thus parametrized, subsumes GLC($\mathbb{B}_{\geq}$).

We present a meaning-preserving translation from GLC($\mathbb{B}_{\geq}$) to SDC($\mathbb{B}_{\geq}$). The types and terms of GLC($\mathbb{B}_{\geq}$) are translated as shown in Figure 4.4. The translation function, $\overline{\ }$, is defined by a straightforward recursion. However, note that since function and product types are graded in GLC but ungraded in SDC, we need to use the graded modality to translate them. This, in turn, requires that we appropriately lock and unlock while translating the introduction and elimination forms for these types. Now coming to typing judgments, since both the calculi use graded contexts, we can leave the contexts as such, after translating the types; however, we need to set the grade on the translated typing judgment to 1. This translation, then, preserves typing and meaning, as shown by the lemma below.

Lemma 4.15 If $\Gamma \vdash a : A$ in GLC($\mathbb{B}_{\geq}$), then $\vec{\Gamma} \vdash \overline{a} :^{1} \overline{A}$ in SDC($\mathbb{B}_{\geq}$). Further, if $\vdash a \rightsquigarrow a'$ in GLC($\mathbb{B}_{\geq}$), then $\vdash \overline{a} \rightsquigarrow^{*} \overline{a'}$ in SDC($\mathbb{B}_{\geq}$).

Here, $\vec{\Gamma}$ denotes the context Γ, but with the types translated. (We use the symbol $\vec{\Gamma}$ in lieu of $\overline{\Gamma}$ because the latter already has a meaning attached to it.) Note the use of the multi-step reduction relation in the above lemma. The multi-step relation is necessary here because if a GLC-term reduces via the β-rule for functions (or pairs), the translated SDC-term would need to use the β-rules for both functions (or pairs) and modal terms to step to the translated reduct.

The above lemma helps us understand no usage in GLC($\mathbb{B}_{\geq}$) through the lens of noninterference in SDC($\mathbb{B}_{\geq}$): unused arguments to computations are essentially high-security inputs, if the output is considered to be of low-security. As such, with regard to GLC($\mathbb{B}_{\geq}$), we may interpret grades marking no usage and potential usage, i.e., 0 and 1, as high-security and low-security labels respectively. When we read the typing rules of GLC with this interpretation in mind, we understand them in a new light. Rule GLC-Case is a prime example of one such typing rule.

GLC-Case

$$\frac{q <: 1 \qquad \Gamma_1 \vdash a : A_1 + A_2 \qquad \Gamma_2, x_1 :^{q} A_1 \vdash b_1 : B \qquad \Gamma_2, x_2 :^{q} A_2 \vdash b_2 : B \qquad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor}{q \cdot \Gamma_1 + \Gamma_2 \vdash \mathbf{case}_q\ a\ \mathbf{of}\ x_1.b_1; x_2.b_2 : B}$$

GLC-CaseUnsound

$$\frac{\Gamma_1 \vdash a : A_1 + A_2 \qquad \Gamma_2, x_1 :^{q} A_1 \vdash b_1 : B \qquad \Gamma_2, x_2 :^{q} A_2 \vdash b_2 : B \qquad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor}{q \cdot \Gamma_1 + \Gamma_2 \vdash \mathbf{case}_q\ a\ \mathbf{of}\ x_1.b_1; x_2.b_2 : B}$$

Consider what happens if we remove the precondition $q <: 1$ from rule GLC-Case and use rule GLC-CaseUnsound instead. Rule GLC-CaseUnsound would make the type system unsound because it would allow high-security inputs to interfere in low-security outputs. For example, using this rule, one can derive: $x :^{0} \mathbf{Unit} + \mathbf{Unit} \vdash \mathbf{case}_0\ x\ \mathbf{of}\ y_1.2; y_2.5 : \mathbf{Int}$. Here, x is a high-security input whereas $\mathbf{case}_0\ x\ \mathbf{of}\ y_1.2; y_2.5$ is a low-security term. But this term outputs 2 when $x = \mathbf{inj}_1\ \mathbf{unit}$ and 5 when $x = \mathbf{inj}_2\ \mathbf{unit}$, thereby breaching security. Note that one cannot derive this term using rule GLC-Case because $\neg(0 <: 1)$ in $\mathbb{B}_{\geq}$. This shows how the precondition $q <: 1$ in rule GLC-Case helps prevent information leaks.

The above discussion shows that no usage and potential usage can be understood in terms of high-security and low-security respectively. Further, these usages can be analyzed not only by GLC but also by SDC. However, when it comes to analyzing other usages, particularly linear usage, SDC is not helpful because SDC is parametrized by lattices, where both the binary operators are idempotent. But linear usage is not idempotent with respect to addition, meaning, linear usage + linear usage ≠ linear usage. Therefore, we may say that though SDC can analyze some usages, it is not as general as GLC in this regard.

4.5.2 Dependency Analysis in GLC

We saw that $\text{GLC}(\mathbb{B}_{\geq})$ ensures inputs from world 0 do not interfere in outputs from world 1. So $\text{GLC}(\mathbb{B}_{\geq})$ can enforce dependency constraints. However, $\text{GLC}(\mathbb{B}_{\geq})$ is not as good as $\text{SDC}(\mathbb{B}_{\geq})$, when it comes to dependency analysis. This is so because $\text{GLC}(\mathbb{B}_{\geq})$ does not support certain key aspects of dependency analysis. For example, $\text{GLC}(\mathbb{B}_{\geq})$ cannot derive a monadic join operator of type ${}^1(\Box^0 \Box^0 A) \to \Box^0 A$. Further, $\text{GLC}(\mathbb{B}_{\geq})$ does not allow an arbitrary function to use 0-marked arguments, even when the function itself is boxed under a 0-modality. As such, the type $\Box^0({}^0\mathbf{Bool} \to \mathbf{Bool})$ has only two distinct elements in $\text{GLC}(\mathbb{B}_{\geq})$, as opposed to four distinct elements in $\text{SDC}(\mathbb{B}_{\geq})$, shown in Section 3.2.1. These shortcomings with regard to dependency analysis are not particular to GLC but are shared by all graded-context type systems designed for usage analysis. We shall discuss these shortcomings in more detail in Section 5.1.1, on our way to designing a unified type system for linearity and dependency analyses. But for now, we move on to designing GRAD, our graded-context dependent type system for linearity (and general usage) analysis.

4.6 Graded-Context Dependent Type System

4.6.1 The Basics

What distinguishes the dependently-typed system from its simply-typed counterpart is that the former allows term variables to appear in types. When variables can appear in both terms and types, counting their usage is not straightforward. Recall that in Sections 1.1.9, 1.1.10 and 1.1.11, we discussed three valid approaches to accounting usage in dependent type systems. The approach we follow in GRAD, discussed in Section 1.1.10, treats usage in types and terms uniformly but counts usage only for the expression that appears as a term in any given typing judgment. The variable rule of GRAD, rule GRAD-VAR, makes this point clear:

$$\textsc{GraD-Var}\quad \frac{\Gamma \vdash A : s}{0 \cdot \Gamma, x{:}^1 A \vdash x : A}$$

We account the resources used by A in Γ. But when A appears as the type of x, we zero out the resources used by A and take into account only the resource used by x, which is one copy of x. We employ this accounting principle in GRAD.

Now, GRAD is a generic Pure Type System (PTS) characterized by a triple, $(\mathcal{S}, \mathcal{A}, \mathcal{R})$, where $s \in \mathcal{S}$ are sorts, $\mathcal{A}$ is a set of axioms and $\mathcal{R}$ is a ternary relation on sorts. Like GLC, GRAD is parametrized by an arbitrary preordered semiring $\mathcal{Q}$ and has function types, product types, sum types and a Unit type. The types, however, are dependent, as expected. Note that we do not have an explicit modal type in GRAD; we can encode the modal type using Σ-type and **Unit** as follows: $\Box^q A \triangleq \Sigma x{:}^q A.\mathbf{Unit}$. Formally, the types and

terms of GRAD are as shown below:

$$\begin{array}{rcl} \textit{terms, types}\ a,b,c,A,B,C & ::= & s \mid x \mid \\ & & \textbf{Unit} \mid \textbf{unit} \mid \textbf{let unit} = a\ \textbf{in}\ b \mid \\ & & \Pi x :^{q} A.B \mid \lambda^{q} x : A.a \mid a\ b^{q} \mid \\ & & \Sigma x :^{q} A.B \mid (a^{q}, b) \mid \textbf{let}\ (x^{q}, y) = a\ \textbf{in}\ b \mid \\ & & A + B \mid \textbf{inj}_1\ a \mid \textbf{inj}_2\ a \mid \textbf{case}_q\ a\ \textbf{of}\ x_1.b_1; x_2.b_2 \end{array}$$

4.6.2 Type System

We now look at the type system of GRAD, starting with the rules for Π-types.

$$\textsc{GraD-Pi}\ \frac{\Gamma_1 \vdash A : s_1 \quad \Gamma_2, x :^{r} A \vdash B : s_2 \quad \mathcal{R}(s_1, s_2, s_3) \quad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor}{\Gamma_1 + \Gamma_2 \vdash \Pi x :^{q} A.B : s_3}$$

$$\textsc{GraD-Lam}\ \frac{\Gamma_1, x :^{q} A \vdash b : B \quad \Gamma \vdash \Pi x :^{q} A.B : s \quad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma \rfloor}{\Gamma_1 \vdash \lambda^{q} x : A.b : \Pi x :^{q} A.B}$$

$$\textsc{GraD-App}\ \frac{\Gamma_1 \vdash b : \Pi x :^{q} A.B \quad \Gamma_2 \vdash a : A \quad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor}{\Gamma_1 + q \cdot \Gamma_2 \vdash b\ a^{q} : B\{a/x\}}$$

A key point to note in rule GRAD-PI is that the grade annotation, q, on the bound variable x in the Π-type is not related to the usage, r, of that variable in the body, B, of the type. The annotation q is meant for recording the usage of the argument in the body of a function having this type, as we see in rule GRAD-LAM.

Allowing a bound variable to be used differently in the body of a type and in the body of a function having that type enables GRAD to represent parametric polymorphism. To illustrate, the System F type, $\forall \alpha.\alpha \to \alpha$, can be expressed in GRAD (parametrized over a zero-unusable semiring) as $\Pi x :^{0} \texttt{Type}.^{1} x \to x$. This type is well-formed because, even though the annotation on the variable x is 0, rule GRAD-PI allows x to be used any number of times in the body of the type. However, if we set $r := q$ in rule GRAD-PI, the type $\Pi x :^{0} \texttt{Type}.^{1} x \to x$ becomes ill-formed, thereby showing that parametric polymorphism cannot be expressed under this modification. This is one of the reasons why we chose not to relate q and r in rule GRAD-PI.

Some forms of irrelevant quantifiers in type theories constrain r to be equal to q [Abel and Scherer, 2012]. By coupling the usage of the bound variable in the body of the function with that in the body of its type, these systems rule out the representation of parametrically polymorphic types, such as the one shown above. In GRAD, we can model this restrictive form of irrelevant quantifier with the assistance of the box modality. If, instead of using the type $\Pi x :^{0} A.B$, we use the type $\Pi x :^{1} \square^{0} A.B$, the unboxed argument cannot appear in a relevant position within B because terms of type $\square^{0} A$, upon unboxing, must be used 0 times only. So then, the unboxed argument cannot be used relevantly either in the body of $\Pi x :^{1} \square^{0} A.B$ or in the body of any function having this type.

It is this distinction between the types $\Pi x :^{0} A.B$ and $\Pi x :^{1} (\square^{0} A).B$ (and a similar distinction between $\Sigma x :^{0} A.B$ and $\Sigma x :^{1} (\square^{0} A).B$) that motivates our inclusion of grade annotations on Π-types (and Σ-types) directly. In the simply-typed system, we can derive grade-annotated function types from linear function

types and the box modality as ${}^{q}A \to B \cong \Box^q A \multimap B$; so there is really no need for grade-annotated function types in the simple system. But here, in a dependently-typed setting, grade-annotated Π-types cannot be derived from linear Π-types and the box modality: the derivation does not carry over; as we just saw, $\Pi x{:}^q A.B \ncong (x : \Box^q A) \multimap B$, when $q = 0$. To reiterate, compared to $\Pi x{:}^q A.B$, the type $(x : \Box^q A) \multimap B$ is more restrictive and cannot capture parametric polymorphism when q equals 0. This is the main reason why we include a grade annotation on the Π-type (and similarly on the Σ-type). Note here that we also included grade annotations on the Π-types and Σ-types of DDC for very similar reasons, as pointed out in Section 3.4.2. In sum, explicit grade annotations on Π-types and Σ-types make our calculi more flexible, which is one of the benefits of graded types we alluded to in Section 1.1.7.

Now, we look at the rules for Σ-types.

$$\textsc{GraD-Sigma}\quad \frac{\Gamma_1 \vdash A : s_1 \qquad \Gamma_2, x{:}^r A \vdash B : s_2 \qquad \mathcal{R}(s_1, s_2, s_3) \qquad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor}{\Gamma_1 + \Gamma_2 \vdash \Sigma x{:}^q A.B : s_3}$$

$$\textsc{GraD-Pair}\quad \frac{\Gamma_1 \vdash a : A \qquad \Gamma_2 \vdash b : B\{a/x\} \qquad \Gamma \vdash \Sigma x{:}^q A.B : s \qquad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor = \lfloor \Gamma \rfloor}{q \cdot \Gamma_1 + \Gamma_2 \vdash (a^q, b) : \Sigma x{:}^q A.B}$$

$$\textsc{GraD-LetPair}\quad \frac{\Gamma_1 \vdash a : \Sigma x{:}^q A_1.A_2 \qquad \Gamma_2, x{:}^q A_1, y{:}^1 A_2 \vdash b : B\{(x^q, y)/z\} \qquad \Gamma, z{:}^r (\Sigma x{:}^q A_1.A_2) \vdash B : s \qquad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor = \lfloor \Gamma \rfloor}{\Gamma_1 + \Gamma_2 \vdash \mathbf{let}\,(x^q, y) = a\ \mathbf{in}\ b : B\{a/z\}}$$

Rule GraD-Sigma is similar to rule GraD-Pi: the grade q on the bound variable x is not related to how x is used in the body of the type. This grade q records the number of available copies of the first component of a pair having this type, as we see in rule GraD-Pair. The elimination rule GraD-LetPair is similar to its counterpart in GLC; however, it implements a dependent pattern-matching.

Recall that Σ-types that are eliminated via pattern-matching are called weak Σ-types as opposed to strong Σ-types, which are eliminated via projections. By this definition, Σ-types in GraD are weak Σ-types. There does not seem to be a straightforward way to include strong Σ-types in GraD. The reason behind this is that the pairing rule *adds up* the resources used by both of the components, making it difficult for any projection rule to figure out what part of that sum belongs to a single projection. We could alternatively use a pairing rule that does not add up the resources used by the individual components, but instead, lets both the components share the same set of resources. Pairs introduced by such a rule can indeed be eliminated by projections. But the downside of such pairs is that both the components *must use* the same set of resources. Another alternative could be to define projections for certain kinds of pairs only, for example, pairs of type $\Sigma x{:}^0 A.B$, whose first components do not use any resource. Now, given that the resources used by such pairs essentially belong to their second components, one can define their second projections, at least in a simply-typed setting, as shown by Moon et al. [2021]. However, problems arise in a dependently-typed

setting, where the first projection is necessary to type-check the second projection. Since the first component of such pairs cannot be used, one cannot define their first projections. But then, one would not be able to define their second projections either. So, in sum, none of the two alternatives mentioned above provide us a satisfactory way of including strong Σ-types in GRAD. Note here that both $\text{DDC}^{\top}$ and DDC can encode strong Σ-types, as shown in Sections 3.3.3 and 3.4.3. However, this encoding is made possible thanks to their graded typing judgments, whereby the judgments checking the first and second projections can have different grades associated with them. GRAD does not have this flexibility and as such, cannot follow $\text{DDC}^{\top}$ and DDC in encoding strong Σ-types.

Next, we look at the conversion rule. The conversion rule GRAD-CONV, shown below, is fairly standard. It uses the definitional equality relation. Definitional equality for GRAD is essentially β-equivalence, under call-by-name reduction. We show some of the congruence rules for the equality relation below. They are standard, but note that these rules need to ensure that the grade annotations on the terms being equated match up.

GRAD-CONV
$$\frac{\Gamma_1 \vdash a : A \qquad \Gamma \vdash B : s \qquad A =_\beta B \qquad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma \rfloor}{\Gamma_1 \vdash a : B}$$

EQ-PI
$$\frac{A_1 =_\beta A_2 \qquad B_1 =_\beta B_2}{\Pi x :^r A_1 . B_1 =_\beta \Pi x :^r A_2 . B_2}$$

EQ-LAM
$$\frac{A_1 =_\beta A_2 \qquad b_1 =_\beta b_2}{\lambda^r x : A_1 . b_1 =_\beta \lambda^r x : A_2 . b_2}$$

EQ-APP
$$\frac{b_1 =_\beta b_2 \qquad a_1 =_\beta a_2}{b_1 \; {a_1}^r =_\beta b_2 \; {a_2}^r}$$

The type system of GRAD also includes rules for Unit and sum type, which are similar to their counterparts in GLC and as such, are elided here. In addition to these rules, GRAD has rules for axioms, weakening and subusaging, which are shown below and are as expected.

GRAD-AXIOM
$$\frac{\mathcal{A}(s_1, s_2)}{\varnothing \vdash s_1 : s_2}$$

GRAD-WEAK
$$\frac{\Gamma \vdash a : A \qquad \Gamma_0 \vdash B : s \qquad \lfloor \Gamma_0 \rfloor = \lfloor \Gamma \rfloor}{\Gamma, y :^0 B \vdash a : A}$$

GRAD-SUB
$$\frac{\Gamma_1 \vdash a : A \qquad \Gamma_2 <: \Gamma_1}{\Gamma_2 \vdash a : A}$$

Now that we have seen the type system, let us consider a few examples of derivable and non-derivable terms in $\text{GRAD}(\mathcal{Q}_{\text{Lin}})$:

$$\varnothing \vdash \lambda^0 x . \lambda^1 y . y : \Pi x :^0 s . \Pi y :^1 x . x$$
$$\varnothing \nvdash \lambda^0 x . \lambda^0 y . y : \Pi x :^0 s . \Pi y :^0 x . x$$
$$\varnothing \vdash \lambda^0 x . \lambda^1 y . \textbf{let}\,({y_1}^1, y_2) = y \textbf{ in } ({y_2}^1, y_1) : \Pi x :^0 s . \Pi y :^1 {}^1 A \times A . {}^1 A \times A$$
$$\varnothing \nvdash \lambda^0 x . \lambda^1 y . \textbf{let}\,({y_1}^1, y_2) = y \textbf{ in } ({y_1}^1, y_1) : \Pi x :^0 s . \Pi y :^1 {}^1 A \times A . {}^1 A \times A$$

Note that we use ${}^{q}A \to B$ and ${}^{q}A \times B$ to denote the non-dependent Pi-type and Sigma-type, $\Pi z :^{q} A.B$ and $\Sigma z{:}^{q} A.B$ respectively, where z is assumed to be fresh. Now, the first term in the above list is the polymorphic identity function, which uses a linear resource exactly once, whereas the third term is a polymorphic swap function, which swaps two linear resources. On the other hand, the second term uses a no-usage resource whereas the fourth term copies one linear resource while discarding the other. As such, the first and third terms are derivable in GraD($\mathcal{Q}_{\text{Lin}}$) whereas the second and fourth ones are not.

Next, we look at the metatheory of GraD.

4.6.3 Metatheory

We present the weakening and substitution lemmas for GraD below. With respect to usage analysis, these lemmas are similar to their GLC counterparts.

Lemma 4.16 (Weakening) If $\Gamma_1, \Gamma_2 \vdash a : A$ and $\Gamma \vdash C : s$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$, then $\Gamma_1, z :^{0} C, \Gamma_2 \vdash a : A$.

Lemma 4.17 (Substitution) If $\Gamma_1, z :^{r} C, \Gamma_2 \vdash a : A$ and $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$, then $\Gamma_1 + r \cdot \Gamma, \Gamma_2\{c/z\} \vdash a\{c/z\} : A\{c/z\}$.

Now, akin to GLC, GraD analyzes usage with respect to a call-by-name semantics. So we prove type-soundness for GraD with respect to a small-step call-by-name reduction relation. The reduction relation for GraD is exactly the same as that of GLC. With respect to this reduction relation, GraD enjoys the standard type-soundness properties of preservation and progress, as shown below.

Theorem 4.18 (Preservation) If $\Gamma \vdash a : A$ and $\vdash a \leadsto a'$, then $\Gamma \vdash a' : A$.

Theorem 4.19 (Progress) If $\varnothing \vdash a : A$, then either a is a value or there exists some a' such that $\vdash a \leadsto a'$.

Next, we develop a heap semantics for GraD, similar to that of GLC.

4.7 Heap Semantics for GraD

GraD enjoys a heap reduction relation same as that of GLC. However, the presence of dependent types in GraD causes an issue with regard to well-typedness of terms during reduction. In GLC, we could delay substitutions in a term by loading them into the heap, without being concerned about how that might affect the type of the term. In GraD, however, delayed substitutions can cause the term to 'lag behind' the type. Below, we consider an example that illustrates this point.

The polymorphic identity function, $\lambda^{0} x{:}s.\lambda^{1} y{:}x.y$, has type $\Pi x :^{0} s.\Pi y :^{1} x.x$. Applying the function to the type argument **Unit**, we get $(\lambda^{0} x{:}s.\lambda^{1} y{:}x.y)\,\mathbf{Unit}^{0}$ of type $\Pi y :^{1} \mathbf{Unit}.\mathbf{Unit}$. Now, by rule Heap-AppBeta, $[\varnothing]\,(\lambda^{0} x{:}s.\lambda^{1} y{:}x.y)\,\mathbf{Unit}^{0} \Rightarrow [x \overset{0}{\mapsto} \mathbf{Unit}]\,\lambda^{1} y{:}x.y$. Unless we look at the definition of x in the heap, we have no reason to believe that $\lambda^{1} y{:}x.y$ has type $\Pi y :^{1} \mathbf{Unit}.\mathbf{Unit}$. The delayed substitution, $x \overset{0}{\mapsto} \mathbf{Unit}$, causes the term $\lambda^{1} y{:}x.y$ to lag behind the type $\Pi y :^{1} \mathbf{Unit}.\mathbf{Unit}$. In order to align the types of the redex and the reduct,

we need to inform the type system about the new definition loaded into the heap. Observe that this issue is not usage-related and does not exist in GLC, where types cannot depend upon term variables. But the issue presents itself in GRAD because GRAD is dependently-typed and the heap-based reduction relation delays substitutions. The good news is that the issue can be resolved with a straightforward extension, whereby the type system of GRAD is informed about the definitions in the heap.

4.7.1 A Dependently-Typed Language with Definitions

In this section, we extend GRAD with support for definitions. Definitions are used in deriving type equalities *only*. From the perspective of the type system, they are essentially a bookkeeping device added to enable reasoning with respect to heap semantics.
First, we extend graded contexts with definitions that mimic delayed substitutions:

$$\textit{graded contexts} \quad \Gamma \quad ::= \quad \varnothing \mid \Gamma, x :^q A \mid \Gamma, x = a :^q A$$

Next, we modify the conversion rule and add two new rules to the type system, as shown below.

$\boxed{\Gamma \vdash a : A}$ *(Typing rules for dependent system with definitions)*

$$\textsc{GraD-Conv-Def}\ \frac{\Gamma_1 \vdash a : A \qquad \Gamma \vdash B : s \qquad A\{\Gamma_1\} =_\beta B\{\Gamma_1\} \qquad \lfloor\Gamma_1\rfloor = \lfloor\Gamma\rfloor}{\Gamma_1 \vdash a : B}$$

$$\textsc{GraD-Var-Def}\ \frac{\Gamma \vdash a : A}{0 \cdot \Gamma, x = a :^1 A \vdash x : A}$$

$$\textsc{GraD-Weak-Def}\ \frac{\Gamma \vdash a : A \qquad \Gamma_0 \vdash b : B \qquad \lfloor\Gamma_0\rfloor = \lfloor\Gamma\rfloor}{\Gamma, y = b :^0 B \vdash a : A}$$

Here, $A\{\Gamma\}$ denotes the type obtained by substituting in A, in reverse order, the definitions of the variables that are defined in Γ. Further, $\lfloor\Gamma\rfloor$ denotes the context Γ, without the grades on the variable assumptions and definitions.

Observe that definitions act like mere variable assumptions in rules GRAD-VAR-DEF and GRAD-WEAK-DEF, which mirror rules GRAD-VAR and GRAD-WEAK respectively. But definitions become important in the conversion rule GRAD-CONV-DEF, which substitutes them before comparing the types for β-equivalence. With this modified conversion rule, the term $\lambda^1 y : x.y$ can be assigned type $\Pi y :^1 \mathbf{Unit}.\mathbf{Unit}$ in a context that defines x to be **Unit**. Note here that the above modifications extend the type system of GRAD; they do not add any restrictions to the original system.

Now, the type system of GRAD, with the above-mentioned extensions, enjoys all the syntactic soundness properties listed in Section 4.6.3. Furthermore, the extended system enjoys additional properties presented below.
Since definitions act only on types, they do not add extra resource demands to any typing derivation. As a result, we can always add an appropriate definition to a standard contextual assumption in any typing

judgment, as shown in the lemma below. Note that in this lemma, the resources used by the definiens (i.e., Γ) do not matter.

Lemma 4.20 (Inserting definitions) If $\Gamma_1, z :^r C, \Gamma_2 \vdash a : A$ and $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$, then $\Gamma_1, z = c :^r C, \Gamma_2 \vdash a : A$.

We can also weaken the context of any typing judgment with new and unused definitions, analogous to usual weakening with variable assumptions.

Lemma 4.21 (Weakening with definitions) If $\Gamma_1, \Gamma_2 \vdash a : A$ and $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$, then $\Gamma_1, z = c :^0 C, \Gamma_2 \vdash a : A$.

Further, we can also substitute the contextual definitions in any typing judgment, as shown in the following lemma. Note that in this lemma, z is substituted by c and not by an arbitrary c' satisfying $\Gamma \vdash c' : C$. This restriction is necessary to ensure well-typedness of the substituted term. To see why, consider the judgment: $z = {}^1\mathbf{Unit} \to \mathbf{Unit} :^0 s \vdash \lambda^1 y : z. y\ \mathbf{unit}^1 : {}^1({}^1\mathbf{Unit} \to \mathbf{Unit}) \to \mathbf{Unit}$. Now, if we substitute z by an arbitrary c' satisfying $\varnothing \vdash c' : s$ (such as, $c' = \mathbf{Unit}$), the substituted term, $\lambda^1 y : c'. y\ \mathbf{unit}^1$, may not be well-typed. But we may always substitute z by its definition, while maintaining well-typedness.

Lemma 4.22 (Substituting definitions) if $\Gamma_1, z = c :^r C, \Gamma_2 \vdash a : A$ and $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$, then $\Gamma_1 + r \cdot \Gamma, \Gamma_2\{c/z\} \vdash a\{c/z\} : A\{c/z\}$.

Next, we move on to heap reduction and compatibility relations. Since these relations make use of contexts and we have modified contexts to include definitions, we need to modify these relations as well. The modifications required, however, are minor. As far as the reduction relation is concerned, only the β-rules that load new assignments into the heap need to be updated: the added context in these rules should now remember the assignments of the new variables. For example, rule Heap-AppBeta gets updated to rule Heap-AppBeta-Def, as shown below.

Heap-AppBeta

$$\frac{y\ \mathit{fresh} \qquad b' = b\{y/x\}}{[H]\,(\lambda^q x : A_0. b)\ a^q \Rightarrow^r_S [H, y \overset{r \cdot q}{\mapsto} (\Gamma | a | A)\,;\, \mathbf{0}\,;\, y :^{r \cdot q} A]\ b'}$$

Heap-AppBeta-Def

$$\frac{y\ \mathit{fresh} \qquad b' = b\{y/x\}}{[H]\,(\lambda^q x : A_0. b)\ a^q \Rightarrow^r_S [H, y \overset{r \cdot q}{\mapsto} (\Gamma | a | A)\,;\, \mathbf{0}\,;\, y = a :^{r \cdot q} A]\ b'}$$

As far as the compatibility relation is concerned, rule Compat-Cons also needs to ensure that the context remembers the definitions of the domain variables. So, we update rule Compat-Cons to rule Compat-Cons-Def, as shown below. With this update, we have the following interesting property: if $H \Vdash \Gamma$, then H and Γ contain the same list of definitions. Then if $\Gamma \vdash a : A$, the conversion rule Grad-Conv-Def can make use of the definitions in H while type-checking a. In this way, the heap can inform the type system about delayed substitutions.

$$\frac{H \Vdash \Gamma_1 + (q \cdot \Gamma_2) \qquad \Gamma_2 \vdash a : A}{H, x \overset{q}{\mapsto} (\Gamma_2 | a | A) \Vdash \Gamma_1, x :^q A}\ \textsc{Compat-Cons}$$

$$\frac{H \Vdash \Gamma_1 + (q \cdot \Gamma_2) \qquad \Gamma_2 \vdash a : A}{H, x \overset{q}{\mapsto} (\Gamma_2 | a | A) \Vdash \Gamma_1, x = a :^q A}\ \textsc{Compat-Cons-Def}$$

These are all the modifications we need in the reduction and compatibility relations.
This also completes our description of the extended system.

Now, we establish a correspondence between the original system and the extended one. Towards this end, we relate the original typing and heap compatibility judgments to their counterparts in the extended system. To distinguish, let us denote the typing and compatibility judgments of the original system as $\Gamma \vdash_o a : A$ and $H \Vdash_o \Gamma$ respectively and those of the extended system as $\Gamma \vdash_e a : A$ and $H \Vdash_e \Gamma$ respectively. Now, for $H \Vdash_o \Gamma$, let Γ_H denote Γ with the variables defined according to the assignments in H. Also, let H_H denote H with the variables in the embedded contexts in H defined according to H. Then, we can show that given a derivation in the original system, adding appropriate definitions to the context gives us a derivation in the extended system. In other words, typing and compatibility judgments in the original system can be elaborated to typing and compatibility judgments in the extended system, as shown in the lemma below.

Lemma 4.23 (Elaboration) If $H \Vdash_o \Gamma$ and $\Gamma \vdash_o a : A$, then $H_H \Vdash_e \Gamma_H$ and $\Gamma_H \vdash_e a : A$.

Conversely, we can show that given a derivation in the extended system, substituting the definitions gives us a derivation in the original system. In other words, a typing judgment in the extended system, with definitions substituted, is a typing judgment of the original system, as shown by the lemma below.

Lemma 4.24 (Multi-substitution) If $H \Vdash_e \Gamma$ and $\Gamma \vdash_e a : A$, then $\varnothing \vdash_o a\{H\} : A\{H\}$.

From the above lemmas, we see that the original and extended systems are essentially equivalent. So then, if we can show that the extended system is sound with respect to heap semantics, we can conclude that the same for the original one. Next, we show that the extended system is indeed sound with respect to heap semantics.

4.7.2 Heap Soundness Theorem

GRAD, extended with definitions, is sound with respect to heap semantics. Note the statement of the soundness theorem below is the same as that of the soundness theorem for GLC.

Theorem 4.25 (Soundness) If $H \Vdash \Gamma$ and $\Gamma \vdash a : A$, then either a is a value or there exists H', $\mathbf{u'}$, Γ'_0 and Γ' such that:

- $[H]\, a \Rightarrow^1_S [H' ; \mathbf{u'} ; \Gamma'_0]\, a'$
- $\Gamma' \vdash a' : A$
- $H' \Vdash \Gamma'$
- $\overline{\Gamma} \diamond \mathbf{0} \ +\ \mathbf{u'} \times \langle H' \rangle \ +\ \mathbf{0} \diamond \overline{\Gamma'_0} \ \ <:\ \ \overline{\Gamma'} \ +\ \mathbf{u'} \ +\ (\mathbf{0} \diamond \overline{\Gamma'_0}) \times \langle H' \rangle$

The soundness theorem above shows that usage analysis in GRAD is correct. This soundness theorem is similar in spirit to theorems showing correctness of usage analysis via operational methods in graded-context type systems, such as, Abel and Bernardy [2020], Brunel et al. [2014]. But this theorem can be proved by simple induction on the typing derivation; it does not require extra machinery. In contrast, the proof of soundness in Brunel et al. [2014] requires a realizability model over and above the reduction relation. The proof of soundness in Abel and Bernardy [2020], referred to as modality preservation, is more similar to the above theorem. However, modality preservation is not employed by Abel and Bernardy [2020] to reason about no usage and linear usage; in fact, they develop an alternative relational semantics for this purpose. On the other hand, our soundness theorem can be readily employed to reason about such usages, as we saw in Section 4.4 and shall also see below.

We can use the above soundness theorem to prove the usual preservation and progress lemmas. We can also use this theorem to reason about no usage and linear usage, as discussed in Section 4.4. Further, using this theorem, we can prove several interesting corollaries related to polymorphic types.

Corollary 4.26 In GRAD parametrized over a zero-unusable semiring, if $\varnothing \vdash f : \Pi x :^0 s.x$ and $\varnothing \vdash A : s$, then $f\ A^0$ must diverge.

Corollary 4.27 In GRAD parametrized over a zero-unusable semiring and a strongly normalizing PTS, if $\varnothing \vdash f : \Pi x :^0 s.\Pi y :^1 x.x$ and $\varnothing \vdash A : s$ and $\varnothing \vdash a : A$, then $f\ A^0\ a^1 =_\beta a$.

Corollary 4.28 In GRAD parametrized over a one-linear semiring and a strongly normalizing PTS, if $\varnothing \vdash f : \Pi x :^0 s.\Pi y :^1 {}^1x \times x.{}^1x \times x$ and $\varnothing \vdash A : s$ and $\varnothing \vdash a_1 : A$ and $\varnothing \vdash a_2 : A$, then $f\ A^0\ ({a_1}^1, a_2)^1 =_\beta ({a_1}^1, a_2)$ or $f\ A^0\ ({a_1}^1, a_2)^1 =_\beta ({a_2}^1, a_1)$.

The above corollaries are similar to 'free theorems' that follow from parametric polymorphism [Wadler, 1989]. As a matter of fact, the second corollary is the poster child of 'free theorems'. But notwithstanding this similarity, the motivations behind the two are different: the 'free theorems' are a consequence of parametric treatment of type variables, whereas the above corollaries are a consequence of restrictions on variable usage. Interestingly, it turns out that parametric polymorphism is related to no usage. Mishra-Linger [2008] first made this observation, though in the context of irrelevance analysis. Mishra-Linger [2008] showed that parametric polymorphism can be understood in terms of run-time irrelevance. Now, run-time irrelevance, as presented in the previous chapter, is very similar to no usage; we shall elaborate on this point in the following section. So, it does not come as a surprise that 'free theorems' related to parametric polymorphism follow from soundness of usage analysis.

As we pointed out above, there is a close relation between run-time irrelevance and no usage. In the next section, we make this relation precise, as we compare GRAD with DDC$^\top$ and DDC.

4.8 GraD and DDC$^{\top}$/DDC: A Comparison

4.8.1 Technical Similarities and Differences

To begin with, observe that GRAD, DDC$^{\top}$ and DDC are alike on some important technical aspects:

1. In all the three calculi, the contexts of typing judgments are graded. Further, the calculi also employ graded types for their respective analyses.

2. In all the three calculi, the analysis carried out pertains to the expression that appears as a term in a given typing judgment. The variable rules of the calculi illustrate this point nicely:

$$\textsc{GraD-Var}\ \frac{\Gamma \vdash A : s}{0 \cdot \Gamma, x :^{1} A \vdash x : A} \qquad \textsc{DCT-Var}\ \frac{\Omega \vdash A :^{\top} s \qquad \ell_0 \sqsubseteq \ell}{\Omega, x :^{\ell_0} A \vdash x :^{\ell} A} \qquad \textsc{Ddc-Var}\ \frac{\begin{array}{c}\Omega \Vdash A :^{\top} s \\ \ell_0 \sqsubseteq \ell \qquad \ell \sqsubseteq \mathcal{C}\end{array}}{\Omega, x :^{\ell_0} A \vdash x :^{\ell} A}$$

   In rule GRAD-VAR, the premise and conclusion judgments account the resources used by the expressions appearing as terms, i.e., A and x respectively. Similarly, in rules DCT-VAR and DDC-VAR, the premise and conclusion judgments track dependencies for the expressions appearing as terms, i.e., A and x respectively. This feature enables these calculi to decouple the analysis in terms from the analysis in their types and contributes to their minimalistic designs.

3. In all the three calculi, the treatment of an assumption in a type is not related to its treatment in a term having that type. The Pi rules of the calculi illustrate this point nicely:

$$\textsc{GraD-Pi}\ \frac{\begin{array}{c}\Gamma_1 \vdash A : s_1 \\ \Gamma_2, x :^{r} A \vdash B : s_2 \\ \mathcal{R}(s_1, s_2, s_3) \qquad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor\end{array}}{\Gamma_1 + \Gamma_2 \vdash \Pi x :^{q} A.B : s_3} \qquad \textsc{DCT-Pi}\ \frac{\begin{array}{c}\Omega \vdash A :^{\ell} s_1 \\ \Omega, x :^{\ell} A \vdash B :^{\ell} s_2 \\ (s_1, s_2, s_3) \in \mathcal{R}\end{array}}{\Omega \vdash \Pi x :^{\ell_0} A.B :^{\ell} s_3} \qquad \textsc{Ddc-Pi}\ \frac{\begin{array}{c}\Omega \vdash A :^{\ell} s_1 \\ \Omega, x :^{\ell} A \vdash B :^{\ell} s_2 \\ (s_1, s_2, s_3) \in \mathcal{R}\end{array}}{\Omega \vdash \Pi x :^{\ell_0} A.B :^{\ell} s_3}$$

   In rule GRAD-PI, r is not related to q. Similarly, in rules DCT-PI and DDC-PI, ℓ is not related to ℓ_0. This feature automatically decouples usage/dependency constraints in types from those in terms. Note how the feature mentioned above (at point number 2) aligns with this one.

4. In all the three calculi, the annotations on bound variables in types signify how those variables are constrained in terms having those types. For example, the annotation on the bound variable in a Π-type signifies how that variable is constrained in the body of a function having that type.

Observe that the last three features pointed out above correspond to the three design questions that came up while we were considering the problem of combining linearity analysis with dependent types in Section 1.1.8. Thereafter in Section 1.1.10, we saw how GRAD answers these questions. Here, we find that DDC$^{\top}$

and DDC answer these questions in the same way as GRAD does. The reader would have noticed that we didn't keep these questions in mind while designing $\text{DDC}^{\top}$ and DDC. Nevertheless, our design choices for these calculi have followed those for GRAD. This conspicuous resemblance between the design choices stems from the basic principles that guided our design of all the three calculi: simplicity and minimalism. Recall that in Sections 1.1.9, 1.1.10 and 1.1.11, we discussed how our design choices help realize these principles in GRAD, as compared to other calculi that make different design choices. In the light of that discussion, it does not come as a surprise that guided by the same principles, we made the same design choices for $\text{DDC}^{\top}$ and DDC.

We see that there are several technical similarities between GRAD and $\text{DDC}^{\top}$/DDC. But there are also several clear technical differences between them. GRAD is parametrized by preordered semirings whereas $\text{DDC}^{\top}$ and DDC are parametrized by lattices. GRAD employs a typing judgment that has a fixed grade to the right of the turnstile whereas $\text{DDC}^{\top}$ and DDC employ typing judgments that allow the grade to the right of the turnstile to vary. GRAD extensively uses context operations in typing rules whereas $\text{DDC}^{\top}$ and DDC seldom use them. This last difference, however, is mostly of a presentational nature because we shall see in Section 5.4.1 that the typing rules of $\text{DDC}^{\top}$/DDC can be presented in a way that makes similar use of context operations as GRAD. But the other two differences pointed out above are more basic in nature, as we saw in Sections 1.3.1 and 3.5.2.

In the next chapter, we shall make use of the technical similarities between GRAD and $\text{DDC}^{\top}$/DDC and reconcile their technical differences as we design a unified calculus for linearity and dependency analyses in pure type systems. But for now, we consider how the two analyses, as carried out in these calculi, compare with one another.

4.8.2 Similarities and Differences in Analyses

In Section 4.5, we compared GLC with SDC. We saw that over the boolean semiring $\mathbb{B}_{\geq}$, SDC subsumes GLC, implying that no usage and potential usage can be understood in terms of high-security and low-security levels respectively. We can carry this argument forward and show that over the same semiring, the dependent version of SDC, i.e. $\text{DDC}^{\top}$, subsumes the dependent version of GLC, i.e. GRAD.

Towards this end, we present a meaning-preserving translation from $\text{GRAD}(\mathbb{B}_{\geq})$ to $\text{DDC}^{\top}(\mathbb{B}_{\geq})$, both parametrized over the same PTS. (Recall here that by the construction described in Section 4.5.1, $\mathbb{B}_{\geq}$ is also a lattice, isomorphic to $\mathcal{L}_2$.) The translation function, $\overline{}$, is defined by straightforward recursion on terms; we present some of the cases in Figure 4.5. This translation preserves typing and meaning, as we see in the lemma below.

Lemma 4.29 If $\Gamma \vdash a : A$ in $\text{GRAD}(\mathbb{B}_{\geq})$, then $\overrightarrow{\Gamma} \vdash \overline{a} :^{1} \overline{A}$ in $\text{DDC}^{\top}(\mathbb{B}_{\geq})$, parametrized over the same PTS. Further, if $\vdash a \leadsto a'$ in $\text{GRAD}(\mathbb{B}_{\geq})$, then $\vdash \overline{a} \leadsto \overline{a'}$ in $\text{DDC}^{\top}(\mathbb{B}_{\geq})$.

Note that $\overrightarrow{\Gamma}$ here denotes the context Γ, but with the types translated. (We use the symbol $\overrightarrow{\Gamma}$ in lieu of $\overline{\Gamma}$ because the latter already has a meaning attached to it.) Now, from the above lemma, we can see that no

$$\overline{x} = x \qquad \overline{s} = s \qquad \overline{\mathbf{Unit}} = \mathbf{Unit}$$

$$\overline{\Pi x :^q A.B} = \Pi x :^q \overline{A}.\overline{B} \qquad \overline{\lambda^q x : A.b} = \lambda^q x : \overline{A}.\overline{b} \qquad \overline{b\ a^q} = \overline{b}\ \overline{a}^q$$

Figure 4.5: Term translation from GraD($\mathbb{B}_{\geq}$) to DDC$^{\top}$($\mathbb{B}_{\geq}$) (Excerpt)

usage in GraD($\mathbb{B}_{\geq}$) can be understood in terms of run-time irrelevance in DDC$^{\top}$($\mathbb{B}_{\geq}$). Conversely, run-time irrelevance can also be understood in terms of no usage. However, we would not be able to show this result by means of a translation from DDC$^{\top}$($\mathbb{B}_{\geq}$) to GraD($\mathbb{B}_{\geq}$) because certain terms derivable in the former do not have equivalents in the latter. This situation is similar to that between SDC($\mathbb{B}_{\geq}$) and GLC($\mathbb{B}_{\geq}$), discussed in Section 4.5.2. Recall that we could not translate SDC($\mathbb{B}_{\geq}$) to GLC($\mathbb{B}_{\geq}$) because the latter does not derive a join operator and also does not allow an arbitrary function to return high-security values, even when the function itself is wrapped under a high-security label. These shortcomings of GLC carry over to its dependent version, GraD, thereby thwarting a translation from DDC$^{\top}$. However, in the next chapter, we shall show that these shortcomings can be addressed by a simple modification to the typing judgment of GLC and GraD. That modification will help us show (in Section 5.6.3) that run-time irrelevance and no usage analyses are essentially the same.

Moving ahead with our comparison between GraD and DDC$^{\top}$, note that owing to the shortcomings mentioned above, GraD is not well-suited for dependency analysis; on the other hand, DDC$^{\top}$, being parametrized by lattices, cannot help in linearity analysis. Now, DDC is a more general calculus than DDC$^{\top}$ but when it comes to linearity analysis, both the calculi are equally ineffective. So, we see that both GraD and DDC$^{\top}$/DDC have their strengths and weaknesses. Moreover, these strengths complement one another. In the next chapter, we devise a way to harness these complementary strengths on our way to designing a unified calculus for linearity and dependency analyses.

4.9 Discussions and Related Work

4.9.1 GraD as a General Coeffect Calculus

Till now, we have employed GraD for usage analysis only. However, GraD, like other graded-context type systems [Brunel et al., 2014, Ghica and Smith, 2014, Petricek et al., 2014], can be employed as a general coeffect calculus. Recall that coeffects model how the environment affects computations. To give an example, usage of data is a coeffect because by restricting usage, the environment can disallow computations from progressing. Other examples of coeffects include the number of steps required to evaluate programs [Brunel et al., 2014], the number of threads required to run parallel programs [Ghica and Smith, 2014], the number of past values of a variable required in stream-processing computations [Petricek et al., 2014], etc. Now, Brunel et al. [2014], Ghica and Smith [2014] and Petricek et al. [2014] have shown that graded-context type systems, parametrized over semirings, can be employed as general coeffect calculi. Since GraD is also

a graded-context type system parametrized over semirings, one may follow the constructions presented by these authors to analyze coeffects other than usage in GRAD. In this dissertation, however, we do not explore other coeffects because our focus is on usage and dependency analyses only.

4.9.2 Technical Comparison with QTT and GrTT

In Sections 1.1.9, 1.1.10 and 1.1.11, we discussed how QTT [Atkey, 2018] and GRTT [Moon et al., 2021] compare against GRAD. Here, we look at some of the finer technical differences between these three calculi.

- First, note that all three calculi are parametrized by semirings. However, there are some technical differences with regard to the exact specification of the parametrizing structure. QTT requires the parametrizing semiring to be both zerosumfree and entire (see Section 4.4.2 for definitions); these properties are essential to admit substitution in QTT. On the other hand, GRAD and GRTT do not impose any such restrictions on the parametrizing semiring.

  In this regard, an interesting point to note is that GRTT and QTT, unlike GRAD, do not include a notion of order in the parametrizing structure. Now, without an order, there is no straightforward way to allow subusaging. In fact, GRTT and QTT do not allow subusaging. In Section 1.1.6, we pointed how subusaging enables the same type system to analyze both exact and bounded usage. In the absence of subusaging, GRTT and QTT can only analyze exact usage, but not bounded usage. This limitation has consequences. One consequence is that these calculi cannot allow linear usage of unrestricted resources, i.e., the typing judgment, $x :^{\omega} A \vdash x : A$, does not hold in GRTT or in QTT, parametrized over the semiring $\{0, 1, \omega\}$. Another consequence is that these calculi cannot reason about affine (at most once) usage. These shortcomings, however, may be addressed by including a notion of order in the parametrizing semiring and following that up with a subusaging rule.

- Second, recall that QTT tracks usage in types and terms differently: types live in a resource-agnostic world whereas terms live in a resource-aware world. So, QTT employs two forms of typing judgments: a resource-agnostic form for types, $\Gamma \vdash A :^0 A'$, and a resource-aware one for terms, $\Gamma \vdash a :^1 A$. On the other hand, both GRAD and GRTT employ only one form of typing judgment, which is resource-aware. We pointed out earlier that a single form of typing judgment results in a simpler and more uniform calculus. Further, it also leads to less restrictions on the parametrizing structure, as described above. QTT requires the parametrizing semiring to be zerosumfree and entire precisely owing to the dichotomy introduced in the calculus by its two constituent fragments. However, there is also a benefit that ensues from having a resource-agnostic fragment in addition to a resource-aware one. Unlike GRAD and GRTT, QTT can support strong Σ-types in its resource-agnostic fragment. In the next chapter, we shall design a calculus that incorporates the best of both QTT and GRAD into a single system. That calculus would treat types and terms uniformly and would also have a resource-agnostic fragment that supports strong Σ-types.

4.9.3 Heap Semantics for Linear Calculi

Computational and operational interpretations of linear logic have been explored in several works, most notably in Chirimar et al. [2000] and Turner and Wadler [1999]. Turner and Wadler [1999], building upon Chirimar et al. [2000], provide two heap-based operational interpretations of linear logic. In the first interpretation, nonlinear values are recomputed every time they are required during reduction, akin to our heap semantics. In the second interpretation, nonlinear values are computed only once and the results are memoized, akin to a call-by-need semantics. Turner and Wadler [1999] show that linear logic, under the first interpretation, enjoys the single pointer property, meaning, at run time, any linear resource has exactly one pointer to it. But under the second interpretation, linear logic only satisfies a weaker version of this property, which guarantees the maintenance of a single pointer, meaning, if there is a single pointer to some linear data to begin with, no new pointers will be added to that data during the course of reduction.

Our heap semantics for GLC and GRAD is based on the first interpretation mentioned above. In line with this interpretation, we recompute values every time they are required and do not memoize results of any computations. As a result, our calculi enjoy the (stronger version of the) single pointer property, as shown in Lemma 4.14. Our heap semantics may, in fact, be regarded as a generalization of the heap-based operational interpretation (the first one) of Turner and Wadler [1999]. The generalization is along two directions: first, from their simply-typed setting to our dependently-typed setting and second, from just a linear/nonlinear usage analysis in their case to usage analysis over an arbitrary preordered semiring in our case.

However, unlike Turner and Wadler [1999], we do not explore the other heap-based interpretation, i.e., call-by-need interpretation, for our calculi. In Section 4.2.1, we pointed out that our type system accounts usage, implicitly assuming a call-by-name reduction strategy. As such, we would not be able to interpret our calculi with respect to a call-by-need reduction strategy. If we wish to do so, we would first need to modify our type system to account usage from a call-by-need perspective. While such a modification might be possible, it would not be straightforward because usage demands vary drastically as we move from a call-by-name reduction strategy to a call-by-need one.

Turner and Wadler [1999] employ their heap-based interpretations to show useful properties like safety of in-place update of linear resources. In this chapter, we have employed our heap semantics to prove a variety of useful properties in addition to plain soundness. Heap semantics, as presented in Turner and Wadler [1999] and as generalized in this chapter, is also useful in proving soundness properties of other calculi that analyze usage in some form. We give two examples below. In a recent work, Marshall et al. [2022] adapted our heap semantics to prove soundness for a calculus that combines linearity with uniqueness, the property of having a unique reference. In the next chapter, we shall see that our heap semantics, with minor modifications, can also help us prove noninterference for a dependency calculus.

4.10 Conclusion

Graded-context type systems provide a general framework for analyzing usage of resources in programs. In this chapter, we extended this framework to dependently-typed languages. We designed a graded-context dependently-typed calculus, GRAD, that can analyze a wide variety of usages, including no usage and linear usage. We presented a standard substitution-based semantics and a usage-aware heap-based semantics for GRAD. We then showed that GRAD is sound with respect to both the semantics. Given that the heap-based semantics ensures fairness of usage, soundness with respect to this semantics implies GRAD analyzes usages correctly. In literature, there are a few other calculi for analyzing usages in dependently-typed languages. What makes GRAD unique among them is its minimalist design. This feature makes GRAD a theoretically elegant calculus and also makes it suitable for implementation in practical programming languages.

Chapter 5

Combined Linearity and Dependency Analysis in Pure Type Systems

In Chapter 3, we analyzed dependencies in pure type systems. In Chapter 4, we analyzed linearity in pure type systems. In this chapter, we shall unify these analyses. There are several benefits to our prospective unification of linearity and dependency analyses.

- First, from a theoretical perspective, it would unify the standard calculi for linearity analysis with the standard calculi for dependency analysis. These two sets of calculi have developed independent of one another and it is only recently that researchers have gained insight into the common structure unifying them. This insight has led to the development of calculi that can analyze linearity and restricted forms of dependency. A calculus unifying linearity analysis with a general dependency analysis, however, is still missing. But such a calculus could be beneficial because it could show us concretely how the two analyses correspond to each other.
- Second, from a practical perspective, it would allow programmers to use the same type system for both linearity and dependency analyses. Presently, programmers use different type systems for this purpose. For example, Haskell programmers use Linear Haskell extension [Bernardy et al., 2018] for linearity analysis and LIO library [Stefan et al., 2017] for dependency analysis. A unified analysis can help unify the type systems of Linear Haskell and LIO library, and likewise for similar type systems in other programming languages.
- Third, it would allow a combination of the two analyses. A combined analysis is more powerful than the individual analyses done separately because it allows arbitrary combination of usage and flow constraints. For example, in a combined analysis, a piece of data may be simultaneously linear and secure; then, that piece of data must be used exactly once, and that too, in a secure context. We illustrate this point through the System F type: $\forall\alpha.\alpha \times \alpha \to \alpha \times \alpha$. Up to equality, this

type has exactly four inhabitants: $\Lambda\alpha.\lambda(x,y).(x,y)$ and $\Lambda\alpha.\lambda(x,y).(y,x)$ and $\Lambda\alpha.\lambda(x,y).(x,x)$ and $\Lambda\alpha.\lambda(x,y).(y,y)$. The type $\forall\alpha.{}^{1}\alpha \times {}^{1}\alpha \to {}^{1}\alpha \times {}^{1}\alpha$ imposes linearity constraints and as such, has only the inhabitants: $\Lambda\alpha.\lambda(x,y).(x,y)$ and $\Lambda\alpha.\lambda(x,y).(y,x)$. The type $\forall\alpha.{}^{\mathbf{H}}\alpha \times {}^{\mathbf{L}}\alpha \to {}^{\mathbf{H}}\alpha \times {}^{\mathbf{L}}\alpha$ imposes dependency constraints and as such, has only the inhabitants: $\Lambda\alpha.\lambda(x,y).(x,y)$ and $\Lambda\alpha.\lambda(x,y).(y,y)$. The type $\forall\alpha.{}^{(1,\mathbf{H})}\alpha \times {}^{(1,\mathbf{L})}\alpha \to {}^{(1,\mathbf{H})}\alpha \times {}^{(1,\mathbf{L})}\alpha$ imposes both linearity and dependency constraints and as such, has $\Lambda\alpha.\lambda(x,y).(x,y)$ as the sole inhabitant. Observe that without a combined analysis, it would be difficult to express the type $\forall\alpha.{}^{(1,\mathbf{H})}\alpha \times {}^{(1,\mathbf{L})}\alpha \to {}^{(1,\mathbf{H})}\alpha \times {}^{(1,\mathbf{L})}\alpha$ because the type requires that linearity and dependency constraints be simultaneously imposed. (Here, the superscripts 1, $\mathbf{L}$ and $\mathbf{H}$ denote linear, low-security and high-security nature of data respectively.)

A unification of linearity and dependency analyses, though desirable, is not straightforward. Recall from Section 1.3 the two key differences between the analyses that need to be resolved to achieve such a unification: the differences in the parametrizing algebraic structures and the differences in the modalities used for the analyses. In this chapter, we shall resolve these differences. Our work here carries forward recent advances in graded-context type systems [Abel and Bernardy, 2020, Orchard et al., 2019] that show the two analyses can be viewed through the same lens and as such, can also be unified. However, existing graded-context type systems do not fully unify linearity analysis with a general dependency analysis. The problem with these type systems is that though they analyze linearity (and other usages) nicely, they do not perform very well, when it comes to analyzing dependencies. There are several aspects of dependency analysis that these type systems cannot capture. We discuss them in detail in Section 5.1.1. Notwithstanding these shortcomings, graded-context type systems take us quite close to unifying linearity and dependency analyses. In this chapter, we show that by systematically extending graded-context type systems, we can design a calculus for unified linearity and dependency analysis.

Concretely, we design LDC (for Linearity Dependency Calculus), which can analyze both linearity and dependency using the same mechanism. Following the style adopted in the last two chapters, we first present SLDC, the simply-typed version of LDC, and thereafter extend SLDC to dependent types. SLDC subsumes the standard simply-typed calculi in literature for linearity and dependency analyses, thereby showing that it is a general calculus for the two analyses in simple type systems. LDC, on the other hand, subsumes GRAD, parametrized over the linearity preordered semiring, and $\text{DDC}^{\top}$, parametrized over an arbitrary lattice, thereby showing that it is a general calculus for linearity and dependency analyses in pure type systems. Both SLDC and LDC enjoy a heap semantics similar to that of GLC and GRAD. Interestingly, this heap semantics can be used to show not only correctness of usage analysis but also that of dependency analysis, i.e., noninterference.

In sum, we make the following contributions in this chapter:

- We present two calculi, SLDC and LDC, which analyze linearity and dependency, using the same mechanism, in simple type systems and pure type systems respectively.
- We show that SLDC subsumes the standard calculi for analyzing linearity and dependency in simple type systems, such as, Linear Nonlinear λ-calculus of Benton [1994], DCC of Abadi et al. [1999] and

Sealing Calculus of Shikuma and Igarashi [2006]. We also show that LDC subsumes the calculi we designed for linearity and dependency analyses in pure type systems, i.e., GRAD, parametrized over the linearity preordered semiring, and DDC$^{\top}$, parametrized over an arbitrary lattice.

- We design a heap semantics for SLDC and LDC and show soundness of both linearity and dependency analyses using this semantics.
- We show that SLDC and LDC can carry out a combined linearity and dependency analysis.

5.1 Challenges and Resolution

By now, the reader would have noticed the similarities between linearity and dependency analyses not just on an abstract level but also on a concrete level. On an abstract level, the two analyses address the same problem of modeling multiple worlds that interact according to a set of given constraints. On a concrete level, the two analyses employ the same technical gadgets to achieve their ends. Chapters 3 and 4 show how these gadgets, such as, graded contexts, graded types and context operations, enable dependency and linearity analyses in pure type systems. These similarities, both on abstract and concrete levels, will form the basis of unification of the two analyses in this chapter. But in order to achieve this unification, we need to resolve some challenges.

The key challenge is to reconcile the difference in the form of the typing judgments employed for dependency and linearity analyses in Chapters 3 and 4 respectively. Recall that the typing judgment employed for dependency analysis allows an arbitrary grade to the right of the turnstile whereas the typing judgment employed for linearity analysis fixes the grade to the right of the turnstile to 1. In Section 3.5.2, we pointed out the reason behind this difference in the form of typing judgments. Now since we want to unify the two analyses, we need to come up with a single form of typing judgment that is suitable for both of them. There are two immediate alternatives that we can try: the two forms of typing judgments that we have already employed. Put differently, we can try to analyze dependencies using the same form of typing judgment we employed for linearity analysis and vice-versa. Below, we shall explore these alternatives one after the other.

5.1.1 Dependency Analysis Through Graded-Context Type Systems

In Chapter 4, we saw how graded-context type systems enable linearity analysis, and more generally, usage analysis, in pure type systems. Now given the close connection between linearity analysis and dependency analysis, it is natural to wonder if these graded-context type systems can also be employed for dependency analysis. (Our type systems for dependency analysis employ graded contexts as well, but here the term 'graded-context type systems' is used to refer exclusively to the graded-context type systems parametrized by preordered semirings, and geared for usage accounting.) Recall that in Section 4.5, we showed that graded-context type systems can indeed carry out some dependency analyses. The question, then, is whether these type systems can capture all the key aspects of dependency analysis. We consider this question below.

At the outset, note that graded-context type systems are parametrized over preordered semirings whereas dependencies are analyzed over lattice structures. In Section 1.3.1, we pointed out the irreconcilable differences between preordered semirings and lattices. But for the moment being, let us focus on distributive lattices, which may indeed be viewed as preordered semirings, where multiplication, addition and order are given by join, meet and lattice order respectively. So, we refine the above question as: can graded-context type systems capture all the key aspects of dependency analysis over distributive lattices? Unfortunately, the answer to this question is no. There are two key aspects of dependency analysis that these type systems cannot capture. We point them out below.

1. In Chapter 2, we showed that dependency analysis is both monadic and comonadic in nature. As such, the modality employed for dependency analysis needs to support the standard monadic and comonadic operations of join and fork respectively. Now, the modality employed in graded-context type systems can support a fork operation (see Section 4.2.2) but it does not support a join operation. In fact, a join operator cannot be derived for this modality in general, meaning, in a graded-context type system parametrized over $\mathcal{Q}$, no function of type $\Box^{q_1} \Box^{q_2} A \to \Box^{q_1 \cdot q_2} A$ is derivable, for an arbitrary type A and $q_1, q_2 \in \mathcal{Q}$. Proposition D.1 presents a model-theoretic proof of this claim. Note that this shortcoming is common to all standard graded-context type systems in literature [Abel and Bernardy, 2020, Atkey, 2018, Brunel et al., 2014, Ghica and Smith, 2014, Orchard et al., 2019, Petricek et al., 2014] and is an outcome, as we shall see, of the very nature of the typing judgment employed by these systems. Further, note that this shortcoming is inconsequential for usage analysis but significantly limits dependency analysis.

   An ad hoc fix to this shortcoming might be to add an explicit join operator to the type system, as follows:

$$\textsc{GC-Join}\quad \frac{\Gamma \vdash a : \Box^{q_1} \Box^{q_2} A}{\Gamma \vdash \mathbf{join}^{q_1, q_2}\ a : \Box^{q_1 \cdot q_2} A}$$

   But this fix is not very satisfactory because:

   - This **join** operator raises issues in operational semantics, especially with call-by-name reduction. To understand why, consider the question: how should $\mathbf{join}^{q_1, q_2}\ b$ reduce in a call-by-name calculus? To reduce terms headed by **join**, one might come up with the following rules:

$$\textsc{Step-JoinLeft}\quad \frac{\vdash a \leadsto a'}{\vdash \mathbf{join}^{q_1, q_2}\ a \leadsto \mathbf{join}^{q_1, q_2}\ a'} \qquad \textsc{Step-JoinBeta}\quad \frac{}{\vdash \mathbf{join}^{q_1, q_2}\ \mathbf{box}_{q_1}\ \mathbf{box}_{q_2}\ a \leadsto \mathbf{box}_{q_1 \cdot q_2}\ a}$$

     Then, to reduce $\mathbf{join}^{q_1, q_2}\ b$, one would first reduce b to a value and thereafter attempt to apply rule Step-JoinBeta. However, notice that such an attempt might not succeed even for well-typed terms. This is so because in a call-by-name calculus, $\mathbf{box}_q\ a'$ is a value, irrespective of whether a' itself is a value or not. As such, the reduction of $\mathbf{join}^{q_1, q_2}\ b$ might just stop at a term like $\mathbf{join}^{q_1, q_2}\ \mathbf{box}_{q_1}\ b'$, where b' is not headed by $\mathbf{box}_{q_2}$. See that $\mathbf{join}^{q_1, q_2}\ \mathbf{box}_{q_1}\ b'$ cannot step

via rule Step-JoinBeta. Given this situation, to maintain type-soundness, one would be forced to declare terms like $\mathbf{join}^{q_1,q_2}\ \mathbf{box}_{q_1}\ b'$ as values, thereby allowing unprincipled values within the calculus.

- Further, the addition of this **join** operator breaks the symmetry of the type system because rule GC-Join is neither an introduction rule nor an elimination rule for $\Box$. Owing to this break in symmetry, there is no principled way to ensure that **join** is the inverse of **fork**, which is already derivable in the type system, as shown in Section 4.2.2. Thus, this **join** operator does not bode well for the equational theory of the system too.

In light of these drawbacks, we conclude that adding an ad hoc **join** operator to enable dependency analysis in a graded-context type system is not the right approach. So, we avoid this approach. In this regard, note that Orchard et al. [2019] add a similar join operator to their graded-context type system but the semantics they use is call-by-value instead of call-by-name. In Section 4.2.1, we pointed out that the graded-context type systems we design account usage with respect to a call-by-name semantics and here we saw that a join operator does not fit well with a call-by-name semantics. As such, the approach of Orchard et al. [2019] cannot be adapted to our setting.

2. Another key aspect of dependency analysis not supported by graded-context type systems pertains to the treatment of functions wrapped under high-security labels. It is desirable that a dependency calculus allows any function to return high-security values, given the function itself is wrapped under a high-security label. Note that this is true of our dependency calculi GMCC_{e} and SDC, as we showed in Sections 2.10.2 and 3.2.1 respectively. But this is not true of graded-context type systems. More precisely, in a graded-context type system parametrized by the lattice $\mathcal{L}_2 = \mathbf{L} \sqsubseteq \mathbf{H}$, the type $\Box^{\mathbf{H}}(\Box^{\mathbf{H}}\mathbf{Bool} \to \mathbf{Bool})$ contains only two distinct terms, $\mathbf{box}_{\mathbf{H}}\ (\lambda x.\mathbf{true})$ and $\mathbf{box}_{\mathbf{H}}\ (\lambda x.\mathbf{false})$. Put differently, the type system cannot derive terms corresponding to $\mathbf{split}^{\mathbf{H}}(\lambda x.\mathbf{merge}^{\mathbf{H}} x)$ and $\mathbf{split}^{\mathbf{H}}(\lambda x.\mathbf{not}\ (\mathbf{merge}^{\mathbf{H}} x))$ of $\text{GMCC}_{\text{e}}(\mathcal{L}_2)$.

   On a closer look, we find that this shortcoming of graded-context type systems stem from the fact that these systems fix the grade to the right of the turnstile in their typing judgments to 1. Had these systems allowed an arbitrary grade in this position, they could have derived terms corresponding to $\mathbf{split}^{\mathbf{H}}(\lambda x.\mathbf{merge}^{\mathbf{H}} x)$ and $\mathbf{split}^{\mathbf{H}}(\lambda x.\mathbf{not}\ (\mathbf{merge}^{\mathbf{H}} x))$. This observation is similar to one we made in Section 2.10.2: $\text{GMCC}(\mathcal{L}_2)$, while being very similar to $\text{GMCC}_{\text{e}}(\mathcal{L}_2)$, also cannot derive terms corresponding to the ones above, simply because the former does not employ grades either to the left or to the right of the turnstile in its typing judgments.

Hence, we see that there are several impediments to dependency analysis in graded-context type systems. Therefore, we shall not attempt to analyze dependencies using graded-context type systems. The other alternative for unifying linearity and dependency analyses, pointed out above, is to find whether the type systems we designed for dependency analyses can be adapted to analyze linearity as well. We explore this alternative next.

5.1.2 Linearity Analysis Through Dependency Calculi

Our dependency calculi from Chapter 3 point towards an alternative way of unifying linearity and dependency analyses. These calculi, in spite of being very similar to graded-context type systems, avoid their above-mentioned limitations, thanks to the form of the typing judgment they employ. But being parametrized over lattices, they cannot analyze linearity. However, note that they can carry out some usage analyses, as we saw in Sections 4.5.1 and 4.8.2. Now the question is, can these calculi be adapted to analyze linearity? Fortunately, the answer to this question is yes. To analyze linearity through these calculi, we first need to modify the parametrizing structure from a lattice $\mathcal{L}$ to a preordered semiring $\mathcal{Q}$. With this modification, the typing judgment would look like:

$$x_1 :^{q_1} A_1, x_2 :^{q_2} A_2, \ldots, x_n :^{q_n} A_n \vdash b :^{q} B \tag{5.1}$$

where $q_i, q \in \mathcal{Q}$. There is, however, a known danger awaiting a calculus that employs this judgment form to analyze usage. Atkey [2018] found that a type system that employs this judgment form and is parametrized by an arbitrary semiring does not admit substitution. Now, an arbitrary semiring may be viewed as a preordered semiring, where the order relation is discrete. So, the above finding applies here and poses a danger to our plan as well. But there is, as it were, a silver lining! We find that though substitution is inadmissible over some preordered semirings, it is in fact admissible over several preordered semirings that interest us, including $\mathcal{Q}_{\text{Lin}}$ and $\mathcal{Q}_{\text{Aff}}$, the preordered semirings for analyzing linear and affine usage. So, we may employ the above judgment form for analyzing linearity.

Thus, we have found a single form of typing judgment that may be employed for both linearity and dependency analyses. In the rest of the chapter, we shall present our Linearity Dependency Calculus, LDC, which employs this judgment form to unify the two analyses. First, we shall show how LDC analyzes linearity and other usages. Then, we shall show how the calculus analyzes dependencies. Finally, we shall show how to carry out a combined analysis using the calculus. For linearity and other usage analyses, we shall parametrize LDC over certain preordered semirings, described in appropriate sections. For dependency analysis, we shall parametrize LDC over an arbitrary lattice. For a combined analysis, we shall parametrize LDC over the cartesian product of the structures used for the individual analyses, as discussed in Section 1.3.1. But we shall start with SLDC, the simply-typed version of the calculus.

5.2 Linearity and Dependency Analyses over Simple Types

5.2.1 Type System for Linearity Analysis

First, we shall analyze exact usage and bounded usage. We shall add unrestricted usage, referred to by ω, to our calculus in Section 5.5. Recall from Section 4.1 that exact and bounded usage are analyzed via $\mathbb{N}_=$ (the semiring of natural numbers with discrete order) and $\mathbb{N}_\geq$ (the semiring of natural numbers with descending natural order) respectively. Now, let $\mathcal{Q}_{\mathbb{N}}$ vary over $\{\mathbb{N}_=, \mathbb{N}_\geq\}$. The grammar of SLDC($\mathcal{Q}_{\mathbb{N}}$) appears in Figure

types, A, B, C	$::= \mathbf{Unit} \mid {}^{r}A \to B \mid {}^{r}A_1 \times A_2 \mid A_1 + A_2$
terms, a, b, c	$::= x \mid \lambda^{r} x : A.b \mid b\ a^{r} \mid \mathbf{unit} \mid \mathbf{let}_{q_0}\ \mathbf{unit} = a\ \mathbf{in}\ b \mid$
	$(a_1{}^{r}, a_2) \mid \mathbf{let}_{q_0}\ (x^{r}, y) = a\ \mathbf{in}\ b \mid \mathbf{inj}_1\ a_1 \mid \mathbf{inj}_2\ a_2 \mid \mathbf{case}_{q_0}\ a\ \mathbf{of}\ x_1.b_1\ ;\ x_2.b_2$
contexts, Γ	$::= \varnothing \mid \Gamma, x :^{q} A$

FIGURE 5.1: Grammar of SLDC($\mathcal{Q}_{\mathbb{N}}$)

$\boxed{\Gamma \vdash a :^{q} A}$ *(Typing)*

SLDC-VAR
$$\frac{}{0 \cdot \Gamma_1, x :^{q} A, 0 \cdot \Gamma_2 \vdash x :^{q} A}$$

SLDC-LAM
$$\frac{\Gamma, x :^{q \cdot r} A \vdash b :^{q} B}{\Gamma \vdash \lambda^{r} x : A.b :^{q} {}^{r}A \to B}$$

SLDC-APP
$$\frac{\Gamma_1 \vdash b :^{q} {}^{r}A \to B \qquad \Gamma_2 \vdash a :^{q \cdot r} A \qquad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor}{\Gamma_1 + \Gamma_2 \vdash b\ a^{r} :^{q} B}$$

SLDC-PAIR
$$\frac{\Gamma_1 \vdash a_1 :^{q \cdot r} A_1 \qquad \Gamma_2 \vdash a_2 :^{q} A_2 \qquad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor}{\Gamma_1 + \Gamma_2 \vdash (a_1{}^{r}, a_2) :^{q} {}^{r}A_1 \times A_2}$$

SLDC-LETPAIR
$$\frac{\Gamma_1 \vdash a :^{q \cdot q_0} {}^{r}A_1 \times A_2 \qquad \Gamma_2, x :^{q \cdot q_0 \cdot r} A_1, y :^{q \cdot q_0} A_2 \vdash b :^{q} B \qquad q_0 <: 1 \qquad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor}{\Gamma_1 + \Gamma_2 \vdash \mathbf{let}_{q_0}\ (x^{r}, y) = a\ \mathbf{in}\ b :^{q} B}$$

SLDC-UNIT
$$\frac{}{0 \cdot \Gamma \vdash \mathbf{unit} :^{q} \mathbf{Unit}}$$

SLDC-LETUNIT
$$\frac{\Gamma_1 \vdash a :^{q \cdot q_0} \mathbf{Unit} \qquad \Gamma_2 \vdash b :^{q} B \qquad q_0 <: 1 \qquad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor}{\Gamma_1 + \Gamma_2 \vdash \mathbf{let}_{q_0}\ \mathbf{unit} = a\ \mathbf{in}\ b :^{q} B}$$

SLDC-INJ1
$$\frac{\Gamma \vdash a_1 :^{q} A_1}{\Gamma \vdash \mathbf{inj}_1\ a_1 :^{q} A_1 + A_2}$$

SLDC-INJ2
$$\frac{\Gamma \vdash a_2 :^{q} A_2}{\Gamma \vdash \mathbf{inj}_2\ a_2 :^{q} A_1 + A_2}$$

SLDC-CASE
$$\frac{\Gamma_1 \vdash a :^{q \cdot q_0} A_1 + A_2 \qquad \Gamma_2, x_1 :^{q \cdot q_0} A_1 \vdash b_1 :^{q} B \qquad \Gamma_2, x_2 :^{q \cdot q_0} A_2 \vdash b_2 :^{q} B \qquad q_0 <: 1 \qquad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor}{\Gamma_1 + \Gamma_2 \vdash \mathbf{case}_{q_0}\ a\ \mathbf{of}\ x_1.b_1\ ;\ x_2.b_2 :^{q} B}$$

SLDC-SUBL
$$\frac{\Gamma' \vdash a :^{q} A \qquad \Gamma <: \Gamma'}{\Gamma \vdash a :^{q} A}$$

SLDC-SUBR
$$\frac{\Gamma \vdash a :^{q} A \qquad q <: q'}{\Gamma \vdash a :^{q'} A}$$

FIGURE 5.2: Typing rules of SLDC($\mathcal{Q}_{\mathbb{N}}$)

5.1. It is similar to that of GLC: there are function types, product types, sum types and a **Unit** type, along with their introduction and elimination forms. SLDC does not have an explicit graded modal type because it can be encoded using product and **Unit** types. Next, like GLC, SLDC uses graded contexts. However, unlike GLC, SLDC employs a typing judgment that allows an arbitrary grade to the right of the turnstile, as motivated in the last section. The typing judgment of SLDC, $x_1 :^{q_1} A_1, x_2 :^{q_2} A_2, \ldots, x_n :^{q_n} A_n \vdash a :^{q} A$, may be intuitively understood as: one can produce q copies of a of type A, using q_i copies of x_i of type A_i, where $i = 1, 2, \ldots, n$. As is the case with GLC, this use is with respect to a call-by-name reduction strategy. With this understanding of the typing judgment, let us look at the type system that appears in Figure 5.2. Note here that the operations and relation on contexts are defined in the same way as in Chapter 4.

Most of the typing rules in Figure 5.2 are as expected, when seen in the light of the typing rules of GLC. However, there are some distinctions worth noting between the two sets of typing rules:

- Owing to the flexibility offered by the new form of typing judgment, rules SLDC-App, SLDC-Pair, and SLDC-Case, unlike their counterparts in GLC, do not have to multiply contexts of premise judgments with appropriate grades.
- The elimination rules SLDC-LetUnit, SLDC-LetPair, and SLDC-Case have a side condition, $q_0 <: 1$. The reason behind the side condition is the same as that of having a similar condition in rule GLC-Case: for reducing any of these elimination forms, we first need to reduce the term a, implying that in any case, we would need the resources for reducing at least one copy of a. Observe that in these rules, we may set q_0 to 1 but allowing any $q_0 <: 1$ makes the rules more flexible. For example, given this flexibility, in rule SLDC-LetPair, b may use $q \cdot q_0$ copies of y in lieu of just q copies of y. Also observe that this extra grade is tracked in the terms through an annotation; this annotation aids in reasoning about usage (as we shall see in Section 5.3.1).
- Rules SLDC-SubL and SLDC-SubR help discard resources to the left and to the right of the turnstile respectively. Rule SLDC-SubL is similar to rule GLC-Sub. However, rule SLDC-SubR does not have a counterpart in GLC because in GLC, the grade to the right of the turnstile is fixed at 1. But this rule is crucial to SLDC because it is required to prove the substitution lemma for the calculus.

We see that there are some differences between the typing rules of GLC and SLDC, but these differences are technical rather than essential in nature. However, we would like to emphasize here that unlike GLC, SLDC cannot be parametrized by an *arbitrary* preordered semiring. This is so because substitution becomes inadmissible when the calculus is parametrized by certain preordered semirings, as alluded to in Section 5.1.2.

Next, we look at a few examples of derivable and non-derivable terms in SLDC($\mathbb{N}_{\geq}$).

$$\varnothing \vdash \lambda^1 x.x :^1 {}^1A \to A \text{ but } \varnothing \nvdash \lambda^0 x.x :^1 {}^0A \to A$$
$$\varnothing \vdash \lambda^1 x.x :^2 {}^1A \to A \text{ but } \varnothing \nvdash \lambda^1 x.(x^2, \mathbf{unit}) :^1 {}^1A \to {}^2A \times \mathbf{Unit}$$
$$\varnothing \vdash \lambda^1 x.\mathbf{let}_1\ (y^1, z) = x\ \mathbf{in}\ y :^1 {}^1({}^1A \times A) \to A \text{ but } \varnothing \nvdash \lambda^1 x.(x^1, x) :^1 {}^1A \to {}^1A \times A$$

On a closer look, we find that the derivable terms use resources fairly whereas the non-derivable ones do not. All the terms in the first column output arguments that are marked with grade 1; the last term in the column additionally discards an argument marked with grade 1. Since grade 1 represents affine usage in $\mathbb{N}_{\geq}$, all these terms are derivable in SLDC($\mathbb{N}_{\geq}$). On the other hand, the terms in the second column 'copy' their arguments. Since copying is not permitted by $\mathbb{N}_{\geq}$, none these terms is derivable in SLDC($\mathbb{N}_{\geq}$). Moreover, owing to the same reason, the types of these terms are essentially uninhabited.

Note that in order to produce 0 copies of any term, we do not need any resources. So, in the 0-world, any annotated type is inhabited, provided its unannotated counterpart is so. For example, we can derive the

following judgments: $\varnothing \vdash \lambda^0 x.x :^0 {}^0A \to A$ and $\varnothing \vdash \lambda^1 x.(x^2, \textbf{unit}) :^0 {}^1A \to {}^2A \times \textbf{Unit}$ and $\varnothing \vdash \lambda^1 x.(x^1, x) :^0 {}^1A \to {}^1A \times A$. But resources do not have any meaning in the 0-world, i.e., the judgment $\Gamma \vdash a :^0 A$ conveys no more information than its corresponding standard λ-calculus counterpart.

Next, we look at the operational semantics and metatheory of SLDC($\mathcal{Q}_{\mathbb{N}}$).

5.2.2 Metatheory of Linearity Analysis

First, we consider some syntactic properties. The calculus satisfies the multiplication lemma stated below. This lemma says that if we need Γ resources to produce q copies of a, then we would need $r_0 \cdot \Gamma$ resources to produce $r_0 \cdot q$ copies of a.

Lemma 5.1 (Multiplication) If $\Gamma \vdash a :^q A$, then $r_0 \cdot \Gamma \vdash a :^{r_0 \cdot q} A$.

The calculus also satisfies a factorization lemma, stated below. This lemma says that if a context Γ produces q copies of a, then Γ can be split into q parts, whereby each part produces 1 copy of a. We need the precondition, $q \neq 0$, since resources don't have any meaning in the 0-world (but they are meaningful in all other worlds, in particular, the 1-world).

Lemma 5.2 (Factorization) If $\Gamma \vdash a :^q A$ and $q \neq 0$, then there exists Γ' such that $\Gamma' \vdash a :^1 A$ and $\Gamma <: q \cdot \Gamma'$.

Using the above two lemmas, we can prove a splitting lemma, stated below. This lemma says that if we have the resources, Γ, to produce $q_1 + q_2$ copies of a, then we can split Γ into two parts, Γ_1 and Γ_2, such that Γ_1 and Γ_2 can produce q_1 and q_2 copies of a respectively. Atkey [2018] showed that the splitting lemma does not hold in a type system that is similar to ours, when parametrized over certain semirings. However, we find that the splitting lemma does hold for SLDC, when parametrized over the preordered semirings $\mathbb{N}_=$ and $\mathbb{N}_\geq$.

Lemma 5.3 (Splitting) If $\Gamma \vdash a :^{q_1 + q_2} A$, then there exists Γ_1 and Γ_2 such that $\Gamma_1 \vdash a :^{q_1} A$ and $\Gamma_2 \vdash a :^{q_2} A$ and $\Gamma = \Gamma_1 + \Gamma_2$.

Together, the multiplication and factorization lemmas show that usage analyses in SLDC do not really need the extra flexibility gained by having an arbitrary grade to the right of the turnstile in typing judgments. This observation is along expected lines because we have seen in Chapter 4 that usage analyses can be carried out, without any issue, even when the grade to the right of the turnstile in typing judgments is kept fixed at 1. However, as discussed in Section 5.1, this extra flexibility is crucial to analyzing dependencies in the calculus. So even though we don't need this extra flexibility for usage analyses, we still support it because it is needed for dependency analyses.

Next, we look at weakening and substitution lemmas. The weakening lemma is similar to that for GLC. The substitution lemma, however, is somewhat different. For substitution, we ensure that the number of available copies of the substitute matches the allowed usage of the variable. Then, to type-check the substituted term, we just add the two contexts; we don't need any context multiplication here.

Lemma 5.4 (Weakening) If $\Gamma_1, \Gamma_2 \vdash a :^q A$, then $\Gamma_1, z :^0 C, \Gamma_2 \vdash a :^q A$.

$\boxed{\vdash a \rightsquigarrow a'}$ *(Step rules)*

$$\frac{\text{STEP-APPBETA}}{\vdash (\lambda^r x : A.b)\ a^r \rightsquigarrow b\{a/x\}} \qquad \frac{\text{STEP-LETPAIRBETA}}{\vdash \mathbf{let}_{q_0}\ (x^r, y) = (a_1{}^r, a_2)\ \mathbf{in}\ b \rightsquigarrow b\{a_1/x\}\{a_2/y\}}$$

FIGURE 5.3: Small-step reduction for SLDC (Excerpt)

Lemma 5.5 (Substitution) If $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash a :^q A$ and $\Gamma \vdash c :^{r_0} C$ where $\lfloor \Gamma_1 \rfloor = \lfloor \Gamma \rfloor$, then $\Gamma_1 + \Gamma, \Gamma_2 \vdash a\{c/z\} :^q A$.

Now, we consider the operational semantics of the language. We pointed out that SLDC analyzes usage with respect to a call-by-name reduction strategy. So, we use a call-by-name small-step semantics for the calculus. All the step rules are standard other than the β-rules that appear in Figure 5.3. These rules ensure that the grade in the introduction form matches with that in the elimination form, similar to the β-rules for the calculi presented in the last two chapters. With this operational semantics, the calculus enjoys the standard type soundness property.

Theorem 5.6 (Preservation) If $\Gamma \vdash a :^q A$ and $\vdash a \rightsquigarrow a'$, then $\Gamma \vdash a' :^q A$.

Theorem 5.7 (Progress) If $\varnothing \vdash a :^q A$, then either a is a value or there exists a' such that $\vdash a \rightsquigarrow a'$.

From Chapter 4, we know that standard substitution-based semantics cannot really model usage. So, in Section 5.3, we shall present a heap-based semantics for SLDC, similar to that for GLC. Then, via soundness with respect to this semantics, we shall show that SLDC analyzes usage correctly. But for now, we move on to dependency analysis using SLDC.

5.2.3 Type System for Dependency Analysis

Let $\mathcal{L} = (L, \sqcap, \sqcup, \top, \bot, \sqsubseteq)$ be an arbitrary lattice. We use ℓ, m to denote the elements of L. Now, if we interpret $+, \cdot, 0, 1$ and $<:$ as $\sqcap, \sqcup, \top, \bot$ and $\sqsubseteq$ respectively, and use ℓ and m in place of q and r, then we have a dependency calculus in the type system presented in Figure 5.2. We present a few selected typing rules with this changed notation in Figure 5.4. For the sake of distinction, we add an extra letter 'D' to the names of the rules.

The type system is now parametrized by $\mathcal{L}$ in lieu of $\mathcal{Q}_{\mathbb{N}}$. We define $\ell \sqcup \Gamma$, $\Gamma_1 \sqcap \Gamma_2$ and $\Gamma \sqsubseteq \Gamma'$ in the same way as their counterparts $q \cdot \Gamma$, $\Gamma_1 + \Gamma_2$ and $\Gamma <: \Gamma'$ respectively. The typing judgment $x_1 :^{\ell_1} A_1, x_2 :^{\ell_2} A_2, \ldots, x_n :^{\ell_n} A_n \vdash b :^{\ell} B$ may be read as: b is observable at ℓ, assuming x_i is observable at ℓ_i, for $i = 1, 2, \ldots, n$. With this reading, let us look at the typing rules in Figure 5.4. The first thing we notice is the similarity of this type system with that of SDC, presented in Section 3.2.1. Most of the typing rules are just variants of the ones for SDC. However, there are some technical differences:

- The key technical difference is that SLDC, unlike SDC, has grades on function and product types. However, SLDC does not have a graded modal type, similar to $S_m\ A$ of SDC. Nevertheless, the graded

$\boxed{\Gamma \vdash a :^{\ell} A}$ *(Selected typing rules)*

$$\text{SLDC-VarD}\quad \frac{}{\top \sqcup \Gamma_1, x :^{\ell} A, \top \sqcup \Gamma_2 \vdash x :^{\ell} A}$$

$$\text{SLDC-LamD}\quad \frac{\Gamma, x :^{\ell \sqcup m} A \vdash b :^{\ell} B}{\Gamma \vdash \lambda^m x : A.b :^{\ell} {}^{m}A \to B}$$

$$\text{SLDC-AppD}\quad \frac{\Gamma_1 \vdash b :^{\ell} {}^{m}A \to B \qquad \Gamma_2 \vdash a :^{\ell \sqcup m} A \qquad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor}{\Gamma_1 \sqcap \Gamma_2 \vdash b\ a^m :^{\ell} B}$$

$$\text{SLDC-PairD}\quad \frac{\Gamma_1 \vdash a_1 :^{\ell \sqcup m} A_1 \qquad \Gamma_2 \vdash a_2 :^{\ell} A_2 \qquad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor}{\Gamma_1 \sqcap \Gamma_2 \vdash ({a_1}^m, a_2) :^{\ell} {}^{m}A_1 \times A_2}$$

$$\text{SLDC-LetPairD}\quad \frac{\Gamma_1 \vdash a :^{\ell \sqcup \ell_0} {}^{m}A_1 \times A_2 \qquad \Gamma_2, x :^{\ell \sqcup \ell_0 \sqcup m} A_1, y :^{\ell \sqcup \ell_0} A_2 \vdash b :^{\ell} B \qquad \ell_0 \sqsubseteq \bot \qquad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor}{\Gamma_1 \sqcap \Gamma_2 \vdash \mathbf{let}_{\ell_0}\ (x^m, y) = a\ \mathbf{in}\ b :^{\ell} B}$$

$$\text{SLDC-SubLD}\quad \frac{\Gamma' \vdash a :^{\ell} A \qquad \Gamma \sqsubseteq \Gamma'}{\Gamma \vdash a :^{\ell} A}$$

$$\text{SLDC-SubRD}\quad \frac{\Gamma \vdash a :^{\ell} A \qquad \ell \sqsubseteq \ell'}{\Gamma \vdash a :^{\ell'} A}$$

FIGURE 5.4: Typing rules of SLDC($\mathcal{L}$) (Excerpt)

modal type $S_m\, A$ of SDC can be encoded as ${}^{m}A \times \mathbf{Unit}$ in SLDC. Conversely, graded function and product types of SLDC, ${}^{m}A \to B$ and ${}^{m}A \times B$, can be encoded as $S_m\, A \to B$ and $S_m\, A \times B$ respectively in SDC.

- Another technical difference is that SLDC has explicit typing rules for relaxing the grades on the contextual assumptions (rule ST-SubLD) and for strengthening the grade of the observer (rule ST-SubRD). On the other hand, SDC does not explicitly include such typing rules but owing to a lenient variable rule SDC-Var, these rules are admissible in the calculus: refer to the narrowing and subsumption lemmas of SDC, lemmas 3.1 and 3.4 respectively.

- Yet another technical difference is that SLDC extensively uses context operations in its typing rules whereas SDC makes no such use. This difference stems from the fact that unlike SDC, SLDC has typing rules where the same contextual assumption appears at different grades in different premise typing judgments (such as rule SLDC-AppD), thereby necessitating the use of context operations in the conclusion judgment. This difference, though seemingly significant, is rather inconsequential from a dependency perspective. To see why, consider rule SLDC-AppD and a variant of this rule, where both the premise judgments use the same graded context:

$$\text{SLDC-AppD}\quad \frac{\Gamma_1 \vdash b :^{\ell} {}^{m}A \to B \qquad \Gamma_2 \vdash a :^{\ell \sqcup m} A \qquad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor}{\Gamma_1 \sqcap \Gamma_2 \vdash b\ a^m :^{\ell} B} \qquad \text{SLDC-AppDV}\quad \frac{\Gamma \vdash b :^{\ell} {}^{m}A \to B \qquad \Gamma \vdash a :^{\ell \sqcup m} A}{\Gamma \vdash b\ a^m :^{\ell} B}$$

Now, replacing rule SLDC-AppD with rule SLDC-AppDV does not make any difference to dependency analysis in the calculus. This is so because: one can derive rule SLDC-AppD using rules SLDC-AppDV and SLDC-SubLD; also, one can go the other way and derive rule SLDC-AppDV using rule SLDC-AppD.

- A presentational difference between SLDC and SDC lies in some of the elimination rules: the elimination rules for pair, unit and sum have a side condition in SLDC. For example, in rule SLDC-LetPairD, we have the side condition, $\ell_0 \sqsubseteq \bot$. Now, since $\bot$ is the bottom element of the lattice, this condition immediately forces ℓ_0 to be $\bot$. So, rule SLDC-LetPairD may be simplified with ℓ_0 set to $\bot$, thereby bringing it closer to rule SDC-LetPair. The same is true of the elimination rules for unit and sum. However, we present rule SLDC-LetPairD without this simplification in order to emphasize its similarity with rule SLDC-LetPair.

Next, we consider a few examples of derivable and non-derivable terms in LDC($\mathcal{L}_\diamond$), where $\mathcal{L}_\diamond$ is the lattice: $\mathbf{L} \sqsubseteq \mathbf{M_1}, \mathbf{M_2} \sqsubseteq \mathbf{H}$ (introduced in Section 1.2.2). Let $\mathbf{Bool} \triangleq \mathbf{Unit} + \mathbf{Unit}$ and let $\mathbf{true} \triangleq \mathbf{inj}_1\ \mathbf{unit}$ and $\mathbf{false} \triangleq \mathbf{inj}_2\ \mathbf{unit}$ and $\mathbf{not}\ b \triangleq \mathbf{case}\ b\ \mathbf{of}\ x_1.\mathbf{false}\ ;\ x_2.\mathbf{true}$. Then,

$$\varnothing \vdash \lambda^{\mathbf{L}} x.x :^{\mathbf{H}}\ {}^{\mathbf{L}}\mathbf{Bool} \to \mathbf{Bool} \text{ but } \varnothing \nvdash \lambda^{\mathbf{H}} x.x :^{\mathbf{L}}\ {}^{\mathbf{H}}\mathbf{Bool} \to \mathbf{Bool}$$
$$\varnothing \vdash \mathbf{lock}^{\mathbf{H}}\ (\lambda^{\mathbf{H}} x.x) :^{\mathbf{L}} S_{\mathbf{H}}\ ({}^{\mathbf{H}}\mathbf{Bool} \to \mathbf{Bool}) \text{ but } \varnothing \nvdash \lambda^{\mathbf{H}} x.\mathbf{not}\ x :^{\mathbf{L}}\ {}^{\mathbf{H}}\mathbf{Bool} \to \mathbf{Bool}$$
$$\varnothing \vdash \lambda^{\mathbf{H}} x.\mathbf{lock}^{\mathbf{M_1}}\ \mathbf{lock}^{\mathbf{M_2}}\ x :^{\mathbf{L}}\ {}^{\mathbf{H}}\mathbf{Bool} \to S_{\mathbf{M_1}}\ S_{\mathbf{M_2}}\ \mathbf{Bool} \text{ but } x :^{\mathbf{H}} \mathbf{Bool} \nvdash x :^{\mathbf{M_1}} \mathbf{Bool}$$

Here, $S_m\ A \triangleq {}^m A \times \mathbf{Unit}$ and $\mathbf{lock}^{\ell}\ a \triangleq (a^{\ell}, \mathbf{unit})$. On a closer look, we find that the non-derivable terms above violate the dependency structure imposed by $\mathcal{L}_\diamond$. The first and the second terms transfer information from $\mathbf{H}$ to $\mathbf{L}$ while the third one does so from $\mathbf{H}$ to $\mathbf{M_1}$. The derivable terms, on the other hand, respect the imposed dependency structure. The first term transfers information from $\mathbf{L}$ to $\mathbf{H}$, the second one from $\mathbf{H}$ to $\mathbf{H}$, while the third one emboxes $\mathbf{H}$ information in an $\mathbf{M_2}$ box nested within an $\mathbf{M_1}$ box. Observe the difference between the first non-derivable term and the second derivable term: the function $\lambda^{\mathbf{H}} x.x$, ill-typed at $\mathbf{L}$, becomes well-typed, when wrapped under a high-security label. We touched upon this aspect of dependency analysis several times in this dissertation: any function should be allowed to return high-security values, given the function itself is wrapped under a high-security label. Like GMCC$_\mathrm{e}$ and SDC, SLDC supports this aspect of dependency analysis, as we see above.

In Section 5.1.1, we saw that graded-context type systems cannot derive a monadic join operator. But SLDC, through its use of graded typing judgments, can derive such an operator. Below, we present the terms that witness the standard join and fork operations in SLDC($\mathcal{L}$). (Here, erased grades are assumed to be $\bot$.) These terms are similar to their counterparts in SDC($\mathcal{L}$), presented in Section 3.2.1. These terms show that the graded modality for dependency analysis in SLDC is both monadic and comonadic in nature, similar to the graded modalities of our dependency calculi, GMCC$_\mathrm{e}$ and SDC.

$$\varnothing \vdash c_1 : S_{\ell_1}\ S_{\ell_2}\ A \to S_{\ell_1 \sqcup \ell_2}\ A \text{ and } \varnothing \vdash c_2 : S_{\ell_1 \sqcup \ell_2}\ A \to S_{\ell_1}\ T_{\ell_2}\ A \text{ where,}$$
$$c_1 := \lambda x.\mathbf{lock}^{\ell_1 \sqcup \ell_2}\ (\mathbf{let}\ ({y_2}^{\ell_2}, _) = (\mathbf{let}\ ({y_1}^{\ell_1}, _) = x\ \mathbf{in}\ y_1)\ \mathbf{in}\ y_2)$$

$$c_2 \coloneqq \lambda x.\mathbf{lock}^{\ell_1}\ \mathbf{lock}^{\ell_2}\ (\mathbf{let}\ (y^{\ell_1 \sqcup \ell_2}, _) = x\ \mathbf{in}\ y)$$

Next, we look at the metatheory of SLDC($\mathcal{L}$).

5.2.4 Metatheory of Dependency Analysis

We consider the dependency counterparts of the properties of SLDC($\mathcal{Q}_{\mathbb{N}}$), presented in Section 5.2.2. Most of these properties are true of SLDC($\mathcal{L}$). SLDC($\mathcal{L}$) satisfies the multiplication lemma. The lemma says that we can always simultaneously upgrade the context and the level at which the derived term is observed. Observe the similarity between this lemma and the multiplication lemma for SDC, lemma 3.3.

Lemma 5.8 (Multiplication) If $\Gamma \vdash a :^{\ell} A$, then $m_0 \sqcup \Gamma \vdash a :^{m_0 \sqcup \ell} A$.

The splitting lemma is also true. Since $\sqcap$ is idempotent, it follows directly from rule SLDC-SubRD.

Lemma 5.9 (Splitting) If $\Gamma \vdash a :^{q_1 \sqcap q_2} A$, then there exists Γ_1 and Γ_2 such that $\Gamma_1 \vdash a :^{q_1} A$ and $\Gamma_2 \vdash a :^{q_2} A$ and $\Gamma = \Gamma_1 \sqcap \Gamma_2$.

The factorization lemma, however, is not true here because if true, it would allow secure information to leak. To see why, consider the following SLDC($\mathcal{L}_\diamond$) judgment: $\varnothing \vdash \lambda^{\mathbf{H}} x.x :^{\mathbf{H}}\ ^{\mathbf{H}}\mathbf{Bool} \to \mathbf{Bool}$. If the factorization lemma were true, from this judgment, we would have: $\varnothing \vdash \lambda^{\mathbf{H}} x.x :^{\mathbf{L}}\ ^{\mathbf{H}}\mathbf{Bool} \to \mathbf{Bool}$, a non-constant function from high-security booleans to low-security booleans, representing a leak of secure information. Note that in SLDC($\mathcal{Q}_{\mathbb{N}}$), the factorization lemma is required for proving the splitting lemma which, in turn, is necessary to show substitution. However, in SLDC($\mathcal{L}$), the splitting lemma holds trivially and does not need any factorization lemma.

Next, we consider weakening and substitution lemmas. The weakening lemma adds an extra assumption at the highest security level to the context. The substitution lemma, on the other hand, substitutes an assumption held at m_0 with a term derived at m_0. Observe that these lemmas are very similar to their SDC counterparts.

Lemma 5.10 (Weakening) If $\Gamma_1, \Gamma_2 \vdash a :^{\ell} A$, then $\Gamma_1, z :^{\top} C, \Gamma_2 \vdash a :^{\ell} A$.

Lemma 5.11 (Substitution) If $\Gamma_1, z :^{m_0} C, \Gamma_2 \vdash a :^{\ell} A$ and $\Gamma \vdash c :^{m_0} C$ where $\lfloor \Gamma_1 \rfloor = \lfloor \Gamma \rfloor$, then $\Gamma_1 \sqcap \Gamma, \Gamma_2 \vdash a\{c/z\} :^{\ell} A$.

Now, SLDC($\mathcal{L}$) can be given the same call-by-name semantics as SLDC($\mathcal{Q}_{\mathbb{N}}$). With respect to this semantics, SLDC($\mathcal{L}$) enjoys the standard type soundness property.

Theorem 5.12 (Preservation) If $\Gamma \vdash a :^{\ell} A$ and $\vdash a \leadsto a'$, then $\Gamma \vdash a' :^{\ell} A$.

Theorem 5.13 (Progress) If $\varnothing \vdash a :^{\ell} A$, then either a is a value or there exists a' such that $\vdash a \leadsto a'$.

Now, as is the case with usage analyses, soundness of dependency analyses too do not follow from the standard type soundness theorems. Recall from Chapter 3 that to prove soundness or noninterference, we had to

design an indexed indistinguishability relation. In essence, the relation identifies parts of terms that remain unobservable at any given dependency level. We showed that the operational semantics of the dependency calculi we designed respects this relation, thereby showing that these calculi satisfy noninterference. We could follow the same strategy here to prove noninterference for SLDC($\mathcal{L}$). However, we do not really need to design an explicit indexed indistinguishability relation for SLDC because such a relation is implicitly provided by the heap semantics, as we shall see in Section 5.3.2. Interestingly, we can employ the same heap semantics to reason about both usage and dependency analyses in SLDC. This heap semantics helps us prove soundness of the two analyses and various other useful properties, pointed out in Section 5.3.3. But before presenting the heap semantics, we shall elaborate on the relation between SLDC($\mathcal{L}$) and SDC($\mathcal{L}$).

5.2.5 SLDC and SDC: A Comparison

In Section 5.2.3, we pointed out the similarities between the typing rules for dependency analysis in SLDC and SDC. In Section 5.2.4, we pointed out the similarities between the metatheoretic properties of dependency analysis in the two calculi. We see that as far as dependency analysis is concerned, the two calculi are very similar. In fact, we can go further and show that the two calculi are equivalent with regard to dependency analysis. Since this equivalence can be established in a straightforward manner, we present the details only in the appendices.

Lemma 5.14 (Equivalence) With regard to dependency analysis, SLDC is equivalent to SDC.

This equivalence shows that SLDC, like SDC, is a general dependency calculus. However, note that unlike SLDC, SDC cannot analyze linear usage because SDC can only be parametrized by lattices and not by preordered semirings like $\mathbb{N}_=$ and $\mathbb{N}_\geq$ (which cannot be viewed as lattices). So in substance, SLDC is an extension of SDC with usage analyses.

Next, we show soundness of usage and dependency analyses in SLDC via a heap semantics for the calculus.

5.3 Heap Semantics for SLDC

In this section, we present a heap semantics for SLDC, similar to the one for GLC, presented in Section 4.3. Further, we employ the same heap semantics for analyzing both usages and dependencies in SLDC. This is the main difference between the heap semantics in this chapter and the one in Chapter 4, which is employed for usage analysis only. If we put this difference aside, both the semantics are essentially the same.

Recall that heap semantics shows how a term reduces in a heap that assigns values to the free variables of the term. Heaps are ordered lists of variable and term pairs, where the terms may be seen as the definitions of the corresponding variables. To every variable and term pair in a heap, we assign a weight, which may be either a resource label, $q \in \mathcal{Q}_\mathbb{N}$, or a dependency label, $\ell \in \mathcal{L}$. A heap where every variable and term pair has a weight associated with it is referred to as a weighted heap. We assume our weighted heaps satisfy

uniqueness, i.e., no variable is defined twice, and acyclicity, i.e., the definition of a variable does not refer to itself or to other variables appearing subsequently in the heap. We model reduction as an interaction between a term and a weighted heap that defines the free variables of the term.

5.3.1 Reduction Relation

The heap-based reduction rules appear in Figure 5.5. At the outset, note that we use the operators of preordered semiring to present these reduction rules and the lemmas that follow in Section 5.3.2. To read them from a dependency perspective, the reader needs to interpret these operators in terms of their lattice counterparts, i.e., $+, \cdot, 0, 1$ and $<:$ as $\sqcap, \sqcup, \top, \bot$ and $\sqsubseteq$ respectively, as pointed out in Section 5.2.3. Now, with regard to the reduction relation, $[H]a \Longrightarrow^{q}_{S} [H']a'$, there are a few things to note :

- This reduction relation is a simplified version of the one we saw in Chapter 4. We have simplified the relation for a cleaner presentation. Note that owing to this simplification, we cannot state certain lemmas like conservation (lemma 4.7), count balance (lemma 4.9), etc. But this limitation does not pose any problem with regard to stating and proving the soundness theorem. The extra arguments in the reduction relation in Chapter 4 were meant to help us better understand the way heap-based reductions happen. Since we have gained a good understanding of heap-based reductions, we can now do away with these arguments.
- From a usage perspective, the judgment above may be read as: q copies of a use resources from heap H to produce q copies of a', with H' being the left-over resources. The heap regarded as a memory, resource labels on assignments correspond to the maximum allowed usages of the respective data items, where usage of a data item is the number of times it may be looked up during reduction. This reading of the heap reduction relation is the same as in Chapter 4.
- From a dependency perspective, the judgment above may be read as: under the dependency constraints imposed by H, the term a steps to a' at level q, with H' being the updated constraints. The heap regarded as a memory, dependency labels on assignments correspond to the minimum clearances necessary for accessing the respective data items, whereas the label on the judgment corresponds to the security clearance of the user.
- As in Chapter 4, S here denotes a support set of variables that must be avoided while choosing fresh names.

Now, we consider the rules presented in Figure 5.5. The most interesting of these rules is rule HeapLD-Var. There are a few things to note with regard to this rule:

- From a usage perspective, this rule may be read as: to step q copies of x, we need to look-up the value of x for q times, thereby using up q resources. This reading is similar to that of rule Heap-Var from Chapter 4.

Heap, $H ::= \varnothing \mid H, x \overset{q}{\mapsto} a$

$\boxed{[H]a \Longrightarrow^{q}_{S} [H']a'}$ *(Selected heap step rules)*

$$\textsc{HeapLD-Var}\quad \frac{}{[H_1, x \overset{r+q}{\mapsto} a, H_2]x \Longrightarrow^{q}_{S} [H_1, x \overset{r}{\mapsto} a, H_2]a}$$

$$\textsc{HeapLD-SubR}\quad \frac{[H]a \Longrightarrow^{q_1}_{S} [H']a' \qquad q_1 <: q_2}{[H]a \Longrightarrow^{q_2}_{S} [H']a'}$$

$$\textsc{HeapLD-AppL}\quad \frac{[H]b \Longrightarrow^{q}_{S \cup \text{fv}\, a} [H']b'}{[H]b\ a^r \Longrightarrow^{q}_{S} [H']b'\ a^r}$$

$$\textsc{HeapLD-AppBeta}\quad \frac{y\ \textit{fresh} \qquad b' = b\{y/x\}}{[H](\lambda^r x : A.b)\ a^r \Longrightarrow^{q}_{S} [H, y \overset{q \cdot r}{\mapsto} a]b'}$$

$$\textsc{HeapLD-LetPairBeta}\quad \frac{\begin{array}{c} x'\ \textit{fresh} \\ y'\ \textit{fresh} \qquad b' = b\{x'/x\}\{y'/y\} \end{array}}{[H]\mathbf{let}_{q_0}\ (x^r, y) = ({a_1}^r, a_2)\ \mathbf{in}\ b \Longrightarrow^{q}_{S} [H, x' \overset{q \cdot q_0 \cdot r}{\mapsto} a_1, y' \overset{q \cdot q_0}{\mapsto} a_2]b'}$$

FIGURE 5.5: Heap semantics for SLDC (Excerpt)

- From a dependency perspective, this rule may be read as: the data pointed to by x, requiring a minimum clearance of $r \sqcap q$, can be observed at q. This is so because $r \sqcap q \sqsubseteq q$. Note that the minimum clearance of the data gets updated to r in the new heap. This update, though unnecessary from a dependency perspective, is harmless because $r \sqcap q \sqsubseteq r$, implying that the minimum clearance of the data would not be any more lenient in the new heap, under any circumstances. Ensuring that rules do not introduce leniency on the minimum clearance of data stipulated by the starting heap is important for enforcing dependency constraints in the semantics. Conversely, one would not want the rules, especially the variable rule, to introduce strictness on the minimum clearance either. Essentially, one would want the minimum clearance to remain the same. Observe that in this rule, the minimum clearance can, in fact, remain the same because one may set $r := r_0 \sqcap q$ (for some r_0), in which case, $r \sqcap q = r$.

- Unlike rule HEAP-VAR from Chapter 4, this rule does not have any precondition. Recall that rule HEAP-VAR has the precondition, $q <: 1$. We added this precondition to rule HEAP-VAR because we were interested only in reducing one or more copies of x. This interest was prompted by the form of the typing judgments in Chapter 4, which fix the grade to the right of the turnstile to 1. But given that the typing judgments in this chapter allow the grade to the right of the turnstile to vary, we cannot be interested solely in reducing one or more copies of x. More specifically, we should allow reductions involving any number of copies of a term. As such, we do not restrict q in any way in rule HEAPLD-VAR.

Next, we look at the rule for allowing subusage: rule HEAPLD-SUBR. From a usage perspective, this rule is essentially the same as rule HEAP-SUBR from Chapter 4. From a dependency perspective, this rule is there to enable subsumption. Note here that in the above reduction relation, we do not have any rule corresponding to HEAP-SUBL from Chapter 4. We do not include such a rule here because it is not necessary. Rule HEAP-SUBL is necessary in Chapter 4 because the calculi in that chapter are parametrized by an arbitrary preordered semiring. However, the calculi in this chapter are parametrized by specific preordered

semirings and lattices. These parametrizing structures all satisfy a special condition: if $q_1 <: q_2$, then there exists q_0 such that $q_1 = q_0 + q_2$. This condition holds for $\mathbb{N}_=$, $\mathbb{N}_\geq$ and any lattice $\mathcal{L}$. Moreover, this condition also holds for the preordered semirings we use in this chapter later, and listed out in Section 5.5. But note that this condition does not hold for an arbitrary preordered semiring, for example, the preordered semiring, $\{0, 1\}$, with $1 + 1 = 1$ and $0 <: 1$. In sum, since the algebraic structures in this chapter satisfy the special condition mentioned above, a rule like HeapLD-SubL becomes superfluous here.

Moving ahead, the left step rules for application and let-pair are as one would expect and as such, are elided. But note the multiplication by q_0 in the β-rule for pairs, shown in Figure 5.5. We didn't require this multiplication in the corresponding rule from Chapter 4 because q_0 is implicitly 1 there. Now that we have seen some of the step rules, let us consider some reductions that go through and some that don't.

$$\text{In SLDC}(\mathbb{N}_=),\ [x \overset{1}{\mapsto} \mathbf{true}]x \Longrightarrow^1_S [x \overset{0}{\mapsto} \mathbf{true}]\mathbf{true} \text{ but not } [x \overset{0}{\mapsto} \mathbf{true}]x \Longrightarrow^1_S [x \overset{0}{\mapsto} \mathbf{true}]\mathbf{true}$$
$$\text{In SLDC}(\mathbb{N}_\geq),\ [x \overset{2}{\mapsto} \mathbf{true}]x \Longrightarrow^1_S [x \overset{0}{\mapsto} \mathbf{true}]\mathbf{true} \text{ but not } [x \overset{1}{\mapsto} \mathbf{true}]x \Longrightarrow^2_S [x \overset{0}{\mapsto} \mathbf{true}]\mathbf{true}$$
$$\text{In SLDC}(\mathcal{L}_2),\ [x \overset{\mathbf{L}}{\mapsto} \mathbf{true}]x \Longrightarrow^{\mathbf{H}}_S [x \overset{\mathbf{L}}{\mapsto} \mathbf{true}]\mathbf{true} \text{ but not } [x \overset{\mathbf{H}}{\mapsto} \mathbf{true}]x \Longrightarrow^{\mathbf{L}}_S [x \overset{\mathbf{H}}{\mapsto} \mathbf{true}]\mathbf{true}$$

5.3.2 Fair Enforcement of Usage and Dependency Constraints

Looking at the reduction rules in Figure 5.5, we find that they enforce usage and dependency constraints fairly. The only rule that allows usage of resources is rule HeapLD-Var. This rule ensures that a look-up goes through only if the starting heap can provide adequate resources. The rule also ensures that the necessary resources are taken away from the heap after a successful look-up. Similarly, the only rule that allows information to flow from heap to program is again rule HeapLD-Var. This rule ensures that information flows only to users who have the necessary clearance.

The following lemmas formalize the arguments presented above. The first lemma says that a definition that is not usable/observable at some point during reduction does not become usable/observable at a later point. The second lemma says that an unusable/unobservable definition does not play any role in reduction. Note that with regard to usage analysis, if $\neg(\exists q_0, r <: q_0 + q)$, then a resource appearing at grade r is not usable at world q; similarly with regard to dependency analysis, if $\neg(\exists q_0, r <: q_0 + q)$, or equivalently $\neg(r \sqsubseteq q)$, then data appearing at grade r is not observable at world q. (Here, $|H|$ denotes the length of H.)

Lemma 5.15 (Unchanged) If $[H_1, x \overset{r}{\mapsto} a, H_2]c \Longrightarrow^q_S [H'_1, x \overset{r'}{\mapsto} a, H'_2]c'$ (where $|H_1| = |H'_1|$) and $\neg(\exists q_0, r <: q_0 + q)$, then $r' = r$.

Lemma 5.16 (Irrelevant) If $[H_1, x \overset{r}{\mapsto} a, H_2]c \Longrightarrow^q_{S \cup \mathrm{fv}\, b} [H'_1, x \overset{r'}{\mapsto} a, H'_2]c'$ (where $|H_1| = |H'_1|$) and $\neg(\exists q_0, r <: q_0 + q)$, then $[H_1, x \overset{r}{\mapsto} b, H_2]c \Longrightarrow^q_{S \cup \mathrm{fv}\, a} [H'_1, x \overset{r'}{\mapsto} b, H'_2]c'$.

There are a few points to note with regard to the above lemmas:

- First, observe how the same lemmas apply to both usage and dependency analyses. This similarity stems from the fact that unusable data is treated in usage analysis in the same way as unobservable data in dependency analysis.

- Second, observe how similar the above lemmas are to lemmas 4.11 and 4.12 respectively. In fact, in the statements above, if we set q and r to 1 and 0 respectively, they become almost identical to those of lemmas 4.11 and 4.12. However, we don't specify q and r as such because unlike the type systems in Chapter 4, the type systems in this chapter do not employ a fixed reference (the grade to the right of the turnstile in typing judgments can vary here). Also note that in the lemmas above, we do not need to explicitly ensure the parametrizing preordered semiring is zero-unusable. This is so because all the preordered semirings used to parametrize SLDC are zero-unusable.

- Next, the relation $[H]a \Longrightarrow_S^{\ell} [H']a'$ is designed to formalize reductions from the perspective of an observer at level ℓ. Now as per the second lemma above, such an observer cannot distinguish between data items held at level m, whenever $\neg(m \sqsubseteq \ell)$. So, in effect, for a given ℓ, the relation $[H]a \Longrightarrow_S^{\ell} [H']a'$ models reductions that implicitly homogenize all information held at levels m for which $\neg(m \sqsubseteq \ell)$. Then, to prove noninterference for the calculus, we just need to show soundness with respect to this reduction relation. Recall that in Chapter 3, we proved noninterference by showing that reductions respect the indexed indistinguishability relation, $\sim_{\ell}$, which also homogenizes all information held at levels m for which $\neg(m \sqsubseteq \ell)$. So with regard to proving noninterference, we are essentially following in the footsteps of Chapter 3.

Next, we present the soundness theorem, which in conjunction with the above lemmas, will help us show noninterference and correctness of usage accounting in SLDC.

5.3.3 Soundness With Respect To Heap Semantics

The key idea behind usage analysis through heap semantics is that, if a heap contains the right amount of resources to evaluate some number of copies of a term, as judged by the type system, then the evaluation of that many number of copies of the term in that heap does not get stuck. Since the heap-based reduction rules enforce fairness of resource usage, this would mean that the type system accounts resource usage correctly. The key idea behind dependency analysis through heap semantics is similar. If a heap sets the right access permissions for a user, as judged by the type system, then the evaluation, in that heap, of any program observable to that user does not get stuck. Since the heap-based reduction rules enforce dependencies appropriately, this would mean that the type system analyzes dependencies correctly.

The compatibility relation, $H \Vdash \Gamma$, presented below, formalizes the idea that the heap H contains the right amount of resources or has set the right access permissions for evaluating any term type-checked in context Γ. Observe that this relation is similar to the compatibility relation from Chapter 4. A technical difference between the two relations is that for the one presented below, we don't need to use context multiplication because we can directly derive a term at grade q.

$\boxed{H \Vdash \Gamma}$ *(Compatibility)*

$$\text{Compat-Empty}\quad \frac{}{\varnothing \Vdash \varnothing}$$

$$\text{Compat-Cons}\quad \frac{H \Vdash \Gamma_1 + \Gamma_2 \qquad \Gamma_2 \vdash a :^q A}{H, x \overset{q}{\mapsto} a \Vdash \Gamma_1, x :^q A}$$

Next, we present the soundness theorem. The soundness theorem says that if a heap H is compatible with a context Γ, then the evaluation, starting with heap H, of a term type-checked in context Γ, does not get stuck.

Theorem 5.17 (Soundness) If $H \Vdash \Gamma$ and $\Gamma \vdash a :^q A$, then either a is a value or there exists H', Γ', a' such that:

- $[H]a \Longrightarrow^q_S [H']a'$
- $H' \Vdash \Gamma'$
- $\Gamma' \vdash a' :^q A$

Note that this soundness theorem applies to both usage and dependency analyses. Next, note the similarity between this theorem and the soundness theorem for GLC, theorem 4.10. If in the statement above, we set q to 1, then it becomes almost identical to that of theorem 4.10. However, the theorem above does not have the fourth clause because in this chapter, we don't track the arguments necessary for stating that clause.

Next, we present some corollaries of the soundness theorem.

Corollary 5.18 (No Usage) In SLDC($\mathcal{Q}_{\mathbb{N}}$): Let $\varnothing \vdash f :^1 {}^0 A \to \mathbf{Bool}$. Then, for any $\varnothing \vdash a_1 :^0 A$ and $\varnothing \vdash a_2 :^0 A$, the terms $f\ {a_1}^0$ and $f\ {a_2}^0$ have the same operational behavior, i.e., either both the terms diverge or both reduce to the same value.

The above corollary also holds in LDC($\mathcal{L}$) with 0 and 1 replaced by **H** and **L** respectively. In LDC($\mathcal{L}$), this corollary shows noninterference of high-security inputs in low-security outputs. More generally, we can show that:

Corollary 5.19 (Noninterference) In SLDC($\mathcal{L}$): Let $\varnothing \vdash f :^\ell {}^m A \to \mathbf{Bool}$ such that $\neg(m \sqsubseteq \ell)$. Then, for any $\varnothing \vdash a_1 :^m A$ and $\varnothing \vdash a_2 :^m A$, the terms $f\ {a_1}^m$ and $f\ {a_2}^m$ have the same operational behavior.

The similarity between the above two corollaries shows once again how no usage mirrors noninterference and vice-versa. Moving ahead, apart from no usage and noninterference, we can also employ the soundness theorem to reason about other usages like affine usage, as shown in the corollary below. Note that when the parametrizing preordered semiring is $\mathbb{N}_\geq$, 1 represents affine and not strict linear usage.

Corollary 5.20 (Affine Usage) In SLDC($\mathbb{N}_\geq$): Let $\varnothing \vdash f :^1 {}^1 A \to \mathbf{Bool}$. Then, for any $\varnothing \vdash a :^1 A$, the term $f\ a^1$ uses a at most once during reduction.

Now that we have seen the syntax and semantics of SLDC, we generalize the calculus to its PTS version.

5.4 Linearity and Dependency Analyses in Pure Type Systems

In this section, we extend SLDC to LDC, a unified calculus for linearity and dependency analyses in Pure Type Systems. Recall that a Pure Type System is characterized by a triple, $(\mathcal{S}, \mathcal{A}, \mathcal{R})$, where $\mathcal{S}$ is a set of sorts, $\mathcal{A}$ is a set of axioms and $\mathcal{R}$ is a ternary relation on sorts. Now, the grammar of LDC is similar to that of SLDC. As far as types and terms are concerned, we just need to add to them the sorts in $\mathcal{S}$ and generalize ${}^{r}A \to B$ and ${}^{r}A \times B$ to $\Pi x :^{r} A.B$ and $\Sigma x :^{r} A.B$ respectively. As far as usage and dependency analyses are concerned, we use the same parametrizing structures, i.e., $\mathcal{Q}_{\mathbb{N}}$ and $\mathcal{L}$.

However, there is an important distinction between SLDC and LDC. In LDC, we need to extend the analyses to dependent types. In Chapters 3 and 4, we have already seen such extensions. In Chapter 3, DDC$^{\top}$ extends dependency analysis in simply-typed SDC to dependent types. In Chapter 4, GRAD extends usage analysis in simply-typed GLC to dependent types. We follow these extensions as we extend usage and dependency analyses in simply-typed SLDC to dependently-typed LDC. For usage analysis in LDC, we follow the principle of GRAD. For dependency analysis in LDC, we follow the principle of DDC$^{\top}$. We can simultaneously follow the principles of GRAD and DDC$^{\top}$ because both these calculi essentially employ the same principle, when it comes to extending their respective analyses to dependent types. That principle, pointed out in Section 4.8.1, may be summed up as: analyze usage/dependencies in types and terms independent of one another. To elaborate: GRAD treats usage in types and terms uniformly but counts usage only for the expression that appears as a term in a given typing judgment; similarly, DDC$^{\top}$ treats dependencies in types and terms uniformly but tracks dependencies only for the expression that appears as a term in a given typing judgment. We shall follow this very principle while analyzing usages and dependencies in LDC.

Now, we look at the type system of LDC.

5.4.1 Type System of LDC

The typing and equality rules of LDC appear in Figures 5.6 and 5.7 respectively. There are a few points to note with regard to these rules:

- First and foremost, these rules apply to both usage and dependency analyses. To read the rules from a dependency perspective, the reader needs to interpret the operators $+, \cdot, 0, 1$ and $<:$ as $\sqcap, \sqcup, \top, \bot$ and $\sqsubseteq$ respectively, as pointed out in Section 5.2.3.
- Given a judgment $\Gamma \vdash a :^{0} A$, where $\overline{\Gamma}$ is a 0-vector, we use the judgment $\lfloor \Gamma \rfloor \vdash_{0} a : A$ as its shorthand (with the dependency analogue denoted as $\lfloor \Gamma \rfloor \vdash_{\top} a : A$). The judgment $\Gamma \vdash a :^{0} A$ is essentially a standard PTS-judgment because in world 0 (and $\top$), usage (and dependency) constraints are not meaningful. Another way to look at this judgment is that it 'switches off' usage (and dependency) analysis for the term under consideration.

$\boxed{\Gamma \vdash a :^{q} A}$ *(Typing)*

LDC-Axiom
$$\frac{c : s \in \mathcal{A}}{\varnothing \vdash c :^{q} s}$$

LDC-Var
$$\frac{\Delta \vdash_0 A : s \qquad \lfloor \Gamma \rfloor = \Delta}{0 \cdot \Gamma, x :^{q} A \vdash x :^{q} A}$$

LDC-Weak
$$\frac{\begin{array}{c}\Gamma \vdash a :^{q} A \\ \Delta \vdash_0 B : s \qquad \lfloor \Gamma \rfloor = \Delta\end{array}}{\Gamma, y :^{0} B \vdash a :^{q} A}$$

LDC-Pi
$$\frac{\begin{array}{c}\Gamma_1 \vdash A :^{q} s_1 \\ \Gamma_2, x :^{r_0} A \vdash B :^{q} s_2 \\ \mathcal{R}(s_1, s_2, s_3) \qquad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor\end{array}}{\Gamma_1 + \Gamma_2 \vdash \Pi x :^{r} A.B :^{q} s_3}$$

LDC-Lam
$$\frac{\begin{array}{c}\Gamma, x :^{q \cdot r} A \vdash b :^{q} B \\ \Delta \vdash_0 \Pi x :^{r} A.B : s \qquad \lfloor \Gamma \rfloor = \Delta\end{array}}{\Gamma \vdash \lambda^{r} x : A.b :^{q} \Pi x :^{r} A.B}$$

LDC-App
$$\frac{\begin{array}{c}\Gamma_1 \vdash b :^{q} \Pi x :^{r} A.B \\ \Gamma_2 \vdash a :^{q \cdot r} A \qquad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor\end{array}}{\Gamma_1 + \Gamma_2 \vdash b\ a^{r} :^{q} B\{a/x\}}$$

LDC-Conv
$$\frac{\begin{array}{c}\Gamma \vdash a :^{q} A \qquad \Delta \vdash_0 B : s \\ A =_\beta B \qquad \lfloor \Gamma \rfloor = \Delta\end{array}}{\Gamma \vdash a :^{q} B}$$

LDC-SubL
$$\frac{\Gamma' \vdash a :^{q} A \qquad \Gamma <: \Gamma'}{\Gamma \vdash a :^{q} A}$$

LDC-SubR
$$\frac{\Gamma \vdash a :^{q} A \qquad q <: q'}{\Gamma \vdash a :^{q'} A}$$

LDC-Sigma
$$\frac{\begin{array}{c}\Gamma_1 \vdash A_1 :^{q} s_1 \\ \Gamma_2, x :^{r_0} A_1 \vdash A_2 :^{q} s_2 \\ \mathcal{R}(s_1, s_2, s_3) \qquad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor\end{array}}{\Gamma_1 + \Gamma_2 \vdash \Sigma x :^{r} A_1.A_2 :^{q} s_3}$$

LDC-Pair
$$\frac{\begin{array}{c}\Delta \vdash_0 \Sigma x :^{r} A_1.A_2 : s \\ \Gamma_1 \vdash a_1 :^{q \cdot r} A_1 \\ \Gamma_2 \vdash a_2 :^{q} A_2\{a_1/x\} \\ \lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor = \Delta\end{array}}{\Gamma_1 + \Gamma_2 \vdash ({a_1}^{r}, a_2) :^{q} \Sigma x :^{r} A_1.A_2}$$

LDC-LetPair
$$\frac{\begin{array}{c}\Delta, z : \Sigma x :^{r} A_1.A_2 \vdash_0 B : s \\ \Gamma_1 \vdash a :^{q \cdot q_0} \Sigma x :^{r} A_1.A_2 \\ \Gamma_2, x :^{q \cdot q_0 \cdot r} A_1, y :^{q \cdot q_0} A_2 \vdash b :^{q} B\{(x^{r}, y)/z\} \\ q_0 <: 1 \qquad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor = \Delta\end{array}}{\Gamma_1 + \Gamma_2 \vdash \mathbf{let}_{q_0}\ (x^{r}, y) = a\ \mathbf{in}\ b :^{q} B\{a/z\}}$$

LDC-Sum
$$\frac{\begin{array}{c}\Gamma_1 \vdash A_1 :^{q} s \\ \Gamma_2 \vdash A_2 :^{q} s \qquad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor\end{array}}{\Gamma_1 + \Gamma_2 \vdash A_1 + A_2 :^{q} s}$$

LDC-Inj
$$\frac{\begin{array}{c}\Delta \vdash_0 A_1 + A_2 : s \\ \Gamma \vdash a_i :^{q} A_i \qquad \lfloor \Gamma \rfloor = \Delta\end{array}}{\Gamma \vdash \mathbf{inj}_i\ a_i :^{q} A_1 + A_2}$$

LDC-Case
$$\frac{\begin{array}{c}\Delta, z : A_1 + A_2 \vdash_0 B : s \\ \Gamma_1 \vdash a :^{q \cdot q_0} A_1 + A_2 \\ \Gamma_2, x_1 :^{q \cdot q_0} A_1 \vdash b_1 :^{q} B\{\mathbf{inj}_1\ x_1/z\} \\ \Gamma_2, x_2 :^{q \cdot q_0} A_2 \vdash b_2 :^{q} B\{\mathbf{inj}_2\ x_2/z\} \\ q_0 <: 1 \qquad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor = \Delta\end{array}}{\Gamma_1 + \Gamma_2 \vdash \mathbf{case}_{q_0}\ a\ \mathbf{of}\ x_1.b_1\ ;\ x_2.b_2 :^{q} B\{a/z\}}$$

FIGURE 5.6: Typing rules of LDC

$\boxed{A =_\beta B}$ *(Definitional equality)*

Eq-PiCong
$$\frac{A_1 =_\beta A_2 \qquad B_1 =_\beta B_2}{\Pi x :^{r} A_1.B_1 =_\beta \Pi x :^{r} A_2.B_2}$$

Eq-LamCong
$$\frac{A_1 =_\beta A_2 \qquad b_1 =_\beta b_2}{\lambda^{r} x : A_1.b_1 =_\beta \lambda^{r} x : A_2.b_2}$$

Eq-AppCong
$$\frac{b_1 =_\beta b_2 \qquad a_1 =_\beta a_2}{b_1\ {a_1}^{r} =_\beta b_2\ {a_2}^{r}}$$

FIGURE 5.7: Equality rules of LDC (Excerpt)

- The judgment $\Delta \vdash_0 A : s$, where Δ denotes ungraded contexts, is used to check well-formedness of types in several rules, for example, rule LDC-Lam, rule LDC-Pair, etc. We use this judgment in these rules because it 'switches off' the analysis in types while we are considering the analysis in terms. We need such 'switch offs' to help us analyze usages (and dependencies) in terms and types independent of one another, in accordance with the adopted principle pointed out earlier. With regard to the use of this judgment, LDC is similar to $\text{DDC}^{\top}$, which also checks well-formedness of types at $\top$, but is different from GraD, which does not have the judgment form $\Gamma \vdash a :^0 A$. This difference, however, is not very significant because GraD realizes similar 'switch-offs' by just zeroing-out the usage requirements of types while accounting the usage requirements of terms.

- In LDC, the annotation on the bound variable in a Π-type indicates how that variable is constrained in the body of a function having the Π-type. This constraint is unrelated to how the variable is constrained in the body of the type itself, as we can see in rule LDC-Pi. Recall that we make the same design choices in GraD and $\text{DDC}^{\top}$, as spelled out in Section 4.8.1.

- Like GraD and $\text{DDC}^{\top}$, LDC uses β-equivalence, under call-by-name reduction, for definitional equality. Some of the congruence rules for the equality relation appear in Figure 5.7. They are fairly standard. However, as in case of GraD and $\text{DDC}^{\top}$, these rules need to ensure that the grade annotations on the terms being equated match up.

- Notwithstanding the presentational differences, the type system of $\text{LDC}(\mathcal{L})$ is equivalent to that of $\text{DDC}^{\top}(\mathcal{L})$. We shall present a formal proof of this equivalence in Section 5.6.

- $\text{LDC}(\mathcal{L})$, being equivalent to $\text{DDC}^{\top}(\mathcal{L})$, can support strong Σ-types. $\text{LDC}(\mathcal{Q}_{\mathbb{N}})$ can also support strong Σ-types, but in the 0 world, meaning, given the judgment $\Delta \vdash_0 a : \Sigma x :^r A_1.A_2$, one can define the first and second projections using rule LDC-LetPair.

Now, we look at a few examples of derivable and non-derivable terms in LDC.

$$\text{In LDC}(\mathbb{N}_=),\ \varnothing \vdash \lambda^0 x.\lambda^1 y.y :^1 \Pi x :^0 s.\Pi y :^1 x.x \text{ but } \varnothing \nvdash \lambda^0 x.\lambda^0 y.y :^1 \Pi x :^0 s.\Pi y :^0 x.x$$
$$\text{In LDC}(\mathbb{N}_\geq),\ \varnothing \vdash \lambda^0 x.\lambda^2 y.y :^2 \Pi x :^0 s.\Pi y :^2 x.x \text{ but } \varnothing \nvdash \lambda^0 x.\lambda^0 y.y :^2 \Pi x :^0 s.\Pi y :^0 x.x$$
$$\text{In LDC}(\mathcal{L}_2),\ \varnothing \vdash \lambda^\top x.\lambda^\bot y.y :^\bot \Pi x :^\top s.\Pi y :^\bot x.x \text{ but } \varnothing \nvdash \lambda^\top x.\lambda^\top y.y :^\bot \Pi x :^\top s.\Pi y :^\top x.x$$

Observe that all the examples above pertain to the polymorphic identity function, annotated in different ways. This function returns its second argument (while ignoring the first one). The terms in the second column have annotations that do not allow the second argument to be returned. As such, these terms are not derivable. On the other hand, the terms in the first column are derivable because the annotations allow the second argument to be returned. In this regard, note the bounded usage of the second argument in the term derived in $\text{LDC}(\mathbb{N}_\geq)$.

Next, we look at the metatheory of LDC.

5.4.2 Metatheory of LDC

We use a standard call-by-name small-step semantics for LDC. The reduction relation for the calculus is the same as that for SLDC. In this regard, LDC is similar to GRAD and $\mathrm{DDC}^{\top}$, both of which are also given a call-by-name semantics. Next, LDC satisfies the PTS analogues of all the lemmas and theorems satisfied by SLDC, pertaining to both usage and dependency analyses, and presented in Sections 5.2.2 and 5.2.4 respectively. Below, we state these lemmas and theorems for LDC without further explanation. Our focus here is on usage analysis. We do not explicitly state the lemmas and theorems pertaining to dependency analysis because when parametrized over lattices, LDC is equivalent to $\mathrm{DDC}^{\top}$, whose metatheoretic properties have already been established in Chapter 3.

Lemma 5.21 (Multiplication) If $\Gamma \vdash a :^{q} A$, then $r_0 \cdot \Gamma \vdash a :^{r_0 \cdot q} A$.

Lemma 5.22 (Factorization) If $\Gamma \vdash a :^{q} A$ and $q \neq 0$, then there exists Γ' such that $\Gamma' \vdash a :^{1} A$ and $\Gamma <: q \cdot \Gamma'$.

Lemma 5.23 (Splitting) If $\Gamma \vdash a :^{q_1 + q_2} A$, then there exists Γ_1 and Γ_2 such that $\Gamma_1 \vdash a :^{q_1} A$ and $\Gamma_2 \vdash a :^{q_2} A$ and $\Gamma = \Gamma_1 + \Gamma_2$.

Lemma 5.24 (Weakening) If $\Gamma_1, \Gamma_2 \vdash a :^{q} A$ and $\lfloor \Gamma_1 \rfloor \vdash_0 C : s$, then $\Gamma_1, z :^{0} C, \Gamma_2 \vdash a :^{q} A$.

Lemma 5.25 (Substitution) If $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash a :^{q} A$ and $\Gamma \vdash c :^{r_0} C$ where $\lfloor \Gamma_1 \rfloor = \lfloor \Gamma \rfloor$, then $\Gamma_1 + \Gamma, \Gamma_2\{c/z\} \vdash a\{c/z\} :^{q} A\{c/z\}$.

Theorem 5.26 (Preservation) If $\Gamma \vdash a :^{q} A$ and $\vdash a \leadsto a'$, then $\Gamma \vdash a' :^{q} A$.

Theorem 5.27 (Progress) If $\varnothing \vdash a :^{q} A$, then either a is a value or there exists a' such that $\vdash a \leadsto a'$.

The above theorems show that LDC is type sound. But standard type soundness theorems are not strong enough to show correctness of usage and dependency analyses. So for this purpose, we design a heap semantics for LDC, as we did for SLDC.

5.4.3 Heap Semantics for LDC

LDC enjoys the same heap reduction relation as SLDC. However, the presence of dependent types causes an identical issue with heap semantics that we pointed out in Section 4.7. Because substitutions are delayed through the heap, the terms and their types may 'get out of sync'. To overcome this issue, we follow the same strategy we used in Section 4.7. We extend the type system of LDC with contexts that allow definitions. Then, we show that the extended calculus is sound with respect to heap semantics. We also show that the original calculus can be elaborated to the extended calculus. By dint of this elaboration, we conclude that the original calculus is also sound with respect to heap semantics. Next, we briefly review the technical details of this construction.

To allow delayed substitution in types, we extend contexts with definitions, which mimic substitutions. The definitions are only used to derive type equalities and are orthogonal to usage and dependency analyses. The variable and weakening rules for definitions and the conversion rule, where definitions are used, are shown in Figure 5.8. We add these extra rules to the type system of LDC. Note that in rule LDC-CONV-DEF,

$\Gamma ::= \varnothing \mid \Gamma, x :^q A \mid \Gamma, x = a :^q A$
$\Delta ::= \varnothing \mid \Delta, x : A \mid \Delta, x = a : A$

$\boxed{\Gamma \vdash a :^q A}$ *(Extra typing rules)*

$$\textsc{LDC-Var-Def}\ \frac{\Delta \vdash_0 a : A \qquad \lfloor \Gamma \rfloor = \Delta}{0 \cdot \Gamma, x = a :^q A \vdash x :^q A}$$

$$\textsc{LDC-Weak-Def}\ \frac{\Gamma \vdash a :^q A \qquad \Delta \vdash_0 b : B \qquad \lfloor \Gamma \rfloor = \Delta}{\Gamma, y = b :^0 B \vdash a :^q A}$$

$$\textsc{LDC-Conv-Def}\ \frac{\Gamma \vdash a :^q A \qquad \Delta \vdash_0 B : s \qquad A\{\Gamma\} =_\beta B\{\Gamma\} \qquad \lfloor \Gamma \rfloor = \Delta}{\Gamma \vdash a :^q B}$$

Figure 5.8: Typing rules with definitions

$\boxed{H \Vdash \Gamma}$ *(Updated compatibility)*

$$\textsc{Compat-Empty-Def}\ \frac{}{\varnothing \Vdash \varnothing}$$

$$\textsc{Compat-Cons-Def}\ \frac{H \Vdash \Gamma_1 + \Gamma_2 \qquad \Gamma_2 \vdash a :^q A}{H, x \overset{q}{\mapsto} a \Vdash \Gamma_1, x = a :^q A}$$

Figure 5.9: Compatibility relation with definitions

$A\{\Gamma\}$ denotes the type A with the definitions in Γ substituted in reverse order. The definitions allow us to communicate to the type system the substitutions that have been delayed by the heap. To enable this communication, we also need to update the compatibility relation, as shown in Figure 5.9.

Next, we show that these extensions do not alter the essential character of the underlying system. The following lemmas establish a correspondence between the underlying system and the extended one. To distinguish, let us denote the typing and the compatibility judgments of the underlying system as $\Gamma \vdash^{\mathbf{u}} a :^q A$ and $H \Vdash^{\mathbf{u}} \Gamma$ respectively and those of the extended system as $\Gamma \vdash^{\mathbf{e}} a :^q A$ and $H \Vdash^{\mathbf{e}} \Gamma$ respectively. Also, for $H \Vdash^{\mathbf{u}} \Gamma$, let Γ_H be the context Γ, with the variables defined according to H. Then, the multi-substitution lemma below says that given a derivation in the extended system, substituting the definitions gives us a derivation in the underlying system. Conversely, the elaboration lemma says that given a derivation in the underlying system, adding appropriate definitions to the context gives us a derivation in the extended system.

Lemma 5.28 (Multi-Substitution) If $H \Vdash^{\mathbf{e}} \Gamma$ and $\Gamma \vdash^{\mathbf{e}} a :^q A$, then $\varnothing \vdash^{\mathbf{u}} a\{\Gamma\} :^q A\{\Gamma\}$.

Lemma 5.29 (Elaboration) If $H \Vdash^{\mathbf{u}} \Gamma$ and $\Gamma \vdash^{\mathbf{u}} a :^q A$, then $H \Vdash^{\mathbf{e}} \Gamma_H$ and $\Gamma_H \vdash^{\mathbf{e}} a :^q A$.

Next, we show that the extended system is sound with respect to heap semantics. The statement of the soundness theorem, shown below, is the same as that for SLDC (Theorem 5.17). Further, this soundness theorem, like theorem 5.17, applies to both usage and dependency analyses.

Theorem 5.30 (Soundness) If $H \Vdash \Gamma$ and $\Gamma \vdash a :^q A$, then either a is a value or there exists H', Γ', a' such that:

- $[H]a \Longrightarrow^q_S [H']a'$

- $H' \Vdash \Gamma'$
- $\Gamma' \vdash a' :^{q} A$

The above theorem shows that usage and dependency analyses in LDC are sound. Several interesting corollaries, similar to the ones we saw in Section 5.3.3, follow from this theorem.

Corollary 5.31 (No Usage) In LDC($\mathcal{Q}_{\mathbb{N}}$): Let $\varnothing \vdash f :^{1} {}^{0}A \to \mathbf{Bool}$. Then, for any $\varnothing \vdash a_1 :^{0} A$ and $\varnothing \vdash a_2 :^{0} A$, the terms $f\ {a_1}^{0}$ and $f\ {a_2}^{0}$ have the same operational behavior.

Corollary 5.32 (Noninterference) In LDC($\mathcal{L}$): Let $\varnothing \vdash f :^{\ell} {}^{m}A \to \mathbf{Bool}$ such that $\neg(m \sqsubseteq \ell)$. Then, for any $\varnothing \vdash a_1 :^{m} A$ and $\varnothing \vdash a_2 :^{m} A$, the terms $f\ {a_1}^{m}$ and $f\ {a_2}^{m}$ have the same operational behavior.

We can also reason about polymorphic types using the soundness theorem, as we did in case of GRAD in Section 4.7.2.

Corollary 5.33 (Null Type) In LDC($\mathcal{Q}_{\mathbb{N}}$), if $\varnothing \vdash f :^{1} \Pi x :^{0} s.x$ and $\varnothing \vdash_{0} A : s$, then $f\ A^{0}$ must diverge.

Corollary 5.34 (Polymorphic Identity Type) In LDC($\mathcal{Q}_{\mathbb{N}}$), given the underlying PTS is strongly normalizing, if $\varnothing \vdash f :^{1} \Pi x :^{0} s.\Pi y :^{1} x.x$ and $\varnothing \vdash_{0} A : s$ and $\varnothing \vdash a :^{1} A$, then $f\ A^{0}\ a^{1} =_{\beta} a$.

5.5 Adding Unrestricted Usage

Till now, we used LDC for analyzing exact usage, bounded usage and dependency. In this section, we use the calculus for analyzing unrestricted usage as well. Unrestricted usage, referred to by ω, is different from exact and bounded usage, referred to by $n \in \mathbb{N}$, in two ways:

- ω is an additive annihilator, meaning $\omega + q = q + \omega = \omega$, for $q \in \mathbb{N} \cup \{\omega\}$. No $n \in \mathbb{N}$ is an additive annihilator.
- ω is a multiplicative annihilator (almost) as well, meaning $\omega \cdot q = q \cdot \omega = \omega$, for $q \in (\mathbb{N} - \{0\}) \cup \{\omega\}$. No $n \in \mathbb{N} - \{0\}$ is a multiplicative annihilator.

To accommodate this behavior of ω, we need to make a change to our type system. But before we make this change, let us fix our set of parametrizing preordered semirings:

- $\mathbb{N}^{\omega}_{=}$, that contains ω and the preordered semiring $\mathbb{N}_{=}$, with $\omega <: q$ for all q
- $\mathbb{N}^{\omega}_{\geq}$, that contains ω and the preordered semiring $\mathbb{N}_{\geq}$, with $\omega <: q$ for all q
- $\mathbb{B}_{\geq}$ and $\mathcal{Q}_{\mathrm{Lin}}$ and $\mathcal{Q}_{\mathrm{Aff}}$, as defined in Section 4.1

Observe that all the preordered semirings listed above include an element that signifies unrestricted usage. That element is denoted by ω in all but $\mathbb{B}_{\geq}$, where it is denoted by 1. But given that 1 behaves like ω in $\mathbb{B}_{\geq}$,

we also refer to it as ω. Next, observe that in all these preordered semirings, ω is the least element. This is to be expected since unrestricted usage includes all possible usages. Going forward, we use $\mathcal{Q}_{\mathbb{N}}^{\omega}$ to denote an arbitrary member of the above set of preordered semirings. Now, we discuss the change necessary to the type system as we move from LDC($\mathcal{Q}_{\mathbb{N}}$) to LDC($\mathcal{Q}_{\mathbb{N}}^{\omega}$).

5.5.1 A Problem and a Modification in Response

When unrestricted usage is allowed, the type systems in Figures 5.2 and 5.6 cannot enforce fairness of resource usage. Consider the following 'unfair' derivation allowed by SLDC in such a situation:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x :^{\omega} A \vdash x :^{\omega} A}\ (\text{SLDC-Var}) \quad \dfrac{}{x :^{\omega} A \vdash x :^{\omega} A}\ (\text{SLDC-Var})}{x :^{\omega} A \vdash (x^1, x) :^{\omega} {}^1A \times A}\ (\text{SLDC-Pair})}{\varnothing \vdash \lambda^1 x : A.(x^1, x) :^{\omega} {}^1A \to {}^1A \times A}\ (\text{SLDC-Lam})}{\varnothing \vdash \lambda^1 x : A.(x^1, x) :^{1} {}^1A \to {}^1A \times A}\ (\text{SLDC-SubR})$$

The judgment $\varnothing \vdash \lambda^1 x : A.(x^1, x) :^1 {}^1A \to {}^1A \times A$ is unfair because it allows copying of resources for any $\mathcal{Q}_{\mathbb{N}}^{\omega}$. Carefully observing the derivation, we find that the unfairness arises when ω, being a multiplicative annihilator (almost), 'tricks' the SLDC-Lam rule into believing that the term uses x once.

This unfairness would lead to a failure in type soundness. To see how, consider the term: $y :^1 A \vdash (\lambda^1 x : A.(x^1, x))\ y^1 :^1 {}^1A \times A$ that type-checks via the above derivation and rule SLDC-App. This term steps to: $\vdash (\lambda^1 x : A.(x^1, x))\ y^1 \leadsto (y^1, y)$. But then, we would have unsoundness because the judgment $y :^1 A \vdash (y^1, y) :^1 {}^1A \times A$ should not hold over any $\mathcal{Q} \in \mathcal{Q}_{\mathbb{N}}^{\omega} - \{\mathbb{B}_{\geq}\}$. Therefore, to ensure fairness and thereby type soundness, we need to modify rule SLDC-Lam and likewise, rule LDC-Lam.

We modify these rules as follows:

SLDC-LamOmega
$$\dfrac{\Gamma, x :^{q \cdot r} A \vdash b :^q B \qquad q = \omega \Rightarrow r = \omega \qquad q_0 \neq 0}{q_0 \cdot \Gamma \vdash \lambda^r x : A.b :^{q_0 \cdot q} {}^rA \to B}$$

LDC-LamOmega
$$\dfrac{\Gamma, x :^{q \cdot r} A \vdash b :^q B \qquad \Delta \vdash_0 \Pi x :^r A.B : s \qquad q = \omega \Rightarrow r = \omega \qquad \lfloor \Gamma \rfloor = \Delta \qquad q_0 \neq 0}{q_0 \cdot \Gamma \vdash \lambda^r x : A.b :^{q_0 \cdot q} \Pi x :^r A.B}$$

There are several points to note regarding the above rules:

- Rules SLDC-LamOmega and LDC-LamOmega are generalizations of rules SLDC-Lam and LDC-Lam respectively, meaning, when usage is analyzed over $\mathcal{Q}_{\mathbb{N}}$, as in Sections 5.2.1 and 5.4.1, replacing the latter rules with the former ones has no effect on the type system.

- These rules impose the constraint: $q = \omega \Rightarrow r = \omega$. This way the (almost) multiplicative annihilator ω won't be able to 'trick' the Lambda-rule into believing that functions use their arguments less than what they actually do. In particular, with these modified rules, the above unfair derivation won't go through. Note that since ω does not multiplicatively annihilate 0, we could have used a more permissive constraint in these rules: $q = \omega \Rightarrow r \in \{0, \omega\}$. However, we don't need this permissiveness because the permissive versions of the rules are admissible in the system, as we show in Section 5.5.2.

- The constraint $q = \omega \Rightarrow r = \omega$, while required for blocking unfair derivations, also blocks some fair ones like the one shown below:

$$\frac{x :^{\omega} A \vdash x :^{\omega} A}{\varnothing \vdash \lambda^1 x : A.x :^{\omega} {}^1 A \to A}$$

  To allow such derivations, while still imposing the constraint, rules SLDC-LamOmega and LDC-LamOmega multiply the conclusion judgment by q_0. This multiplication helps these rules allow the above derivation as:

$$\frac{x :^{1} A \vdash x :^{1} A}{\varnothing \vdash \lambda^1 x : A.x :^{\omega} {}^1 A \to A}$$

- The side condition $q_0 \neq 0$ makes sure that a meaningful judgment is not turned into a meaningless one. Recall that judgments in 0 world are meaningless, as far as usage and dependency analyses are concerned.

- The rules SLDC-LamOmega and LDC-LamOmega may seem specific to usage analysis because the constraint $q = \omega \Rightarrow r = \omega$ does not have an analogue in dependency analysis. This, however, is a minor issue because we can replace the constraint with: $(q + q = q) \wedge (q \neq 0) \Rightarrow r + r = r$.

  With respect to usage analysis, the premise of this new constraint may be satisfied only when $q = \omega$. In such a case, the constraint forces r to equal either 0 or ω. Therefore, this constraint is equivalent to $q = \omega \Rightarrow r \in \{0, \omega\}$. We pointed out above that replacing the constraint $q = \omega \Rightarrow r = \omega$ with $q = \omega \Rightarrow r \in \{0, \omega\}$ in rules SLDC-LamOmega and LDC-LamOmega does not have any effect on the system. So, we may as well replace $q = \omega \Rightarrow r = \omega$ with $(q + q = q) \wedge (q \neq 0) \Rightarrow r + r = r$.

  With respect to dependency analysis, the constraint $(q \sqcap q = q) \wedge (q \neq \top) \Rightarrow r \sqcap r = r$ is a tautology because $r \sqcap r = r$, for all r. Therefore, adding this constraint to rules SLDC-Lam and LDC-Lam does not alter dependency analysis in LDC in any way. This is exactly what we want, since LDC does not need any alteration with respect to dependency analysis.

  In sum, rules SLDC-LamOmega and LDC-LamOmega, with the constraint $(q + q = q) \wedge (q \neq 0) \Rightarrow r + r = r$ in place of $q = \omega \Rightarrow r = \omega$, make sense from the perspective of both usage and dependency analyses and this replacement has no effect on the system.

- Rule SLDC-LamOmega can also be equivalently replaced with the following two simpler rules (and similarly for rule LDC-LamOmega):

$$\text{SLDC-LamOmega0}\quad \frac{\Gamma, x :^{q\cdot r} A \vdash b :^q B \qquad q = \omega \Rightarrow r = \omega}{\Gamma \vdash \lambda^r x : A.b :^q {}^r A \to B} \qquad \text{SLDC-Omega}\quad \frac{\Gamma \vdash a :^q A \qquad q \neq 0}{\omega \cdot \Gamma \vdash a :^\omega A}$$

  Rule SLDC-Omega factors out the multiplication from rule SLDC-LamOmega. However, the former multiplies by ω whereas the latter by an arbitrary $q_0 \neq 0$. But this difference is not significant since a multiplication lemma is admissible in the system under both formulations.

Replacing rules SLDC-Lam and LDC-Lam with rules SLDC-LamOmega and LDC-LamOmega are the only modifications we need to make to the type systems of SLDC and LDC in order to enable them to track unrestricted usage. Now, we pointed out above that these modifications make the concerned rules more general because they help expand usage analyses without affecting dependency analyses. As such, the modified systems may be seen as more expressive versions of the original ones. So here onward, we shall use the names of the original systems to refer to their modified counterparts.

Next, we consider the metatheory and soundness of LDC, as modified above.

5.5.2 Metatheory and Soundness of Modified System

Since the modification described above does not affect dependency analyses, we shall just consider the metatheory and soundness of usage analyses. LDC($\mathcal{Q}_{\mathbb{N}}^{\omega}$) satisfies all the lemmas and theorems satisfied by LDC($\mathcal{Q}_{\mathbb{N}}$), as stated in Section 5.4.2. We shall not restate the lemmas here but would point out an important distinction in the proof of the factorization lemma. To prove the factorization lemma for LDC($\mathcal{Q}_{\mathbb{N}}^{\omega}$), we need an auxiliary lemma on unusable resources, as stated below.

Lemma 5.35 (Unusability) If $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash a :^q A$ and $\neg(\exists q_0, r_0 <: q_0 + q)$, then $\Gamma_1, z :^0 C, \Gamma_2 \vdash a :^q A$.

The above lemma says that an unusable resource may be replaced by a 0-usage resource. The resource $z :^{r_0} C$ is unusable at world q because $\neg(\exists q_0, r_0 <: q_0 + q)$. The above lemma is necessary to prove the factorization lemma, which appears below, because once unrestricted usage is allowed, the multiplicative cancellation property no longer holds, i.e., $q \cdot r_1 <: q \cdot r_2 \not\Rightarrow r_1 <: r_2$ because $\omega \cdot r_1 <: \omega \cdot r_2$ for arbitrary $r_1, r_2 \neq 0$. In the absence of the cancellation property, a proof by induction of the factorization lemma requires the above lemma in several cases, especially when q in the lemma statement equals ω.

Lemma 5.36 (Factorization) If $\Gamma \vdash a :^q A$ and $q \neq 0$, then there exists Γ' such that $\Gamma' \vdash a :^1 A$ and $\Gamma <: q \cdot \Gamma'$.

Using the factorization lemma, we can show that the following permissive variant of rule LDC-LamOmega is admissible in LDC, as pointed out in Section 5.5.1.

$$\text{LDC-LamOmegaV}\quad \frac{\begin{array}{c}\Gamma, x :^{q \cdot r} A \vdash b :^{q} B \\ \Delta \vdash_0 \Pi x :^{r} A.B : s \\ q = \omega \Rightarrow r \in \{0, \omega\} \\ \lfloor \Gamma \rfloor = \Delta \qquad q_0 \neq 0\end{array}}{q_0 \cdot \Gamma \vdash \lambda^{r} x : A.b :^{q_0 \cdot q} \Pi x :^{r} A.B}$$

To see how, first observe that if $q = \omega$ and $r = \omega$, then the above rule follows by rule LDC-LamOmega. So, we just need to derive the rule when $q = \omega$ and $r = 0$. The derivation is as follows. Given $\Gamma, x :^{0} A \vdash b :^{\omega} B$, by the factorization lemma, there exists Γ' and q' such that $\Gamma', x :^{q'} A \vdash b :^{1} B$ where $\Gamma <: \omega \cdot \Gamma'$ and $0 <: \omega \cdot q'$. Then, from $0 <: \omega \cdot q'$, we have, $q' = 0$ because $0 <: q'' \Rightarrow q'' = 0$ in any $\mathcal{Q}_{\mathbb{N}}^{\omega}$. This means, $\Gamma', x :^{0} A \vdash b :^{1} B$. Now, by rule LDC-LamOmega, $\omega \cdot \Gamma' \vdash \lambda^{0} x : A.b :^{\omega} \Pi x :^{0} A.B$ and then by rule LDC-SubL, $\Gamma \vdash \lambda^{0} x : A.b :^{\omega} \Pi x :^{0} A.B$. Finally, by the multiplication lemma, $q_0 \cdot \Gamma \vdash \lambda^{0} x : A.b :^{q_0 \cdot \omega} \Pi x :^{0} A.B$, as required. So, we see that rule LDC-LamOmegaV is derivable in the calculus. As such, we don't need the permissiveness offered by this rule over and above that of rule LDC-LamOmega. On a side note, this argument also shows that when the parametrizing preordered semiring is $\mathbb{B}_{\geq}$, the rules LDC-Lam and LDC-LamOmega become interchangeable.

Next, we show that LDC($\mathcal{Q}_{\mathbb{N}}^{\omega}$) is type sound.

Theorem 5.37 (Preservation) If $\Gamma \vdash a :^{q} A$ and $\vdash a \rightsquigarrow a'$, then $\Gamma \vdash a' :^{q} A$.

Theorem 5.38 (Progress) If $\varnothing \vdash a :^{q} A$, then either a is a value or there exists a' such that $\vdash a \rightsquigarrow a'$.

Now, we move on to soundness with respect to heap semantics. Note that for LDC($\mathcal{Q}_{\mathbb{N}}^{\omega}$), we don't need any modification to our heap semantics. Further, to prove soundness with respect to this semantics, we can follow the exact same strategy laid out in Section 5.4.3. We shall not restate the details here but would just sum up the property in terms of the statement below.

Theorem 5.39 LDC($\mathcal{Q}_{\mathbb{N}}^{\omega}$) satisfies heap soundness.

The above theorem shows that LDC can analyze usage soundly over any $\mathcal{Q}_{\mathbb{N}}^{\omega}$. Recall that $\mathcal{Q}_{\mathbb{N}}^{\omega}$ includes the preordered semirings $\mathcal{Q}_{\text{Lin}}$ and $\mathcal{Q}_{\text{Aff}}$. This, then, shows that LDC can analyze linear and affine usages correctly. LDC can also analyze dependencies in general because when parametrized over lattices, the calculus is equivalent to DDC$^{\top}$, as we show in the following section. In the following section, we also show that when it comes to analyzing linear and affine usages, LDC subsumes GraD. Therefore, we can say that LDC is a general calculus for linearity and dependency analyses in pure type systems. For analyzing linearity and other usages, we parametrize the calculus over a suitable $\mathcal{Q}_{\mathbb{N}}^{\omega}$. For analyzing dependencies, we parametrize the calculus over a suitable lattice. For a combined usage and dependency analysis, we parametrize the calculus over the cartesian product of the structures used for the individual analyses, as elaborated in Section 1.3.1 and also in the following one.

5.6 LDC: A General Calculus for Linearity and Dependency Analyses

In this section, we show that LDC is a general calculus for linearity and dependency analyses. We have already seen some examples of usage (including linear usage) and dependency analyses in LDC in Section 5.4.1. Here, we show how to carry out a combined usage and dependency analysis in LDC. Thereafter, by comparison with GRAD and $\text{DDC}^{\top}$, we show that LDC is a very general calculus, when it comes to usage and dependency analyses.

5.6.1 Combined Usage and Dependency Analysis

For a combined usage and dependency analysis, we parametrize LDC over the cartesian product of the structures employed for the individual analyses, i.e., preordered semirings in $\mathcal{Q}_{\mathbb{N}}^{\omega}$ and lattices. Recall from Section 1.3.1 that the cartesian product of a preordered semiring $\mathcal{Q} = (Q, +, \cdot, 0, 1, <:)$ and a lattice $\mathcal{L} = (L, \sqcap, \sqcup, \top, \bot, \sqsubseteq)$, denoted $\mathcal{Q} \times \mathcal{L}$, is defined as the set $Q \times L$, together with two constants, $(0, \top)$ and $(1, \bot)$, and two binary operators, $+$ and $\cdot$, defined as, $(q_1, \ell_1) + (q_2, \ell_2) \triangleq (q_1 + q_2, \ell_1 \sqcap \ell_2)$ and $(q_1, \ell_1) \cdot (q_2, \ell_2) \triangleq (q_1 \cdot q_2, \ell_1 \sqcup \ell_2)$, and a binary order relation, $<:$, defined as, $(q_1, \ell_1) <: (q_2, \ell_2) \triangleq q_1 <: q_2 \wedge \ell_1 \sqsubseteq \ell_2$. Now, given a preordered semiring $\mathcal{Q}$ in $\mathcal{Q}_{\mathbb{N}}^{\omega}$ and a lattice $\mathcal{L}$, we can parametrize LDC over $\mathcal{Q} \times \mathcal{L}$ and interpret the typing rules of the calculus in terms of the above definitions. With this parametrization, the calculus can simultaneously analyze usages and dependencies. Below, we present some examples that illustrate how.

Let $\mathcal{Q} \coloneqq \mathcal{Q}_{\text{Lin}}$ and $\mathcal{L} \coloneqq \mathcal{L}_2$. Then, in $\text{LDC}(\mathcal{Q} \times \mathcal{L})$, we have,

$$\alpha :^{(0,\mathbf{H})} s, z :^{(1,\mathbf{L})} {}^{(1,\mathbf{L})}\alpha \times \alpha \vdash \mathbf{let}_{(1,\mathbf{L})} (x^{(1,\mathbf{L})}, y) = z \,\mathbf{in}\, (y^{(1,\mathbf{L})}, x) :^{(1,\mathbf{L})} {}^{(1,\mathbf{L})}\alpha \times \alpha \tag{5.2}$$

$$\alpha :^{(0,\mathbf{H})} s, z :^{(1,\mathbf{L})} {}^{(1,\mathbf{L})}\alpha \times \alpha \nvdash \mathbf{let}_{(1,\mathbf{L})} (x^{(1,\mathbf{L})}, y) = z \,\mathbf{in}\, (y^{(1,\mathbf{L})}, y) :^{(1,\mathbf{L})} {}^{(1,\mathbf{L})}\alpha \times \alpha \tag{5.3}$$

$$\alpha :^{(0,\mathbf{H})} s, z :^{(\omega,\mathbf{L})} {}^{(\omega,\mathbf{H})}\alpha \times \alpha \nvdash \mathbf{let}_{(1,\mathbf{L})} (x^{(\omega,\mathbf{H})}, y) = z \,\mathbf{in}\, (y^{(\omega,\mathbf{H})}, x) :^{(\omega,\mathbf{L})} {}^{(\omega,\mathbf{H})}\alpha \times \alpha \tag{5.4}$$

$$\alpha :^{(0,\mathbf{H})} s, z :^{(\omega,\mathbf{L})} {}^{(\omega,\mathbf{H})}\alpha \times \alpha \vdash \mathbf{let}_{(1,\mathbf{L})} (x^{(\omega,\mathbf{H})}, y) = z \,\mathbf{in}\, (y^{(\omega,\mathbf{H})}, y) :^{(\omega,\mathbf{L})} {}^{(\omega,\mathbf{H})}\alpha \times \alpha \tag{5.5}$$

$$\alpha :^{(0,\mathbf{H})} s, z :^{(1,\mathbf{L})} {}^{(1,\mathbf{H})}\alpha \times \alpha \nvdash \mathbf{let}_{(1,\mathbf{L})} (x^{(1,\mathbf{H})}, y) = z \,\mathbf{in}\, (y^{(1,\mathbf{H})}, x) :^{(1,\mathbf{L})} {}^{(1,\mathbf{H})}\alpha \times \alpha \tag{5.6}$$

$$\alpha :^{(0,\mathbf{H})} s, z :^{(1,\mathbf{L})} {}^{(1,\mathbf{H})}\alpha \times \alpha \nvdash \mathbf{let}_{(1,\mathbf{L})} (x^{(1,\mathbf{H})}, y) = z \,\mathbf{in}\, (y^{(1,\mathbf{H})}, y) :^{(1,\mathbf{L})} {}^{(1,\mathbf{H})}\alpha \times \alpha \tag{5.7}$$

$$\alpha :^{(0,\mathbf{H})} s, z :^{(1,\mathbf{L})} {}^{(1,\mathbf{H})}\alpha \times \alpha \vdash \mathbf{let}_{(1,\mathbf{L})} (x^{(1,\mathbf{H})}, y) = z \,\mathbf{in}\, (x^{(1,\mathbf{H})}, y) :^{(1,\mathbf{L})} {}^{(1,\mathbf{H})}\alpha \times \alpha \tag{5.8}$$

The above examples, discussed in the introduction to the chapter, show how combined usage and dependency analysis works. In examples (5.2) and (5.3), we solely impose usage constraints whereas in examples (5.4) and (5.5), we solely impose dependency constraints. In examples (5.6), (5.7) and (5.8), we impose usage and dependency constraints together. When usage and dependency constraints are imposed together, terms that violate either of them do not type-check, as we can see by carefully going through examples (5.2)-(5.8).

Next, we compare LDC with GRAD and $\text{DDC}^{\top}$.

5.6.2 Comparison with GraD and DDC$^{\top}$

GRAD can be employed as a general coeffect calculus, as pointed out in Section 4.9.1. LDC, however, cannot be employed as such. The reason behind this shortcoming is that unlike GRAD, LDC cannot be parametrized by an arbitrary preordered semiring. Recall that in Section 5.5, we presented a list of preordered semirings that may parametrize LDC. Further, in Section 5.1.2, we pointed out why an arbitrary preordered semiring cannot parametrize LDC. So, while comparing LDC with GRAD, we restrict ourselves to the list of preordered semirings, presented in Section 5.5 and denoted by $\mathcal{Q}_{\mathbb{N}}^{\omega}$, that may parametrize both the calculi. With this restriction, we are essentially considering a comparison of the two calculi with respect to usage analyses only.

Next, we show that over any $\mathcal{Q}_{\mathbb{N}}^{\omega}$, LDC subsumes GRAD. Towards this end, we present a meaning-preserving translation from GRAD to LDC. The translation function, $\overline{}$, is defined by a straightforward recursion. We present a few cases of the function definition below. Note that if we ignore the subscript grade in the elimination forms for pair and unit, $\overline{}$ is exactly an identity function.

$$\overline{x} = x \qquad \overline{({a_1}^r, a_2)} = (\overline{a_1}^r, \overline{a_2}) \qquad \overline{\mathbf{let}\ (x^r, y) = a\ \mathbf{in}\ b} = \mathbf{let}_1\ (x^r, y) = \overline{a}\ \mathbf{in}\ \overline{b}$$

$$\overline{\mathbf{unit}} = \mathbf{unit} \qquad \overline{\mathbf{inj}_i\ a} = \mathbf{inj}_i\ \overline{a} \qquad \overline{\mathbf{case}_{q_0}\ a\ \mathbf{of}\ x_1.b_1\ ;\ x_2.b_2} = \mathbf{case}_{q_0}\ \overline{a}\ \mathbf{of}\ x_1.\overline{b_1}\ ;\ x_2.\overline{b_2}$$

This translation preserves typing and meaning, as shown by the lemma below. Following the convention set in Section 4.5.1, we use $\vec{\Gamma}$ to denote Γ, with the types translated.

Lemma 5.40 (LDC and GRAD) For any $\mathcal{Q} \in \mathcal{Q}_{\mathbb{N}}^{\omega}$, if $\Gamma \vdash a : A$ in GRAD($\mathcal{Q}$), then $\vec{\Gamma} \vdash \overline{a} :^1 \overline{A}$ in LDC($\mathcal{Q}$). Further, if $\vdash a \rightsquigarrow a'$ in GRAD($\mathcal{Q}$), then $\vdash \overline{a} \rightsquigarrow \overline{a'}$ in LDC($\mathcal{Q}$).

The above lemma shows that when it comes to usage analyses over $\mathcal{Q}_{\mathbb{N}}^{\omega}$, LDC is no less expressive than GRAD. As a matter of fact, LDC is somewhat more expressive than GRAD in this regard. We argue why below. Recall from Section 4.5.1 that the preordered semiring $\mathbb{B}_{\geq}$ is isomorphic to lattice $\mathcal{L}_2 \coloneqq \mathbf{L} \sqsubseteq \mathbf{H}$. Then, LDC($\mathbb{B}_{\geq}$) $\cong$ LDC($\mathcal{L}_2$) and GRAD($\mathbb{B}_{\geq}$) $\cong$ GRAD($\mathcal{L}_2$). Now, as pointed out in Section 5.1.1, the type $\Box^{\mathbf{H}}(\Box^{\mathbf{H}}\mathbf{Bool} \to \mathbf{Bool})$ contains only two distinct terms in GRAD($\mathcal{L}_2$) whereas the same type contains four distinct terms in SLDC($\mathcal{L}_2$) (refer to Section 5.2.3), and therefore in LDC($\mathcal{L}_2$). As such, LDC($\mathcal{L}_2$) or LDC($\mathbb{B}_{\geq}$) is more expressive than GRAD($\mathcal{L}_2$) or GRAD($\mathbb{B}_{\geq}$).

Now, we move on to dependency analyses and compare LDC with DDC$^{\top}$. Both LDC and DDC$^{\top}$ are general dependency calculi parametrized by arbitrary lattices. Here, we show that the two calculi are equivalent over any parametrizing lattice. We establish the equivalence in a straightforward manner as follows. Observe that owing to the syntactic similarity of the two calculi, the translation functions are identities in both the directions.

Lemma 5.41 (LDC and DDC$^{\top}$) For any lattice $\mathcal{L}$, $\Gamma \vdash a :^{\ell} A$ in DDC$^{\top}$($\mathcal{L}$) if and only if $\Gamma \vdash a :^{\ell} A$ in LDC($\mathcal{L}$). Further, $\vdash a \rightsquigarrow a'$ in DDC$^{\top}$($\mathcal{L}$) if and only if $\vdash a \rightsquigarrow a'$ in LDC($\mathcal{L}$).

The above lemma shows that when it comes to dependency analysis, LDC is as expressive as DDC$^{\top}$. LDC, however, does not internalize dependency analysis and as such, is not as expressive as DDC. In particular,

unlike DDC, LDC does not analyze compile-time irrelevance. An interesting future work would be to add compile-time irrelevance analysis to LDC.

5.6.3 No Usage and Run-Time Irrelevance

Before moving ahead, we would like to draw the reader's attention to an interesting result. We pointed out above that $\text{LDC}(\mathbb{B}_{\geq}) \cong \text{LDC}(\mathcal{L}_2)$. Now on one hand, $\text{LDC}(\mathbb{B}_{\geq})$, like $\text{Grad}(\mathbb{B}_{\geq})$, is a calculus for analyzing no usage. On the other hand, by Lemma 5.41, $\text{LDC}(\mathcal{L}_2)$ is equivalent to $\text{DDC}^{\top}(\mathcal{L}_2)$, which is a calculus for analyzing run-time irrelevance, as shown in Section 3.3.5. So then, from the above isomorphism, we conclude that no usage and run-time irrelevance analyses are essentially the same, as referred to in Section 4.8.2. This result is a fine testimony to the close connection between usage and dependency analyses.

5.7 Discussions and Related Work

5.7.1 SLDC and LNL λ-calculus

In Section 1.1.3, we introduced two split-context type systems, LNL λ-calculus of Benton [1994] and DILL of Barber [1996]. The discussion in that section focused on DILL. Here, we consider the LNL λ-calculus. The LNL λ-calculus is similar to DILL in its use of split-contexts. But it is also different from DILL because it uses two distinct forms of typing judgments, which correspond to linear and nonlinear worlds respectively. Recall that DILL uses a single form of typing judgment, which corresponds to the linear world. This difference between DILL and LNL λ-calculus is emblematic of the difference between GLC and SLDC, with regard to the form of their typing judgments. Similar to DILL, GLC uses a single form of typing judgment, which corresponds to the linear world. Likewise, similar to LNL λ-calculus, SLDC uses graded typing judgments, where the grade (to the right of the turnstile in a typing judgment) corresponds to the world under consideration. In Section 1.1.7, we discussed in detail how graded-context type systems, of which GLC is but a representative, generalize DILL. In this section, we show how SLDC generalizes LNL λ-calculus.

First, note that the LNL λ-calculus analyzes linear and unrestricted usages only. So, we compare it with $\text{SLDC}(\mathcal{Q}_{\text{Lin}})$. Next, to analyze linear and unrestricted usages, the LNL λ-calculus employs two forms of contexts and two forms of typing judgments. The two forms of contexts, linear and nonlinear, correspond to assumptions held at grades 1 and ω respectively in $\text{SLDC}(\mathcal{Q}_{\text{Lin}})$. The two forms of typing judgments, linear and nonlinear, correspond to derivations in worlds 1 and ω respectively in $\text{SLDC}(\mathcal{Q}_{\text{Lin}})$. The LNL λ-calculus has two sets of types: standard intuitionistic types and their linear variants, such as, standard function type, $X \to Y$, and its linear variant, $A \multimap B$; standard product type, $X \times Y$, and its linear variant, $A \otimes B$. Additionally, the calculus has two type constructors, F and G, via which the linear and nonlinear worlds interact. As far as the terms of the calculus are concerned, they include the introduction and elimination forms for these types.

linear types, A, B, C	$::= I \mid A \otimes B \mid A \multimap B \mid F\,X$
nonlinear types, X, Y, Z	$::= 1 \mid X \times Y \mid X \to Y \mid G\,A$
linear terms, e, f	$::= a \mid b \mid * \mid \mathbf{let}\ * = e\ \mathbf{in}\ f \mid e \otimes f \mid \mathbf{let}\ a \otimes b = e\ \mathbf{in}\ f \mid \lambda a : A.e \mid e\ f \mid$
	$\mathbf{derelict}\ s \mid F\ s \mid \mathbf{let}\ F(x) = e\ \mathbf{in}\ f$
nonlinear terms, s, t	$::= x \mid y \mid () \mid (s, t) \mid \mathbf{fst}(s) \mid \mathbf{snd}(s) \mid \lambda x : X.s \mid s\ t \mid G\ e$
linear contexts, Γ	$::= \varnothing \mid \Gamma, a : A$
nonlinear contexts, Θ	$::= \varnothing \mid \Theta, x : X$

FIGURE 5.10: Grammar of LNL λ-calculus

In the LNL λ-calculus, linear and nonlinear contexts (which are multisets of assumptions) are denoted by Γ and Θ respectively, while linear and nonlinear judgments are written as $\Theta; \Gamma \vdash_{\mathcal{L}} e : A$ and $\Theta \vdash_{\mathcal{C}} s : X$ respectively. The calculus uses A, B, C for linear types; X, Y, Z for nonlinear types; a, b for linear variables; x, y for nonlinear variables; e, f for linear terms; and s, t for nonlinear terms. The grammar of the calculus appears in Figure 5.10; typing rules in Figure 5.11; and β-rules in Figure 5.12. This presentation is as given in Benton [1994].

Now, we present a meaning-preserving translation from LNL λ-calculus to SLDC($\mathcal{Q}_{\text{Lin}}$). The translation function for types and terms is given in Figure 5.13. Observe that both the linear and nonlinear function types of LNL λ-calculus get translated to the same underlying function type in SLDC($\mathcal{Q}_{\text{Lin}}$), with the difference between them captured by the differing grade annotations. The same is true of linear and nonlinear product types. Further, observe that no extra type constructors are required for translating F and G.

Next, we extend the translation function to contexts: a linear context Γ and a nonlinear context Θ of LNL λ-calculus are translated to $\overline{\Gamma}^1$ and $\overline{\Theta}^\omega$ of SLDC($\mathcal{Q}_{\text{Lin}}$) respectively. Here, $\overline{\Gamma}^1$ and $\overline{\Theta}^\omega$ denote Γ and Θ, with the types translated, and all assumptions held at grades 1 and ω respectively.

This translation preserves typing and meaning, as shown by the lemma below. Here, $_ =_\beta _$ denotes the equivalence relation generated by the β-rules presented in Figure 5.12. We use the same notation for denoting the β-equivalence relation of SLDC, which is generated according to the rules presented in Figure 5.3.

Lemma 5.42 The translation from LNL λ-calculus to SLDC($\mathcal{Q}_{\text{Lin}}$) is sound:

- If $\Theta; \Gamma \vdash_{\mathcal{L}} e : A$, then $\overline{\Theta}^\omega, \overline{\Gamma}^1 \vdash \overline{e} :^1 \overline{A}$.
- If $\Theta \vdash_{\mathcal{C}} s : X$, then $\overline{\Theta}^\omega \vdash \overline{s} :^\omega \overline{X}$.
- If $e =_\beta f$, then $\overline{e} =_\beta \overline{f}$. If $s =_\beta t$ then $\overline{s} =_\beta \overline{t}$.

The above translation shows that SLDC($\mathcal{Q}_{\text{Lin}}$) is no less expressive than the LNL λ-calculus. As a matter of fact, SLDC($\mathcal{Q}_{\text{Lin}}$) is more expressive than the LNL λ-calculus because the latter does not model 0-usage.

$\boxed{\Theta;\Gamma \vdash_{\mathcal{L}} e : A}$ *(Linear typing rules)*

$$\textsc{L-Var}\ \frac{}{\Theta; a : A \vdash_{\mathcal{L}} a : A} \qquad \textsc{L-Unit}\ \frac{}{\Theta; \varnothing \vdash_{\mathcal{L}} * : I}$$

$$\textsc{L-LetUnit}\ \frac{\Theta;\Gamma_1 \vdash_{\mathcal{L}} e : I \quad \Theta;\Gamma_2 \vdash_{\mathcal{L}} f : A}{\Theta;\Gamma_1,\Gamma_2 \vdash_{\mathcal{L}} \mathbf{let}\ * = e\ \mathbf{in}\ f : A} \qquad \textsc{L-Pair}\ \frac{\Theta;\Gamma_1 \vdash_{\mathcal{L}} e : A \quad \Theta;\Gamma_2 \vdash_{\mathcal{L}} f : B}{\Theta;\Gamma_1,\Gamma_2 \vdash_{\mathcal{L}} e \otimes f : A \otimes B}$$

$$\textsc{L-LetPair}\ \frac{\Theta;\Gamma_1 \vdash_{\mathcal{L}} e : A \otimes B \quad \Theta;\Gamma_2, a : A, b : B \vdash_{\mathcal{L}} f : C}{\Theta;\Gamma_1,\Gamma_2 \vdash_{\mathcal{L}} \mathbf{let}\ a \otimes b = e\ \mathbf{in}\ f : C} \qquad \textsc{L-Lam}\ \frac{\Theta;\Gamma, a : A \vdash_{\mathcal{L}} e : B}{\Theta;\Gamma \vdash_{\mathcal{L}} \lambda a : A.e : A \multimap B}$$

$$\textsc{L-App}\ \frac{\Theta;\Gamma_1 \vdash_{\mathcal{L}} e : A \multimap B \quad \Theta;\Gamma_2 \vdash_{\mathcal{L}} f : A}{\Theta;\Gamma_1,\Gamma_2 \vdash_{\mathcal{L}} e\, f : B}$$

$$\textsc{L-FIntro}\ \frac{\Theta \vdash_{\mathcal{C}} s : X}{\Theta;\varnothing \vdash_{\mathcal{L}} F\, s : F\, X} \qquad \textsc{L-FElim}\ \frac{\Theta;\Gamma_1 \vdash_{\mathcal{L}} e : F\, X \quad \Theta, x : X;\Gamma_2 \vdash_{\mathcal{L}} f : A}{\Theta;\Gamma_1,\Gamma_2 \vdash_{\mathcal{L}} \mathbf{let}\ F(x) = e\ \mathbf{in}\ f : A} \qquad \textsc{L-GElim}\ \frac{\Theta \vdash_{\mathcal{C}} s : G\, A}{\Theta;\varnothing \vdash_{\mathcal{L}} \mathbf{derelict}\ s : A}$$

$\boxed{\Theta \vdash_{\mathcal{C}} s : X}$ *(Nonlinear typing rules)*

$$\textsc{NL-Var}\ \frac{}{\Theta, x : X \vdash_{\mathcal{C}} x : X} \qquad \textsc{NL-Unit}\ \frac{}{\Theta \vdash_{\mathcal{C}} () : 1} \qquad \textsc{NL-Pair}\ \frac{\Theta \vdash_{\mathcal{C}} s : X \quad \Theta \vdash_{\mathcal{C}} t : Y}{\Theta \vdash_{\mathcal{C}} (s,t) : X \times Y} \qquad \textsc{NL-Fst}\ \frac{\Theta \vdash_{\mathcal{C}} s : X \times Y}{\Theta \vdash_{\mathcal{C}} \mathbf{fst}(s) : X}$$

$$\textsc{NL-Snd}\ \frac{\Theta \vdash_{\mathcal{C}} s : X \times Y}{\Theta \vdash_{\mathcal{C}} \mathbf{snd}(s) : Y} \qquad \textsc{NL-Lam}\ \frac{\Theta, x : X \vdash_{\mathcal{C}} s : Y}{\Theta \vdash_{\mathcal{C}} \lambda x : X.s : X \to Y} \qquad \textsc{NL-App}\ \frac{\Theta \vdash_{\mathcal{C}} s : X \to Y \quad \Theta \vdash_{\mathcal{C}} t : X}{\Theta \vdash_{\mathcal{C}} s\, t : Y} \qquad \textsc{NL-GIntro}\ \frac{\Theta;\varnothing \vdash_{\mathcal{L}} e : A}{\Theta \vdash_{\mathcal{C}} G\, e : G\, A}$$

FIGURE 5.11: Typing rules of LNL λ-calculus

$\boxed{e \to_\beta f \text{ and } s \to_\beta t}$ *(β-rules)*

$$\textsc{Beta-LetUnitL}\ \frac{}{\mathbf{let}\ * = *\ \mathbf{in}\ f \to_\beta f} \qquad \textsc{Beta-LetPairL}\ \frac{}{\mathbf{let}\ a_1 \otimes a_2 = e_1 \otimes e_2\ \mathbf{in}\ f \to_\beta f\{e_1/a_1\}\{e_2/a_2\}} \qquad \textsc{Beta-Fst}\ \frac{}{\mathbf{fst}((s,t)) \to_\beta s}$$

$$\textsc{Beta-Snd}\ \frac{}{\mathbf{snd}((s,t)) \to_\beta t} \qquad \textsc{Beta-AppL}\ \frac{}{(\lambda a : A.e)\, f \to_\beta e\{f/a\}} \qquad \textsc{Beta-AppNL}\ \frac{}{(\lambda x : X.s)\, t \to_\beta s\{t/x\}}$$

$$\textsc{Beta-F}\ \frac{}{\mathbf{let}\ F(x) = F\, s\ \mathbf{in}\ e \to_\beta e\{s/x\}} \qquad \textsc{Beta-G}\ \frac{}{\mathbf{derelict}\ (G\, e) \to_\beta e}$$

FIGURE 5.12: β-rules of LNL λ-calculus

$$\overline{1} = \mathbf{Unit} \qquad \overline{X \times Y} = {}^{\omega}\overline{X} \times \overline{Y} \qquad \overline{I} = \mathbf{Unit} \qquad \overline{A \otimes B} = {}^{1}\overline{A} \times \overline{B}$$

$$\overline{X \to Y} = {}^{\omega}\overline{X} \to \overline{Y} \qquad \overline{G\,A} = \overline{A} \qquad \overline{A \multimap B} = {}^{1}\overline{A} \to \overline{B} \qquad \overline{F\,X} = {}^{\omega}\overline{X} \times \mathbf{Unit}$$

$\overline{x} = x$	$\overline{a} = a$
$\overline{()} = \mathbf{unit}$	$\overline{*} = \mathbf{unit}$
$\overline{(s,t)} = (\overline{s}^{\omega}, \overline{t})$	$\overline{e \otimes f} = (\overline{e}^{1}, \overline{f})$
$\overline{\mathbf{fst}(s)} = \mathbf{let}_1\ (x^{\omega}, y) = \overline{s}\ \mathbf{in}\ x\ \ (x, y \text{ fresh})$	$\overline{\mathbf{let}\ a \otimes b = e\ \mathbf{in}\ f} = \mathbf{let}_1\ (a^{1}, b) = \overline{e}\ \mathbf{in}\ \overline{f}$
$\overline{\mathbf{snd}(s)} = \mathbf{let}_1\ (x^{\omega}, y) = \overline{s}\ \mathbf{in}\ y\ \ (x, y \text{ fresh})$	$\overline{\mathbf{let}\ * = e\ \mathbf{in}\ f} = \mathbf{let}_1\ \mathbf{unit} = \overline{e}\ \mathbf{in}\ \overline{f}$
$\overline{\lambda x : X.s} = \lambda^{\omega} x : \overline{X}.\overline{s}$	$\overline{\lambda a : A.e} = \lambda^{1} a : \overline{A}.\overline{e}$
$\overline{s\,t} = \overline{s}\,\overline{t}^{\omega}$	$\overline{e\,f} = \overline{e}\,\overline{f}^{1}$
$\overline{G\,e} = \overline{e}$	$\overline{\mathbf{derelict}\ s} = \overline{s}$
$\overline{F\,s} = (\overline{s}^{\omega}, \mathbf{unit})$	$\overline{\mathbf{let}\ F(x) = e\ \mathbf{in}\ f} = \mathbf{let}_1\ (x^{\omega}, y) = \overline{e}\ \mathbf{in}\ \mathbf{let}_1\ \mathbf{unit} = y\ \mathbf{in}\ \overline{f}$ (y fresh)

FIGURE 5.13: Type and term translation from LNL λ-calculus to SLDC($\mathcal{Q}_{\mathrm{Lin}}$)

This is also the reason why a translation in the other direction from SLDC($\mathcal{Q}_{\mathrm{Lin}}$) to LNL λ-calculus would fail.

Now, we discuss why SLDC may be seen as a generalization of the LNL λ-calculus. The LNL λ-calculus is designed to model linear and nonlinear usages only. The calculus models these usages by employing two distinct forms of typing judgments. SLDC, on the other hand, models a wide variety of usages over the set of preordered semirings listed in Section 5.5. SLDC models these usages by employing a typing judgment that is graded by the elements of the parametrizing structure. The typing judgment of SLDC may be seen as a generalization of that of the LNL λ-calculus: the latter has only two distinct forms, while the former has as many 'forms' as the number of elements of the parametrizing structure. (The translation above shows us that the two distinct forms of typing judgments of LNL λ-calculus correspond to two 'forms' of typing judgments of SLDC($\mathcal{Q}_{\mathrm{Lin}}$).) Further, SLDC employs graded contexts, which, as discussed in Section 1.1.7, may be seen as generalization of split contexts, employed by LNL λ-calculus. So, both the typing context and typing judgment of SLDC may be seen as generalizations of those of the LNL λ-calculus. Owing to these generalizations, SLDC can analyze a wider variety of usages than the LNL λ-calculus. In sum, we can say that SLDC is a generalized LNL λ-calculus.

5.7.2 LDC and McBride's System

In Section 1.1.9, we introduced McBride's system, which is the first-of-its-kind linear dependent type system. Interestingly, LDC is very similar to McBride's system. The key similarity between the two systems is that both of them employ graded typing judgments, i.e., typing judgments with a variable grade to the right

of the turnstile. This feature distinguishes these two systems from other graded-context type systems in literature, where typing judgments have a fixed grade to the right of the turnstile. In this chapter, we saw how graded typing judgments help unify linearity and dependency analyses. However, graded typing judgments are best described as double-edged swords. Recall from Section 1.1.9 that McBride's system does not admit substitution. The main reason behind this inadmissibility of substitution is the variable nature of the grade to the right of the turnstile in typing judgments. As a matter of fact, when this grade is restricted to $\{0, 1\}$, substitution becomes admissible in McBride's system, as shown by Atkey [2018].

At this point, the reader may wonder, if the variable nature of the grade to the right of the turnstile is indeed to blame for inadmissibility of substitution in McBride's system, then why is substitution admissible in LDC, which also shares this feature? The answer lies in the restriction we impose on structures that may parametrize LDC. While McBride's system may be parametrized by an arbitrary semiring, LDC may be parametrized by only those preordered semirings that appear in the list presented in Section 5.5. When we withdraw this restriction and allow an arbitrary preordered semiring to parametrize LDC, substitution no longer remains admissible in the calculus. In sum, LDC avoids the issues of McBride's system by restricting the parametrizing structures. This approach is in contrast to that taken by Atkey [2018], Moon et al. [2021] and GRAD, all of which solve the same problem, but by putting restrictions on the form of the typing judgment.

5.7.3 Other Calculi Related to LDC

LDC presents a unified perspective on usage and dependency analyses. There is a precedent to this presentation: Benton and Wadler [1996] observed that models of linear logic also provide models of Moggi's computational metalanguage. Now, broadly speaking, usage analysis may be seen as fine-grained linear logic in action whereas dependency analysis may be seen as fine-grained computational metalanguage in action. So, the unified perspective of LDC is in line with the observation made by Benton and Wadler [1996].

Next, linearity and dependency have been traditionally analyzed as a coeffect and an effect respectively. There is existing work in literature [Gaboardi et al., 2016] on combining effects and coeffects. Gaboardi et al. [2016] present a calculus that employs distinct graded modalities for analyzing coeffects and effects, which are then combined by allowing the modalities to interact via graded distributive laws. In contrast, LDC employs the same graded modality for analyzing both linearity and dependency, but draws the grades from different algebraic structures while carrying out the two analyses. Further, the graded modality employed in LDC is both monadic and comonadic, whereas the graded modalities employed in Gaboardi et al. [2016] are either monadic or comonadic, but not necessarily both. The main reason behind these differences is that while the motivation of Gaboardi et al. [2016] is a calculus for combining general coeffects and effects, our motivation is a calculus for specifically combining linearity and dependency. Owing to this specific nature, our calculus needs neither multiple modalities nor graded distributive laws. Finally, note that the calculus presented in Gaboardi et al. [2016] is simply-typed, whereas LDC is dependently-typed. Designing a dependently-typed calculus for combining general coeffects and effects remains an open problem.

5.8 Conclusion

In this chapter, we showed that linearity and dependency analyses can be systematically unified and combined into a single calculus. More specifically, we designed LDC, a calculus for combined linearity and dependency analysis in pure type systems. LDC, by analyzing both linearity and dependency using the same mechanism, seamlessly combines the two analyses into a single one. By virtue of this combination, LDC shines light on some of the nuances of these analyses, for example, the calculus shows that both no usage and run-time irrelevance can be analyzed in exactly the same way. We showed that linearity and dependency analyses in LDC are correct via soundness with respect to a heap-based semantics for the calculus. Now, with regard to linearity analysis, LDC is as expressive as GRAD, whereas with regard to dependency analysis, LDC is as expressive as $\text{DDC}^{\top}$. These results show that LDC is a general calculus for linearity and dependency analyses in dependent type systems. To sum up, LDC smoothly combines linearity and dependency analyses in the presence of dependent types — this is the key novel contribution of the calculus.

Chapter 6

Conclusion

In this dissertation, we designed calculi for dependency and linearity analyses in pure type systems. We first designed GMCC_e, a calculus for dependency analysis in simple type systems. GMCC_e has a nice categorical interpretation and subsumes standard dependency calculi from literature. Then, we extended GMCC_e (more accurately, its variant SDC) to $\text{DDC}^\top$ and DDC. Both $\text{DDC}^\top$ and DDC can analyze dependencies in pure type systems. However, DDC is more general than $\text{DDC}^\top$ because not only does it analyze dependencies but also it makes use of dependency analysis in the type system itself. We used $\text{DDC}^\top$ and DDC to analyze fine-grained notions of irrelevance in dependently-typed programs. Next, we designed GRAD, a calculus for linearity and other usage analyses in pure type systems. We used GRAD to reason about various forms of usages in dependently-typed programs, such as, no usage, linear usage, exact usage, bounded usage, etc. Thereafter, we designed LDC, a calculus for combined linearity and dependency analysis in pure type systems. LDC, essentially an integration of GRAD and $\text{DDC}^\top$, can reason about both usages and dependencies in pure type systems. The calculi we designed in this dissertation, i.e., GMCC_e, $\text{DDC}^\top$, DDC, GRAD and LDC, provide a systematic way for analyzing dependency, linearity or a combination of the two in any pure type system.

In this concluding chapter, we shall look back at the calculi we designed and reflect on the key ideas behind their design. To begin with, the design of our calculi is based on the observation that dependency and linearity analyses can be modeled as constrained interactions among a set of worlds. The worlds may be high-security and low-security with the constraint that information never leaks from the former to the latter, or they may be linear and nonlinear with the constraint the derivations in the latter do not make use of assumptions from the former. In Section 1.4.1, we saw that grading can help formalize multiple interacting worlds in a single system. This is the reason why grading is useful to dependency and linearity analyses, and thereby, to the design of our calculi. In fact, it is through the tools and techniques of grading that we built our calculi and addressed the various challenges that came along the way.

Our calculi employ grading in a variety of ways. GMCC_e, $\text{DDC}^\top$, DDC, GRAD and LDC all employ graded contexts. Additionally, all of them barring GRAD employ graded typing judgments; and all of them barring GMCC_e employ graded types and terms. Furthermore, DDC employs a graded definitional equality relation

in its type system. Mirroring the syntax, the semantics of these calculi also make extensive use of grading. The categorical semantics of GMCC_{e} employs graded monads and comonads. The indistinguishability relation of $\text{DDC}^{\top}$ and DDC employs grade-indexing. The heap reduction relation of GRAD and LDC employs grades as weights of assignments in the heap. In sum, we can say that grading is the cornerstone of all the calculi we designed in this dissertation.

Now, it is true that the dissertation focuses almost exclusively on dependency and linearity analyses. Nevertheless, the constructions related to grading that are presented here can turn out to be useful in addressing challenges pertaining to other analyses. So next, we shall summarize the key ideas related to grading that have helped us in analyzing dependency and linearity, ideas which could also be useful in other analyses as well.

6.1 Grading: The Key Ideas of Our Work

In Section 1.4, we discussed how grading helps formalize multiple interacting worlds in a single system. Such worlds, in and of themselves, may have very different sets of rules. But with the gadgets provided by grading, we can honor and accommodate these differences in the same system. Throughout this dissertation, we have seen these gadgets in action. Below, we recapitulate them one by one:

- **Graded Contexts:** A graded context is essentially a fusion of a standard context and a grade vector. The components of the grade vector signify the worlds the corresponding contextual assumptions belong to. Now, one could very well incorporate such information in the types of the assumptions themselves. But this approach is comparatively inflexible. This is so because it couples together contextual assumptions and the respective worlds they belong to. As a consequence, typing becomes intertwined with the analysis being carried out, thereby resulting in a less flexible type system. Section 1.1.2 discusses problems caused by such reduced flexibility in linear type systems.

  Graded contexts, on the other hand, decouple contextual assumptions and the respective worlds they belong to, thereby making way for a system where typing can be fairly orthogonal to the analysis being carried out. Such orthogonality is especially desirable in case of dependent type systems, where both types and terms can depend upon contextual assumptions. In such cases, orthogonality allows the type system to treat contextual assumptions differently in a term and in its type. In Section 1.1.8, we saw why this flexibility is necessary to combine linearity with dependent types. In Chapter 3, we saw why this flexibility is necessary for analyzing run-time irrelevance in dependent type systems. In fact, without this flexibility, we could not have designed our dependently-typed calculi for linearity and dependency analyses.

- **Graded Typing Judgments:** A graded typing judgment is a typing judgment that associates a grade with the term being typed. The grade signifies the world the term belongs to. Now, one could incorporate this information in the type of the term itself. But such an approach would again intertwine typing with the analysis being carried out, just as we discussed above. Graded typing judgments, on

the other hand, decouple the term and the world it belongs to, thereby making way for a system where typing can be fairly orthogonal to the analysis being carried out. Below, we illustrate this point with an example.

Recall that we chose GMCC_e and not GMCC as the basis for designing our dependency calculi for dependent type systems. One of the reasons behind this choice is that GMCC_e, unlike GMCC, employs graded typing judgments, which provide the flexibility to treat a term and its type differently, with regard to dependency constraints. Chapter 3 shows why such flexibility is essential to analyzing run-time irrelevance in dependent type systems. To see how the type system makes use of this flexibility, we refer to rule DCT-Lam. This rule checks the term $\lambda^{\ell_0} x : A.b$ at ℓ but its type $\Pi x :^{\ell_0} A.B$ at $\top$, meaning, the type is not subjected to any dependency constraints whereas the term is. Without the flexibility offered by graded typing judgments, this differential treatment of a term and its type would have been difficult to attain. In this regard, note that graded contexts can, to a certain extent, fill in the role played by the grade on the typing judgment. But graded contexts are not a substitute for graded typing judgments, as we saw in Section 5.1.1. In fact, graded contexts and graded typing judgments complement each other and can be used together for designing type systems, as we did in designing $\text{DDC}^\top$, DDC and LDC.

- **Graded Types:** A graded type is a type that directly includes a grade annotation. Note that types that include grade annotations via modalities only are not referred to as graded types, but as graded modal types. To give an example, $^{\mathbf{H}}\mathbf{Bool} \to \mathbf{Bool}$ is a graded type but $S_{\mathbf{H}}\,\mathbf{Bool} \to \mathbf{Bool}$ is a graded modal type. In simple type systems, a graded type can usually be replaced with the corresponding graded modal type. However, in dependent type systems, graded types cannot be replaced with graded modal types because the former offer more flexibility over the latter, as pointed out in Sections 3.4.2 and 4.6.2. By decoupling the grade from the type, graded types allow the same bound variable to be used in the body of a term and in the body of its type under restrictions that are totally different from one another. With graded modal types, such a treatment is not possible. Refer to Sections 3.4.2 and 4.6.2 for concrete examples that show why.

- **Graded Terms:** A graded term is a term that includes a grade annotation. For example, $\lambda^{\ell_0} x : A.b$ and (a^{ℓ_0}, b) and $\mathbf{lock}^{\ell_0}\ a$ are all graded terms. We have made extensive use of graded terms in all our calculi. In our calculi, graded terms serve a key purpose. They enable us to reason about dependency and linearity constraints without considering typing information. For instance, the indexed indistinguishability relation, which we used to prove noninterference, is untyped; but still, the relation can reason about dependencies because grade annotations on terms convey the necessary information: refer to Section 3.2.3, and in particular, rule Ind-Lock, to see how. Similarly, the heap reduction relation, which we used to prove soundness of linearity analysis, is untyped; but still, the relation can reason about usages because grade annotations on terms convey the necessary information: refer to Section 4.3.2, and in particular, rules Heap-AppBeta and Heap-LetPairBeta, to see how. In this way, graded terms support reasoning about the analysis being carried out.

- **Graded Equality Relation:** A graded equality relation is a family of equality relations, indexed by grades. For example, the definitional equality relation of DDC, presented in Section 3.4.2, is a graded

equality relation. A graded equality relation helps us switch between different notions of equality that we may need to work with. Different notions of equality, corresponding to different worlds, emerge naturally in graded type systems, as we saw in Sections 1.4.4 and 3.2.3. Now, when there are multiple notions of equality, we may need to work with one notion of equality sometimes whereas another one at other times. A graded equality relation enables smooth switching between equality relations, as we saw in case of DDC$^{\top}$ and DDC, which use the equality relation at grades $\top$ and $\mathcal{C}$ respectively.

Graded contexts, graded typing judgments, graded types, graded terms and graded equality relations have helped us design our calculi for dependency and linearity analyses. The key benefit of using these gadgets is that they help decouple the analyses from typing, thereby enabling simpler design. In the next section, we shall point out some other analyses that may benefit from such decoupling.

Grading is undoubtedly the main tool we use in the syntactic design of our calculi. However, our use of grading is not limited to syntactic design. We also make use of grading in the semantics of our calculi. Throughout this dissertation, we saw how grades shape the semantics we develop. The semantic role played by grades should not come as a surprise, given that grades signify worlds, which are indeed semantic in nature. Further, the semantic role played by grades is no less important than their syntactic role because it is through 'graded semantics' and not standard semantics that we proved soundness of our calculi. So next, we recap the role played by grades in semantics:

- **Graded Monads and Comonads:** In Chapter 2, we used graded monads and comonads to give semantics to our dependency calculi, GMCC and GMCC$_{\text{e}}$. This semantics helped us prove noninterference for the calculi in a relatively straightforward manner, via the presence-absence test. In that chapter, we also gave semantics to λ° using graded monads and comonads and proved noninterference for the calculus using the same presence-absence test. Now, recall how the presence-absence test works: it checks for noninterference of world ℓ_2 in world ℓ_1 by checking if world ℓ_1 can be 'present' while world ℓ_2 is 'absent'. To carry out such a test, we need a semantics that models both the worlds under consideration. We can design such a semantics by grading monads and comonads, as we saw in Chapter 2.

- **Indexed Indistinguishability Relation:** In Chapter 3, we used the indexed indistinguishability relation to show noninterference for our dependency calculi, SDC, DDC$^{\top}$ and DDC. This relation is indexed by grades, which signify the perspective of observers from the corresponding worlds. The relation is defined by taking only grade information (but not typing information) into account. As such, almost the same relation can be used for both the simply-typed and dependently-typed calculi. This design results in relatively easy proof of noninterference for the dependently-typed calculi. So, the indexed indistinguishability relation shows that grade-only relations can help us prove soundness properties, and that too, in an easy way.

- **Heap Reduction Relation:** In Chapter 4, we used the heap reduction relation to show soundness of usage analysis in GLC and GRAD. In Chapter 5, we used this relation to show soundness of both

usage and dependency analyses in SLDC and LDC. Now, the heap reduction relation associates grades with both the redex and the assignments in the heap and uses these grades to check whether any given reduction is fair. Put differently, this relation ensures fairness just by considering the grades and without taking typing information into account. As such, the relation has a simple definition; but it can be used to prove powerful results, such as corollaries 4.27 and 4.28.

So, we see that grades have played a vital role in the semantics of our calculi as well. The key benefit of using grades in semantics is that they help bypass typing, thereby facilitating easier proofs of soundness. In sum, on both syntactic and semantic fronts, grades help decouple typing from the analysis being carried out, which facilitates simpler design of calculi and easier proofs of soundness.

Next, we discuss some areas other than dependency and linearity analyses that may benefit from our work on grading.

6.2 Areas That May Benefit from Our Work on Grading

The power of grading is best utilized in dependent type systems. As we have seen, grading can support complex analyses in dependent type systems by enabling an orthogonal treatment of typing and the concerned analysis. The full spectrum of analyses (in dependent type systems) that can benefit from grading is, however, difficult to ascertain at this point, given that research in this field is far from complete. Some researchers are indeed working on this question, in its various facets; but as yet, we don't have a complete answer. So here, we shall not try to tackle this question head-on but shall just point out *some* other analyses that could potentially benefit from grading.

6.2.1 Dependent Session Types

The π-calculus, introduced by Milner et al. [1992], is a theory of interacting systems. The calculus models interactions in terms of agents or processes, which communicate via channels. In its initial presentation, the calculus didn't have any types. But over time, typing formalisms were added to the calculus to express precise invariants on interactions. One such formalism is provided by session types [Honda et al., 1998]. In this formalism, exactly two processes communicate via a channel in a dual manner, for instance, if one process sends a message, the other process receives the message; similarly, if one process offers choices, the other process chooses from them. Now, session types are closely connected to linear logic. In fact, Caires and Pfenning [2010] showed that a system of session types for the π-calculus corresponds exactly to the sequent calculus proof system for Dual Intuitionistic Linear Logic (DILL), introduced in Section 1.1.3. This system of session types, referred to as πDILL, closely resembles DILL in its presentation. For example, πDILL includes the standard connectives of linear logic, which are introduced and eliminated via rules that segregate contextual assumptions into nonlinear and linear zones, just like DILL. Now, πDILL is simply-typed but it was later extended to dependent types by Toninho et al. [2011]. Their system may be viewed

as a theory of session types for formalizing interactions over a base dependently-typed language. Next, we shall briefly review their system, discuss its limitations and indicate how our work on grading might address those limitations.

We motivate the system of Toninho et al. [2011] with the examples they provide. A minimal bank service may be specified through the session type: $\texttt{String} \multimap ((\texttt{Nat} \multimap (\texttt{Nat} \otimes \mathbf{1})) \,\&\, (\texttt{Nat} \otimes \mathbf{1}))$. This service requires a customer to first input the account id (a string), after which the customer is offered two choices: either input an amount to deposit (a natural number) and get a receipt of that deposit (another natural number) or find out the account balance (a natural number); the process ends thereafter. Now, this service specification can be made much more precise using dependent types. For instance, the receipt for the deposit can specify the account id and the amount deposited; the 'balance statement' can include the account id in addition to the account balance. To support these functionalities, we can refine the type as:

$$\forall s : \texttt{String}.\texttt{valid}(s) \multimap ((\forall n : \texttt{Nat}.\texttt{dep}(s, n) \multimap \texttt{rcpt}(s, n) \otimes \mathbf{1}) \,\&\, (\exists m : \texttt{Nat}.\texttt{bal}(s, m) \otimes \mathbf{1}))$$

The type above specifies that the bank, upon receiving an account id s from a customer, checks if the id is valid, and then allows the customer to choose between deposit and balance query. If the customer chooses to deposit, the bank asks for the deposit amount and the deposit order (which also notes the account id and the amount); upon receiving them, the bank sends a receipt of the deposit, noting down the account id and the amount deposited. If the customer chooses to query the balance, the bank sends the balance amount and an appropriate 'balance statement'. Now, this refined type is more precise and better reflects interactions between banks and customers. This type is an example of a dependent session type and is supported by the system of Toninho et al. [2011].

Observe that the type above employs connectives of linear logic and is dependent in nature. Since the system of Toninho et al. [2011] can express types like this one, it may be viewed as a linear dependent type system. As a matter of fact, it may be viewed as an extension of DILL to dependent types. Now, in Section 1.1.4, we pointed out the key limitation of linear dependent type systems that extend split-context systems like DILL to dependent types: in such systems, types cannot depend upon linear assumptions. The system of Toninho et al. [2011] also faces this limitation. For instance, the following type cannot be expressed in their system:

$$(s : \texttt{String}) \multimap \texttt{valid}(s) \multimap ((\forall n : \texttt{Nat}.\texttt{dep}(s, n) \multimap \texttt{rcpt}(s, n) \otimes \mathbf{1}) \,\&\, (\exists m : \texttt{Nat}.\texttt{bal}(s, m) \otimes \mathbf{1}))$$

The type above specifies a service where the account id received from the customer is used exactly once, say, while verifying the validity of the id. Note that the id may be used arbitrarily in the body of the type, but exactly once in the body of the term. Types like the one above express finer invariants and so, allowing them could be practically useful. From a theoretical point of view, allowing such types could lead to a more general calculus for dependent session types. Now, to allow such types, one could follow our work on grading.

Recall that in Section 1.1.8, we discussed how graded-context dependent systems (as opposed to split-context dependent systems) allow types to depend upon linear assumptions. Now, GRAD is a graded-context dependent system and offers this flexibility. Though designed for a different purpose, I believe that GRAD

can be suitably molded to serve as a calculus for dependent session types. In its new form, GRAD could allow session types to depend upon linear assumptions. To realize this goal, work needs to be done on GRAD, but the goal is worth pursuing. This is so because GRAD can bring not only the above-mentioned flexibility but also several other benefits of grading to the arena of dependent session types. For instance, Toninho et al. [2011] use a separate mechanism for enforcing proof irrelevance in their system of dependent session types, but with GRAD, such a mechanism would be unnecessary because GRAD can already enforce proof irrelevance via no usage. To the best of my knowledge, the utility of grading, especially that of graded-context systems, has not been explored with regard to dependent session types because the trend in literature on this topic [Thiemann and Vasconcelos, 2019, Toninho and Yoshida, 2018] has been to employ split-context systems. From Sections 1.1.7 and 6.1, we know the advantages of graded-context systems over split-context systems. I believe that these advantages can be carried over to the arena of dependent session types and the exercise would result in a more general and useful theory of the subject.

6.2.2 Generic Type and Effect System for Dependent Types

In Section 1.4.3, we discussed how type and effect systems employ grading to reason about side effects of programs. Now, the type and effect systems in literature work with simple/polymorphic types only. To the best of my knowledge, there is no generic type and effect system for dependent types, even though such a system [Marino and Millstein, 2009] has been designed for simple types more than a decade ago. Undoubtedly, the problem of designing a generic type and effect system for dependent types is challenging because one additionally needs to consider the possibility of side effects of types themselves.

Though not in the context of type and effect systems, some researchers [Ahman et al., 2016, Vákár, 2016] have tackled the above-mentioned challenge, but by devising a workaround. Ahman et al. [2016] and Vákár [2016] combine effects and dependent types but require types to depend upon values only, and not upon computations that may have effects. To impose this restriction, they need a careful analysis of values and computations in the style of a call-by-push-value calculus [Levy, 2003], with the result being that their effect calculi are fairly complex. Now, the restriction Ahman et al. [2016] and Vákár [2016] impose on dependent types may remind the reader of another restriction: types should not depend upon linear assumptions. Recall that the initial linear dependent type systems [Krishnaswami et al., 2015, Vákár, 2015] imposed this restriction. But with the flexibility offered by grading, we did away with it and in the process, designed a simpler and more uniform linear dependent type system. This experience suggests to me that the restriction Ahman et al. [2016] and Vákár [2016] impose on dependent types can also be done away with, thereby paving the way for a simpler and more uniform effect calculus for dependent types. I believe that type and effect systems, with their intrinsic graded nature, can show us how to design such a calculus. So next, I discuss some ideas that might help design a generic type and effect system for dependent types.

Marino and Millstein [2009] design a generic type and effect system for simple types, which could serve as our starting point. Their key insight is that effects can be represented in terms of their duals, privileges (or capabilities), which are appropriately granted and checked by the type system. For instance, if the effects

under consideration are exceptions, the privilege (or capability) can be `canThrow`, which is granted in the body of a `try` expression and checked (whether it is held) in a `throw` expression. One can think of privileges as grades, granting privileges as operating on grades and checking privileges as checking relations between grades. Now, using the insight mentioned above, Marino and Millstein [2009] design a type and effect system where typing and effect analysis (done via privileges) are fairly orthogonal to each other. This orthogonality is similar to the orthogonality that helped us extend dependency and linearity analyses to dependent type systems. I think that its underlying orthogonality can help us extend the system of Marino and Millstein [2009] to dependent types. This orthogonality can also help us treat types and terms on an equal footing in a dependent setting, thereby not requiring us to carry out a value/computation analysis as in Ahman et al. [2016] and Vákár [2016].

6.3 Epilogue

We have walked a long way. Starting out with the goal of analyzing dependency and linearity in pure type systems, we have walked across many uncharted terrains, both rough and smooth. In this journey, we faced several challenges but fortunately, with the guidance of the wisdom from the past, we have been able to overcome them. Now, when I look back, I see that our journey has made new trails on the terrains we walked across. I sincerely believe that these trails will help future travelers who tread this way.

Appendix A

Dependency Analysis in Simple Type Systems

A.1 Proofs of Lemmas/Theorems Stated in Section 2.2

Theorem A.1 (Theorem 2.1) If $\Gamma \vdash a : A$ in GMC, then $[\![a]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], [\![A]\!])$. Further, if $\Gamma \vdash a_1 : A$ and $\Gamma \vdash a_2 : A$ such that $a_1 \equiv a_2$ in GMC, then $[\![a_1]\!] = [\![a_2]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], [\![A]\!])$.

Proof. Let $\Gamma \vdash a : A$. We show $[\![a]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], [\![A]\!])$ by induction on the typing derivation.

- λ-calculus. Standard.
- Rule M-Return. Have: $\Gamma \vdash \mathbf{ret}\ a : T_1\ A$ where $\Gamma \vdash a : A$.
  By IH, $[\![a]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], [\![A]\!])$.
  Now, $[\![\mathbf{ret}\ a]\!] = [\![\Gamma]\!] \xrightarrow{[\![a]\!]} [\![A]\!] \xrightarrow{\eta_{[\![A]\!]}} \mathbf{T}_1[\![A]\!]$.
  Therefore, $[\![\mathbf{ret}\ a]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], [\![T_1\ A]\!])$.
- Rule M-Fmap. Have: $\Gamma \vdash \mathbf{lift}^m f : T_m\ A \to T_m\ B$ where $\Gamma \vdash f : A \to B$.
  By IH, $[\![\Gamma]\!] \xrightarrow{[\![f]\!]} [\![B]\!]^{[\![A]\!]}$.
  Now, $[\![\mathbf{lift}^m f]\!] = \Lambda\Big([\![\Gamma]\!] \times \mathbf{T}_m[\![A]\!] \xrightarrow{t^{\mathbf{T}_m}_{[\![\Gamma]\!],[\![A]\!]}} \mathbf{T}_m([\![\Gamma]\!] \times [\![A]\!]) \xrightarrow{\mathbf{T}_m(\Lambda^{-1}[\![f]\!])} \mathbf{T}_m[\![B]\!]\Big)$.
  Therefore, $[\![\mathbf{lift}^m f]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], [\![T_m\ A \to T_m\ B]\!])$.
- Rule M-Join. Have: $\Gamma \vdash \mathbf{join}^{m_1,m_2} a : T_{m_1 \cdot m_2}\ A$ where $\Gamma \vdash a : T_{m_1}\ T_{m_2}\ A$.
  By IH, $[\![\Gamma]\!] \xrightarrow{[\![a]\!]} \mathbf{T}_{m_1}\mathbf{T}_{m_2}[\![A]\!]$.
  Now, $[\![\mathbf{join}^{m_1,m_2} a]\!] = [\![\Gamma]\!] \xrightarrow{[\![a]\!]} \mathbf{T}_{m_1}\mathbf{T}_{m_2}[\![A]\!] \xrightarrow{\mu^{m_1,m_2}_{[\![A]\!]}} \mathbf{T}_{m_1 \cdot m_2}[\![A]\!]$.
  Therefore, $[\![\mathbf{join}^{m_1,m_2} a]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], [\![T_{m_1 \cdot m_2}\ A]\!])$.

- Rule M-Up. Have: $\Gamma \vdash \mathbf{up}^{m_1,m_2}\, a : T_{m_2}\ A$ where $\Gamma \vdash a : T_{m_1}\ A$ and $m_1 <: m_2$.
  By IH, $[\![\Gamma]\!] \xrightarrow{[\![a]\!]} \mathbf{T}_{m_1}[\![A]\!]$.
  Now, $[\![\mathbf{up}^{m_1,m_2}\, a]\!] = [\![\Gamma]\!] \xrightarrow{[\![a]\!]} \mathbf{T}_{m_1}[\![A]\!] \xrightarrow{\mathbf{T}^{m_1 <: m_2}_{[\![A]\!]}} \mathbf{T}_{m_2}[\![A]\!]$.
  Therefore, $[\![\mathbf{up}^{m_1,m_2}\, a]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], [\![T_{m_2}\ A]\!])$.

Next, we show that if $\Gamma \vdash a_1 : A$ and $\Gamma \vdash a_2 : A$ such that $a_1 \equiv a_2$ in GMC, then $[\![a_1]\!] = [\![a_2]\!]$.
By inversion on $a_1 \equiv a_2$:

- λ-calculus. Standard.
- $\mathbf{lift}^m(\lambda x.x) \equiv \lambda x.x$.
  Given: $\Gamma \vdash \mathbf{lift}^m(\lambda x : A.x) : T_m\ A \to T_m\ A$ and $\Gamma \vdash \lambda x : T_m\ A.x : T_m\ A \to T_m\ A$.
  Now,

$$\begin{aligned}
&[\![\mathbf{lift}^m(\lambda x : A.x)]\!] \\
&= \Lambda\Big(\mathbf{T}_m(\Lambda^{-1}[\![\lambda x : A.x]\!]) \circ t^{\mathbf{T}_m}_{[\![\Gamma]\!],[\![A]\!]}\Big) \\
&= \Lambda(\mathbf{T}_m \pi_2 \circ t^{\mathbf{T}_m}_{[\![\Gamma]\!],[\![A]\!]}) \\
&= \Lambda\pi_2\ \text{[By naturality and strength]} \\
&= [\![\lambda x : T_m\ A.x]\!].
\end{aligned}$$

- $\mathbf{lift}^m(\lambda x.g\ (f\ x)) \equiv \lambda x.(\mathbf{lift}^m g)\ ((\mathbf{lift}^m f)\ x)$.
  Given: $\Gamma \vdash \mathbf{lift}^m(\lambda x : A.g\ (f\ x)) : T_m\ A \to T_m\ C$ and $\Gamma \vdash \lambda x : T_m\ A.(\mathbf{lift}^m g)\ ((\mathbf{lift}^m f)\ x) : T_m\ A \to T_m\ C$ where $\Gamma \vdash f : A \to B$ and $\Gamma \vdash g : B \to C$.
  Now,

$$\begin{aligned}
&[\![\mathbf{lift}^m(\lambda x : A.g\ (f\ x))]\!] \\
&= \Lambda\Big(\mathbf{T}_m(\Lambda^{-1}[\![\lambda x : A.g\ (f\ x)]\!]) \circ t^{\mathbf{T}_m}_{[\![\Gamma]\!],[\![A]\!]}\Big) \\
&= \Lambda\Big(\mathbf{T}_m(\Lambda^{-1}[\![g]\!] \circ \langle \pi_1, \Lambda^{-1}[\![f]\!]\rangle) \circ t^{\mathbf{T}_m}_{[\![\Gamma]\!],[\![A]\!]}\Big).
\end{aligned}$$

  Next,

$$\begin{aligned}
&[\![\lambda x : T_m\ A.(\mathbf{lift}^m g)\ ((\mathbf{lift}^m f)\ x)]\!] \\
&= \Lambda\Big(\Lambda^{-1}[\![\mathbf{lift}^m g]\!] \circ \langle \pi_1, \Lambda^{-1}\mathbf{lift}^m f\rangle\Big) \\
&= \Lambda\Big(\mathbf{T}_m(\Lambda^{-1}[\![g]\!]) \circ t^{\mathbf{T}_m}_{[\![\Gamma]\!],[\![B]\!]} \circ \langle \pi_1, \mathbf{T}_m(\Lambda^{-1}[\![f]\!]) \circ t^{\mathbf{T}_m}_{[\![\Gamma]\!],[\![A]\!]}\rangle\Big).
\end{aligned}$$

  The morphisms above are equal because the squares below commute:

  The right square commutes by naturality whereas the left one commutes because the diagram below commutes:

$$
\begin{array}{ccccc}
\llbracket\Gamma\rrbracket \times \mathbf{T}_m\llbracket A\rrbracket & \xrightarrow{\langle \pi_1, t^{\mathbf{T}_m}_{\llbracket\Gamma\rrbracket,\llbracket A\rrbracket}\rangle} & \llbracket\Gamma\rrbracket \times \mathbf{T}_m(\llbracket\Gamma\rrbracket \times \llbracket A\rrbracket) & \xrightarrow{\mathrm{id}_{\llbracket\Gamma\rrbracket} \times \mathbf{T}_m(\Lambda^{-1}\llbracket f\rrbracket)} & \llbracket\Gamma\rrbracket \times \mathbf{T}_m\llbracket B\rrbracket \\
\downarrow{\scriptstyle t^{\mathbf{T}_m}_{\llbracket\Gamma\rrbracket,\llbracket A\rrbracket}} & & \downarrow{\scriptstyle t^{\mathbf{T}_m}_{\llbracket\Gamma\rrbracket,\llbracket\Gamma\rrbracket\times\llbracket A\rrbracket}} & & \downarrow{\scriptstyle t^{\mathbf{T}_m}_{\llbracket\Gamma\rrbracket,\llbracket B\rrbracket}} \\
\mathbf{T}_m(\llbracket\Gamma\rrbracket \times \llbracket A\rrbracket) & \xrightarrow{\mathbf{T}_m\langle \pi_1, \mathrm{id}_{\llbracket\Gamma\rrbracket\times\llbracket A\rrbracket}\rangle} & \mathbf{T}_m(\llbracket\Gamma\rrbracket \times (\llbracket\Gamma\rrbracket \times \llbracket A\rrbracket)) & \xrightarrow{\mathbf{T}_m(\mathrm{id}_{\llbracket\Gamma\rrbracket} \times \Lambda^{-1}\llbracket f\rrbracket)} & \mathbf{T}_m(\llbracket\Gamma\rrbracket \times \llbracket B\rrbracket)
\end{array}
$$

$$
\begin{array}{ccccc}
\llbracket\Gamma\rrbracket \times \mathbf{T}_m\llbracket A\rrbracket & & \xrightarrow{\qquad t^{\mathbf{T}_m}_{\llbracket\Gamma\rrbracket,\llbracket A\rrbracket} \qquad} & & \mathbf{T}_m(\llbracket\Gamma\rrbracket \times \llbracket A\rrbracket) \\
\downarrow{\scriptstyle \langle \mathrm{id}_{\llbracket\Gamma\rrbracket}, \mathrm{id}_{\llbracket\Gamma\rrbracket}\rangle \times \mathbf{T}_m \mathrm{id}_{\llbracket A\rrbracket}} & & & & \downarrow{\scriptstyle \mathbf{T}_m(\langle \mathrm{id}_{\llbracket\Gamma\rrbracket}, \mathrm{id}_{\llbracket\Gamma\rrbracket}\rangle \times \mathrm{id}_{\llbracket A\rrbracket})} \\
(\llbracket\Gamma\rrbracket \times \llbracket\Gamma\rrbracket) \times \mathbf{T}_m\llbracket A\rrbracket & & \xrightarrow{\qquad t^{\mathbf{T}_m}_{\llbracket\Gamma\rrbracket\times\llbracket\Gamma\rrbracket,\llbracket A\rrbracket} \qquad} & & \mathbf{T}_m((\llbracket\Gamma\rrbracket \times \llbracket\Gamma\rrbracket) \times \llbracket A\rrbracket) \\
\downarrow{\scriptstyle \alpha^{-1}_{\llbracket\Gamma\rrbracket,\llbracket\Gamma\rrbracket,\mathbf{T}_m\llbracket A\rrbracket}} & & & & \downarrow{\scriptstyle \mathbf{T}_m \alpha^{-1}_{\llbracket\Gamma\rrbracket,\llbracket\Gamma\rrbracket,\llbracket A\rrbracket}} \\
\llbracket\Gamma\rrbracket \times (\llbracket\Gamma\rrbracket \times \mathbf{T}_m\llbracket A\rrbracket) & \xrightarrow{\mathrm{id}_{\llbracket\Gamma\rrbracket} \times t^{\mathbf{T}_m}_{\llbracket\Gamma\rrbracket,\llbracket A\rrbracket}} & \llbracket\Gamma\rrbracket \times \mathbf{T}_m(\llbracket\Gamma\rrbracket \times \llbracket A\rrbracket) & \xrightarrow{t^{\mathbf{T}_m}_{\llbracket\Gamma\rrbracket,\llbracket\Gamma\rrbracket\times\llbracket A\rrbracket}} & \mathbf{T}_m(\llbracket\Gamma\rrbracket \times (\llbracket\Gamma\rrbracket \times \llbracket A\rrbracket))
\end{array}
$$

FIGURE A.1: Commutative diagram

The square above commutes by naturality whereas the rectangle below commutes by strength.

- $\mathbf{up}^{m_1,m_1}\, a \equiv a$.
  Given: $\Gamma \vdash \mathbf{up}^{m_1,m_1}\, a : T_{m_1}\ A$ where $\Gamma \vdash a : T_{m_1}\ A$.
  Since $\mathbf{T}$ is a functor, $\mathbf{T}^{m_1 <: m_1}_{\llbracket A\rrbracket} = \mathrm{id}_{\mathbf{T}_{m_1}\llbracket A\rrbracket}$.
  Therefore, $\llbracket \mathbf{up}^{m_1,m_1}\, a\rrbracket = \mathbf{T}^{m_1 <: m_1}_{\llbracket A\rrbracket} \circ \llbracket a\rrbracket = \llbracket a\rrbracket$.

- $\mathbf{up}^{m_2,m_3}(\mathbf{up}^{m_1,m_2}\, a) \equiv \mathbf{up}^{m_1,m_3}\, a$.
  Given: $\Gamma \vdash \mathbf{up}^{m_2,m_3}(\mathbf{up}^{m_1,m_2}\, a) : T_{m_3}\ A$ and $\Gamma \vdash \mathbf{up}^{m_1,m_3}\, a : T_{m_3}\ A$ where $\Gamma \vdash a : T_{m_1}\ A$ and $m_1 <: m_2$ and $m_2 <: m_3$.
  Since $\mathbf{T}$ is a functor, $\mathbf{T}^{m_2 <: m_3}_{\llbracket A\rrbracket} \circ \mathbf{T}^{m_1 <: m_2}_{\llbracket A\rrbracket} = \mathbf{T}^{m_1 <: m_3}_{\llbracket A\rrbracket}$.
  Therefore, $\llbracket \mathbf{up}^{m_2,m_3}(\mathbf{up}^{m_1,m_2}\, a)\rrbracket = \mathbf{T}^{m_2 <: m_3}_{\llbracket A\rrbracket} \circ \mathbf{T}^{m_1 <: m_2}_{\llbracket A\rrbracket} \circ \llbracket a\rrbracket = \mathbf{T}^{m_1 <: m_3}_{\llbracket A\rrbracket} \circ \llbracket a\rrbracket = \llbracket \mathbf{up}^{m_1,m_3}\, a\rrbracket$.

- $(\mathbf{up}^{m_1,m_1'}\, a)\ {}^{m_1'}\!\!\gg\!\!=^{m_2}\ f \equiv \mathbf{up}^{m_1\cdot m_2, m_1'\cdot m_2}(a\ {}^{m_1}\!\!\gg\!\!=^{m_2}\ f)$.
  Given: $\Gamma \vdash (\mathbf{up}^{m_1,m_1'}\, a)\ {}^{m_1'}\!\!\gg\!\!=^{m_2}\ f : T_{m_1'\cdot m_2}\ B$ and $\Gamma \vdash \mathbf{up}^{m_1\cdot m_2, m_1'\cdot m_2}(a\ {}^{m_1}\!\!\gg\!\!=^{m_2}\ f) : T_{m_1'\cdot m_2}\ B$ where $\Gamma \vdash a : T_{m_1}\ A$ and $\Gamma \vdash f : A \to T_{m_2}\ B$ and $m_1 <: m_1'$.
  Now,

$$
\begin{aligned}
& \llbracket (\mathbf{up}^{m_1,m_1'}\, a)\ {}^{m_1'}\!\!\gg\!\!=^{m_2}\ f\rrbracket \\
= & \llbracket \mathbf{join}^{m_1',m_2}((\mathbf{lift}^{m_1'} f)\ (\mathbf{up}^{m_1,m_1'}\, a))\rrbracket \\
= & \mu^{m_1',m_2}_{\llbracket B\rrbracket} \circ \llbracket (\mathbf{lift}^{m_1'} f)\ (\mathbf{up}^{m_1,m_1'}\, a)\rrbracket \\
= & \mu^{m_1',m_2}_{\llbracket B\rrbracket} \circ \mathrm{app} \circ \langle \llbracket \mathbf{lift}^{m_1'} f\rrbracket, \llbracket \mathbf{up}^{m_1,m_1'}\, a\rrbracket\rangle
\end{aligned}
$$

$$
\begin{aligned}
&= \mu^{m_1', m_2}_{[\![B]\!]} \circ \mathrm{app} \circ \langle \Lambda\Big(\mathbf{T}_{m_1'}(\Lambda^{-1}[\![f]\!]) \circ t^{\mathbf{T}_{m_1'}}_{[\![\Gamma]\!],[\![A]\!]}\Big), \mathbf{T}^{m_1 <: m_1'}_{[\![A]\!]} \circ [\![a]\!] \rangle \\
&= \mu^{m_1', m_2}_{[\![B]\!]} \circ \mathbf{T}_{m_1'}(\Lambda^{-1}[\![f]\!]) \circ t^{\mathbf{T}_{m_1'}}_{[\![\Gamma]\!],[\![A]\!]} \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, \mathbf{T}^{m_1 <: m_1'}_{[\![A]\!]} \circ [\![a]\!] \rangle \\
&= \mu^{m_1', m_2}_{[\![B]\!]} \circ \mathbf{T}_{m_1'}(\Lambda^{-1}[\![f]\!]) \circ t^{\mathbf{T}_{m_1'}}_{[\![\Gamma]\!],[\![A]\!]} \circ (\mathrm{id}_{[\![\Gamma]\!]} \times \mathbf{T}^{m_1 <: m_1'}_{[\![A]\!]}) \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![a]\!] \rangle.
\end{aligned}
$$

Next,

$$
\begin{aligned}
& [\![\mathbf{up}^{m_1 \cdot m_2, m_1' \cdot m_2}(a \ {}^{m_1}\!\!\gg\!\!=^{m_2} f)]\!] \\
&= \mathbf{T}^{m_1 \cdot m_2 <: m_1' \cdot m_2}_{[\![B]\!]} \circ [\![a \ {}^{m_1}\!\!\gg\!\!=^{m_2} f]\!] \\
&= \mathbf{T}^{m_1 \cdot m_2 <: m_1' \cdot m_2}_{[\![B]\!]} \circ [\![\mathbf{join}^{m_1, m_2}((\mathbf{lift}^{m_1} f)\ a)]\!] \\
&= \mathbf{T}^{m_1 \cdot m_2 <: m_1' \cdot m_2}_{[\![B]\!]} \circ \mu^{m_1, m_2}_{[\![B]\!]} \circ [\![(\mathbf{lift}^{m_1} f)\ a]\!] \\
&= \mathbf{T}^{m_1 \cdot m_2 <: m_1' \cdot m_2}_{[\![B]\!]} \circ \mu^{m_1, m_2}_{[\![B]\!]} \circ \mathrm{app} \circ \langle \Lambda\Big(\mathbf{T}_{m_1}(\Lambda^{-1}[\![f]\!]) \circ t^{\mathbf{T}_{m_1}}_{[\![\Gamma]\!],[\![A]\!]}\Big), [\![a]\!] \rangle \\
&= \mathbf{T}^{m_1 \cdot m_2 <: m_1' \cdot m_2}_{[\![B]\!]} \circ \mu^{m_1, m_2}_{[\![B]\!]} \circ \mathbf{T}_{m_1}(\Lambda^{-1}[\![f]\!]) \circ t^{\mathbf{T}_{m_1}}_{[\![\Gamma]\!],[\![A]\!]} \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![a]\!] \rangle.
\end{aligned}
$$

The morphisms above are equal because the diagram below commutes:

$$
\begin{array}{ccccccc}
[\![\Gamma]\!] \times \mathbf{T}_{m_1}[\![A]\!] & \xrightarrow{t^{\mathbf{T}_{m_1}}_{[\![\Gamma]\!],[\![A]\!]}} & \mathbf{T}_{m_1}([\![\Gamma]\!] \times [\![A]\!]) & \xrightarrow{\mathbf{T}_{m_1}(\Lambda^{-1}[\![f]\!])} & \mathbf{T}_{m_1}\mathbf{T}_{m_2}[\![B]\!] & \xrightarrow{\mu^{m_1, m_2}_{[\![B]\!]}} & \mathbf{T}_{m_1 \cdot m_2}[\![B]\!] \\
\downarrow{\scriptstyle \mathrm{id}_{[\![\Gamma]\!]} \times \mathbf{T}^{m_1 <: m_1'}_{[\![A]\!]}} & & \downarrow{\scriptstyle \mathbf{T}^{m_1 <: m_1'}_{[\![\Gamma]\!] \times [\![A]\!]}} & & \downarrow{\scriptstyle \mathbf{T}^{m_1 <: m_1'}_{\mathbf{T}_{m_2}[\![B]\!]}} & & \downarrow{\scriptstyle \mathbf{T}^{m_1 \cdot m_2 <: m_1' \cdot m_2}_{[\![B]\!]}} \\
[\![\Gamma]\!] \times \mathbf{T}_{m_1'}[\![A]\!] & \xrightarrow{t^{\mathbf{T}_{m_1'}}_{[\![\Gamma]\!],[\![A]\!]}} & \mathbf{T}_{m_1'}([\![\Gamma]\!] \times [\![A]\!]) & \xrightarrow{\mathbf{T}_{m_1'}(\Lambda^{-1}[\![f]\!])} & \mathbf{T}_{m_1'}\mathbf{T}_{m_2}[\![B]\!] & \xrightarrow{\mu^{m_1', m_2}_{[\![B]\!]}} & \mathbf{T}_{m_1' \cdot m_2}[\![B]\!]
\end{array}
$$

The leftmost square commutes because $\mathbf{T}^{m_1 <: m_1'}$ is a *strong* natural transformation; the middle one commutes because $\mathbf{T}^{m_1 <: m_1'}$ is a natural transformation; the rightmost one commutes because μ is natural in its first component.

- $a \ {}^{m_1}\!\!\gg\!\!=^{m_2'} (\lambda x.\mathbf{up}^{m_2, m_2'} b) \equiv \mathbf{up}^{m_1 \cdot m_2, m_1 \cdot m_2'}(a \ {}^{m_1}\!\!\gg\!\!=^{m_2} \lambda x.b)$.
  Given: $\Gamma \vdash a \ {}^{m_1}\!\!\gg\!\!=^{m_2'} (\lambda x : A.\mathbf{up}^{m_2, m_2'} b) : T_{m_1 \cdot m_2'}\ B$ and $\Gamma \vdash \mathbf{up}^{m_1 \cdot m_2, m_1 \cdot m_2'}(a \ {}^{m_1}\!\!\gg\!\!=^{m_2} \lambda x : A.b) : T_{m_1 \cdot m_2'}\ B$ where $\Gamma \vdash a : T_{m_1}\ A$ and $\Gamma, x : A \vdash b : T_{m_2}\ B$.
  Now,

$$
\begin{aligned}
& [\![a \ {}^{m_1}\!\!\gg\!\!=^{m_2'} (\lambda x : A.\mathbf{up}^{m_2, m_2'} b)]\!] \\
&= [\![\mathbf{join}^{m_1, m_2'}((\mathbf{lift}^{m_1}(\lambda x : A.\mathbf{up}^{m_2, m_2'} b))\ a)]\!] \\
&= \mu^{m_1, m_2'}_{[\![B]\!]} \circ [\![(\mathbf{lift}^{m_1}(\lambda x : A.\mathbf{up}^{m_2, m_2'} b))\ a]\!] \\
&= \mu^{m_1, m_2'}_{[\![B]\!]} \circ \mathrm{app} \circ \langle \Lambda\Big(\mathbf{T}_{m_1}(\Lambda^{-1}[\![\lambda x : A.\mathbf{up}^{m_2, m_2'} b]\!]) \circ t^{\mathbf{T}_{m_1}}_{[\![\Gamma]\!],[\![A]\!]}\Big), [\![a]\!] \rangle \\
&= \mu^{m_1, m_2'}_{[\![B]\!]} \circ \mathbf{T}_{m_1}(\mathbf{T}^{m_2 <: m_2'}_{[\![B]\!]} \circ [\![b]\!]) \circ t^{\mathbf{T}_{m_1}}_{[\![\Gamma]\!],[\![A]\!]} \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![a]\!] \rangle \\
&= \mu^{m_1, m_2'}_{[\![B]\!]} \circ \mathbf{T}_{m_1}\mathbf{T}^{m_2 <: m_2'}_{[\![B]\!]} \circ \mathbf{T}_{m_1}[\![b]\!] \circ t^{\mathbf{T}_{m_1}}_{[\![\Gamma]\!],[\![A]\!]} \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![a]\!] \rangle.
\end{aligned}
$$

$$
\begin{array}{ccccccc}
[\![\Gamma]\!] \times [\![A]\!] & \xrightarrow{t^{\mathbf{Id}}_{[\![\Gamma]\!],[\![A]\!]}} & [\![\Gamma]\!] \times [\![A]\!] & \xrightarrow{\Lambda^{-1}[\![f]\!]} & \mathbf{T}_m[\![B]\!] & \xrightarrow{\mathrm{id}_{\mathbf{T}_m[\![B]\!]}} & \\
\downarrow{\scriptstyle \mathrm{id}_{[\![\Gamma]\!]}\times\eta_{[\![A]\!]}} & & \downarrow{\scriptstyle \eta_{[\![\Gamma]\!]\times[\![A]\!]}} & & \downarrow{\scriptstyle \eta_{\mathbf{T}_m[\![B]\!]}} & & \searrow \\
[\![\Gamma]\!] \times \mathbf{T}_1[\![A]\!] & \xrightarrow{t^{\mathbf{T}_1}_{[\![\Gamma]\!],[\![A]\!]}} & \mathbf{T}_1([\![\Gamma]\!] \times [\![A]\!]) & \xrightarrow{\mathbf{T}_1(\Lambda^{-1}[\![f]\!])} & \mathbf{T}_1\mathbf{T}_m[\![B]\!] & \xrightarrow{\mu^{1,m}_{[\![B]\!]}} & \mathbf{T}_m[\![B]\!]
\end{array}
$$

Figure A.2: Commutative diagram

Next,

$$
\begin{aligned}
&[\![\mathbf{up}^{m_1\cdot m_2, m_1\cdot m_2'}(a \ {}^{m_1}\!\!\gg\!\!=^{m_2} \ \lambda x : A.b)]\!] \\
=\ & \mathbf{T}^{m_1\cdot m_2 <: m_1\cdot m_2'}_{[\![B]\!]} \circ [\![\mathbf{join}^{m_1,m_2}((\mathbf{lift}^{m_1}(\lambda x : A.b))\ a)]\!] \\
=\ & \mathbf{T}^{m_1\cdot m_2 <: m_1\cdot m_2'}_{[\![B]\!]} \circ \mu^{m_1,m_2}_{[\![B]\!]} \circ \mathrm{app} \circ \langle \Lambda\Big(\mathbf{T}_{m_1}(\Lambda^{-1}[\![\lambda x : A.b]\!]) \circ t^{\mathbf{T}_{m_1}}_{[\![\Gamma]\!],[\![A]\!]}\Big), [\![a]\!]\rangle \\
=\ & \mathbf{T}^{m_1\cdot m_2 <: m_1\cdot m_2'}_{[\![B]\!]} \circ \mu^{m_1,m_2}_{[\![B]\!]} \circ \mathbf{T}_{m_1}[\![b]\!] \circ t^{\mathbf{T}_{m_1}}_{[\![\Gamma]\!],[\![A]\!]} \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![a]\!]\rangle.
\end{aligned}
$$

The two morphisms above are equal because μ is natural in its second component.

- $(\mathbf{ret}\ a)\ {}^{1}\!\!\gg\!\!=^{m} f \equiv f\ a$.
  Given: $\Gamma \vdash (\mathbf{ret}\ a)\ {}^{1}\!\!\gg\!\!=^{m} f : T_m\ B$ and $\Gamma \vdash f\ a : T_m\ B$ where $\Gamma \vdash a : A$ and $\Gamma \vdash f : A \to T_m\ B$.
  Now,

$$
\begin{aligned}
&[\![(\mathbf{ret}\ a)\ {}^{1}\!\!\gg\!\!=^{m} f]\!] \\
=&[\![\mathbf{join}^{1,m}((\mathbf{lift}^1 f)\ (\mathbf{ret}\ a))]\!] \\
=&\mu^{1,m}_{[\![B]\!]} \circ \mathrm{app} \circ \langle [\![\mathbf{lift}^1 f]\!], [\![\mathbf{ret}\ a]\!]\rangle \\
=&\mu^{1,m}_{[\![B]\!]} \circ \mathrm{app} \circ \langle \Lambda\Big(\mathbf{T}_1(\Lambda^{-1}[\![f]\!]) \circ t^{\mathbf{T}_1}_{[\![\Gamma]\!],[\![A]\!]}\Big), \eta_{[\![A]\!]} \circ [\![a]\!]\rangle \\
=&\mu^{1,m}_{[\![B]\!]} \circ \mathbf{T}_1(\Lambda^{-1}[\![f]\!]) \circ t^{\mathbf{T}_1}_{[\![\Gamma]\!],[\![A]\!]} \circ (\mathrm{id}_{[\![\Gamma]\!]} \times \eta_{[\![A]\!]}) \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![a]\!]\rangle.
\end{aligned}
$$

  And,

$$
[\![f\ a]\!] = \Lambda^{-1}[\![f]\!] \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![a]\!]\rangle.
$$

  The above two morphisms are equal because the diagram in Figure A.2 commutes. The left square commutes because η is a *strong* natural transformation; the square to the right commutes because η is a natural transformation; the triangle to the right commutes because $\mathbf{T}$ is a lax monoidal functor. Also, note $t^{\mathbf{Id}}_{[\![\Gamma]\!],[\![A]\!]} = \mathrm{id}_{[\![\Gamma]\!]\times[\![A]\!]}$.

- $a\ {}^{m_1}\!\!\gg\!\!=^{1}\ (\lambda x : A.\mathbf{ret}\ x) \equiv a$.
  Given: $\Gamma \vdash a\ {}^{m_1}\!\!\gg\!\!=^{1}\ (\lambda x : A.\mathbf{ret}\ x) : T_{m_1}\ A$ and $\Gamma \vdash a : T_{m_1}\ A$.

$$
\begin{array}{ccccccc}
\llbracket\Gamma\rrbracket \times \mathbf{T}_{m_1}\mathbf{T}_{m_2}\llbracket B\rrbracket & \xrightarrow{\mathbf{T}_{m_1} t^{\mathbf{T}_{m_2}}_{\llbracket\Gamma\rrbracket,\llbracket B\rrbracket} \circ t^{\mathbf{T}_{m_1}}_{\llbracket\Gamma\rrbracket,\mathbf{T}_{m_2}\llbracket B\rrbracket}} & \mathbf{T}_{m_1}\mathbf{T}_{m_2}(\llbracket\Gamma\rrbracket \times \llbracket B\rrbracket) & \xrightarrow{\mathbf{T}_{m_1}\mathbf{T}_{m_2}(\Lambda^{-1}\llbracket g\rrbracket)} & \mathbf{T}_{m_1}\mathbf{T}_{m_2}\mathbf{T}_{m_3}\llbracket C\rrbracket & \xrightarrow{\mathbf{T}_{m_1}\mu^{m_2,m_3}_{\llbracket C\rrbracket}} & \mathbf{T}_{m_1}\mathbf{T}_{m_2\cdot m_3}\llbracket C\rrbracket \\
\downarrow{\scriptstyle \mathrm{id}_{\llbracket\Gamma\rrbracket}\times\mu^{m_1,m_2}_{\llbracket B\rrbracket}} & & \downarrow{\scriptstyle \mu^{m_1,m_2}_{\llbracket\Gamma\rrbracket\times\llbracket B\rrbracket}} & & \downarrow{\scriptstyle \mu^{m_1,m_2}_{\mathbf{T}_{m_3}\llbracket C\rrbracket}} & & \downarrow{\scriptstyle \mu^{m_1,m_2\cdot m_3}_{\llbracket C\rrbracket}} \\
\llbracket\Gamma\rrbracket \times \mathbf{T}_{m_1\cdot m_2}\llbracket B\rrbracket & \xrightarrow{t^{\mathbf{T}_{m_1\cdot m_2}}_{\llbracket\Gamma\rrbracket,\llbracket B\rrbracket}} & \mathbf{T}_{m_1\cdot m_2}(\llbracket\Gamma\rrbracket \times \llbracket B\rrbracket) & \xrightarrow{\mathbf{T}_{m_1\cdot m_2}(\Lambda^{-1}\llbracket g\rrbracket)} & \mathbf{T}_{m_1\cdot m_2}\mathbf{T}_{m_3}\llbracket C\rrbracket & \xrightarrow{\mu^{m_1\cdot m_2,m_3}_{\llbracket C\rrbracket}} & \mathbf{T}_{m_1\cdot m_2\cdot m_3}\llbracket C\rrbracket
\end{array}
$$

FIGURE A.3: Commutative diagram

Now,

$$
\begin{aligned}
&\llbracket a \ {}^{m_1}\!\!\gg\!\!=^{1} (\lambda x : A.\mathbf{ret}\ x)\rrbracket \\
=&\llbracket \mathbf{join}^{m_1,1}((\mathbf{lift}^{m_1}(\lambda x : A.\mathbf{ret}\ x))\ a)\rrbracket \\
=&\mu^{m_1,1}_{\llbracket A\rrbracket} \circ \mathrm{app} \circ \langle \Lambda\Big(\mathbf{T}_{m_1}(\Lambda^{-1}\llbracket \lambda x : A.\mathbf{ret}\ x\rrbracket) \circ t^{\mathbf{T}_{m_1}}_{\llbracket\Gamma\rrbracket,\llbracket A\rrbracket}\Big), \llbracket a\rrbracket\rangle \\
=&\mu^{m_1,1}_{\llbracket A\rrbracket} \circ \mathbf{T}_{m_1}(\eta_{\llbracket A\rrbracket} \circ \pi_2) \circ t^{\mathbf{T}_{m_1}}_{\llbracket\Gamma\rrbracket,\llbracket A\rrbracket} \circ \langle \mathrm{id}_{\llbracket\Gamma\rrbracket}, \llbracket a\rrbracket\rangle \\
=&\mu^{m_1,1}_{\llbracket A\rrbracket} \circ \mathbf{T}_{m_1}\eta_{\llbracket A\rrbracket} \circ \mathbf{T}_{m_1}\pi_2 \circ t^{\mathbf{T}_{m_1}}_{\llbracket\Gamma\rrbracket,\llbracket A\rrbracket} \circ \langle \mathrm{id}_{\llbracket\Gamma\rrbracket}, \llbracket a\rrbracket\rangle \\
=&\mu^{m_1,1}_{\llbracket A\rrbracket} \circ \mathbf{T}_{m_1}\eta_{\llbracket A\rrbracket} \circ \pi_2 \circ \langle \mathrm{id}_{\llbracket\Gamma\rrbracket}, \llbracket a\rrbracket\rangle \ \text{[By naturality and strength]} \\
=&\mu^{m_1,1}_{\llbracket A\rrbracket} \circ \mathbf{T}_{m_1}\eta_{\llbracket A\rrbracket} \circ \llbracket a\rrbracket \\
=&\llbracket a\rrbracket \ [\because \mathbf{T} \text{ is a lax monoidal functor}].
\end{aligned}
$$

- $(a \ {}^{m_1}\!\!\gg\!\!=^{m_2} f) \ {}^{m_1\cdot m_2}\!\!\gg\!\!=^{m_3} g \equiv a \ {}^{m_1}\!\!\gg\!\!=^{m_2\cdot m_3} (\lambda x : A.(f\ x \ {}^{m_2}\!\!\gg\!\!=^{m_3} g))$.
  Given: $\Gamma \vdash (a \ {}^{m_1}\!\!\gg\!\!=^{m_2} f) \ {}^{m_1\cdot m_2}\!\!\gg\!\!=^{m_3} g : T_{m_1\cdot m_2\cdot m_3}\ C$ and $\Gamma \vdash a \ {}^{m_1}\!\!\gg\!\!=^{m_2\cdot m_3} (\lambda x : A.(f\ x \ {}^{m_2}\!\!\gg\!\!=^{m_3} g)) : T_{m_1\cdot m_2\cdot m_3}\ C$ where $\Gamma \vdash a : T_{m_1}\ A$ and $\Gamma \vdash f : A \to T_{m_2}\ B$ and $\Gamma \vdash g : B \to T_{m_3}\ C$.

  Now,

$$
\begin{aligned}
&\llbracket a \ {}^{m_1}\!\!\gg\!\!=^{m_2\cdot m_3} (\lambda x : A.(f\ x \ {}^{m_2}\!\!\gg\!\!=^{m_3} g))\rrbracket \\
=&\llbracket \mathbf{join}^{m_1,m_2\cdot m_3}((\mathbf{lift}^{m_1}(\lambda x : A.(f\ x \ {}^{m_2}\!\!\gg\!\!=^{m_3} g)))\ a)\rrbracket \\
=&\mu^{m_1,m_2\cdot m_3}_{\llbracket C\rrbracket} \circ \mathrm{app} \circ \langle \llbracket \mathbf{lift}^{m_1}(\lambda x : A.(f\ x \ {}^{m_2}\!\!\gg\!\!=^{m_3} g))\rrbracket, \llbracket a\rrbracket\rangle \\
=&\mu^{m_1,m_2\cdot m_3}_{\llbracket C\rrbracket} \circ \mathrm{app} \circ \langle \Lambda\Big(\mathbf{T}_{m_1}(\Lambda^{-1}\llbracket \lambda x : A.(f\ x \ {}^{m_2}\!\!\gg\!\!=^{m_3} g)\rrbracket) \circ t^{\mathbf{T}_{m_1}}_{\llbracket\Gamma\rrbracket,\llbracket A\rrbracket}\Big), \llbracket a\rrbracket\rangle \\
=&\mu^{m_1,m_2\cdot m_3}_{\llbracket C\rrbracket} \circ \mathbf{T}_{m_1}(\Lambda^{-1}\llbracket \lambda x : A.(f\ x \ {}^{m_2}\!\!\gg\!\!=^{m_3} g)\rrbracket) \circ t^{\mathbf{T}_{m_1}}_{\llbracket\Gamma\rrbracket,\llbracket A\rrbracket} \circ \langle \mathrm{id}_{\llbracket\Gamma\rrbracket}, \llbracket a\rrbracket\rangle \\
=&\mu^{m_1,m_2\cdot m_3}_{\llbracket C\rrbracket} \circ \mathbf{T}_{m_1}\llbracket f\ x \ {}^{m_2}\!\!\gg\!\!=^{m_3} g\rrbracket \circ t^{\mathbf{T}_{m_1}}_{\llbracket\Gamma\rrbracket,\llbracket A\rrbracket} \circ \langle \mathrm{id}_{\llbracket\Gamma\rrbracket}, \llbracket a\rrbracket\rangle \\
=&\mu^{m_1,m_2\cdot m_3}_{\llbracket C\rrbracket} \circ \mathbf{T}_{m_1}\llbracket \mathbf{join}^{m_2,m_3}((\mathbf{lift}^{m_2} g)\ (f\ x))\rrbracket \circ t^{\mathbf{T}_{m_1}}_{\llbracket\Gamma\rrbracket,\llbracket A\rrbracket} \circ \langle \mathrm{id}_{\llbracket\Gamma\rrbracket}, \llbracket a\rrbracket\rangle \\
=&\mu^{m_1,m_2\cdot m_3}_{\llbracket C\rrbracket} \circ \mathbf{T}_{m_1}(\mu^{m_2,m_3}_{\llbracket C\rrbracket} \circ \mathrm{app} \circ \langle \llbracket \mathbf{lift}^{m_2} g\rrbracket, \llbracket f\ x\rrbracket\rangle) \circ t^{\mathbf{T}_{m_1}}_{\llbracket\Gamma\rrbracket,\llbracket A\rrbracket} \circ \langle \mathrm{id}_{\llbracket\Gamma\rrbracket}, \llbracket a\rrbracket\rangle \\
=&\mu^{m_1,m_2\cdot m_3}_{\llbracket C\rrbracket} \circ \mathbf{T}_{m_1}\mu^{m_2,m_3}_{\llbracket C\rrbracket} \circ \mathbf{T}_{m_1}\mathbf{T}_{m_2}(\Lambda^{-1}\llbracket g\rrbracket \circ (\pi_1 \times \mathrm{id})) \circ \mathbf{T}_{m_1} t^{\mathbf{T}_{m_2}}_{\llbracket\Gamma\rrbracket\times\llbracket A\rrbracket,\llbracket B\rrbracket} \circ \mathbf{T}_{m_1}\langle \mathrm{id}, \Lambda^{-1}\llbracket f\rrbracket\rangle \circ
\end{aligned}
$$

$$
\begin{array}{ccccc}
\llbracket\Gamma\rrbracket \times \mathbf{T}_{m_1}\llbracket A\rrbracket & \xrightarrow{t^{\mathbf{T}_{m_1}}_{\llbracket\Gamma\rrbracket,\llbracket A\rrbracket}} & \mathbf{T}_{m_1}(\llbracket\Gamma\rrbracket \times \llbracket A\rrbracket) & \xrightarrow{\mathbf{T}_{m_1}\langle \mathrm{id},\Lambda^{-1}\llbracket f\rrbracket\rangle} & \mathbf{T}_{m_1}((\llbracket\Gamma\rrbracket \times \llbracket A\rrbracket) \times \mathbf{T}_{m_2}\llbracket B\rrbracket) \\
\downarrow{\scriptstyle \langle \pi_1, t^{\mathbf{T}_{m_1}}_{\llbracket\Gamma\rrbracket,\llbracket A\rrbracket}\rangle} & & \downarrow{\scriptstyle \mathbf{T}_{m_1}\langle \pi_1,\mathrm{id}\rangle} & & \downarrow{\scriptstyle \mathbf{T}_{m_1}\langle \pi_1\circ\pi_1,\pi_2\rangle} \\
\llbracket\Gamma\rrbracket \times \mathbf{T}_{m_1}(\llbracket\Gamma\rrbracket \times \llbracket A\rrbracket) & \xrightarrow{t^{\mathbf{T}_{m_1}}_{\llbracket\Gamma\rrbracket,\llbracket\Gamma\rrbracket\times\llbracket A\rrbracket}} & \mathbf{T}_{m_1}(\llbracket\Gamma\rrbracket \times (\llbracket\Gamma\rrbracket \times \llbracket A\rrbracket)) & \xrightarrow{\mathbf{T}_{m_1}(\mathrm{id}\times\Lambda^{-1}\llbracket f\rrbracket)} & \mathbf{T}_{m_1}(\llbracket\Gamma\rrbracket \times \mathbf{T}_{m_2}\llbracket B\rrbracket)
\end{array}
$$

$$
\llbracket\Gamma\rrbracket \times \mathbf{T}_{m_1}(\llbracket\Gamma\rrbracket \times \llbracket A\rrbracket) \xrightarrow{\mathrm{id}\times\mathbf{T}_{m_1}(\Lambda^{-1}\llbracket f\rrbracket)} \llbracket\Gamma\rrbracket \times \mathbf{T}_{m_1}\mathbf{T}_{m_2}\llbracket B\rrbracket \xrightarrow{t^{\mathbf{T}_{m_1}}_{\llbracket\Gamma\rrbracket,\mathbf{T}_{m_2}\llbracket B\rrbracket}} \mathbf{T}_{m_1}(\llbracket\Gamma\rrbracket \times \mathbf{T}_{m_2}\llbracket B\rrbracket)
$$

Figure A.4: Commutative diagram

$$
\begin{aligned}
& t^{\mathbf{T}_{m_1}}_{\llbracket\Gamma\rrbracket,\llbracket A\rrbracket} \circ \langle \mathrm{id}, \llbracket a\rrbracket\rangle \\
=& \mu^{m_1,m_2\cdot m_3}_{\llbracket C\rrbracket} \circ \mathbf{T}_{m_1}\mu^{m_2,m_3}_{\llbracket C\rrbracket} \circ \mathbf{T}_{m_1}\mathbf{T}_{m_2}(\Lambda^{-1}\llbracket g\rrbracket) \circ \mathbf{T}_{m_1} t^{\mathbf{T}_{m_2}}_{\llbracket\Gamma\rrbracket,\llbracket B\rrbracket} \circ \mathbf{T}_{m_1}(\pi_1 \times \mathrm{id}) \circ \mathbf{T}_{m_1}\langle \mathrm{id}, \Lambda^{-1}\llbracket f\rrbracket\rangle \\
& \circ t^{\mathbf{T}_{m_1}}_{\llbracket\Gamma\rrbracket,\llbracket A\rrbracket} \circ \langle \mathrm{id}, \llbracket a\rrbracket\rangle \quad \text{[By naturality of } t^{\mathbf{T}_{m_2}}\text{]}.
\end{aligned}
$$

Next,

$$
\begin{aligned}
& \llbracket (a \ {}^{m_1}\!\!\gg\!\!=^{m_2} f) \ {}^{m_1\cdot m_2}\!\!\gg\!\!=^{m_3} g\rrbracket \\
=& \llbracket \mathbf{join}^{m_1\cdot m_2,m_3}((\mathbf{lift}^{m_1\cdot m_2} g)\ (a \ {}^{m_1}\!\!\gg\!\!=^{m_2} f))\rrbracket \\
=& \mu^{m_1\cdot m_2,m_3}_{\llbracket C\rrbracket} \circ \mathrm{app} \circ \langle \llbracket \mathbf{lift}^{m_1\cdot m_2} g\rrbracket, \llbracket a \ {}^{m_1}\!\!\gg\!\!=^{m_2} f\rrbracket\rangle \\
=& \mu^{m_1\cdot m_2,m_3}_{\llbracket C\rrbracket} \circ \mathrm{app} \circ \langle \Lambda\Big(\mathbf{T}_{m_1\cdot m_2}(\Lambda^{-1}\llbracket g\rrbracket) \circ t^{\mathbf{T}_{m_1\cdot m_2}}_{\llbracket\Gamma\rrbracket,\llbracket B\rrbracket}\Big), \llbracket a \ {}^{m_1}\!\!\gg\!\!=^{m_2} f\rrbracket\rangle \\
=& \mu^{m_1\cdot m_2,m_3}_{\llbracket C\rrbracket} \circ \mathbf{T}_{m_1\cdot m_2}(\Lambda^{-1}\llbracket g\rrbracket) \circ t^{\mathbf{T}_{m_1\cdot m_2}}_{\llbracket\Gamma\rrbracket,\llbracket B\rrbracket} \circ \langle \mathrm{id}_{\llbracket\Gamma\rrbracket}, \llbracket \mathbf{join}^{m_1,m_2}((\mathbf{lift}^{m_1} f)\ a)\rrbracket\rangle \\
=& \mu^{m_1\cdot m_2,m_3}_{\llbracket C\rrbracket} \circ \mathbf{T}_{m_1\cdot m_2}(\Lambda^{-1}\llbracket g\rrbracket) \circ t^{\mathbf{T}_{m_1\cdot m_2}}_{\llbracket\Gamma\rrbracket,\llbracket B\rrbracket} \\
& \quad \circ \langle \mathrm{id}_{\llbracket\Gamma\rrbracket}, \mu^{m_1,m_2}_{\llbracket B\rrbracket} \circ \mathbf{T}_{m_1}(\Lambda^{-1}\llbracket f\rrbracket) \circ t^{\mathbf{T}_{m_1}}_{\llbracket\Gamma\rrbracket,\llbracket A\rrbracket} \circ \langle \mathrm{id}_{\llbracket\Gamma\rrbracket}, \llbracket a\rrbracket\rangle\rangle \\
=& \mu^{m_1\cdot m_2,m_3}_{\llbracket C\rrbracket} \circ \mathbf{T}_{m_1\cdot m_2}(\Lambda^{-1}\llbracket g\rrbracket) \circ t^{\mathbf{T}_{m_1\cdot m_2}}_{\llbracket\Gamma\rrbracket,\llbracket B\rrbracket} \circ (\mathrm{id}_{\llbracket\Gamma\rrbracket} \times \mu^{m_1,m_2}_{\llbracket B\rrbracket}) \\
& \quad \circ \langle \mathrm{id}_{\llbracket\Gamma\rrbracket}, \mathbf{T}_{m_1}(\Lambda^{-1}\llbracket f\rrbracket) \circ t^{\mathbf{T}_{m_1}}_{\llbracket\Gamma\rrbracket,\llbracket A\rrbracket} \circ \langle \mathrm{id}_{\llbracket\Gamma\rrbracket}, \llbracket a\rrbracket\rangle\rangle \\
=& \mu^{m_1\cdot m_2,m_3}_{\llbracket C\rrbracket} \circ \mathbf{T}_{m_1\cdot m_2}(\Lambda^{-1}\llbracket g\rrbracket) \circ t^{\mathbf{T}_{m_1\cdot m_2}}_{\llbracket\Gamma\rrbracket,\llbracket B\rrbracket} \circ (\mathrm{id} \times \mu^{m_1,m_2}_{\llbracket B\rrbracket}) \circ \Big(\mathrm{id} \times (\mathbf{T}_{m_1}(\Lambda^{-1}\llbracket f\rrbracket) \circ t^{\mathbf{T}_{m_1}}_{\llbracket\Gamma\rrbracket,\llbracket A\rrbracket})\Big) \\
& \quad \circ \langle \pi_1, \mathrm{id}\rangle \circ \langle \mathrm{id}, \llbracket a\rrbracket\rangle \\
=& \mu^{m_1,m_2\cdot m_3}_{\llbracket C\rrbracket} \circ \mathbf{T}_{m_1}\mu^{m_2,m_3}_{\llbracket C\rrbracket} \circ \mathbf{T}_{m_1}\mathbf{T}_{m_2}(\Lambda^{-1}\llbracket g\rrbracket) \circ \mathbf{T}_{m_1} t^{\mathbf{T}_{m_2}}_{\llbracket\Gamma\rrbracket,\llbracket B\rrbracket} \circ t^{\mathbf{T}_{m_1}}_{\llbracket\Gamma\rrbracket,\mathbf{T}_{m_2}\llbracket B\rrbracket} \\
& \quad \circ \Big(\mathrm{id} \times (\mathbf{T}_{m_1}(\Lambda^{-1}\llbracket f\rrbracket) \circ t^{\mathbf{T}_{m_1}}_{\llbracket\Gamma\rrbracket,\llbracket A\rrbracket})\Big) \circ \langle \pi_1, \mathrm{id}\rangle \circ \langle \mathrm{id}, \llbracket a\rrbracket\rangle \quad \text{[By Figure A.3]}
\end{aligned}
$$

The diagram in Figure A.3 commutes: the leftmost square commutes because μ^{m_1,m_2} is a *strong* natural transformation; the middle one commutes because μ^{m_1,m_2} is a natural transformation; the rightmost one commutes because $\mathbf{T}$ is a lax monoidal functor.

Next, to show that the above morphisms are equal, we just need to show that:

$$t^{\mathbf{T}_{m_1}}_{[\![\Gamma]\!],\mathbf{T}_{m_2}[\![B]\!]} \circ \Big(\mathrm{id} \times (\mathbf{T}_{m_1}(\Lambda^{-1}[\![f]\!]) \circ t^{\mathbf{T}_{m_1}}_{[\![\Gamma]\!],[\![A]\!]})\Big) \circ \langle \pi_1, \mathrm{id}\rangle = \mathbf{T}_{m_1}(\pi_1 \times \mathrm{id}) \circ \mathbf{T}_{m_1}\langle \mathrm{id}, \Lambda^{-1}[\![f]\!]\rangle \circ t^{\mathbf{T}_{m_1}}_{[\![\Gamma]\!],[\![A]\!]}$$

These two morphisms are equal by the commutative diagram in Figure A.4. The diagram in Figure A.4 commutes: the bottom subfigure commutes by naturality of $t^{\mathbf{T}_{m_1}}$; the right one commutes by properties of products; the left one commutes by the commutativity of the diagram in Figure A.1.

□

A.2 Proofs of Lemmas/Theorems Stated in Section 2.3

Theorem A.2 (Theorem 2.2) Let $\mathcal{L}$ be a bounded join-semilattice. If $\Gamma \vdash a : A$ in GMC($\mathcal{L}$), then $\overline{\Gamma} \vdash \overline{a} : \overline{A}$ in DCC($\mathcal{L}$). Further, if $\Gamma \vdash a_1 : A$ and $\Gamma \vdash a_2 : A$ such that $a_1 \equiv a_2$ in GMC($\mathcal{L}$), then $\overline{a_1} \simeq \overline{a_2}$ in DCC($\mathcal{L}$).

Proof. By induction on $\Gamma \vdash a : A$.

- λ-calculus. By IH.
- Rule M-Return. Have: $\Gamma \vdash \mathbf{ret}\ a : T_\bot\ A$ where $\Gamma \vdash a : A$.
  By IH, $\overline{\Gamma} \vdash \overline{a} : \overline{A}$. Therefore, $\overline{\Gamma} \vdash \mathbf{eta}^\bot\ \overline{a} : T_\bot\ \overline{A}$.
- Rule M-Fmap. Have: $\Gamma \vdash \mathbf{lift}^\ell f : T_\ell\ A \to T_\ell\ B$ where $\Gamma \vdash f : A \to B$.
  By IH, $\overline{\Gamma} \vdash \overline{f} : \overline{A} \to \overline{B}$.
  Now,

$$\dfrac{\dfrac{\dfrac{}{\overline{\Gamma}, x : T_\ell\ \overline{A} \vdash x : T_\ell\ \overline{A}} \quad \dfrac{\dfrac{}{\overline{\Gamma}, x : T_\ell\ \overline{A}, y : \overline{A} \vdash \overline{f}\ y : \overline{B}}\ (\textsc{App})}{\overline{\Gamma}, x : T_\ell\ \overline{A}, y : \overline{A} \vdash \mathbf{eta}^\ell\ (\overline{f}\ y) : T_\ell\ \overline{B}}\ (\textsc{Eta}) \quad \dfrac{\ell \sqsubseteq \ell}{\ell \sqsubseteq T_\ell\ \overline{B}}\ (\textsc{Prot-Mon})}{\overline{\Gamma}, x : T_\ell\ \overline{A} \vdash \mathbf{bind}^\ell\ y = x\ \mathbf{in}\ \mathbf{eta}^\ell\ (\overline{f}\ y) : T_\ell\ \overline{B}}\ (\textsc{Bind})}{\overline{\Gamma} \vdash \lambda x : T_\ell\ \overline{A}.\mathbf{bind}^\ell\ y = x\ \mathbf{in}\ \mathbf{eta}^\ell\ (\overline{f}\ y) : T_\ell\ \overline{A} \to T_\ell\ \overline{B}}\ (\textsc{Lam})$$

- Rule M-Join. Have $\Gamma \vdash \mathbf{join}^{\ell_1,\ell_2} a : T_{\ell_1 \sqcup \ell_2}\ A$ where $\Gamma \vdash a : T_{\ell_1}\ T_{\ell_2}\ A$.
  By IH, $\overline{\Gamma} \vdash \overline{a} : T_{\ell_1}\ T_{\ell_2}\ \overline{A}$.
  Now,

$$\dfrac{\dfrac{}{\overline{\Gamma} \vdash \overline{a} : T_{\ell_1}\ T_{\ell_2}\ \overline{A}} \quad \dfrac{\dfrac{}{\overline{\Gamma}, x : T_{\ell_2}\ \overline{A} \vdash x : T_{\ell_2}\ \overline{A}} \quad \dfrac{\dfrac{}{\overline{\Gamma}, x : T_{\ell_2}\ \overline{A}, y : \overline{A} \vdash y : \overline{A}}}{\overline{\Gamma}, x : T_{\ell_2}\ \overline{A}, y : \overline{A} \vdash \mathbf{eta}^{\ell_1 \sqcup \ell_2}\ y : T_{\ell_1 \sqcup \ell_2}\ \overline{A}}\ (\textsc{Eta})}{\overline{\Gamma}, x : T_{\ell_2}\ \overline{A} \vdash \mathbf{bind}^{\ell_2}\ y = x\ \mathbf{in}\ \mathbf{eta}^{\ell_1 \sqcup \ell_2}\ y : T_{\ell_1 \sqcup \ell_2}\ \overline{A}}\ (\textsc{Bind})}{\overline{\Gamma} \vdash \mathbf{bind}^{\ell_1}\ x = \overline{a}\ \mathbf{in}\ \mathbf{bind}^{\ell_2}\ y = x\ \mathbf{in}\ \mathbf{eta}^{\ell_1 \sqcup \ell_2}\ y : T_{\ell_1 \sqcup \ell_2}\ \overline{A}}\ (\textsc{Bind})$$

The above derivation uses the judgments $\ell_2 \sqsubseteq T_{\ell_1 \sqcup \ell_2}\, \overline{A}$ and $\ell_1 \sqsubseteq T_{\ell_1 \sqcup \ell_2}\, \overline{A}$ on the first and the second applications of the bind rule respectively.

- Rule M-Up. Have: $\Gamma \vdash \mathbf{up}^{\ell_1,\ell_2}\, a : T_{\ell_2}\, A$ where $\Gamma \vdash a : T_{\ell_1}\, A$ and $\ell_1 \sqsubseteq \ell_2$.
  By IH, $\overline{\Gamma} \vdash \overline{a} : T_{\ell_1}\, \overline{A}$.
  Now,
  $$\dfrac{\overline{\overline{\Gamma} \vdash \overline{a} : T_{\ell_1}\, \overline{A}} \quad \dfrac{\overline{\Gamma}, x : \overline{A} \vdash x : \overline{A}}{\overline{\Gamma}, x : \overline{A} \vdash \mathbf{eta}^{\ell_2}\, x : T_{\ell_2}\, \overline{A}}\ (\textsc{Eta}) \quad \dfrac{\ell_1 \sqsubseteq \ell_2}{\ell_1 \sqsubseteq T_{\ell_2}\, \overline{A}}\ (\textsc{Prot-Mon})}{\overline{\Gamma} \vdash \mathbf{bind}^{\ell_1}\, x = \overline{a}\ \mathbf{in}\ \mathbf{eta}^{\ell_2}\, x : T_{\ell_2}\, \overline{A}}\ (\textsc{Bind})$$
  .

Now, for $\Gamma \vdash a_1 : A$ and $\Gamma \vdash a_2 : A$, if $a_1 \equiv a_2$, then $\lfloor \overline{a_1} \rfloor$ and $\lfloor \overline{a_2} \rfloor$ are equal as λ-terms. Hence, $\overline{a_1} \simeq \overline{a_2}$ in DCC. □

A.3 Proofs of Lemmas/Theorems Stated in Section 2.4

Theorem A.3 (Theorem 2.3) If $\Gamma \vdash a : A$ in GCC, then $[\![a]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], [\![A]\!])$. Further, if $\Gamma \vdash a_1 : A$ and $\Gamma \vdash a_2 : A$ such that $a_1 \equiv a_2$ in GCC, then $[\![a_1]\!] = [\![a_2]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], [\![A]\!])$.

Proof. Follows from Theorem A.1 by duality. □

A.4 Proofs of Lemmas/Theorems Stated in Section 2.5

Theorem A.4 (Theorem 2.4) If $\Gamma \vdash a : A$ in GCC($\mathcal{L}$), then $\overline{\Gamma} \vdash \overline{a} : \overline{A}$ in DCC$_e$($\mathcal{L}$). Further, if $\Gamma \vdash a_1 : A$ and $\Gamma \vdash a_2 : A$ such that $a_1 \equiv a_2$ in GCC($\mathcal{L}$), then $\overline{a_1} \simeq \overline{a_2}$ in DCC$_e$($\mathcal{L}$).

Proof. By induction on $\Gamma \vdash a : A$. Only the cases **extr** and **fork** are new; for the other cases, follow the proof of Theorem A.2.

- Rule C-Extract. Have: $\Gamma \vdash \mathbf{extr}\, a : A$ where $\Gamma \vdash a : D_\bot\, A$.
  By IH, $\overline{\Gamma} \vdash \overline{a} : T_\bot\, \overline{A}$.
  Now,
  $$\dfrac{\overline{\overline{\Gamma} \vdash \overline{a} : T_\bot\, \overline{A}} \quad \overline{\overline{\Gamma}, x : \overline{A} \vdash x : \overline{A}} \quad \overline{\bot \sqsubseteq \overline{A}}\ (\textsc{Prot-Min})}{\overline{\Gamma} \vdash \mathbf{bind}^{\bot}\, x = \overline{a}\ \mathbf{in}\ x : \overline{A}}\ (\textsc{Bind})$$
- Rule C-Fork. Have: $\Gamma \vdash \mathbf{fork}^{\ell_1,\ell_2}\, a : D_{\ell_1}\, D_{\ell_2}\, A$ where $\Gamma \vdash a : D_{\ell_1 \sqcup \ell_2}\, A$.
  By IH, $\overline{\Gamma} \vdash \overline{a} : T_{\ell_1 \sqcup \ell_2}\, \overline{A}$.

Now,

$$\dfrac{\dfrac{}{\overline{\Gamma} \vdash \overline{a} : T_{\ell_1 \sqcup \ell_2}\, \overline{A}} \qquad \dfrac{\dfrac{\dfrac{}{\overline{\Gamma}, x : \overline{A} \vdash x : \overline{A}}}{\overline{\Gamma}, x : \overline{A} \vdash \mathbf{eta}^{\ell_2}\, x : T_{\ell_2}\, \overline{A}}}{\overline{\Gamma}, x : \overline{A} \vdash \mathbf{eta}^{\ell_1}\, \mathbf{eta}^{\ell_2}\, x : T_{\ell_1}\, T_{\ell_2}\, \overline{A}} \qquad \dfrac{\dfrac{}{\ell_1 \sqsubseteq T_{\ell_1}\, T_{\ell_2}\, \overline{A}}\ (\textsc{Mon}) \quad \dfrac{\dfrac{}{\ell_2 \sqsubseteq T_{\ell_2}\, \overline{A}}\ (\textsc{Mon})}{\ell_2 \sqsubseteq T_{\ell_1}\, T_{\ell_2}\, \overline{A}}\ (\textsc{Aldy})}{\ell_1 \sqcup \ell_2 \sqsubseteq T_{\ell_1}\, T_{\ell_2}\, \overline{A}}\ (\textsc{Comb})}{\overline{\Gamma} \vdash \mathbf{bind}^{\ell_1 \sqcup \ell_2}\, x = \overline{a}\ \mathbf{in}\ \mathbf{eta}^{\ell_1}\, \mathbf{eta}^{\ell_2}\, x : T_{\ell_1}\, T_{\ell_2}\, \overline{A}}\ (\textsc{Bind})$$

Now, for $\Gamma \vdash a_1 : A$ and $\Gamma \vdash a_2 : A$, if $a_1 \equiv a_2$, then $\lfloor \overline{a_1} \rfloor$ and $\lfloor \overline{a_2} \rfloor$ are equal as λ-terms. Hence, $\overline{a_1} \simeq \overline{a_2}$ in DCC$_e$. □

A.5 Proofs of Lemmas/Theorems Stated in Section 2.6

Theorem A.5 (Theorem 2.5) If $\Gamma \vdash a : A$ in GMCC, then $[\![a]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], [\![A]\!])$. Further, if $\Gamma \vdash a_1 : A$ and $\Gamma \vdash a_2 : A$ such that $a_1 \equiv a_2$ in GMCC, then $[\![a_1]\!] = [\![a_2]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], [\![A]\!])$.

Proof. Let $\Gamma \vdash a : A$. Then, by Theorems A.1 and A.3, $[\![a]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], [\![A]\!])$ because a strong monoidal functor is also a lax and an oplax monoidal functor. For $\Gamma \vdash a_1 : A$ and $\Gamma \vdash a_2 : A$, if $a_1 \equiv a_2$, then $[\![a_1]\!] = [\![a_2]\!]$ by Theorems A.1 and A.3 and the equations listed in Section 2.6.2.

□

Theorem A.6 (Theorem 2.6) Given any preordered monoid $\mathcal{M}$, for typing derivations $\Gamma \vdash a_1 : A$ and $\Gamma \vdash a_2 : A$ in GMCC($\mathcal{M}$), if $[\![a_1]\!] = [\![a_2]\!]$ in all models of GMCC($\mathcal{M}$), then $a_1 \equiv a_2$ is derivable in GMCC($\mathcal{M}$).

Proof. We use standard term-model construction for proving this theorem. First, fix the preordered monoid, $\mathcal{M}$. Next, construct the freely generated bicartesian closed category, $\mathbb{F}$, from the syntax of GMCC($\mathcal{M}$), as follows:
$\mathrm{Obj}(\mathbb{F}), A, B ::= \mathbf{Unit} \mid \mathbf{Void} \mid A \times B \mid A + B \mid A \to B \mid S_m\ A$.
$\mathrm{Hom}_{\mathbb{F}}(A, B) = \{t \mid \varnothing \vdash t : A \to B\} / =_{\beta\eta}$.
The objects of $\mathbb{F}$ are the types of GMCC($\mathcal{M}$) while the morphisms are the terms of GMCC($\mathcal{M}$) quotiented by $\beta\eta$-equivalence. This is the classifying category of GMCC($\mathcal{M}$).

Now, we define a strong monoidal functor $\mathbb{S}$ from $\mathbf{C}(\mathcal{M})$ to $\mathbf{End}^{\mathrm{s}}_{\mathbb{F}}$.
$\mathbb{S}(m) := \mathbb{S}_m$ where $\mathbb{S}_m\ A := S_m\ A$ and $\mathbb{S}_m(A \xrightarrow{t} B) := S_m\ A \xrightarrow{\mathbf{lift}^m\, t} S_m\ B$.
$\mathbb{S}(m_1 <: m_2) := \mathbb{S}_{m_1} \xrightarrow{\mathbb{S}^{m_1 <: m_2}} \mathbb{S}_{m_2}$ where $\mathbb{S}^{m_1 <: m_2}_A := \lambda x : S_{m_1}\ A.\mathbf{up}^{m_1, m_2}\, x$.

Need to check that $\mathbb{S}$ is well-defined. In other words, need to show that $\mathbb{S}_m$s are strong endofunctors and $\mathbb{S}^{m_1 <: m_2}$s are strong natural transformations.

$\mathbb{S}_m$ is a functor because:
For $A \in \text{Obj}(\mathbb{F})$, $\mathbb{S}_m\, \text{id}_A = \textbf{lift}^m \text{id}_A = \textbf{lift}^m(\lambda x : A.x) = \lambda x : S_m\, A.x = \text{id}_{S_m\, A}$.
For $f \in \text{Hom}_{\mathbb{F}}(A, B)$ and $g \in \text{Hom}_{\mathbb{F}}(B, C)$,
$\mathbb{S}_m\, (g \circ f) = \textbf{lift}^m(g \circ f) = \textbf{lift}^m(\lambda x.g\, (f\, x)) = \lambda x.(\textbf{lift}^m g)\, ((\textbf{lift}^m f)\, x) = \mathbb{S}_m\, g \circ \mathbb{S}_m\, f$.

Now we define strength of $\mathbb{S}_m$, $t^{\mathbb{S}_m}$.
We have, $x : A \times S_m\, B \vdash \lambda y.(\textbf{proj}_1\, x, y) : B \to A \times B$.
Then, $x : A \times S_m\, B \vdash \textbf{lift}^m(\lambda y.(\textbf{proj}_1\, x, y)) : S_m\, B \to S_m\, (A \times B)$.
And, $x : A \times S_m\, B \vdash (\textbf{lift}^m(\lambda y.(\textbf{proj}_1\, x, y)))\, (\textbf{proj}_2\, x) : S_m\, (A \times B)$.
So, $\varnothing \vdash \lambda x.(\textbf{lift}^m(\lambda y.(\textbf{proj}_1\, x, y)))\, (\textbf{proj}_2\, x) : A \times S_m\, B \to S_m\, (A \times B)$.
We define: $t^{\mathbb{S}_m}_{A,B} \triangleq \lambda x.(\textbf{lift}^m(\lambda y.(\textbf{proj}_1\, x, y)))\, (\textbf{proj}_2\, x)$.
Check that $t^{\mathbb{S}_m}_{A,B}$ is natural in both A and B:

The left diagram in Figure A.5 commutes because:

$$
\begin{aligned}
& t^{\mathbb{S}_m}_{A',B} \circ (f \times \text{id}) \\
=& \lambda x.t^{\mathbb{S}_m}_{A',B}\, ((f \times \text{id})\, x) \\
=& \lambda x.t^{\mathbb{S}_m}_{A',B}\, (f\, \textbf{proj}_1\, x, \textbf{proj}_2\, x) \\
=& \lambda x.(\textbf{lift}^m(\lambda y.(f\, \textbf{proj}_1\, x, y)))\, (\textbf{proj}_2\, x)
\end{aligned}
$$

and

$$
\begin{aligned}
& \mathbb{S}_m\, (f \times \text{id}) \circ t^{\mathbb{S}_m}_{A,B} \\
=& \lambda x.(\textbf{lift}^m(f \times \text{id}))\, ((\textbf{lift}^m(\lambda y.(\textbf{proj}_1\, x, y)))\, (\textbf{proj}_2\, x)) \\
=& \lambda x.(\lambda z.(\textbf{lift}^m(f \times \text{id}))\, ((\textbf{lift}^m(\lambda y.(\textbf{proj}_1\, x, y)))\, z))\, (\textbf{proj}_2\, x) \\
=& \lambda x.(\textbf{lift}^m(\lambda z.(f \times \text{id})\, ((\lambda y.(\textbf{proj}_1\, x, y))\, z)))\, (\textbf{proj}_2\, x) \\
=& \lambda x.(\textbf{lift}^m(\lambda z.(f\, \textbf{proj}_1\, x, z)))\, (\textbf{proj}_2\, x)
\end{aligned}
$$

The right diagram in Figure A.5 commutes because:

$$
\begin{aligned}
& t^{\mathbb{S}_m}_{A,B'} \circ \text{id} \times \mathbb{S}_m\, g \\
=& \lambda x.t^{\mathbb{S}_m}_{A,B'}\, (\textbf{proj}_1\, x, (\textbf{lift}^m g)\, (\textbf{proj}_2\, x)) \\
=& \lambda x.(\textbf{lift}^m(\lambda y.(\textbf{proj}_1\, x, y)))\, ((\textbf{lift}^m g)\, (\textbf{proj}_2\, x)) \\
=& \lambda x.(\lambda z.(\textbf{lift}^m \lambda y.(\textbf{proj}_1\, x, y))\, ((\textbf{lift}^m g)\, z))\, \textbf{proj}_2\, x \\
=& \lambda x.(\textbf{lift}^m(\lambda z.(\lambda y.(\textbf{proj}_1\, x, y))\, (g\, z)))\, (\textbf{proj}_2\, x) \\
=& \lambda x.(\textbf{lift}^m(\lambda z.(\textbf{proj}_1\, x, g\, z)))\, (\textbf{proj}_2\, x)
\end{aligned}
$$

$$
\begin{array}{ccc}
A \times S_m\ B & \xrightarrow{t^{\mathbb{S}_m}_{A,B}} & S_m\ (A \times B) \\
\big\downarrow f\times\mathrm{id} & & \big\downarrow \mathbb{S}_m\ (f\times\mathrm{id}) \\
A' \times S_m\ B & \xrightarrow{t^{\mathbb{S}_m}_{A',B}} & S_m\ (A' \times B)
\end{array}
\qquad
\begin{array}{ccc}
A \times S_m\ B & \xrightarrow{t^{\mathbb{S}_m}_{A,B}} & S_m\ (A \times B) \\
\big\downarrow \mathrm{id}\times\mathbb{S}_m\ g & & \big\downarrow \mathbb{S}_m\ (\mathrm{id}\times g) \\
A \times S_m\ B' & \xrightarrow{t^{\mathbb{S}_m}_{A,B'}} & S_m\ (A \times B')
\end{array}
$$

Figure A.5: Commutative diagram

$$
\begin{array}{ccccc}
(A \times B) \times \mathbb{S}_m\ C & & \xrightarrow{t^{\mathbb{S}_m}_{A\times B,C}} & & \mathbb{S}_m\ ((A \times B) \times C) \\
\big\downarrow \alpha^{-1} & & & & \big\downarrow \mathbb{S}_m\ \alpha^{-1} \\
A \times (B \times \mathbb{S}_m\ C) & \xrightarrow{\mathrm{id}\times t^{\mathbb{S}_m}_{B,C}} & A \times \mathbb{S}_m\ (B \times C) & \xrightarrow{t^{\mathbb{S}_m}_{A,B\times C}} & \mathbb{S}_m\ (A \times (B \times C))
\end{array}
\qquad
\begin{array}{ccc}
\top \times \mathbb{S}_m\ A & \xrightarrow{t^{\mathbb{S}_m}_{\top,A}} & \mathbb{S}_m(\top \times A) \\
& \searrow^{\pi_2} & \big\downarrow \mathbb{S}_m\ \pi_2 \\
& & \mathbb{S}_m\ A
\end{array}
$$

Figure A.6: Commutative diagram

and

$$
\begin{aligned}
&\mathbb{S}_m\ (\mathrm{id} \times g) \circ t^{\mathbb{S}_m}_{A,B} \\
=&\lambda x.(\mathbf{lift}^m(\mathrm{id} \times g))\ (\mathbf{lift}^m(\lambda y.(\mathbf{proj}_1\ x, y)))\ (\mathbf{proj}_2\ x) \\
=&\lambda x.(\lambda z.(\mathbf{lift}^m \mathrm{id} \times g)\ (\mathbf{lift}^m \lambda y.(\mathbf{proj}_1\ x, y))\ z)\ \mathbf{proj}_2\ x \\
=&\lambda x.(\mathbf{lift}^m(\lambda z.(\mathrm{id} \times g)\ (\lambda y.(\mathbf{proj}_1\ x, y))\ z))\ (\mathbf{proj}_2\ x) \\
=&\lambda x.(\mathbf{lift}^m(\lambda z.(\mathrm{id} \times g)\ (\mathbf{proj}_1\ x, z)))\ (\mathbf{proj}_2\ x) \\
=&\lambda x.(\mathbf{lift}^m(\lambda z.(\mathbf{proj}_1\ x, g\ z)))\ (\mathbf{proj}_2\ x)
\end{aligned}
$$

Now check that $t^{\mathbb{S}_m}$ satisfies the axioms for strength.

The left diagram in Figure A.6 commutes because:

$$
\begin{aligned}
&t^{\mathbb{S}_m}_{A,B\times C} \circ (\mathrm{id} \times t^{\mathbb{S}_m}_{B,C}) \circ \alpha^{-1} \\
=&\lambda x.t^{\mathbb{S}_m}_{A,B\times C}\ (\mathrm{id} \times t^{\mathbb{S}_m}_{B,C})\ (\mathbf{proj}_1\ \mathbf{proj}_1\ x, (\mathbf{proj}_2\ \mathbf{proj}_1\ x, \mathbf{proj}_2\ x)) \\
=&\lambda x.t^{\mathbb{S}_m}_{A,B\times C}\ (\mathbf{proj}_1\ \mathbf{proj}_1\ x, (\mathbf{lift}^m(\lambda y.(\mathbf{proj}_2\ \mathbf{proj}_1\ x, y)))\ (\mathbf{proj}_2\ x)) \\
=&\lambda x.(\mathbf{lift}^m(\lambda z.(\mathbf{proj}_1\ \mathbf{proj}_1\ x, z)))\ (\mathbf{lift}^m(\lambda y.(\mathbf{proj}_2\ \mathbf{proj}_1\ x, y)))\ (\mathbf{proj}_2\ x) \\
=&\lambda x.(\lambda w.(\mathbf{lift}^m(\lambda z.(\mathbf{proj}_1\ \mathbf{proj}_1\ x, z)))\ (\mathbf{lift}^m(\lambda y.(\mathbf{proj}_2\ \mathbf{proj}_1\ x, y)))\ w)\ (\mathbf{proj}_2\ x) \\
=&\lambda x.(\mathbf{lift}^m(\lambda w.(\lambda z.(\mathbf{proj}_1\ \mathbf{proj}_1\ x, z))\ (\lambda y.(\mathbf{proj}_2\ \mathbf{proj}_1\ x, y))\ w))\ (\mathbf{proj}_2\ x) \\
=&\lambda x.(\mathbf{lift}^m(\lambda w.(\lambda z.(\mathbf{proj}_1\ \mathbf{proj}_1\ x, z))\ (\mathbf{proj}_2\ \mathbf{proj}_1\ x, w)))\ (\mathbf{proj}_2\ x) \\
=&\lambda x.(\mathbf{lift}^m(\lambda w.(\mathbf{proj}_1\ \mathbf{proj}_1\ x, (\mathbf{proj}_2\ \mathbf{proj}_1\ x, w))))\ (\mathbf{proj}_2\ x)
\end{aligned}
$$

and

$$
\begin{aligned}
&\mathbb{S}_m\ \alpha^{-1} \circ t^{\mathbb{S}_m}_{A\times B,C} \\
=&\lambda x.\mathbb{S}_m\ \alpha^{-1}\ (\mathbf{lift}^m(\lambda y.(\mathbf{proj}_1\ x, y)))\ (\mathbf{proj}_2\ x)
\end{aligned}
$$

$$=\lambda x.(\mathbf{lift}^m(\lambda y.(\mathbf{proj}_1\,\mathbf{proj}_1\,y,(\mathbf{proj}_2\,\mathbf{proj}_1\,y,\mathbf{proj}_2\,y))))\,(\mathbf{lift}^m(\lambda y.(\mathbf{proj}_1\,x,y)))\,(\mathbf{proj}_2\,x)$$
$$=\lambda x.(\lambda z.(\mathbf{lift}^m(\lambda y.(\mathbf{proj}_1\,\mathbf{proj}_1\,y,(\mathbf{proj}_2\,\mathbf{proj}_1\,y,\mathbf{proj}_2\,y))))\,(\mathbf{lift}^m(\lambda y.(\mathbf{proj}_1\,x,y)))\,z)\,(\mathbf{proj}_2\,x)$$
$$=\lambda x.(\mathbf{lift}^m(\lambda z.(\lambda y.(\mathbf{proj}_1\,\mathbf{proj}_1\,y,(\mathbf{proj}_2\,\mathbf{proj}_1\,y,\mathbf{proj}_2\,y)))\,(\lambda y.(\mathbf{proj}_1\,x,y))\,z))\,(\mathbf{proj}_2\,x)$$
$$=\lambda x.(\mathbf{lift}^m(\lambda z.(\lambda y.(\mathbf{proj}_1\,\mathbf{proj}_1\,y,(\mathbf{proj}_2\,\mathbf{proj}_1\,y,\mathbf{proj}_2\,y)))\,(\mathbf{proj}_1\,x,z)))\,(\mathbf{proj}_2\,x)$$
$$=\lambda x.(\mathbf{lift}^m(\lambda z.(\mathbf{proj}_1\,\mathbf{proj}_1\,x,(\mathbf{proj}_2\,\mathbf{proj}_1\,x,z))))\,(\mathbf{proj}_2\,x).$$

The right diagram in Figure A.6 commutes because:

$$\mathbb{S}_m\,\pi_2\circ t^{\mathbb{S}_m}_{\top,A}$$
$$=\lambda x.(\mathbf{lift}^m(\lambda y.\mathbf{proj}_2\,y))\,(\mathbf{lift}^m(\lambda y.(\mathbf{proj}_1\,x,y)))\,(\mathbf{proj}_2\,x)$$
$$=\lambda x.(\lambda z.(\mathbf{lift}^m(\lambda y.\mathbf{proj}_2\,y))\,(\mathbf{lift}^m(\lambda y.(\mathbf{proj}_1\,x,y)))\,z)\,(\mathbf{proj}_2\,x)$$
$$=\lambda x.(\mathbf{lift}^m(\lambda z.(\lambda y.\mathbf{proj}_2\,y)\,(\lambda y.(\mathbf{proj}_1\,x,y))\,z))\,(\mathbf{proj}_2\,x)$$
$$=\lambda x.(\mathbf{lift}^m(\lambda z.(\lambda y.\mathbf{proj}_2\,y)\,(\mathbf{proj}_1\,x,z)))\,(\mathbf{proj}_2\,x)$$
$$=\lambda x.(\mathbf{lift}^m(\lambda z.z))\,(\mathbf{proj}_2\,x)$$
$$=\lambda x.(\lambda z.z)\,(\mathbf{proj}_2\,x)$$
$$=\lambda x.\mathbf{proj}_2\,x$$
$$=\pi_2.$$

So $\mathbb{S}^m$ is a strong endofunctor.

Now, we need to check that $\mathbb{S}^{m_1<:m_2}$ is a strong natural transformation. To show this, we shall use the following equality:

$$\mathbf{lift}^m f=\lambda y.(y\ {}^m\!\!\gg\!\!=^1\ \lambda x.\mathbf{ret}\,(f\,x)) \tag{A.1}$$

$$\lambda y.(y\ {}^m\!\!\gg\!\!=^1\ \lambda x.\mathbf{ret}\,(f\,x))$$
$$=\lambda y.\mathbf{join}^{m,1}(\mathbf{lift}^m(\lambda x.\mathbf{ret}\,(f\,x)))\,y$$
$$=\lambda y.\mathbf{join}^{m,1}(\mathbf{lift}^m(\lambda x.(\lambda z.\mathbf{ret}\,z)\,(f\,x)))\,y$$
$$=\lambda y.\mathbf{join}^{m,1}(\lambda x.(\mathbf{lift}^m(\lambda z.\mathbf{ret}\,z))\,(\mathbf{lift}^m f)\,x)\,y$$
$$=\lambda y.\mathbf{join}^{m,1}(\mathbf{lift}^m(\lambda z.\mathbf{ret}\,z))\,(\mathbf{lift}^m f)\,y$$
$$=\lambda y.(\mathbf{lift}^m f)\,y\ {}^m\!\!\gg\!\!=^1\ \lambda z.\mathbf{ret}\,z$$
$$=\lambda y.(\mathbf{lift}^m f)\,y\ \text{[By Equation (2.8)]}$$
$$=\mathbf{lift}^m f$$

$$
\begin{array}{ccc}
S_{m_1}\ A & \xrightarrow{\mathbb{S}^{m_1<:m_2}_A} & S_{m_2}\ A \\
\downarrow{\scriptstyle \mathbb{S}_{m_1} f} & & \downarrow{\scriptstyle \mathbb{S}_{m_2} f} \\
S_{m_1}\ B & \xrightarrow{\mathbb{S}^{m_1<:m_2}_B} & S_{m_2}\ B
\end{array}
\qquad\qquad
\begin{array}{ccc}
A \times S_{m_1}\ B & \xrightarrow{\mathrm{id}\times\mathbb{S}^{m_1<:m_2}_B} & A \times S_{m_2}\ B \\
\downarrow{\scriptstyle t^{\mathbb{S}_{m_1}}_{A,B}} & & \downarrow{\scriptstyle t^{\mathbb{S}_{m_2}}_{A,B}} \\
S_{m_1}\ (A \times B) & \xrightarrow{\mathbb{S}^{m_1<:m_2}_{A\times B}} & S_{m_2}\ (A \times B)
\end{array}
$$

FIGURE A.7: Commutative diagram

As a side note, we can dualize the above argument to show that:

$$\mathbf{lift}^m f = \lambda y.((\lambda x.f\ (\mathbf{extr}\ x))\ {}^1\!\ll\!=^m\ y) \tag{A.2}$$

The left diagram in Figure A.7 commutes because:

$$
\begin{aligned}
&\mathbb{S}_{m_2} f \circ \mathbb{S}^{m_1<:m_2}_A \\
=&\lambda x.(\mathbf{lift}^{m_2} f)\ (\mathbf{up}^{m_1,m_2} x) \\
=&\lambda x.(\lambda y.(y\ {}^{m_2}\!\gg\!=^1\ \lambda z.\mathbf{ret}\ (f\ z)))\ (\mathbf{up}^{m_1,m_2} x) \\
=&\lambda x.\mathbf{up}^{m_1,m_2} x\ {}^{m_2}\!\gg\!=^1\ \lambda z.\mathbf{ret}\ (f\ z) \\
=&\lambda x.\mathbf{up}^{m_1,m_2}(x\ {}^{m_1}\!\gg\!=^1\ \lambda z.\mathbf{ret}\ (f\ z))\ \text{[By Equation (2.5)]} \\
=&\lambda x.\mathbf{up}^{m_1,m_2}((\mathbf{lift}^{m_1} f)\ x) \\
=&\mathbb{S}^{m_1<:m_2}_B \circ \mathbb{S}_{m_1} f
\end{aligned}
$$

The right diagram in Figure A.7 commutes because:

$$
\begin{aligned}
&t^{\mathbb{S}_{m_2}}_{A,B} \circ \mathrm{id} \times \mathbb{S}^{m_1<:m_2}_B \\
=&\lambda x.t^{\mathbb{S}_{m_2}}_{A,B}\ (\mathbf{proj}_1\ x, \mathbf{up}^{m_1,m_2}\mathbf{proj}_2\ x) \\
=&\lambda x.(\mathbf{lift}^{m_2}(\lambda y.(\mathbf{proj}_1\ x, y)))\ (\mathbf{up}^{m_1,m_2}\mathbf{proj}_2\ x) \\
=&\lambda x.\mathbf{up}^{m_1,m_2}\mathbf{proj}_2\ x\ {}^{m_2}\!\gg\!=^1\ \lambda z.\mathbf{ret}\ ((\lambda y.(\mathbf{proj}_1\ x, y))\ z) \\
=&\lambda x.\mathbf{up}^{m_1,m_2}\mathbf{proj}_2\ x\ {}^{m_2}\!\gg\!=^1\ \lambda z.\mathbf{ret}\ (\mathbf{proj}_1\ x, z) \\
=&\lambda x.\mathbf{up}^{m_1,m_2}(\mathbf{proj}_2\ x\ {}^{m_1}\!\gg\!=^1\ \lambda z.\mathbf{ret}\ (\mathbf{proj}_1\ x, z))\ \text{[By Equation (2.5)]} \\
=&\lambda x.\mathbf{up}^{m_1,m_2}((\lambda y.(y\ {}^{m_1}\!\gg\!=^1\ \lambda z.\mathbf{ret}\ ((\lambda w.(\mathbf{proj}_1\ x, w))\ z)))\ (\mathbf{proj}_2\ x)) \\
=&\lambda x.\mathbf{up}^{m_1,m_2}((\mathbf{lift}^{m_1}(\lambda w.(\mathbf{proj}_1\ x, w)))\ (\mathbf{proj}_2\ x)) \\
=&\mathbb{S}^{m_1<:m_2}_{A\times B} \circ t^{\mathbb{S}_{m_1}}_{A,B}
\end{aligned}
$$

This shows that $\mathbb{S}^{m_1<:m_2}$ is a strong natural transformation. Hence $\mathbb{S}$ is well-defined.
Next we show that $\mathbb{S}$ is indeed a functor.
We have: $\mathbb{S}^{m<:m}_A = \lambda x.\mathbf{up}^{m,m} x = \lambda x.x = \mathrm{id}_{\mathbb{S}_m\ A}$.
And, $\mathbb{S}^{m_2<:m_3}_A \circ \mathbb{S}^{m_1<:m_2}_A = \lambda x.\mathbf{up}^{m_2,m_3}(\mathbf{up}^{m_1,m_2} x) = \lambda x.\mathbf{up}^{m_1,m_3} x = \mathbb{S}^{m_1<:m_3}_A$.

Hence, $\mathbb{S}$ is a functor. Next we show that $\mathbb{S}$ is strong monoidal.

Define: $\eta : \mathbf{Id} \to \mathbb{S}_1$ and $\mu^{m_1,m_2} : \mathbb{S}_{m_1} \circ \mathbb{S}_{m_2} \to \mathbb{S}_{m_1 \cdot m_2}$ as:
$\eta_A = \lambda x.\mathbf{ret}\ x$ and $\mu^{m_1,m_2}_A = \lambda x.\mathbf{join}^{m_1,m_2} x$.

Need to check that η and μ^{m_1,m_2} are strong natural transformations.

The diagrams in Figure A.8 commute because:

$$\begin{aligned}
&\mathbb{S}_1 f \circ \eta_A\\
=&\lambda x.(\mathbf{lift}^1 f)\ (\mathbf{ret}\ x)\\
=&\lambda x.\mathbf{ret}\ x\ {}^1\!\!\gg\!\!=^1 \lambda y.\mathbf{ret}\ (f\ y)\\
=&\lambda x.(\lambda y.\mathbf{ret}\ (f\ y))\ x\\
=&\lambda x.\mathbf{ret}\ (f\ x)\\
=&\eta_B \circ f
\end{aligned}$$

$$\begin{aligned}
&t^{\mathbb{S}_1}_{A,B} \circ (\mathrm{id} \times \eta_B)\\
=&\ \lambda x.(\mathbf{lift}^1 (\lambda y.(\mathbf{proj}_1\ x, y)))\ (\mathbf{ret}\ \mathbf{proj}_2\ x)\\
=&\ \lambda x.\mathbf{ret}\ \mathbf{proj}_2\ x\ {}^1\!\!\gg\!\!=^1 \lambda z.\mathbf{ret}\ ((\lambda y.(\mathbf{proj}_1\ x, y))\ z)\\
=&\ \lambda x.\mathbf{ret}\ \mathbf{proj}_2\ x\ {}^1\!\!\gg\!\!=^1 \lambda z.\mathbf{ret}\ (\mathbf{proj}_1\ x, z)\\
=&\ \lambda x.(\lambda z.\mathbf{ret}\ (\mathbf{proj}_1\ x, z))\ (\mathbf{proj}_2\ x)\\
=&\ \lambda x.\mathbf{ret}\ (\mathbf{proj}_1\ x, \mathbf{proj}_2\ x) = \lambda x.\mathbf{ret}\ x = \eta_{A \times B}
\end{aligned}$$

This shows that η is a strong natural transformation.
Next, we show that μ^{m_1,m_2} is a strong natural transformation.
Note that

$$a\ {}^{m_1}\!\!\gg\!\!=^{m_2} \lambda y.y = \mathbf{join}^{m_1,m_2}((\mathbf{lift}^{m_1}(\lambda y.y))\ a) = \mathbf{join}^{m_1,m_2}((\lambda y.y)\ a) = \mathbf{join}^{m_1,m_2} a \qquad \text{(A.3)}$$

Now, the left diagram in Figure A.9 commutes because:

$$\begin{aligned}
&\mu^{m_1,m_2}_B \circ \mathbb{S}_{m_1} \mathbb{S}_{m_2} f\\
=&\lambda x.\mathbf{join}^{m_1,m_2}((\mathbf{lift}^{m_1}(\mathbf{lift}^{m_2} f))\ x)\\
=&\lambda x.x\ {}^{m_1}\!\!\gg\!\!=^{m_2} \lambda y.(\mathbf{lift}^{m_2} f)\ y\\
=&\lambda x.x\ {}^{m_1}\!\!\gg\!\!=^{m_2} \lambda y.(y\ {}^{m_2}\!\!\gg\!\!=^1 \lambda z.\mathbf{ret}\ (f\ z))\\
=&\lambda x.(x\ {}^{m_1}\!\!\gg\!\!=^{m_2} \lambda y.y)\ {}^{m_1 \cdot m_2}\!\!\gg\!\!=^1 \lambda z.\mathbf{ret}\ (f\ z)\ \text{[By Equation (2.9)]}\\
=&\lambda x.\mathbf{join}^{m_1,m_2} x\ {}^{m_1 \cdot m_2}\!\!\gg\!\!=^1 \lambda z.\mathbf{ret}\ (f\ z)\\
=&\lambda x.(\mathbf{lift}^{m_1 \cdot m_2} f)\ (\mathbf{join}^{m_1,m_2} x)\\
=&\mathbb{S}_{m_1 \cdot m_2} f \circ \mu^{m_1,m_2}_A
\end{aligned}$$

The right diagram in Figure A.9 commutes because:

$$\begin{aligned}
&\mu^{m_1,m_2}_{A \times B} \circ \mathbb{S}_{m_1} t^{\mathbb{S}_{m_2}}_{A,B} \circ t^{\mathbb{S}_{m_1}}_{A,\mathbb{S}_{m_2} B}\\
=&\lambda x.(\lambda y.\mathbf{join}^{m_1,m_2} y)\ (\mathbf{lift}^{m_1} t^{\mathbb{S}_{m_2}}_{A,B})\ (\lambda z.(\mathbf{lift}^{m_1}(\lambda y.(\mathbf{proj}_1\ z, y)))\ (\mathbf{proj}_2\ z))\ x
\end{aligned}$$

$$
\begin{array}{ccc}
A & \xrightarrow{\eta_A} & \mathbb{S}_1\ A \\
\downarrow{\scriptstyle f} & & \downarrow{\scriptstyle \mathbb{S}_1\ f} \\
B & \xrightarrow{\eta_B} & \mathbb{S}_1\ B
\end{array}
\qquad\qquad
\begin{array}{ccc}
A \times B & \xrightarrow{\mathrm{id}\times\eta_B} & A \times \mathbb{S}_1\ B \\
\downarrow{\scriptstyle \mathrm{id}} & & \downarrow{\scriptstyle t^{\mathbb{S}_1}_{A,B}} \\
A \times B & \xrightarrow{\eta_{A\times B}} & \mathbb{S}_1\ (A \times B)
\end{array}
$$

Figure A.8: Commutative diagram

$$
\begin{array}{ccc}
\mathbb{S}_{m_1}\ \mathbb{S}_{m_2}\ A & \xrightarrow{\mu^{m_1,m_2}_A} & \mathbb{S}_{m_1\cdot m_2}\ A \\
\downarrow{\scriptstyle \mathbb{S}_{m_1}\ \mathbb{S}_{m_2}\ f} & & \downarrow{\scriptstyle \mathbb{S}_{m_1\cdot m_2}\ f} \\
\mathbb{S}_{m_1}\ \mathbb{S}_{m_2}\ B & \xrightarrow{\mu^{m_1,m_2}_B} & \mathbb{S}_{m_1\cdot m_2}\ B
\end{array}
\qquad
\begin{array}{ccc}
A \times \mathbb{S}_{m_1}\ \mathbb{S}_{m_2}\ B & \xrightarrow{\mathrm{id}\times\mu^{m_1,m_2}_B} & A \times \mathbb{S}_{m_1\cdot m_2}\ B \\
{\scriptstyle \mathbb{S}_{m_1}\ t^{\mathbb{S}_{m_2}}_{A,B} \circ t^{\mathbb{S}_{m_1}}_{A,\mathbb{S}_{m_2}\ B}}\downarrow & & \downarrow{\scriptstyle t^{\mathbb{S}_{m_1\cdot m_2}}_{A,B}} \\
\mathbb{S}_{m_1}\ \mathbb{S}_{m_2}\ (A \times B) & \xrightarrow{\mu^{m_1,m_2}_{A\times B}} & \mathbb{S}_{m_1\cdot m_2}\ (A \times B)
\end{array}
$$

Figure A.9: Commutative diagram

$$
\begin{array}{ccc}
\mathbb{S}_{m_1}\ \mathbb{S}_{m_2}\ A & \xrightarrow{\mu^{m_1,m_2}_A} & \mathbb{S}_{m_1\cdot m_2}\ A \\
{\scriptstyle \mathbb{S}^{m_1 <: m_1'}_{\mathbb{S}_{m_2}\ A}}\downarrow & & \downarrow{\scriptstyle \mathbb{S}^{m_1\cdot m_2 <: m_1'\cdot m_2}_A} \\
\mathbb{S}_{m_1'}\ \mathbb{S}_{m_2}\ A & \xrightarrow{\mu^{m_1',m_2}_A} & \mathbb{S}_{m_1'\cdot m_2}\ A
\end{array}
\qquad
\begin{array}{ccc}
\mathbb{S}_{m_1}\ \mathbb{S}_{m_2}\ A & \xrightarrow{\mu^{m_1,m_2}_A} & \mathbb{S}_{m_1\cdot m_2}\ A \\
{\scriptstyle \mathbb{S}_{m_1}\ \mathbb{S}^{m_2 <: m_2'}_A}\downarrow & & \downarrow{\scriptstyle \mathbb{S}^{m_1\cdot m_2 <: m_1\cdot m_2'}_A} \\
\mathbb{S}_{m_1}\ \mathbb{S}_{m_2'}\ A & \xrightarrow{\mu^{m_1,m_2'}_A} & \mathbb{S}_{m_1\cdot m_2'}\ A
\end{array}
$$

Figure A.10: Commutative diagram

$$
\begin{aligned}
&=\lambda x.(\lambda y.\mathbf{join}^{m_1,m_2}\, y)\ (\mathbf{lift}^{m_1} t^{\mathbb{S}_{m_2}}_{A,B})\ (\mathbf{lift}^{m_1}(\lambda y.(\mathbf{proj}_1\ x, y)))\ (\mathbf{proj}_2\ x) \\
&=\lambda x.(\lambda y.\mathbf{join}^{m_1,m_2}\, y)\ (\lambda z.(\mathbf{lift}^{m_1} t^{\mathbb{S}_{m_2}}_{A,B})\ (\mathbf{lift}^{m_1}(\lambda y.(\mathbf{proj}_1\ x, y)))\ z)\ (\mathbf{proj}_2\ x) \\
&=\lambda x.(\lambda y.\mathbf{join}^{m_1,m_2}\, y)\ (\mathbf{lift}^{m_1}(\lambda z.t^{\mathbb{S}_{m_2}}_{A,B}\ (\mathbf{proj}_1\ x, z)))\ (\mathbf{proj}_2\ x) \\
&=\lambda x.(\lambda y.\mathbf{join}^{m_1,m_2}\, y)\ (\mathbf{lift}^{m_1}(\lambda z.(\lambda u.(\mathbf{lift}^{m_2}(\lambda v.(\mathbf{proj}_1\ u, v)))\ (\mathbf{proj}_2\ u))\ (\mathbf{proj}_1\ x, z))) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (\mathbf{proj}_2\ x) \\
&=\lambda x.\mathbf{join}^{m_1,m_2}((\mathbf{lift}^{m_1}(\lambda z.(\mathbf{lift}^{m_2}(\lambda v.(\mathbf{proj}_1\ x, v)))\ z))\ (\mathbf{proj}_2\ x)) \\
&=\lambda x.\mathbf{proj}_2\ x\ \ {}^{m_1}\!\gg\!\!=^{m_2}\ \lambda z.(\mathbf{lift}^{m_2}(\lambda v.(\mathbf{proj}_1\ x, v)))\ z \\
&=\lambda x.\mathbf{proj}_2\ x\ \ {}^{m_1}\!\gg\!\!=^{m_2}\ \lambda z.(z\ \ {}^{m_2}\!\gg\!\!=^{1}\ \lambda u.\mathbf{ret}\ (\mathbf{proj}_1\ x, u)) \\
&=\lambda x.(\mathbf{proj}_2\ x\ \ {}^{m_1}\!\gg\!\!=^{m_2}\ (\lambda y.y))\ \ {}^{m_1\cdot m_2}\!\gg\!\!=^{1}\ \lambda u.\mathbf{ret}\ (\mathbf{proj}_1\ x, u)\ \text{[By Equation (2.9)]} \\
&=\lambda x.(\mathbf{join}^{m_1,m_2}(\mathbf{proj}_2\ x))\ \ {}^{m_1\cdot m_2}\!\gg\!\!=^{1}\ \lambda u.\mathbf{ret}\ (\mathbf{proj}_1\ x, u)
\end{aligned}
$$

and

$$
\begin{aligned}
&t^{\mathbb{S}_{m_1\cdot m_2}}_{A,B} \circ \mathrm{id} \times \mu^{m_1,m_2}_B \\
&=\lambda x.(\lambda z.(\mathbf{lift}^{m_1\cdot m_2}(\lambda y.(\mathbf{proj}_1\ z, y)))\ (\mathbf{proj}_2\ z))\ (\mathbf{proj}_1\ x, \mathbf{join}^{m_1,m_2}\mathbf{proj}_2\ x) \\
&=\lambda x.(\mathbf{lift}^{m_1\cdot m_2}(\lambda y.(\mathbf{proj}_1\ x, y)))\ (\mathbf{join}^{m_1,m_2}\mathbf{proj}_2\ x) \\
&=\lambda x.(\mathbf{join}^{m_1,m_2}(\mathbf{proj}_2\ x))\ \ {}^{m_1\cdot m_2}\!\gg\!\!=^{1}\ \lambda u.\mathbf{ret}\ (\mathbf{proj}_1\ x, u).
\end{aligned}
$$

This shows that μ^{m_1,m_2} is a strong natural transformation.

Next, we show that μ^{m_1,m_2} is natural in both m_1 and m_2.

$$
\begin{array}{ccccc}
\mathbb{S}_1\, \mathbb{S}_m\, A & \xleftarrow{\eta_{\mathbb{S}_m\, A}} & \mathbb{S}_m\, A & \xrightarrow{\mathbb{S}_m\, \eta_A} & \mathbb{S}_m\, \mathbb{S}_1\, A \\
& \searrow^{\mu_A^{1,m}} & \big\downarrow \text{id} & \swarrow^{\mu_A^{m,1}} & \\
& & \mathbb{S}_m\, A & &
\end{array}
\qquad
\begin{array}{ccc}
\mathbb{S}_{m_1}\, \mathbb{S}_{m_2}\, \mathbb{S}_{m_3}\, A & \xrightarrow{\mathbb{S}_{m_1}\, \mu_A^{m_2,m_3}} & \mathbb{S}_{m_1}\, \mathbb{S}_{m_2 \cdot m_3}\, A \\
\big\downarrow \mu^{m_1,m_2}_{\mathbb{S}_{m_3}\, A} & & \big\downarrow \mu_A^{m_1,m_2 \cdot m_3} \\
\mathbb{S}_{m_1 \cdot m_2}\, \mathbb{S}_{m_3}\, A & \xrightarrow{\mu_A^{m_1 \cdot m_2, m_3}} & \mathbb{S}_{m_1 \cdot m_2 \cdot m_3}\, A
\end{array}
$$

FIGURE A.11: Commutative diagram

The diagrams in Figure A.10 commute because:

$$
\begin{aligned}
& \mu_A^{m_1',m_2} \circ \mathbb{S}^{m_1 <: m_1'}_{\mathbb{S}_{m_2}\, A} \\
=& \lambda x.\mathbf{join}^{m_1',m_2}(\mathbf{up}^{m_1,m_1'}\, x) \\
=& \lambda x.\mathbf{up}^{m_1,m_1'}\, x \ {}^{m_1'}\!\gg\!=^{m_2}\ \lambda y.y \\
=& \lambda x.\mathbf{up}^{m_1 \cdot m_2, m_1' \cdot m_2}(x \ {}^{m_1}\!\gg\!=^{m_2}\ \lambda y.y) \ [\text{By Eqn. (2.5)}] \\
=& \lambda x.\mathbf{up}^{m_1 \cdot m_2, m_1' \cdot m_2}(\mathbf{join}^{m_1,m_2}\, x) \\
=& \mathbb{S}_A^{m_1 \cdot m_2 <: m_1' \cdot m_2} \circ \mu_A^{m_1,m_2}
\end{aligned}
$$

and

$$
\begin{aligned}
& \mu_A^{m_1,m_2'} \circ \mathbb{S}_{m_1}\, \mathbb{S}_A^{m_2 <: m_2'} \\
=& \lambda x.\mathbf{join}^{m_1,m_2'}(\mathbf{lift}^{m_1}(\lambda y.\mathbf{up}^{m_2,m_2'}\, y))\, x \\
=& \lambda x.x \ {}^{m_1}\!\gg\!=^{m_2'}\ (\lambda y.\mathbf{up}^{m_2,m_2'}\, y) \\
=& \lambda x.\mathbf{up}^{m_1 \cdot m_2, m_1 \cdot m_2'}(x \ {}^{m_1}\!\gg\!=^{m_2}\ \lambda y.y) \ [\text{By Eqn. (2.6)}] \\
=& \lambda x.\mathbf{up}^{m_1 \cdot m_2, m_1 \cdot m_2'}(\mathbf{join}^{m_1,m_2}\, x) \\
=& \mathbb{S}_A^{m_1 \cdot m_2 <: m_1 \cdot m_2'} \circ \mu_A^{m_1,m_2}
\end{aligned}
$$

This shows that μ^{m_1,m_2} is natural in both m_1 and m_2.
Now we show that $\mathbb{S}$ satisfies the axioms for lax monoidal functor.

The left diagram in Figure A.11 commutes because:

$$
\begin{aligned}
& \mu_A^{1,m} \circ \eta_{\mathbb{S}_m\, A} \\
=& \lambda x.\mathbf{join}^{1,m}(\mathbf{ret}\, x) \\
=& \lambda x.\mathbf{ret}\, x \ {}^{1}\!\gg\!=^{m}\ (\lambda y.y) \\
=& \lambda x.(\lambda y.y)\, x \ [\text{By Eqn. (2.7)}] \\
=& \lambda x.x = \text{id}
\end{aligned}
\qquad
\begin{aligned}
& \mu_A^{m,1} \circ \mathbb{S}_m\, \eta_A \\
=& \lambda x.\mathbf{join}^{m,1}(\mathbf{lift}^{m}(\lambda y.\mathbf{ret}\, y))\, x \\
=& \lambda x.x \ {}^{m}\!\gg\!=^{1}\ \lambda y.\mathbf{ret}\, y \\
=& \lambda x.x \ [\text{By Eqn. (2.8)}] \\
=& \text{id}
\end{aligned}
$$

The right diagram in Figure A.11 commutes because:

$$
\begin{aligned}
&\mu_A^{m_1,m_2\cdot m_3} \circ \mathbb{S}_{m_1}\, \mu_A^{m_2,m_3}\\
=&\lambda x.\mathbf{join}^{m_1,m_2\cdot m_3}(\mathbf{lift}^{m_1}(\lambda y.\mathbf{join}^{m_2,m_3}\, y))\, x\\
=&\lambda x.x\ {}^{m_1}\!\!\gg\!\!=^{m_2\cdot m_3}\ \lambda y.\mathbf{join}^{m_2,m_3}\, y\\
=&\lambda x.x\ {}^{m_1}\!\!\gg\!\!=^{m_2\cdot m_3}\ \lambda y.(y\ {}^{m_2}\!\!\gg\!\!=^{m_3}\ \lambda z.z)\\
=&\lambda x.(x\ {}^{m_1}\!\!\gg\!\!=^{m_2}\ (\lambda w.w))\ {}^{m_1\cdot m_2}\!\!\gg\!\!=^{m_3}\ \lambda z.z\ [\text{By Eqn. (2.9)}]\\
=&\lambda x.\mathbf{join}^{m_1\cdot m_2,m_3}(x\ {}^{m_1}\!\!\gg\!\!=^{m_2}\ \lambda w.w)\\
=&\lambda x.\mathbf{join}^{m_1\cdot m_2,m_3}(\mathbf{join}^{m_1,m_2}\, x)\\
=&\mu_A^{m_1\cdot m_2,m_3} \circ \mu_{\mathbb{S}_{m_3}A}^{m_1,m_2}
\end{aligned}
$$

Hence $\mathbb{S}$ is a lax monoidal functor.

Now $\mathbb{S}$ is also a strong monoidal functor because η and μ^{m_1,m_2} are isomorphisms.

Define: $\epsilon : \mathbb{S}_1 \to \mathbf{Id}$ and $\delta^{m_1,m_2} : \mathbb{S}_{m_1\cdot m_2} \to \mathbb{S}_{m_1} \circ \mathbb{S}_{m_2}$ as:

$\epsilon_A = \lambda x.\mathbf{extr}\, x$ and $\delta_A^{m_1,m_2} = \lambda x.\mathbf{fork}^{m_1,m_2}\, x$.

By dualizing the arguments presented above, we can show that ϵ and δ^{m_1,m_2} are strong natural transformations.

Further, η and ϵ are inverses of one another because $\eta_A \circ \epsilon_A = \lambda x.\mathbf{ret}\ (\mathbf{extr}\ x) = \lambda x.x$ and $\epsilon_A \circ \eta_A = \lambda x.\mathbf{extr}\ (\mathbf{ret}\ x) = \lambda x.x$.

Similarly, μ^{m_1,m_2} and δ^{m_1,m_2} are inverses of one another.

Hence, $\mathbb{S}$ is a strong monoidal functor.

As such, $[\![_]\!]_{(\mathbb{F},\mathbb{S})}$ provides a sound interpretation of GMCC($\mathcal{M}$). This is the generic model of GMCC($\mathcal{M}$).

Now we show that if $\Gamma \vdash b : B$ in GMCC($\mathcal{M}$), then $[\![\Gamma \vdash b : B]\!]_{(\mathbb{F},\mathbb{S})} = \lambda x.b\{\text{proj}_i^n\ x/x_i\} \in \text{Hom}_{\mathbb{F}}(\Pi A_i, B)$ where $\Gamma = x_1 : A_1, x_2 : A_2, \ldots, x_n : A_n$ and $\Pi A_i = A_1 \times A_2 \times \ldots \times A_n$. For $n \in \mathbb{N}$ and $1 \leq i \leq n$, $\mathbf{proj}_i^n\ x$ is defined as:

$$
\begin{aligned}
\mathbf{proj}_1^1\ x &= x\\
\mathbf{proj}_i^2\ x &= \mathbf{proj}_i\ x \text{ for } i = 1,2\\
\mathbf{proj}_i^{n+1}\ x &= \mathbf{proj}_i^n\ (\mathbf{proj}_1\ x) \text{ for } 2 \leq n \text{ and } 1 \leq i \leq n\\
\mathbf{proj}_{n+1}^{n+1}\ x &= \mathbf{proj}_2\ x \text{ for } 2 \leq n
\end{aligned}
$$

The proof is by induction on $\Gamma \vdash b : B$.

- λ-calculus. Standard.

- Rule MC-Return. Have: $\Gamma \vdash \textbf{ret}\ b : S_1\ B$ where $\Gamma \vdash b : B$.
  By IH, $[\![\Gamma \vdash b : B]\!] = \lambda x.b\ \{\textbf{proj}_i^n\ x/x_i\}$.
  Now, $[\![\Gamma \vdash \textbf{ret}\ b : S_1\ B]\!] = \eta_B \circ [\![\Gamma \vdash b : B]\!] = \lambda x.(\textbf{ret}\ b)\ \{\textbf{proj}_i^n\ x/x_i\}$.

- Rule MC-Extract. Similar to rule MC-Return.

- Rule MC-Join. Have: $\Gamma \vdash \textbf{join}^{m_1,m_2}\, b : S_{m_1 \cdot m_2}\ B$ where $\Gamma \vdash b : S_{m_1}\ S_{m_2}\ B$.
  By IH, $[\![\Gamma \vdash b : B]\!] = \lambda x.b\ \{\textbf{proj}_i^n\ x/x_i\}$.
  Now, $[\![\Gamma \vdash \textbf{join}^{m_1,m_2}\, b : S_{m_1 \cdot m_2}\ B]\!] = \mu_B^{m_1,m_2} \circ [\![\Gamma \vdash b : B]\!] = \lambda x.(\textbf{join}^{m_1,m_2}\, b)\ \{\textbf{proj}_i^n\ x/x_i\}$.

- Rule MC-Fork. Similar to rule MC-Join.

- Rule MC-Fmap. Have: $\Gamma \vdash \textbf{lift}^m f : S_m\ B \to S_m\ C$ where $\Gamma \vdash f : B \to C$.
  By IH, $[\![\Gamma \vdash f : B \to C]\!] = \lambda x.f\ \{\textbf{proj}_i^n\ x/x_i\}$.
  Now,

$$
\begin{aligned}
&[\![\Gamma \vdash \textbf{lift}^m f : S_m\ B \to S_m\ C]\!] \\
=&\Lambda\Big(\mathbb{S}_m\ (\Lambda^{-1}[\![\Gamma \vdash f : B \to C]\!]) \circ t^{\mathbb{S}_m}_{\Gamma,B}\Big) \\
=&\lambda y.\lambda z.(\mathbb{S}_m\ (\Lambda^{-1}[\![\Gamma \vdash f : B \to C]\!]) \circ t^{\mathbb{S}_m}_{\Gamma,B})\ (y,z) \\
=&\lambda y.\lambda z.(\textbf{lift}^m (\Lambda^{-1}[\![\Gamma \vdash f : B \to C]\!]))\ (\textbf{lift}^m (\lambda w.(y,w)))\ z \\
=&\lambda y.\textbf{lift}^m (\lambda z.(\Lambda^{-1}[\![\Gamma \vdash f : B \to C]\!])\ (y,z)) \\
=&\lambda y.\textbf{lift}^m (\lambda z.(\lambda u.([\![\Gamma \vdash f : B \to C]\!]\ \textbf{proj}_1\ u)\ (\textbf{proj}_2\ u))\ (y,z)) \\
=&\lambda y.\textbf{lift}^m (\lambda z.([\![\Gamma \vdash f : B \to C]\!]\ y)\ z) \\
=&\lambda y.\textbf{lift}^m ([\![\Gamma \vdash f : B \to C]\!]\ y) \\
=&\lambda x.\textbf{lift}^m (f\ \{\textbf{proj}_i^n\ x/x_i\}) \\
=&\lambda x.(\textbf{lift}^m f)\ \{\textbf{proj}_i^n\ x/x_i\}
\end{aligned}
$$

- Rule MC-Up. Have: $\Gamma \vdash \textbf{up}^{m_1,m_2}\, b : S_{m_2}\ B$ where $\Gamma \vdash b : S_{m_1}\ B$ and $m_1 <: m_2$.
  By IH, $[\![\Gamma \vdash b : B]\!] = \lambda x.b\ \{\textbf{proj}_i^n\ x/x_i\}$.
  Now, $[\![\Gamma \vdash \textbf{up}^{m_1,m_2}\, b : S_{m_2}\ B]\!] = \mathbb{S}_B^{m_1 <: m_2} \circ [\![\Gamma \vdash b : B]\!] = \lambda x.(\textbf{up}^{m_1,m_2}\, b)\ \{\textbf{proj}_i^n\ x/x_i\}$.

Hence, $[\![\Gamma \vdash b : B]\!]_{(\mathbb{F},\mathbb{S})} = \lambda x.b\{\text{proj}_i^n\ x/x_i\}$.
Now, let $\Gamma \vdash b_1 : B$ and $\Gamma \vdash b_2 : B$ such that $[\![b_1]\!] = [\![b_2]\!]$ in all models of GMCC($\mathcal{M}$). Then $[\![b_1]\!]_{(\mathbb{F},\mathbb{S})} = [\![b_2]\!]_{(\mathbb{F},\mathbb{S})}$.
Hence, $\lambda x.b_1\ \{\textbf{proj}_i^n\ x/x_i\} \equiv \lambda x.b_2\ \{\textbf{proj}_i^n\ x/x_i\}$. Therefore, $b_1 \equiv b_2$.
This completes the proof. □

Theorem A.7 (Theorem 2.7) The generic model satisfies the universal property.

Proof. Let the parametrizing monoid be $\mathcal{M}$. For GMCC($\mathcal{M}$), let *Cl* be the classifying category and let G be the generic model.

First, we show how to define models of GMCC($\mathcal{M}$) in other categories using models of GMCC($\mathcal{M}$) in Cl. Let $\mathbb{D}$ be any bicartesian closed category and let F be a finite-product-preserving functor from Cl to $\mathbb{D}$. Now, given any model W of GMCC($\mathcal{M}$) in Cl, we can define a model $F(W)$ of GMCC($\mathcal{M}$) in $\mathbb{D}$ as follows:

$$\llbracket A \rrbracket_{F(W)} \triangleq F(\llbracket A \rrbracket_W)$$
$$\llbracket \Gamma \vdash a : A \rrbracket_{F(W)} \triangleq \Pi \llbracket A_i \rrbracket_{F(W)} (= \Pi F(\llbracket A_i \rrbracket_W)) \xrightarrow{p^{-1}} F(\Pi \llbracket A_i \rrbracket_W) \xrightarrow{F(\llbracket a \rrbracket_W)} F(\llbracket A \rrbracket_W)$$

where $\Gamma = x_1 : A_1, x_2 : A_2, \ldots, x_n : A_n$ and p is a witness of the finite-product-preserving property of F. Next, we check that $F(W)$ is indeed a model of GMCC($\mathcal{M}$).

If $\Gamma \vdash a : A$ in GMCC($\mathcal{M}$), then $\llbracket \Gamma \vdash a : A \rrbracket_W \in \mathrm{Hom}_{Cl}(\llbracket \Gamma \rrbracket_W, \llbracket A \rrbracket_W)$.
Therefore, $\llbracket \Gamma \vdash a : A \rrbracket_{F(W)} \in \mathrm{Hom}_{\mathbb{D}}(\llbracket \Gamma \rrbracket_{F(W)}, \llbracket A \rrbracket_{F(W)})$. Hence, $F(W)$ is a well-defined model of GMCC($\mathcal{M}$).

Now, say $\Gamma \vdash a_1 : A$ and $\Gamma \vdash a_2 : A$ such that $a_1 \equiv a_2$ in GMCC($\mathcal{M}$).
Then, $\llbracket \Gamma \vdash a_1 : A \rrbracket_W = \llbracket \Gamma \vdash a_2 : A \rrbracket_W \in \mathrm{Hom}_{Cl}(\llbracket \Gamma \rrbracket_W, \llbracket A \rrbracket_W)$.
So, $F(\llbracket \Gamma \vdash a_1 : A \rrbracket_W) \circ q = F(\llbracket \Gamma \vdash a_2 : A \rrbracket_W) \circ q \in \mathrm{Hom}_{\mathbb{D}}(\llbracket \Gamma \rrbracket_{F(W)}, \llbracket A \rrbracket_{F(W)})$.
Or, $\llbracket \Gamma \vdash a_1 : A \rrbracket_{F(W)} = \llbracket \Gamma \vdash a_2 : A \rrbracket_{F(W)} \in \mathrm{Hom}_{\mathbb{D}}(\llbracket \Gamma \rrbracket_{F(W)}, \llbracket A \rrbracket_{F(W)})$.
Hence, $F(W)$ is a sound model of GMCC($\mathcal{M}$).

Next, we show the universal property: for any given model W' of GMCC($\mathcal{M}$) in any bicartesian closed category $\mathbb{D}$, there exists a *unique* finite-product-preserving functor, F, from Cl to $\mathbb{D}$ such that $F(G) = W'$.

Given W', we define a functor, F, from Cl to $\mathbb{D}$, as follows:

$$F(A) \triangleq \llbracket A \rrbracket_{W'}$$
$$F(A \xrightarrow{a} B) \triangleq \llbracket x : A \vdash a\, x : B \rrbracket_{W'}$$

Since $A \xrightarrow{a} B$ in Cl, we know that $\varnothing \vdash a : A \to B$ in GMCC($\mathcal{M}$).
Therefore, $x : A \vdash a\, x : B$ in GMCC($\mathcal{M}$). So, $\llbracket x : A \vdash a\, x : B \rrbracket_{W'} \in \mathrm{Hom}_{\mathbb{D}}(\llbracket A \rrbracket_{W'}, \llbracket B \rrbracket_{W'})$.
Here, we need to check that the definition of F respects the equivalence relation on morphisms in Cl. In other words, we need to check that if $\varnothing \vdash a_1 : A \to B$ and $\varnothing \vdash a_2 : A \to B$ such that $a_1 \equiv a_2$ in GMCC($\mathcal{M}$), then $\llbracket x : A \vdash a_1\, x : B \rrbracket_{W'} = \llbracket x : A \vdash a_2\, x : B \rrbracket_{W'}$. But this is true because W' is a sound model of GMCC($\mathcal{M}$). So F is well-defined.

Next, we check that F is indeed a functor.

$$F(A \xrightarrow{\mathrm{id}} A) = \llbracket x : A \vdash x : A \rrbracket_{W'} = \llbracket A \rrbracket_{W'} \xrightarrow{\mathrm{id}} \llbracket A \rrbracket_{W'}$$

$$\begin{aligned}
&F(A \xrightarrow{f} B \xrightarrow{g} C) \\
=&\llbracket x : A \vdash g\, f\, x : C \rrbracket_{W'} \\
=&\mathrm{app} \circ \langle \llbracket x : A \vdash g : B \to C \rrbracket_{W'}, \llbracket x : A \vdash f\, x : B \rrbracket_{W'} \rangle \\
=&\mathrm{app} \circ \langle \llbracket x : A \vdash g : B \to C \rrbracket_{W'}, \mathrm{app} \circ \langle \llbracket x : A \vdash f : A \to B \rrbracket_{W'}, \llbracket x : A \vdash x : A \rrbracket_{W'} \rangle \rangle
\end{aligned}$$

$$
\begin{aligned}
&= \text{app} \circ \langle [\![\varnothing \vdash g : B \to C]\!]_{W'} \circ \langle\rangle, \text{app} \circ \langle [\![\varnothing \vdash f : A \to B]\!]_{W'} \circ \langle\rangle, \text{id} \rangle \rangle \\
&= \text{app} \circ \langle [\![\varnothing \vdash g : B \to C]\!]_{W'} \circ \langle\rangle, \text{id} \rangle \circ \text{app} \circ \langle [\![\varnothing \vdash f : A \to B]\!]_{W'} \circ \langle\rangle, \text{id} \rangle \\
&= [\![y : B \vdash g\ y : C]\!]_{W'} \circ [\![x : A \vdash f\ x : B]\!]_{W'} \\
&= F(B \xrightarrow{g} C) \circ F(A \xrightarrow{f} B)
\end{aligned}
$$

Observe that $F(A_1 \times A_2) = [\![A_1 \times A_2]\!]_{W'} = [\![A_1]\!]_{W'} \times [\![A_2]\!]_{W'} = FA_1 \times FA_2$.
Hence, F is a finite-product-preserving functor.

Now, we show that $F(G) = W'$.
$[\![A]\!]_{F(G)} = F([\![A]\!]_G) = F(A) = [\![A]\!]_{W'}$.
Further,

$$
\begin{aligned}
&[\![\Gamma \vdash a : A]\!]_{F(G)} \\
&= F([\![\Gamma \vdash a : A]\!]_G) \\
&= F(\lambda x.a\{\text{proj}_i^n\ x/x_i\}) \quad [\text{Here}, \Gamma = x_1 : A_1, x_2 : A_2, \ldots, x_n : A_n] \\
&= [\![x : \Pi A_i \vdash a\{\text{proj}_i^n\ x/x_i\} : A]\!]_{W'} \\
&= [\![\Gamma \vdash a : A]\!]_{W'}.
\end{aligned}
$$

Therefore, $F(G) = W'$.

Next, we show the uniqueness property. Let F' be a finite-product-preserving functor from Cl to $\mathbb{D}$ such that $F'(G) = W'$. Need to show that $F' = F$.

Now, $F'(A) = F'([\![A]\!]_G) = [\![A]\!]_{W'} = F(A)$.
Also, $F'(A \xrightarrow{a} B) = F'([\![x : A \vdash a\ x : B]\!]_G) = [\![x : A \vdash a\ x : B]\!]_{W'} = F(A \xrightarrow{a} B)$.
Therefore, $F' = F$.
This completes the proof. □

A.6 Proofs of Lemmas/Theorems Stated in Section 2.7

Theorem A.8 (Theorem 2.8) If $\Gamma \vdash a : A$ in GMCC($\mathcal{L}$), then $\overline{\Gamma} \vdash \overline{a} : \overline{A}$ in DCC$_e$($\mathcal{L}$). Further, if $\Gamma \vdash a_1 : A$ and $\Gamma \vdash a_2 : A$ such that $a_1 \equiv a_2$ in GMCC($\mathcal{L}$), then $\overline{a_1} \simeq \overline{a_2}$ in DCC$_e$($\mathcal{L}$).

Proof. Follows from Theorems A.2 and A.4. □

Lemma A.9 (Lemma 2.9) If $\ell \sqsubseteq B$ in DCC$_e$, then there exists a term $\varnothing \vdash j^{\ell}_{\underline{B}} : S_\ell\ \underline{B} \to \underline{B}$ in GMCC such that $\lfloor j^{\ell}_{\underline{B}} \rfloor \equiv \lambda x.x$.

Proof. By induction on $\ell \sqsubseteq B$.

- Rule Prot-Prod. Have: $\ell \sqsubseteq A \times B$ where $\ell \sqsubseteq A$ and $\ell \sqsubseteq B$.
  By IH, $\exists\ \varnothing \vdash j^{\ell}_{\underline{A}} : S_\ell\ \underline{A} \to \underline{A}$ and $\varnothing \vdash j^{\ell}_{\underline{B}} : S_\ell\ \underline{B} \to \underline{B}$ such that $\lfloor j^{\ell}_{\underline{A}} \rfloor \equiv \lambda x.x$ and $\lfloor j^{\ell}_{\underline{B}} \rfloor \equiv \lambda x.x$.
  Now,

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\varnothing \vdash \lambda y.\mathbf{proj}_1\ y : \underline{A} \times \underline{B} \to \underline{A}}}{\varnothing \vdash \mathbf{lift}^{\ell}(\lambda y.\mathbf{proj}_1\ y) : S_\ell\ (\underline{A} \times \underline{B}) \to S_\ell\ \underline{A}}\ (\textsc{Fmap})}{z : S_\ell\ (\underline{A} \times \underline{B}) \vdash (\mathbf{lift}^{\ell}(\lambda y.\mathbf{proj}_1\ y))\ z : S_\ell\ \underline{A}}\ (\textsc{App})}{z : S_\ell\ (\underline{A} \times \underline{B}) \vdash j^{\ell}_{\underline{A}}\ ((\mathbf{lift}^{\ell}(\lambda y.\mathbf{proj}_1\ y))\ z) : \underline{A}}\ (\text{IH})}{\dfrac{z : S_\ell\ (\underline{A} \times \underline{B}) \vdash (j^{\ell}_{\underline{A}}\ ((\mathbf{lift}^{\ell}(\lambda y.\mathbf{proj}_1\ y))\ z), j^{\ell}_{\underline{B}}\ ((\mathbf{lift}^{\ell}(\lambda y.\mathbf{proj}_2\ y))\ z)) : \underline{A} \times \underline{B}}{\varnothing \vdash \lambda z.(j^{\ell}_{\underline{A}}\ ((\mathbf{lift}^{\ell}(\lambda y.\mathbf{proj}_1\ y))\ z), j^{\ell}_{\underline{B}}\ ((\mathbf{lift}^{\ell}(\lambda y.\mathbf{proj}_2\ y))\ z)) : S_\ell\ (\underline{A} \times \underline{B}) \to \underline{A} \times \underline{B}}\ (\textsc{Lam})}\ (\textsc{Pair})
$$

  Note that we omit the derivation of $z : S_\ell\ (\underline{A} \times \underline{B}) \vdash j^{\ell}_{\underline{B}}\ ((\mathbf{lift}^{\ell}(\lambda y.\mathbf{proj}_2\ y))\ z) : \underline{B}$, which is similar to that of $z : S_\ell\ (\underline{A} \times \underline{B}) \vdash j^{\ell}_{\underline{A}}\ ((\mathbf{lift}^{\ell}(\lambda y.\mathbf{proj}_1\ y))\ z) : \underline{A}$.

  Therefore, $j^{\ell}_{\underline{A} \times \underline{B}} = \lambda z.(j^{\ell}_{\underline{A}}\ ((\mathbf{lift}^{\ell}(\lambda y.\mathbf{proj}_1\ y))\ z), j^{\ell}_{\underline{B}}\ ((\mathbf{lift}^{\ell}(\lambda y.\mathbf{proj}_2\ y))\ z))$ and $\lfloor j^{\ell}_{\underline{A} \times \underline{B}} \rfloor \equiv \lambda x.x$.

- Rule Prot-Fun. Have: $\ell \sqsubseteq A \to B$ where $\ell \sqsubseteq B$.
  By IH, $\exists\ \varnothing \vdash j^{\ell}_{\underline{B}} : S_\ell\ \underline{B} \to \underline{B}$ such that $\lfloor j^{\ell}_{\underline{B}} \rfloor \equiv \lambda x.x$.
  Now,

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{y : \underline{A} \vdash \lambda x.x\ y : (\underline{A} \to \underline{B}) \to \underline{B}}}{y : \underline{A} \vdash \mathbf{lift}^{\ell}(\lambda x.x\ y) : S_\ell\ (\underline{A} \to \underline{B}) \to S_\ell\ \underline{B}}\ (\textsc{Fmap})}{z : S_\ell\ (\underline{A} \to \underline{B}), y : \underline{A} \vdash (\mathbf{lift}^{\ell}(\lambda x.x\ y))\ z : S_\ell\ \underline{B}}\ (\textsc{App})}{z : S_\ell\ (\underline{A} \to \underline{B}), y : \underline{A} \vdash j^{\ell}_{\underline{B}}\ ((\mathbf{lift}^{\ell}(\lambda x.x\ y))\ z) : \underline{B}}\ (\text{IH})}{\varnothing \vdash \lambda z.\lambda y.j^{\ell}_{\underline{B}}\ ((\mathbf{lift}^{\ell}(\lambda x.x\ y))\ z) : S_\ell\ (\underline{A} \to \underline{B}) \to (\underline{A} \to \underline{B})}\ (\textsc{Lam})
$$

  Therefore, $j^{\ell}_{\underline{A} \to \underline{B}} = \lambda z.\lambda y.j^{\ell}_{\underline{B}}\ ((\mathbf{lift}^{\ell}(\lambda x.x\ y))\ z)$ and $\lfloor j^{\ell}_{\underline{A} \to \underline{B}} \rfloor \equiv \lambda x.x$.

- Rule Prot-Monad. Have: $\ell_1 \sqsubseteq T_{\ell_2}\ A$ where $\ell_1 \sqsubseteq \ell_2$.
  Now, $\varnothing \vdash \lambda x.\mathbf{join}^{\ell_1,\ell_2} x : S_{\ell_1}\ S_{\ell_2}\ \underline{B} \to S_{\ell_2}\ \underline{B}$.
  Therefore, $j^{\ell_1}_{S_{\ell_2}\ \underline{B}} = \lambda x.\mathbf{join}^{\ell_1,\ell_2} x$ and $\lfloor j^{\ell_1}_{S_{\ell_2}\ \underline{B}} \rfloor \equiv \lambda x.x$.

- Rule Prot-Already. Have: $\ell \sqsubseteq T_{\ell'}\ A$ where $\ell \sqsubseteq A$.
  By IH, $\exists\ \varnothing \vdash j^{\ell}_{\underline{A}} : S_\ell\ \underline{A} \to \underline{A}$ such that $\lfloor j^{\ell}_{\underline{A}} \rfloor \equiv \lambda x.x$.

Now,

$$\dfrac{\dfrac{\dfrac{}{\varnothing \vdash j^{\ell}_{\underline{A}} : S_{\ell}\, \underline{A} \to \underline{A}}\,(\text{IH})}{\varnothing \vdash \mathbf{lift}^{\ell'}\, j^{\ell}_{\underline{A}} : S_{\ell'}\, S_{\ell}\, \underline{A} \to S_{\ell'}\, \underline{A}}\,(\textsc{Fmap}) \quad \dfrac{\dfrac{}{x : S_{\ell}\, S_{\ell'}\, \underline{A} \vdash \mathbf{join}^{\ell,\ell'}\, x : S_{\ell \sqcup \ell'}\, \underline{A}}\,(\textsc{Join})}{x : S_{\ell}\, S_{\ell'}\, \underline{A} \vdash \mathbf{fork}^{\ell',\ell}(\mathbf{join}^{\ell,\ell'}\, x) : S_{\ell'}\, S_{\ell}\, \underline{A}}\,(\textsc{Fork})}{\dfrac{x : S_{\ell}\, S_{\ell'}\, \underline{A} \vdash (\mathbf{lift}^{\ell'}\, j^{\ell}_{\underline{A}})\, (\mathbf{fork}^{\ell',\ell}(\mathbf{join}^{\ell,\ell'}\, x)) : S_{\ell'}\, \underline{A}}{\varnothing \vdash \lambda x.(\mathbf{lift}^{\ell'}\, j^{\ell}_{\underline{A}})\, (\mathbf{fork}^{\ell',\ell}(\mathbf{join}^{\ell,\ell'}\, x)) : S_{\ell}\, S_{\ell'}\, \underline{A} \to S_{\ell'}\, \underline{A}}\,(\textsc{Lam})}\,(\textsc{App})$$

Notice the flip going from **join** to **fork** in the above derivation!

Therefore, $j^{\ell}_{S_{\ell'}\, \underline{A}} = \lambda x.(\mathbf{lift}^{\ell'}\, j^{\ell}_{\underline{A}})\, (\mathbf{fork}^{\ell',\ell}(\mathbf{join}^{\ell,\ell'}\, x))$ and $\lfloor j^{\ell}_{S_{\ell'}\, \underline{A}} \rfloor \equiv \lambda x.x$.

- Rule Prot-Minimum. Have: $\bot \sqsubseteq A$.
  Now, $\varnothing \vdash \lambda x.\mathbf{extr}\, x : S_{\bot}\, \underline{A} \to \underline{A}$.
  Therefore, $j^{\bot}_{\underline{A}} = \lambda x.\mathbf{extr}\, x$ and $\lfloor j^{\bot}_{\underline{A}} \rfloor \equiv \lambda x.x$.

- Rule Prot-Combine. Have: $\ell \sqsubseteq A$ where $\ell_1 \sqsubseteq A$ and $\ell_2 \sqsubseteq A$ and $\ell \sqsubseteq \ell_1 \sqcup \ell_2$.
  By IH, $\exists\ \varnothing \vdash j^{\ell_1}_{\underline{A}} : S_{\ell_1}\, \underline{A} \to \underline{A}$ and $\varnothing \vdash j^{\ell_2}_{\underline{A}} : S_{\ell_2}\, \underline{A} \to \underline{A}$ such that $\lfloor j^{\ell_1}_{\underline{A}} \rfloor \equiv \lambda x.x$ and $\lfloor j^{\ell_2}_{\underline{A}} \rfloor \equiv \lambda x.x$.
  Now,

$$\dfrac{\dfrac{\dfrac{}{\varnothing \vdash j^{\ell_2}_{\underline{A}} : S_{\ell_2}\, \underline{A} \to \underline{A}}\,(\text{IH})}{\varnothing \vdash \mathbf{lift}^{\ell_1} j^{\ell_2}_{\underline{A}} : S_{\ell_1}\, S_{\ell_2}\, \underline{A} \to S_{\ell_1}\, \underline{A}}\,(\textsc{Fmap}) \quad \dfrac{\dfrac{}{x : S_{\ell}\, \underline{A} \vdash \mathbf{up}^{\ell,\ell_1 \sqcup \ell_2} x : S_{\ell_1 \sqcup \ell_2}\, \underline{A}}\,(\textsc{Up})}{x : S_{\ell}\, \underline{A} \vdash \mathbf{fork}^{\ell_1,\ell_2}(\mathbf{up}^{\ell,\ell_1 \sqcup \ell_2} x) : S_{\ell_1}\, S_{\ell_2}\, \underline{A}}\,(\textsc{Fork})}{\dfrac{\dfrac{x : S_{\ell}\, \underline{A} \vdash (\mathbf{lift}^{\ell_1} j^{\ell_2}_{\underline{A}})\, (\mathbf{fork}^{\ell_1,\ell_2}(\mathbf{up}^{\ell,\ell_1 \sqcup \ell_2} x)) : S_{\ell_1}\, \underline{A}}{x : S_{\ell}\, \underline{A} \vdash j^{\ell_1}_{\underline{A}}\, ((\mathbf{lift}^{\ell_1} j^{\ell_2}_{\underline{A}})\, (\mathbf{fork}^{\ell_1,\ell_2}(\mathbf{up}^{\ell,\ell_1 \sqcup \ell_2} x))) : \underline{A}}\,(\text{IH})}{\varnothing \vdash \lambda x.j^{\ell_1}_{\underline{A}}\, ((\mathbf{lift}^{\ell_1} j^{\ell_2}_{\underline{A}})\, (\mathbf{fork}^{\ell_1,\ell_2}(\mathbf{up}^{\ell,\ell_1 \sqcup \ell_2} x))) : S_{\ell}\, \underline{A} \to \underline{A}}\,(\textsc{Lam})}\,(\textsc{App})$$

Therefore, $j^{\ell}_{\underline{A}} = \lambda x.j^{\ell_1}_{\underline{A}}\, ((\mathbf{lift}^{\ell_1} j^{\ell_2}_{\underline{A}})\, (\mathbf{fork}^{\ell_1,\ell_2}(\mathbf{up}^{\ell,\ell_1 \sqcup \ell_2} x)))$ and $\lfloor j^{\ell}_{\underline{A}} \rfloor \equiv \lambda x.x$.

□

Theorem A.10 (Theorem 2.10) If $\Gamma \vdash a : A$ in $\text{DCC}_e(\mathcal{L})$, then $\underline{\Gamma} \vdash \underline{a} : \underline{A}$ in GMCC($\mathcal{L}$). Further, if $\Gamma \vdash a_1 : A$ and $\Gamma \vdash a_2 : A$ such that $a_1 \simeq a_2$ in $\text{DCC}_e(\mathcal{L})$, then $\underline{a_1} \simeq \underline{a_2}$ in GMCC($\mathcal{L}$).

Proof. By induction on $\Gamma \vdash a : A$.

- λ-calculus. By IH.

- Rule DCC-Eta. Have: $\Gamma \vdash \mathbf{eta}^{\ell}\, a : T_{\ell}\, A$ where $\Gamma \vdash a : A$.
  By IH, $\underline{\Gamma} \vdash \underline{a} : \underline{A}$.
  Next, $\underline{\mathbf{eta}^{\ell}\, a} = \mathbf{up}^{\bot,\ell}(\mathbf{ret}\, \underline{a})$.
  Now, $\underline{\Gamma} \vdash \mathbf{ret}\, \underline{a} : S_{\bot}\, \underline{A}$.

So, $\underline{\Gamma} \vdash \mathbf{up}^{\bot,\ell}(\mathbf{ret}\ \underline{a}) : S_\ell\ \underline{A}$.

- Rule DCC-BIND. Have: $\Gamma \vdash \mathbf{bind}^\ell\ x = a\ \mathbf{in}\ b : B$ where $\Gamma \vdash a : T_\ell\ A$ and $\Gamma, x : A \vdash b : B$ and $\ell \sqsubseteq B$.
  By IH, $\underline{\Gamma} \vdash \underline{a} : S_\ell\ \underline{A}$ and $\underline{\Gamma}, x : \underline{A} \vdash \underline{b} : \underline{B}$.
  Now, since $\ell \sqsubseteq B$, by Lemma A.9, $\exists\ \varnothing \vdash j^\ell_{\underline{B}} : S_\ell\ \underline{B} \to \underline{B}$ such that $\lfloor j^\ell_{\underline{B}} \rfloor \equiv \lambda x.x$.
  Next, $\underline{\Gamma} \vdash \lambda x.\underline{b} : \underline{A} \to \underline{B}$.
  So, $\underline{\Gamma} \vdash \mathbf{lift}^\ell(\lambda x.\underline{b}) : S_\ell\ \underline{A} \to S_\ell\ \underline{B}$.
  As such, $\underline{\Gamma} \vdash (\mathbf{lift}^\ell(\lambda x.\underline{b}))\ \underline{a} : S_\ell\ \underline{B}$.
  Then, $\underline{\Gamma} \vdash j^\ell_{\underline{B}}\ ((\mathbf{lift}^\ell(\lambda x.\underline{b}))\ \underline{a}) : \underline{B}$.

Now, we can show that, if $\Gamma \vdash a : A$ in $\mathrm{DCC}_e(\mathcal{L})$, then $\lfloor \underline{a} \rfloor \equiv \lfloor a \rfloor$. The proof is by straightforward induction on the typing judgement. For the **eta**-case, note that $\lfloor \mathbf{up}^{\bot,\ell}(\mathbf{ret}\ \underline{a}) \rfloor = \lfloor \underline{a} \rfloor$. For the **bind**-case, note that $\lfloor j^\ell_{\underline{B}}\ ((\mathbf{lift}^\ell(\lambda x.\underline{b}))\ \underline{a}) \rfloor = \lfloor j^\ell_{\underline{B}} \rfloor\ ((\lambda x.\lfloor \underline{b} \rfloor)\ \lfloor \underline{a} \rfloor) \equiv (\lambda y.y)\ ((\lambda x.\lfloor \underline{b} \rfloor)\ \lfloor \underline{a} \rfloor) \equiv \lfloor \underline{b} \rfloor\{\lfloor \underline{a} \rfloor / x\}$.

So, for $\Gamma \vdash a_1 : A$ and $\Gamma \vdash a_2 : A$, if $a_1 \simeq a_2$ in $\mathrm{DCC}_e(\mathcal{L})$, then we have:

$$\lfloor \underline{a_1} \rfloor \equiv \lfloor a_1 \rfloor \equiv \lfloor a_2 \rfloor \equiv \lfloor \underline{a_2} \rfloor.$$

Hence, $\underline{a_1} \simeq \underline{a_2}$ in GMCC($\mathcal{L}$).

□

Theorem A.11 (Theorem 2.11) Let $\Gamma \vdash a : A$ be any derivation in GMCC($\mathcal{L}$). Then, $\overline{\underline{a}} \equiv a$.

Proof. By induction on $\Gamma \vdash a : A$. Note that $\overline{\underline{A}} = A$.

- λ-calculus. By IH.
- Rule MC-RETURN. Have: $\Gamma \vdash \mathbf{ret}\ a : S_1\ A$ where $\Gamma \vdash a : A$.
  By IH, $\underline{\overline{a}} \equiv a$. Now,
  $$\underline{\overline{\mathbf{ret}\ a}} = \underline{\mathbf{eta}^\bot\ \overline{a}} = \mathbf{up}^{\bot,\bot}(\mathbf{ret}\ \underline{\overline{a}}) \equiv \mathbf{ret}\ \underline{\overline{a}} \equiv \mathbf{ret}\ a.$$
- Rule MC-FMAP. Have: $\Gamma \vdash \mathbf{lift}^\ell f : S_\ell\ A \to S_\ell\ B$ where $\Gamma \vdash f : A \to B$.
  By IH, $\underline{\overline{f}} \equiv f$. Now,
  $$\begin{aligned}&\underline{\overline{\mathbf{lift}^\ell f}}\\ =&\underline{\lambda x : T_\ell\ \overline{A}.\mathbf{bind}^\ell\ y = x\ \mathbf{in}\ \mathbf{eta}^\ell\ (\overline{f}\ y)}\\ =&\lambda x.j^\ell_{S_\ell\ B}\ ((\mathbf{lift}^\ell(\lambda y.\underline{\mathbf{eta}^\ell\ (\overline{f}\ y)}))\ x)\\ =&\lambda x.\mathbf{join}^{\ell,\ell}((\mathbf{lift}^\ell(\lambda y.\mathbf{up}^{\bot,\ell}(\mathbf{ret}\ (\underline{\overline{f}}\ y))))\ x)\\ \equiv&\lambda x.x\ {}^\ell\!\!\gg\!=^\ell \lambda y.\mathbf{up}^{\bot,\ell}(\mathbf{ret}\, f\ y)\end{aligned}$$

$$\equiv \lambda x.\mathbf{up}^{\ell,\ell}(x \ {}^{\ell}\!\gg\!=^{\bot} \lambda y.\mathbf{ret}\,(f\ y)) \text{ [By Equation (2.6)]}$$
$$\equiv \lambda x.x \ {}^{\ell}\!\gg\!=^{\bot} \lambda y.\mathbf{ret}\,(f\ y)$$
$$\equiv \mathbf{lift}^{\ell} f \text{ [By Equation } (A.1)]$$

- Rule MC-Join. Have $\Gamma \vdash \mathbf{join}^{\ell_1,\ell_2}\, a : S_{\ell_1 \sqcup \ell_2}\ A$ where $\Gamma \vdash a : S_{\ell_1}\ S_{\ell_2}\ A$.
  By IH, $\underline{\overline{a}} \equiv a$. Now,

$$\underline{\overline{\mathbf{join}^{\ell_1,\ell_2}\, a}}$$
$$= \underline{\mathbf{bind}^{\ell_1}\ x = \overline{a}\ \mathbf{in}\ \mathbf{bind}^{\ell_2}\ y = x\ \mathbf{in}\ \mathbf{eta}^{\ell_1 \sqcup \ell_2}\ y}$$
$$= j^{\ell_1}_{S_{\ell_1 \sqcup \ell_2}\ A}\,((\mathbf{lift}^{\ell_1}(\lambda x.\underline{\mathbf{bind}^{\ell_2}\ y = x\ \mathbf{in}\ \mathbf{eta}^{\ell_1 \sqcup \ell_2}\ y}))\ \underline{\overline{a}})$$
$$\equiv \mathbf{join}^{\ell_1,\ell_1 \sqcup \ell_2}((\mathbf{lift}^{\ell_1}(\lambda x.j^{\ell_2}_{S_{\ell_1 \sqcup \ell_2}\ A}\,((\mathbf{lift}^{\ell_2}(\lambda y.\underline{\mathbf{eta}^{\ell_1 \sqcup \ell_2}\ y}))\ x)))\ a)$$
$$= \mathbf{join}^{\ell_1,\ell_1 \sqcup \ell_2}((\mathbf{lift}^{\ell_1}(\lambda x.\mathbf{join}^{\ell_2,\ell_1 \sqcup \ell_2}((\mathbf{lift}^{\ell_2}(\lambda y.\mathbf{up}^{\bot,\ell_1 \sqcup \ell_2}(\mathbf{ret}\ y)))\ x)))\ a)$$
$$= \mathbf{join}^{\ell_1,\ell_1 \sqcup \ell_2}((\mathbf{lift}^{\ell_1}(\lambda x.(x \ {}^{\ell_2}\!\gg\!=^{\ell_1 \sqcup \ell_2} (\lambda y.\mathbf{up}^{\bot,\ell_1 \sqcup \ell_2}(\mathbf{ret}\ y)))))\ a)$$
$$\equiv \mathbf{join}^{\ell_1,\ell_1 \sqcup \ell_2}((\mathbf{lift}^{\ell_1}(\lambda x.\mathbf{up}^{\ell_2,\ell_1 \sqcup \ell_2}(x \ {}^{\ell_2}\!\gg\!=^{\bot} \lambda y.\mathbf{ret}\ y)))\ a) \text{ [By Equation (2.6)]}$$
$$\equiv \mathbf{join}^{\ell_1,\ell_1 \sqcup \ell_2}((\mathbf{lift}^{\ell_1}(\lambda x.\mathbf{up}^{\ell_2,\ell_1 \sqcup \ell_2} x))\ a) \text{ [By Equation (2.8)]}$$
$$= a \ {}^{\ell_1}\!\gg\!=^{\ell_1 \sqcup \ell_2} \lambda x.\mathbf{up}^{\ell_2,\ell_1 \sqcup \ell_2} x$$
$$\equiv \mathbf{up}^{\ell_1 \sqcup \ell_2,\ell_1 \sqcup \ell_2}(a \ {}^{\ell_1}\!\gg\!=^{\ell_2} \lambda x.x) \text{ [By Equation (2.6)]}$$
$$\equiv a \ {}^{\ell_1}\!\gg\!=^{\ell_2} \lambda x.x \equiv \mathbf{join}^{\ell_1,\ell_2}\, a \text{ [By Equation } (A.3)]$$

- Rule MC-Up. Have: $\Gamma \vdash \mathbf{up}^{\ell_1,\ell_2}\, a : S_{\ell_2}\ A$ where $\Gamma \vdash a : S_{\ell_1}\ A$ and $\ell_1 \sqsubseteq \ell_2$.
  By IH, $\underline{\overline{a}} \equiv a$. Now,

$$\underline{\overline{\mathbf{up}^{\ell_1,\ell_2}\, a}}$$
$$= \underline{\mathbf{bind}^{\ell_1}\ x = \overline{a}\ \mathbf{in}\ \mathbf{eta}^{\ell_2}\ x}$$
$$= j^{\ell_1}_{S_{\ell_2}\ A}\,((\mathbf{lift}^{\ell_1}(\lambda x.\underline{\mathbf{eta}^{\ell_2}\ x}))\ \underline{\overline{a}})$$
$$= \mathbf{join}^{\ell_1,\ell_2}((\mathbf{lift}^{\ell_1}(\lambda x.\mathbf{up}^{\bot,\ell_2}(\mathbf{ret}\ x)))\ \underline{\overline{a}})$$
$$\equiv a \ {}^{\ell_1}\!\gg\!=^{\ell_2} (\lambda x.\mathbf{up}^{\bot,\ell_2}(\mathbf{ret}\ x))$$
$$\equiv \mathbf{up}^{\ell_1,\ell_2}(a \ {}^{\ell_1}\!\gg\!=^{\bot} \lambda x.\mathbf{ret}\ x) \text{ [By Equation (2.6)]}$$
$$\equiv \mathbf{up}^{\ell_1,\ell_2}\, a \text{ [By Equation (2.8)]}$$

- Rule MC-Extract. Have: $\Gamma \vdash \mathbf{extr}\ a : A$ where $\Gamma \vdash a : S_1\ A$.
  By IH, $\underline{\overline{a}} \equiv a$.
  Now, $\underline{\overline{\mathbf{extr}\ a}} = \underline{\mathbf{bind}^{\bot}\ x = \overline{a}\ \mathbf{in}\ x} = j^{\bot}_{A}\,((\mathbf{lift}^{\bot}(\lambda x.x))\ \underline{\overline{a}}) \equiv j^{\bot}_{A}\ a = \mathbf{extr}\ a$.
- Rule MC-Fork. Have: $\Gamma \vdash \mathbf{fork}^{\ell_1,\ell_2}\, a : S_{\ell_1}\ S_{\ell_2}\ A$ where $\Gamma \vdash a : S_{\ell_1 \sqcup \ell_2}\ A$.
  By IH, $\underline{\overline{a}} \equiv a$. Now,

$\underline{\overline{\mathbf{fork}^{\ell_1,\ell_2}\, a}}$

$= \underline{\mathbf{bind}^{\ell_1 \sqcup \ell_2}\ x = \overline{a}\ \mathbf{in}\ \mathbf{eta}^{\ell_1}\ \mathbf{eta}^{\ell_2}\ x}$

$= j^{\ell_1 \sqcup \ell_2}_{S_{\ell_1}\ S_{\ell_2}\ A}\,((\mathbf{lift}^{\ell_1 \sqcup \ell_2}(\lambda x.\underline{\mathbf{eta}^{\ell_1}\ \mathbf{eta}^{\ell_2}\ x}))\ \underline{\overline{a}})$

$\equiv j^{\ell_1 \sqcup \ell_2}_{S_{\ell_1}\ S_{\ell_2}\ A}\,((\mathbf{lift}^{\ell_1 \sqcup \ell_2}(\lambda x.\mathbf{up}^{\bot,\ell_1}\mathbf{ret}\ \mathbf{up}^{\bot,\ell_2}\mathbf{ret}\ x))\ a)$

$\triangleq j^{\ell_1 \sqcup \ell_2}_{S_{\ell_1}\ S_{\ell_2}\ A}\ t_0$ [Say, $t_0 \triangleq (\mathbf{lift}^{\ell_1 \sqcup \ell_2}(\lambda x.\mathbf{up}^{\bot,\ell_1}\mathbf{ret}\ \mathbf{up}^{\bot,\ell_2}\mathbf{ret}\ x))\ a$]

$= j^{\ell_1}_{S_{\ell_1}\ S_{\ell_2}\ A}\,((\mathbf{lift}^{\ell_1}\, j^{\ell_2}_{S_{\ell_1}\ S_{\ell_2}\ A})\ (\mathbf{fork}^{\ell_1,\ell_2}\, t_0))$

$= j^{\ell_1}_{S_{\ell_1}\ S_{\ell_2}\ A}\,((\mathbf{lift}^{\ell_1}(\lambda z.(\mathbf{lift}^{\ell_1}\, j^{\ell_2}_{S_{\ell_2}\ A})\ (\mathbf{fork}^{\ell_1,\ell_2}(\mathbf{join}^{\ell_2,\ell_1}\, z))))\ (\mathbf{fork}^{\ell_1,\ell_2}\, t_0))$

$\equiv j^{\ell_1}_{S_{\ell_1}\ S_{\ell_2}\ A}\,((\lambda z.(\mathbf{lift}^{\ell_1}(\mathbf{lift}^{\ell_1}\, j^{\ell_2}_{S_{\ell_2}\ A}))\ (\mathbf{lift}^{\ell_1}(\lambda y.\mathbf{fork}^{\ell_1,\ell_2}(\mathbf{join}^{\ell_2,\ell_1}\, y)))\ z)\ (\mathbf{fork}^{\ell_1,\ell_2}\, t_0))$

[By Eqn. (2.11)]

$\equiv j^{\ell_1}_{S_{\ell_1}\ S_{\ell_2}\ A}\,((\mathbf{lift}^{\ell_1}(\mathbf{lift}^{\ell_1}\, j^{\ell_2}_{S_{\ell_2}\ A}))\ ((\mathbf{lift}^{\ell_1}(\lambda y.\mathbf{fork}^{\ell_1,\ell_2}(\mathbf{join}^{\ell_2,\ell_1}\, y)))\ (\mathbf{fork}^{\ell_1,\ell_2}\, t_0)))$

$= j^{\ell_1}_{S_{\ell_1}\ S_{\ell_2}\ A}\,((\mathbf{lift}^{\ell_1}(\mathbf{lift}^{\ell_1}\, j^{\ell_2}_{S_{\ell_2}\ A}))\ ((\lambda y.\mathbf{fork}^{\ell_1,\ell_2}(\mathbf{join}^{\ell_2,\ell_1}\, y))\ {}^{\ell_2}\!\lll^{\ell_1}\ t_0))$

$\triangleq j^{\ell_1}_{S_{\ell_1}\ S_{\ell_2}\ A}\,((\mathbf{lift}^{\ell_1}(\mathbf{lift}^{\ell_1}\, j^{\ell_2}_{S_{\ell_2}\ A}))\ ((\lambda y.\mathbf{fork}^{\ell_1,\ell_2}(\mathbf{join}^{\ell_2,\ell_1}\, y))\ {}^{\ell_2}\!\lll^{\ell_1}\ ((\mathbf{lift}^{\ell_1 \sqcup \ell_2} f)\ a)))$

[Say, $f \triangleq \lambda x.\mathbf{up}^{\bot,\ell_1}\mathbf{ret}\ \mathbf{up}^{\bot,\ell_2}\mathbf{ret}\ x$]

$\equiv j^{\ell_1}_{S_{\ell_1}\ S_{\ell_2}\ A}\,((\mathbf{lift}^{\ell_1}(\mathbf{lift}^{\ell_1}\, j^{\ell_2}_{S_{\ell_2}\ A}))\ ((\lambda y.\mathbf{fork}^{\ell_1,\ell_2}(\mathbf{join}^{\ell_2,\ell_1}\, y))\ {}^{\ell_2}\!\lll^{\ell_1}$

$((\lambda x.f\ (\mathbf{extr}\ x))\ {}^{\bot}\!\lll^{\ell_1 \sqcup \ell_2}\ a)))$ [By Eqn. $(A.2)$]

$\equiv j^{\ell_1}_{S_{\ell_1}\ S_{\ell_2}\ A}\,((\mathbf{lift}^{\ell_1}(\mathbf{lift}^{\ell_1}\, j^{\ell_2}_{S_{\ell_2}\ A}))\ ((\lambda z.(\lambda y.\mathbf{fork}^{\ell_1,\ell_2}(\mathbf{join}^{\ell_2,\ell_1}\, y))$

$((\lambda x.f\ (\mathbf{extr}\ x))\ {}^{\bot}\!\lll^{\ell_2}\ z))\ {}^{\ell_2}\!\lll^{\ell_1}\ a))$ [By Eqn. (2.18)]

$\equiv j^{\ell_1}_{S_{\ell_1}\ S_{\ell_2}\ A}\,((\mathbf{lift}^{\ell_1}(\mathbf{lift}^{\ell_1}\, j^{\ell_2}_{S_{\ell_2}\ A}))\ ((\lambda z.(\lambda y.\mathbf{fork}^{\ell_1,\ell_2}(\mathbf{join}^{\ell_2,\ell_1}\, y))$

$(\mathbf{fork}^{\ell_2,\ell_1}(\mathbf{up}^{\ell_2,\ell_2 \sqcup \ell_1}\mathbf{fork}^{\ell_2,\ell_2}\, z)))\ {}^{\ell_2}\!\lll^{\ell_1}\ a))$

[$\because (\lambda x.f\ (\mathbf{extr}\ x))\ {}^{\bot}\!\lll^{\ell_2}\ z \equiv \mathbf{fork}^{\ell_2,\ell_1}(\mathbf{up}^{\ell_2,\ell_2 \sqcup \ell_1}\mathbf{fork}^{\ell_2,\ell_2}\, z)$ (See below)]

$\equiv j^{\ell_1}_{S_{\ell_1}\ S_{\ell_2}\ A}\,((\mathbf{lift}^{\ell_1}(\mathbf{lift}^{\ell_1}\, j^{\ell_2}_{S_{\ell_2}\ A}))\ ((\lambda z.\mathbf{fork}^{\ell_1,\ell_2}\mathbf{join}^{\ell_2,\ell_1}\mathbf{fork}^{\ell_2,\ell_1}\mathbf{up}^{\ell_2,\ell_2 \sqcup \ell_1}\mathbf{fork}^{\ell_2,\ell_2}\, z)\ {}^{\ell_2}\!\lll^{\ell_1}\ a))$

$\equiv j^{\ell_1}_{S_{\ell_1}\ S_{\ell_2}\ A}\,((\mathbf{lift}^{\ell_1}(\mathbf{lift}^{\ell_1}\, j^{\ell_2}_{S_{\ell_2}\ A}))\ ((\lambda z.\mathbf{fork}^{\ell_1,\ell_2}\mathbf{up}^{\ell_2,\ell_2 \sqcup \ell_1}\mathbf{fork}^{\ell_2,\ell_2}\, z)\ {}^{\ell_2}\!\lll^{\ell_1}\ a))$

$= j^{\ell_1}_{S_{\ell_1}\ S_{\ell_2}\ A}\,((\mathbf{lift}^{\ell_1}(\mathbf{lift}^{\ell_1}\, j^{\ell_2}_{S_{\ell_2}\ A}))\ ((\mathbf{lift}^{\ell_1}(\lambda z.\mathbf{fork}^{\ell_1,\ell_2}\mathbf{up}^{\ell_2,\ell_2 \sqcup \ell_1}\mathbf{fork}^{\ell_2,\ell_2}\, z))\ (\mathbf{fork}^{\ell_1,\ell_2}\, a)))$

$\equiv j^{\ell_1}_{S_{\ell_1}\ S_{\ell_2}\ A}\,((\mathbf{lift}^{\ell_1}(\lambda z.(\mathbf{lift}^{\ell_1}\, j^{\ell_2}_{S_{\ell_2}\ A})\ (\mathbf{fork}^{\ell_1,\ell_2}\mathbf{up}^{\ell_2,\ell_2 \sqcup \ell_1}\mathbf{fork}^{\ell_2,\ell_2}\, z)))\ (\mathbf{fork}^{\ell_1,\ell_2}\, a))$

[By Eqn. (2.11)]

$= j^{\ell_1}_{S_{\ell_1}\ S_{\ell_2}\ A}\,((\mathbf{lift}^{\ell_1}(\lambda z.(\mathbf{lift}^{\ell_1}(\lambda x.\mathbf{join}^{\ell_2,\ell_2}\, x))\ (\mathbf{fork}^{\ell_1,\ell_2}\mathbf{up}^{\ell_2,\ell_2 \sqcup \ell_1}\mathbf{fork}^{\ell_2,\ell_2}\, z)))\ (\mathbf{fork}^{\ell_1,\ell_2}\, a))$

$= j^{\ell_1}_{S_{\ell_1}\ S_{\ell_2}\ A}\,((\mathbf{lift}^{\ell_1}(\lambda z.((\lambda x.\mathbf{join}^{\ell_2,\ell_2}\, x)\ {}^{\ell_2}\!\lll^{\ell_1}\ \mathbf{up}^{\ell_2,\ell_2 \sqcup \ell_1}\mathbf{fork}^{\ell_2,\ell_2}\, z)))\ (\mathbf{fork}^{\ell_1,\ell_2}\, a))$

$\equiv j^{\ell_1}_{S_{\ell_1}\ S_{\ell_2}\ A}\,((\mathbf{lift}^{\ell_1}(\lambda z.(\mathbf{up}^{\bot,\ell_1}(\lambda x.\mathbf{join}^{\ell_2,\ell_2}\, x\ \ {}^{\ell_2}\!\lll^{\bot}\ \mathbf{fork}^{\ell_2,\ell_2}\, z))))\ (\mathbf{fork}^{\ell_1,\ell_2}\, a))$

[By Eqn. (2.14)]

$\equiv j^{\ell_1}_{S_{\ell_1}\ S_{\ell_2}\ A}\,((\mathbf{lift}^{\ell_1}(\lambda z.(\mathbf{up}^{\bot,\ell_1}\mathbf{ret}\ \mathbf{extr}\ (\lambda x.\mathbf{join}^{\ell_2,\ell_2}\, x\ \ {}^{\ell_2}\!\lll^{\bot}\ \mathbf{fork}^{\ell_2,\ell_2}\, z))))\ (\mathbf{fork}^{\ell_1,\ell_2}\, a))$

$$\equiv j^{\ell_1}_{S_{\ell_1}\ S_{\ell_2}\ A}\,((\mathbf{lift}^{\ell_1}(\lambda z.(\mathbf{up}^{\bot,\ell_1}\mathbf{ret}\,(\mathbf{join}^{\ell_2,\ell_2}\mathbf{fork}^{\ell_2,\ell_2}z))))\,(\mathbf{fork}^{\ell_1,\ell_2}a))\ \text{[By Eqn. (2.16)]}$$
$$\equiv j^{\ell_1}_{S_{\ell_1}\ S_{\ell_2}\ A}\,((\mathbf{lift}^{\ell_1}(\lambda z.(\mathbf{up}^{\bot,\ell_1}\mathbf{ret}\,z)))\,(\mathbf{fork}^{\ell_1,\ell_2}a))$$
$$=\mathbf{join}^{\ell_1,\ell_1}((\mathbf{lift}^{\ell_1}(\lambda z.\mathbf{up}^{\bot,\ell_1}\mathbf{ret}\,z))\,(\mathbf{fork}^{\ell_1,\ell_2}a))$$
$$=\mathbf{fork}^{\ell_1,\ell_2}a\ {}^{\ell_1}\!\gg\!=^{\ell_1}\ \lambda z.\mathbf{up}^{\bot,\ell_1}\mathbf{ret}\,z$$
$$\equiv \mathbf{up}^{\ell_1,\ell_1}(\mathbf{fork}^{\ell_1,\ell_2}a\ {}^{\ell_1}\!\gg\!=^{\bot}\ \lambda z.\mathbf{ret}\,z)\ \text{[By Eqn. (2.6)]}$$
$$\equiv \mathbf{fork}^{\ell_1,\ell_2}a\ {}^{\ell_1}\!\gg\!=^{\bot}\ \lambda z.\mathbf{ret}\,z \equiv \mathbf{fork}^{\ell_1,\ell_2}a\ \text{[By Eqn. (2.8)]}$$

Note that:

$$\lambda x.f\ (\mathbf{extr}\ x)\ {}^{\bot}\!\lll^{\ell_2}\ z$$
$$\equiv\lambda x.\mathbf{up}^{\bot,\ell_1}\mathbf{ret}\,\mathbf{up}^{\bot,\ell_2}x\ {}^{\bot}\!\lll^{\ell_2}\ z$$
$$\equiv\lambda x.\mathbf{up}^{\bot,\ell_1}\mathbf{ret}\,x\ {}^{\ell_2}\!\lll^{\ell_2}\ \mathbf{up}^{\ell_2,\ell_2}z\ \text{[By Eqn. (2.15)]}$$
$$\equiv(\mathbf{lift}^{\ell_2}(\lambda x.\mathbf{up}^{\bot,\ell_1}\mathbf{ret}\,x))\,(\mathbf{fork}^{\ell_2,\ell_2}z)$$
$$\equiv(\lambda y.\mathbf{up}^{\bot,\ell_1}y)\ {}^{\bot}\!\lll^{\ell_2}\ \mathbf{fork}^{\ell_2,\ell_2}z\ \text{[By Eqn. (}A\text{.2)]}$$
$$\equiv\lambda y.y\ {}^{\ell_1}\!\lll^{\ell_2}\ \mathbf{up}^{\ell_2,\ell_2\sqcup\ell_1}(\mathbf{fork}^{\ell_2,\ell_2}z)\ \text{[By Eqn. (2.15)]}$$
$$\equiv\mathbf{fork}^{\ell_2,\ell_1}(\mathbf{up}^{\ell_2,\ell_2\sqcup\ell_1}\mathbf{fork}^{\ell_2,\ell_2}z)\ \text{[By Eqn. (2.10)]}$$

□

A.7 Proofs of Lemmas/Theorems Stated in Section 2.8

Lemma A.12 If $\ell \sqsubseteq A$ in DCC$_e$, then $\exists\ k^{\ell}_{[\![A]\!]} \in \mathrm{Hom}_{\mathbb{C}}(\mathbf{S}_\ell[\![A]\!], [\![A]\!])$ such that $k^{\ell}_{[\![A]\!]} \circ \mathbf{S}^{\bot\sqsubseteq\ell}_{[\![A]\!]} \circ \eta_{[\![A]\!]} = \mathrm{id}_{[\![A]\!]}$.

Proof. By induction on $\ell \sqsubseteq A$.

- Rule Prot-Prod. Have: $\ell \sqsubseteq A \times B$ where $\ell \sqsubseteq A$ and $\ell \sqsubseteq B$.
  By IH, $\exists k^{\ell}_{[\![A]\!]} \in \mathrm{Hom}_{\mathbb{C}}(\mathbf{S}_\ell[\![A]\!], [\![A]\!])$ and $k^{\ell}_{[\![B]\!]} \in \mathrm{Hom}_{\mathbb{C}}(\mathbf{S}_\ell[\![B]\!], [\![B]\!])$ such that
  $k^{\ell}_{[\![A]\!]} \circ \mathbf{S}^{\bot\sqsubseteq\ell}_{[\![A]\!]} \circ \eta_{[\![A]\!]} = \mathrm{id}_{[\![A]\!]}$ and $k^{\ell}_{[\![B]\!]} \circ \mathbf{S}^{\bot\sqsubseteq\ell}_{[\![B]\!]} \circ \eta_{[\![B]\!]} = \mathrm{id}_{[\![B]\!]}$.
  Define: $k^{\ell}_{[\![A]\!]\times[\![B]\!]} = \mathbf{S}_\ell([\![A]\!] \times [\![B]\!]) \xrightarrow{\langle \mathbf{S}_\ell\pi_1, \mathbf{S}_\ell\pi_2\rangle} \mathbf{S}_\ell[\![A]\!] \times \mathbf{S}_\ell[\![B]\!] \xrightarrow{k^{\ell}_{[\![A]\!]}\times k^{\ell}_{[\![B]\!]}} [\![A]\!] \times [\![B]\!]$.
  Need to show: $k^{\ell}_{[\![A]\!]\times[\![B]\!]} \circ \mathbf{S}^{\bot\sqsubseteq\ell}_{[\![A]\!]\times[\![B]\!]} \circ \eta_{[\![A]\!]\times[\![B]\!]} = \mathrm{id}_{[\![A]\!]\times[\![B]\!]}$.

  This equation follows from the commutative diagram in Figure A.12. The diagram in Figure A.12 commutes: the triangle commutes by naturality; the square too commutes by naturality; the circular segment commutes by IH.

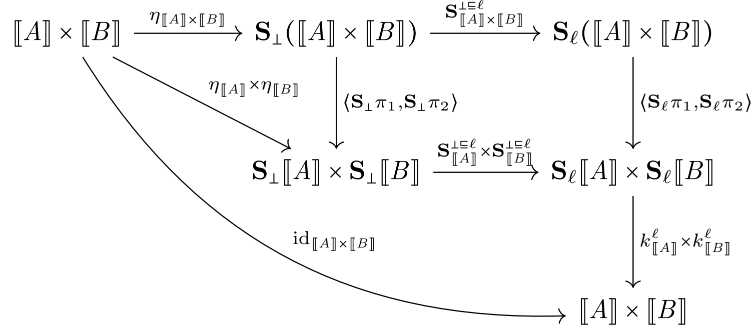


Figure A.12: Commutative diagram

- Rule Prot-Fun. Have: $\ell \sqsubseteq A \to B$ where $\ell \sqsubseteq B$.
  By IH, $\exists k^\ell_{[\![B]\!]} \in \mathrm{Hom}_{\mathbb{C}}(\mathbf{S}_\ell[\![B]\!], [\![B]\!])$ such that $k^\ell_{[\![B]\!]} \circ \mathbf{S}^{\bot\sqsubseteq\ell}_{[\![B]\!]} \circ \eta_{[\![B]\!]} = \mathrm{id}_{[\![B]\!]}$.
  Define: $k^\ell_{[\![B]\!]^{[\![A]\!]}} = \Lambda\Big(\mathbf{S}_\ell[\![B]\!]^{[\![A]\!]} \times [\![A]\!] \xrightarrow{\langle\pi_2,\pi_1\rangle} [\![A]\!] \times \mathbf{S}_\ell[\![B]\!]^{[\![A]\!]} \xrightarrow{t^{\mathbf{S}_\ell}_{[\![A]\!],[\![B]\!]^{[\![A]\!]}}} \mathbf{S}_\ell([\![A]\!] \times [\![B]\!]^{[\![A]\!]}) \xrightarrow{\mathbf{S}_\ell\langle\pi_2,\pi_1\rangle} \mathbf{S}_\ell([\![B]\!]^{[\![A]\!]} \times [\![A]\!]) \xrightarrow{\mathbf{S}_\ell \mathrm{app}} \mathbf{S}_\ell[\![B]\!] \xrightarrow{k^\ell_{[\![B]\!]}} [\![B]\!]\Big)$.

  Need to show: $k^\ell_{[\![B]\!]^{[\![A]\!]}} \circ \mathbf{S}^{\bot\sqsubseteq\ell}_{[\![B]\!]^{[\![A]\!]}} \circ \eta_{[\![B]\!]^{[\![A]\!]}} = \mathrm{id}_{[\![B]\!]^{[\![A]\!]}}$.

  Now,

$$
\begin{aligned}
& k^\ell_{[\![B]\!]^{[\![A]\!]}} \circ \mathbf{S}^{\bot\sqsubseteq\ell}_{[\![B]\!]^{[\![A]\!]}} \circ \eta_{[\![B]\!]^{[\![A]\!]}} \\
=& \Lambda\Big(k^\ell_{[\![B]\!]} \circ \mathbf{S}_\ell \mathrm{app} \circ \mathbf{S}_\ell\langle\pi_2,\pi_1\rangle \circ t^{\mathbf{S}_\ell}_{[\![A]\!],[\![B]\!]^{[\![A]\!]}} \circ \langle\pi_2,\pi_1\rangle\Big) \circ \mathbf{S}^{\bot\sqsubseteq\ell}_{[\![B]\!]^{[\![A]\!]}} \circ \eta_{[\![B]\!]^{[\![A]\!]}} \\
=& \Lambda\Big(k^\ell_{[\![B]\!]} \circ \mathbf{S}_\ell \mathrm{app} \circ \mathbf{S}_\ell\langle\pi_2,\pi_1\rangle \circ t^{\mathbf{S}_\ell}_{[\![A]\!],[\![B]\!]^{[\![A]\!]}} \circ \langle\pi_2,\pi_1\rangle \circ (\mathbf{S}^{\bot\sqsubseteq\ell}_{[\![B]\!]^{[\![A]\!]}} \circ \eta_{[\![B]\!]^{[\![A]\!]}}) \times \mathrm{id}_{[\![A]\!]}\Big) \\
=& \Lambda\Big(k^\ell_{[\![B]\!]} \circ \mathbf{S}^{\bot\sqsubseteq\ell}_{[\![B]\!]} \circ \eta_{[\![B]\!]} \circ \mathrm{app} \circ \langle\pi_2,\pi_1\rangle \circ \langle\pi_2,\pi_1\rangle\Big) \text{ [By Figure A.13]} \\
=& \Lambda(\mathrm{app}) \text{ [By IH]} \\
=& \mathrm{id}_{[\![B]\!]^{[\![A]\!]}}
\end{aligned}
$$

  Note that in Figure A.13, $\overline{\eta}_X \coloneqq \mathbf{S}^{\bot\sqsubseteq\ell}_X \circ \eta_X$. The diagram in this figure commutes by naturality.

- Rule Prot-Monad. Have $\ell_1 \sqsubseteq T_{\ell_2}\ A$ where $\ell_1 \sqsubseteq \ell_2$.
  Define: $k^{\ell_1}_{\mathbf{S}_{\ell_2}[\![A]\!]} = \mu^{\ell_1,\ell_2}_{[\![A]\!]}$.
  Need to show: $k^{\ell_1}_{\mathbf{S}_{\ell_2}[\![A]\!]} \circ \mathbf{S}^{\bot\sqsubseteq\ell_1}_{\mathbf{S}_{\ell_2}[\![A]\!]} \circ \eta_{\mathbf{S}_{\ell_2}[\![A]\!]} = \mathrm{id}_{\mathbf{S}_{\ell_2}[\![A]\!]}$.
  Now,

$$
\begin{aligned}
& k^{\ell_1}_{\mathbf{S}_{\ell_2}[\![A]\!]} \circ \mathbf{S}^{\bot\sqsubseteq\ell_1}_{\mathbf{S}_{\ell_2}[\![A]\!]} \circ \eta_{\mathbf{S}_{\ell_2}[\![A]\!]} \\
=& \mu^{\ell_1,\ell_2}_{[\![A]\!]} \circ \mathbf{S}^{\bot\sqsubseteq\ell_1}_{\mathbf{S}_{\ell_2}[\![A]\!]} \circ \eta_{\mathbf{S}_{\ell_2}[\![A]\!]}
\end{aligned}
$$

$$
\begin{array}{ccccccccc}
\llbracket B\rrbracket^{\llbracket A\rrbracket}\times\llbracket A\rrbracket & \xrightarrow{\langle\pi_2,\pi_1\rangle} & \llbracket A\rrbracket\times\llbracket B\rrbracket^{\llbracket A\rrbracket} & \xrightarrow{\mathrm{id}} & \llbracket A\rrbracket\times\llbracket B\rrbracket^{\llbracket A\rrbracket} & \xrightarrow{\langle\pi_2,\pi_1\rangle} & \llbracket B\rrbracket^{\llbracket A\rrbracket}\times\llbracket A\rrbracket & \xrightarrow{\mathrm{app}} & \llbracket B\rrbracket \\
\downarrow{\scriptstyle \overline{\eta}_{\llbracket B\rrbracket^{\llbracket A\rrbracket}}\times\mathrm{id}} & & \downarrow{\scriptstyle \mathrm{id}_{\llbracket A\rrbracket}\times\overline{\eta}_{\llbracket B\rrbracket^{\llbracket A\rrbracket}}} & & \downarrow{\scriptstyle \overline{\eta}_{\llbracket A\rrbracket\times\llbracket B\rrbracket^{\llbracket A\rrbracket}}} & & \downarrow{\scriptstyle \overline{\eta}_{\llbracket B\rrbracket^{\llbracket A\rrbracket}\times\llbracket A\rrbracket}} & & \downarrow{\scriptstyle \overline{\eta}_{\llbracket B\rrbracket}} \\
\mathbf{S}_\ell\llbracket B\rrbracket^{\llbracket A\rrbracket}\times\llbracket A\rrbracket & \xrightarrow{\langle\pi_2,\pi_1\rangle} & \llbracket A\rrbracket\times\mathbf{S}_\ell\llbracket B\rrbracket^{\llbracket A\rrbracket} & \xrightarrow{t^{\mathbf{S}_\ell}_{\llbracket A\rrbracket,\llbracket B\rrbracket^{\llbracket A\rrbracket}}} & \mathbf{S}_\ell(\llbracket A\rrbracket\times\llbracket B\rrbracket^{\llbracket A\rrbracket}) & \xrightarrow{\mathbf{S}_\ell\langle\pi_2,\pi_1\rangle} & \mathbf{S}_\ell(\llbracket B\rrbracket^{\llbracket A\rrbracket}\times\llbracket A\rrbracket) & \xrightarrow{\mathbf{S}_\ell\mathrm{app}} & \mathbf{S}_\ell\llbracket B\rrbracket
\end{array}
$$

FIGURE A.13: Commutative diagram

$$
\begin{array}{ccccccc}
 & & \mathbf{S}_{\ell'}\llbracket A\rrbracket & & & & \\
 & \swarrow{\scriptstyle \eta_{\mathbf{S}_{\ell'}\llbracket A\rrbracket}} & \downarrow{\scriptstyle \mathrm{id}_{\mathbf{S}_{\ell'}\llbracket A\rrbracket}} & & \overset{\mathrm{id}_{\mathbf{S}_{\ell'}\llbracket A\rrbracket}}{\frown} & & \\
\mathbf{S}_\perp\mathbf{S}_{\ell'}\llbracket A\rrbracket & \xrightarrow{\mu^{\perp,\ell'}_{\llbracket A\rrbracket}} & \mathbf{S}_{\ell'}\llbracket A\rrbracket & \xrightarrow{\delta^{\ell',\perp}_{\llbracket A\rrbracket}} & \mathbf{S}_{\ell'}\mathbf{S}_\perp\llbracket A\rrbracket & \xrightarrow{\mathbf{S}_{\ell'}\epsilon_{\llbracket A\rrbracket}} & \mathbf{S}_{\ell'}\llbracket A\rrbracket \\
\downarrow{\scriptstyle \mathbf{S}^{\perp\sqsubseteq\ell}_{\mathbf{S}_{\ell'}\llbracket A\rrbracket}} & & \downarrow{\scriptstyle \mathbf{S}^{\ell'\sqsubseteq\ell\sqcup\ell'}_{\llbracket A\rrbracket}} & & \downarrow{\scriptstyle \mathbf{S}_{\ell'}\mathbf{S}^{\perp\sqsubseteq\ell}_{\llbracket A\rrbracket}} & \nearrow{\scriptstyle \mathbf{S}_{\ell'}k^{\ell}_{\llbracket A\rrbracket}} & \\
\mathbf{S}_\ell\mathbf{S}_{\ell'}\llbracket A\rrbracket & \xrightarrow{\mu^{\ell,\ell'}_{\llbracket A\rrbracket}} & \mathbf{S}_{\ell\sqcup\ell'}\llbracket A\rrbracket & \xrightarrow{\delta^{\ell',\ell}_{\llbracket A\rrbracket}} & \mathbf{S}_{\ell'}\mathbf{S}_\ell\llbracket A\rrbracket & &
\end{array}
$$

FIGURE A.14: Commutative diagram

$$
\begin{aligned}
&= \mu^{\ell_1,\ell_2}_{\llbracket A\rrbracket} \circ \mathbf{S}^{\perp\sqsubseteq\ell_1}_{\mathbf{S}_{\ell_2}\llbracket A\rrbracket} \circ \delta^{\perp,\ell_2}_{\llbracket A\rrbracket} \circ \mu^{\perp,\ell_2}_{\llbracket A\rrbracket} \circ \eta_{\mathbf{S}_{\ell_2}\llbracket A\rrbracket} \\
&= \mu^{\ell_1,\ell_2}_{\llbracket A\rrbracket} \circ \mathbf{S}^{\perp\sqsubseteq\ell_1}_{\mathbf{S}_{\ell_2}\llbracket A\rrbracket} \circ \delta^{\perp,\ell_2}_{\llbracket A\rrbracket} \circ \mathrm{id}_{\mathbf{S}_{\ell_2}\llbracket A\rrbracket} \text{ [By lax monoidality]} \\
&= \mu^{\ell_1,\ell_2}_{\llbracket A\rrbracket} \circ \delta^{\ell_1,\ell_2}_{\llbracket A\rrbracket} \circ \mathbf{S}^{\ell_2\sqsubseteq\ell_1\sqcup\ell_2}_{\llbracket A\rrbracket} \text{ [By naturality]} \\
&= \mathbf{S}^{\ell_2\sqsubseteq\ell_1\sqcup\ell_2}_{\llbracket A\rrbracket} = \mathbf{S}^{\ell_2\sqsubseteq\ell_2}_{\llbracket A\rrbracket}[\because \ell_1 \sqsubseteq \ell_2] = \mathrm{id}_{\mathbf{S}_{\ell_2}\llbracket A\rrbracket}
\end{aligned}
$$

- Rule PROT-ALREADY. Have: $\ell \sqsubseteq T_{\ell'}\ A$ where $\ell \sqsubseteq A$.
  By IH, $\exists k^{\ell}_{\llbracket A\rrbracket} \in \mathrm{Hom}_{\mathbb{C}}(\mathbf{S}_\ell\llbracket A\rrbracket, \llbracket A\rrbracket)$ such that $k^{\ell}_{\llbracket A\rrbracket} \circ \mathbf{S}^{\perp\sqsubseteq\ell}_{\llbracket A\rrbracket} \circ \eta_{\llbracket A\rrbracket} = \mathrm{id}_{\llbracket A\rrbracket}$.
  Define: $k^{\ell}_{\mathbf{S}_{\ell'}\llbracket A\rrbracket} = \mathbf{S}_\ell\mathbf{S}_{\ell'}\llbracket A\rrbracket \xrightarrow{\mu^{\ell,\ell'}_{\llbracket A\rrbracket}} \mathbf{S}_{\ell\sqcup\ell'}\llbracket A\rrbracket = \mathbf{S}_{\ell'\sqcup\ell}\llbracket A\rrbracket \xrightarrow{\delta^{\ell',\ell}_{\llbracket A\rrbracket}} \mathbf{S}_{\ell'}\mathbf{S}_\ell\llbracket A\rrbracket \xrightarrow{\mathbf{S}_{\ell'}k^{\ell}_{\llbracket A\rrbracket}} \mathbf{S}_{\ell'}\llbracket A\rrbracket$.
  Need to show: $k^{\ell}_{\mathbf{S}_{\ell'}\llbracket A\rrbracket} \circ \mathbf{S}^{\perp\sqsubseteq\ell}_{\mathbf{S}_{\ell'}\llbracket A\rrbracket} \circ \eta_{\mathbf{S}_{\ell'}\llbracket A\rrbracket} = \mathrm{id}_{\mathbf{S}_{\ell'}\llbracket A\rrbracket}$.

  This equation follows from the commutative diagram in Figure A.14. The diagram in Figure A.14 commutes: the left triangle in the top row commutes because $\mathbf{S}$ is a lax monoidal functor; the circular segment in the top row commutes because $\mathbf{S}$ is an oplax monoidal functor; the left square in the bottom row commutes because μ is natural in its first component; the right square in the bottom row commutes because δ is natural in its second component; the triangle to the right in the bottom row commutes by IH.

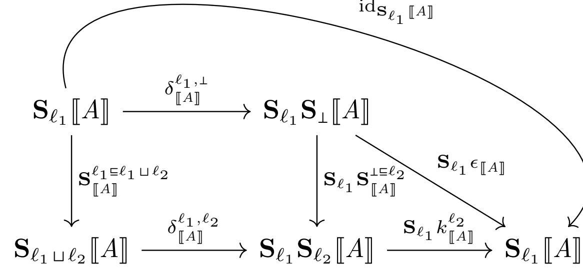


FIGURE A.15: Commutative diagram

- Rule PROT-MINIMUM. Have: $\perp \sqsubseteq A$.
  Define: $k^{\perp}_{\llbracket A \rrbracket} = \epsilon_{\llbracket A \rrbracket}$. Note, $\epsilon_{\llbracket A \rrbracket} \circ \mathbf{S}^{\perp \sqsubseteq \perp}_{\llbracket A \rrbracket} \circ \eta_{\llbracket A \rrbracket} = \mathrm{id}_{\llbracket A \rrbracket}$.
- Rule PROT-COMBINE. Have: $\ell \sqsubseteq A$ where $\ell_1 \sqsubseteq A$ and $\ell_2 \sqsubseteq A$ and $\ell \sqsubseteq \ell_1 \sqcup \ell_2$.
  By IH, $\exists k^{\ell_1}_{\llbracket A \rrbracket} \in \mathrm{Hom}_{\mathbb{C}}(\mathbf{S}_{\ell_1}\llbracket A \rrbracket, \llbracket A \rrbracket)$ and $k^{\ell_2}_{\llbracket A \rrbracket} \in \mathrm{Hom}_{\mathbb{C}}(\mathbf{S}_{\ell_2}\llbracket A \rrbracket, \llbracket A \rrbracket)$ such that:

$$k^{\ell_1}_{\llbracket A \rrbracket} \circ \mathbf{S}^{\perp \sqsubseteq \ell_1}_{\llbracket A \rrbracket} \circ \eta_{\llbracket A \rrbracket} = \mathrm{id}_{\llbracket A \rrbracket}$$
$$k^{\ell_2}_{\llbracket A \rrbracket} \circ \mathbf{S}^{\perp \sqsubseteq \ell_2}_{\llbracket A \rrbracket} \circ \eta_{\llbracket A \rrbracket} = \mathrm{id}_{\llbracket A \rrbracket}.$$

Define: $k^{\ell}_{\llbracket A \rrbracket} = \mathbf{S}_{\ell}\llbracket A \rrbracket \xrightarrow{\mathbf{S}^{\ell \sqsubseteq \ell_1 \sqcup \ell_2}_{\llbracket A \rrbracket}} \mathbf{S}_{\ell_1 \sqcup \ell_2}\llbracket A \rrbracket \xrightarrow{\delta^{\ell_1,\ell_2}_{\llbracket A \rrbracket}} \mathbf{S}_{\ell_1}\mathbf{S}_{\ell_2}\llbracket A \rrbracket \xrightarrow{\mathbf{S}_{\ell_1}k^{\ell_2}_{\llbracket A \rrbracket}} \mathbf{S}_{\ell_1}\llbracket A \rrbracket \xrightarrow{k^{\ell_1}_{\llbracket A \rrbracket}} \llbracket A \rrbracket$.
Now,

$$\begin{aligned}
& k^{\ell}_{\llbracket A \rrbracket} \circ \mathbf{S}^{\perp \sqsubseteq \ell}_{\llbracket A \rrbracket} \circ \eta_{\llbracket A \rrbracket} \\
=& k^{\ell_1}_{\llbracket A \rrbracket} \circ \mathbf{S}_{\ell_1}k^{\ell_2}_{\llbracket A \rrbracket} \circ \delta^{\ell_1,\ell_2}_{\llbracket A \rrbracket} \circ \mathbf{S}^{\ell \sqsubseteq \ell_1 \sqcup \ell_2}_{\llbracket A \rrbracket} \circ \mathbf{S}^{\perp \sqsubseteq \ell}_{\llbracket A \rrbracket} \circ \eta_{\llbracket A \rrbracket} \\
=& k^{\ell_1}_{\llbracket A \rrbracket} \circ \mathbf{S}_{\ell_1}k^{\ell_2}_{\llbracket A \rrbracket} \circ \delta^{\ell_1,\ell_2}_{\llbracket A \rrbracket} \circ \mathbf{S}^{\perp \sqsubseteq \ell_1 \sqcup \ell_2}_{\llbracket A \rrbracket} \circ \eta_{\llbracket A \rrbracket} \ [\because \mathbf{S} \text{ is a functor}] \\
=& k^{\ell_1}_{\llbracket A \rrbracket} \circ \mathbf{S}_{\ell_1}k^{\ell_2}_{\llbracket A \rrbracket} \circ \delta^{\ell_1,\ell_2}_{\llbracket A \rrbracket} \circ \mathbf{S}^{\ell_1 \sqsubseteq \ell_1 \sqcup \ell_2}_{\llbracket A \rrbracket} \circ \mathbf{S}^{\perp \sqsubseteq \ell_1}_{\llbracket A \rrbracket} \circ \eta_{\llbracket A \rrbracket} \ [\because \mathbf{S} \text{ is a functor}] \\
=& k^{\ell_1}_{\llbracket A \rrbracket} \circ \mathrm{id}_{\mathbf{S}_{\ell_1}\llbracket A \rrbracket} \circ \mathbf{S}^{\perp \sqsubseteq \ell_1}_{\llbracket A \rrbracket} \circ \eta_{\llbracket A \rrbracket} \ [\text{By commutative diagram in Figure A.15}] \\
=& \mathrm{id}_{\llbracket A \rrbracket} \ [\text{By IH}]
\end{aligned}$$

The diagram in Figure A.15 commutes: the circular segment commutes because $\mathbf{S}$ is an oplax monoidal functor; the square commutes because δ is natural in its second component; the triangle commutes by IH.

□

Lemma A.13 If $\ell \sqsubseteq A$ in DCC_e, then $\mathbf{S}^{\perp \sqsubseteq \ell}_{\llbracket A \rrbracket} \circ \eta_{\llbracket A \rrbracket} \circ k^{\ell}_{\llbracket A \rrbracket} = \mathrm{id}_{\mathbf{S}_{\ell}\llbracket A \rrbracket}$.

Proof. Let $\ell \sqsubseteq A$. By Lemma A.12, we know that $\exists k^{\ell}_{[\![A]\!]}$ such that $k^{\ell}_{[\![A]\!]} \circ \mathbf{S}^{\bot\sqsubseteq\ell}_{[\![A]\!]} \circ \eta_{[\![A]\!]} = \mathrm{id}_{[\![A]\!]}$.

Now,

$$\begin{aligned} k^{\ell}_{[\![A]\!]} \circ \mathbf{S}^{\bot\sqsubseteq\ell}_{[\![A]\!]} \circ \eta_{[\![A]\!]} &= \mathrm{id}_{[\![A]\!]} \\ \text{or, } \mathbf{S}_\ell k^{\ell}_{[\![A]\!]} \circ \mathbf{S}_\ell \mathbf{S}^{\bot\sqsubseteq\ell}_{[\![A]\!]} \circ \mathbf{S}_\ell \eta_{[\![A]\!]} &= \mathbf{S}_\ell \mathrm{id}_{[\![A]\!]} \; [\because \mathbf{S}_\ell \text{ is a functor}] \\ \text{or, } \mathbf{S}_\ell k^{\ell}_{[\![A]\!]} \circ \mathbf{S}^{\bot\sqsubseteq\ell}_{\mathbf{S}_\ell [\![A]\!]} \circ \eta_{\mathbf{S}_\ell [\![A]\!]} &= \mathrm{id}_{\mathbf{S}_\ell [\![A]\!]} \; [\because (\mathbf{S}_\ell, \mathbf{S}^{\bot\sqsubseteq\ell} \circ \eta, \mu^{\ell,\ell}) \text{ is an idempotent monad}] \\ \text{or, } \mathbf{S}^{\bot\sqsubseteq\ell}_{[\![A]\!]} \circ \eta_{[\![A]\!]} \circ k^{\ell}_{[\![A]\!]} &= \mathrm{id}_{\mathbf{S}_\ell [\![A]\!]} \; [\text{By naturality of } \mathbf{S}^{\bot\sqsubseteq\ell} \circ \eta] \end{aligned}$$

This shows that $\mathbf{S}_\ell [\![A]\!] \cong [\![A]\!]$.

□

Lemma A.14 (Lemma 2.12) If $\ell \sqsubseteq A$, then $\exists$ an isomorphism $k : \mathbf{S}_\ell [\![A]\!] \to [\![A]\!]$.
Further, $k \circ \overline{\eta} = id_{[\![A]\!]}$ and $\overline{\eta} \circ k = \mathrm{id}_{\mathbf{S}_\ell [\![A]\!]}$ where $\overline{\eta} \triangleq \mathbf{S}^{\bot\sqsubseteq\ell} \circ \eta : [\![A]\!] \to \mathbf{S}_\ell [\![A]\!]$.

Proof. Follows by lemma A.12 and A.13. □

Theorem A.15 (Theorem 2.13) If $\Gamma \vdash a : A$ in DCC_e, then $[\![a]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], [\![A]\!])$.

Proof. By induction on $\Gamma \vdash a : A$.

- λ-calculus. Standard.
- Rule DCC-Eta. Have: $\Gamma \vdash \mathbf{eta}^{\ell}\ a : T_\ell\ A$ where $\Gamma \vdash a : A$.
  By IH, $[\![a]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], [\![A]\!])$.
  Therefore, $[\![\mathbf{eta}^{\ell}\ a]\!] = \mathbf{S}^{\bot\sqsubseteq\ell}_{[\![A]\!]} \circ \eta_{[\![A]\!]} \circ [\![a]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], \mathbf{S}_\ell [\![A]\!])$.
- Rule DCC-Bind. Have: $\Gamma \vdash \mathbf{bind}^{\ell}\ x = a\ \mathbf{in}\ b : B$ where $\Gamma \vdash a : T_\ell\ A$ and $\Gamma, x : A \vdash b : B$ and $\ell \sqsubseteq B$.
  By lemma A.12, $\exists k^{\ell}_{[\![B]\!]} \in \mathrm{Hom}_{\mathbb{C}}(\mathbf{S}_\ell [\![B]\!], [\![B]\!])$. Now,
  $[\![\mathbf{bind}^{\ell}\ x = a\ \mathbf{in}\ b]\!] = [\![\Gamma]\!] \xrightarrow{\langle \mathrm{id}_{[\![\Gamma]\!]}, [\![a]\!] \rangle} [\![\Gamma]\!] \times \mathbf{S}_\ell [\![A]\!] \xrightarrow{t^{\mathbf{S}_\ell}_{[\![\Gamma]\!],[\![A]\!]}} \mathbf{S}_\ell([\![\Gamma]\!] \times [\![A]\!]) \xrightarrow{\mathbf{S}_\ell [\![b]\!]} \mathbf{S}_\ell [\![B]\!] \xrightarrow{k^{\ell}_{[\![B]\!]}} [\![B]\!]$.
  Therefore, $[\![\mathbf{bind}^{\ell}\ x = a\ \mathbf{in}\ b]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], [\![B]\!])$.

□

Theorem A.16 (Theorem 2.14) If $\Gamma \vdash a : A$ in DCC_e and $\vdash a \rightsquigarrow a'$, then $[\![a]\!] = [\![a']\!]$.

Proof. By induction on $\Gamma \vdash a : A$.

- λ-calculus. Standard.
- Rule DCC-Eta. Have: $\Gamma \vdash \mathbf{eta}^{\ell}\ a : T_\ell\ A$ where $\Gamma \vdash a : A$.
  Further, $\vdash \mathbf{eta}^{\ell}\ a \rightsquigarrow c$. By inversion on $\vdash \mathbf{eta}^{\ell}\ a \rightsquigarrow c$.

– Rule CBV-Eta. Have: $\vdash \mathbf{eta}^\ell\ a \rightsquigarrow \mathbf{eta}^\ell\ a'$ where $\vdash a \rightsquigarrow a'$.
By IH, $[\![a]\!] = [\![a']\!]$.
Therefore, $[\![\mathbf{eta}^\ell\ a]\!] = \mathbf{S}^{\perp\sqsubseteq\ell}_{[\![A]\!]} \circ \eta_{[\![A]\!]} \circ [\![a]\!] = \mathbf{S}^{\perp\sqsubseteq\ell}_{[\![A]\!]} \circ \eta_{[\![A]\!]} \circ [\![a']\!] = [\![\mathbf{eta}^\ell\ a']\!]$.

- Rule DCC-Bind. Have: $\Gamma \vdash \mathbf{bind}^\ell\ x = a\ \mathbf{in}\ b : B$ where $\Gamma \vdash a : T_\ell\ A$ and $\Gamma, x : A \vdash b : B$ and $\ell \sqsubseteq B$.
Further, $\vdash \mathbf{bind}^\ell\ x = a\ \mathbf{in}\ b \rightsquigarrow c$. By inversion on $\vdash \mathbf{bind}^\ell\ x = a\ \mathbf{in}\ b \rightsquigarrow c$.

  – Rule CBV-BindLeft. Have: $\vdash \mathbf{bind}^\ell\ x = a\ \mathbf{in}\ b \rightsquigarrow \mathbf{bind}^\ell\ x = a'\ \mathbf{in}\ b$ where $\vdash a \rightsquigarrow a'$.
By IH, $[\![a]\!] = [\![a']\!]$.
Therefore,

$$\begin{aligned}[\![\mathbf{bind}^\ell\ x = a\ \mathbf{in}\ b]\!] &= k^\ell_{[\![B]\!]} \circ \mathbf{S}_\ell[\![b]\!] \circ t^{\mathbf{S}_\ell}_{[\![\Gamma]\!],[\![A]\!]} \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![a]\!]\rangle \\ &= k^\ell_{[\![B]\!]} \circ \mathbf{S}_\ell[\![b]\!] \circ t^{\mathbf{S}_\ell}_{[\![\Gamma]\!],[\![A]\!]} \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![a']\!]\rangle = [\![\mathbf{bind}^\ell\ x = a'\ \mathbf{in}\ b]\!].\end{aligned}$$

  – Rule CBV-BindBeta. Have: $\vdash \mathbf{bind}^\ell\ x = \mathbf{eta}^\ell\ v\ \mathbf{in}\ b \rightsquigarrow b\{v/x\}$.
Now,

$$\begin{aligned}&[\![\mathbf{bind}^\ell\ x = \mathbf{eta}^\ell\ v\ \mathbf{in}\ b]\!] \\ =& k^\ell_{[\![B]\!]} \circ \mathbf{S}_\ell[\![b]\!] \circ t^{\mathbf{S}_\ell}_{[\![\Gamma]\!],[\![A]\!]} \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, \mathbf{S}^{\perp\sqsubseteq\ell}_{[\![A]\!]} \circ \eta_{[\![A]\!]} \circ [\![v]\!]\rangle \\ =& k^\ell_{[\![B]\!]} \circ \mathbf{S}_\ell[\![b]\!] \circ t^{\mathbf{S}_\ell}_{[\![\Gamma]\!],[\![A]\!]} \circ (\mathrm{id}_{[\![\Gamma]\!]} \times \mathbf{S}^{\perp\sqsubseteq\ell}_{[\![A]\!]}) \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, \eta_{[\![A]\!]} \circ [\![v]\!]\rangle \\ =& k^\ell_{[\![B]\!]} \circ \mathbf{S}_\ell[\![b]\!] \circ \mathbf{S}^{\perp\sqsubseteq\ell}_{[\![\Gamma]\!]\times[\![A]\!]} \circ t^{\mathbf{S}_\perp}_{[\![\Gamma]\!],[\![A]\!]} \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, \eta_{[\![A]\!]} \circ [\![v]\!]\rangle\ [\because \mathbf{S}^{\perp\sqsubseteq\ell} \text{ is strong}] \\ =& k^\ell_{[\![B]\!]} \circ \mathbf{S}_\ell[\![b]\!] \circ \mathbf{S}^{\perp\sqsubseteq\ell}_{[\![\Gamma]\!]\times[\![A]\!]} \circ t^{\mathbf{S}_\perp}_{[\![\Gamma]\!],[\![A]\!]} \circ (\mathrm{id}_{[\![\Gamma]\!]} \times \eta_{[\![A]\!]}) \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![v]\!]\rangle \\ =& k^\ell_{[\![B]\!]} \circ \mathbf{S}_\ell[\![b]\!] \circ \mathbf{S}^{\perp\sqsubseteq\ell}_{[\![\Gamma]\!]\times[\![A]\!]} \circ \eta_{[\![\Gamma]\!]\times[\![A]\!]} \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![v]\!]\rangle\ [\because \eta \text{ is strong}] \\ =& k^\ell_{[\![B]\!]} \circ \mathbf{S}^{\perp\sqsubseteq\ell}_{[\![B]\!]} \circ \mathbf{S}_\perp[\![b]\!] \circ \eta_{[\![\Gamma]\!]\times[\![A]\!]} \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![v]\!]\rangle\ [\text{By naturality}] \\ =& k^\ell_{[\![B]\!]} \circ \mathbf{S}^{\perp\sqsubseteq\ell}_{[\![B]\!]} \circ \eta_{[\![B]\!]} \circ [\![b]\!] \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![v]\!]\rangle\ [\text{By naturality}] \\ =& \mathrm{id}_{[\![B]\!]} \circ [\![b]\!] \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![v]\!]\rangle\ [\text{By Lemma A.12}] \\ =& [\![b\{v/x\}]\!]\end{aligned}$$

□

Theorem A.17 (Theorem 2.15) Let the interpretation $[\![_]\!]_{(\mathbb{C},\mathbf{S})}$ be injective for ground types.

- Let $\Gamma \vdash b : \mathbf{Bool}$ and v be a value of type $\mathbf{Bool}$. If $[\![b]\!]_{(\mathbb{C},\mathbf{S})} = [\![v]\!]_{(\mathbb{C},\mathbf{S})}$, then $\vdash b \rightsquigarrow^* v$.

- Fix some $\ell \in L$. Let $\Gamma \vdash b : T_\ell\ \mathbf{Bool}$ and v be a value of type $T_\ell\ \mathbf{Bool}$. Suppose, the morphisms $\overline{\eta}_X \triangleq \mathbf{S}^{\perp\sqsubseteq\ell}_X \circ \eta_X$ are mono for any $X \in \mathrm{Obj}(\mathbb{C})$. Now, if $[\![b]\!]_{(\mathbb{C},\mathbf{S})} = [\![v]\!]_{(\mathbb{C},\mathbf{S})}$, then $\vdash b \rightsquigarrow^* v$.

Proof. Let $\Gamma \vdash b : \mathbf{Bool}$ and v be a boolean value such that $[\![b]\!]_{(\mathbb{C},\mathbf{S})} = [\![v]\!]_{(\mathbb{C},\mathbf{S})}$.
We show that $\vdash b \rightsquigarrow^* v$.

First note that DCC_e is strongly normalizing with respect to the reduction relation, $\rightsquigarrow^*$. Further, DCC_e is also type sound with respect to this reduction relation.

Therefore, given $\Gamma \vdash b : \textbf{Bool}$, we know that there exists a value v_0 such that $\Gamma \vdash v_0 : \textbf{Bool}$ and $\vdash b \rightsquigarrow^* v_0$.

Next, by Theorem A.16, $[\![b]\!]_{(\mathbb{C},\mathbf{S})} = [\![v_0]\!]_{(\mathbb{C},\mathbf{S})}$.

Since $[\![b]\!]_{(\mathbb{C},\mathbf{S})} = [\![v]\!]_{(\mathbb{C},\mathbf{S})}$ (given) and $[\![b]\!]_{(\mathbb{C},\mathbf{S})} = [\![v_0]\!]_{(\mathbb{C},\mathbf{S})}$ (above), therefore, $[\![v]\!]_{(\mathbb{C},\mathbf{S})} = [\![v_0]\!]_{(\mathbb{C},\mathbf{S})}$.

But, by injectivity, $v = v_0$.

Thus, $\vdash b \rightsquigarrow^* v$.

For the second part, we use a similar argument.

Given $\Gamma \vdash b : T_\ell\, \textbf{Bool}$, we know that there exists a value v_0 such that $\Gamma \vdash v_0 : T_\ell\, \textbf{Bool}$ and $\vdash b \rightsquigarrow^* v_0$.

By Theorem A.16, $[\![b]\!]_{(\mathbb{C},\mathbf{S})} = [\![v_0]\!]_{(\mathbb{C},\mathbf{S})}$.

Since $[\![b]\!]_{(\mathbb{C},\mathbf{S})} = [\![v]\!]_{(\mathbb{C},\mathbf{S})}$ (given) and $[\![b]\!]_{(\mathbb{C},\mathbf{S})} = [\![v_0]\!]_{(\mathbb{C},\mathbf{S})}$ (above), therefore, $[\![v]\!]_{(\mathbb{C},\mathbf{S})} = [\![v_0]\!]_{(\mathbb{C},\mathbf{S})}$.

Both v and v_0 are values of type $T_\ell\, \textbf{Bool}$. Therefore, $v = \textbf{eta}^\ell\ v'$ and $v_0 = \textbf{eta}^\ell\ v'_0$, for some values v' and v'_0 of type $\textbf{Bool}$.

Then, $[\![\textbf{eta}^\ell\ v']\!]_{(\mathbb{C},\mathbf{S})} = [\![\textbf{eta}^\ell\ v'_0]\!]_{(\mathbb{C},\mathbf{S})}$, or $\mathbf{S}^{\perp \sqsubseteq \ell}_{[\![\textbf{Bool}]\!]} \circ \eta_{[\![\textbf{Bool}]\!]} \circ [\![v']\!]_{(\mathbb{C},\mathbf{S})} = \mathbf{S}^{\perp \sqsubseteq \ell}_{[\![\textbf{Bool}]\!]} \circ \eta_{[\![\textbf{Bool}]\!]} \circ [\![v'_0]\!]_{(\mathbb{C},\mathbf{S})}$.

Since for any $X \in \text{Obj}(\mathbb{C})$, $\overline{\eta}_X$ is mono, so $[\![v']\!]_{(\mathbb{C},\mathbf{S})} = [\![v'_0]\!]_{(\mathbb{C},\mathbf{S})}$.

By injectivity, $v' = v'_0$.

Therefore, $v = v_0$.

Hence, $\vdash b \rightsquigarrow^* v$. $\square$

Theorem A.18 If $\Gamma \vdash a : A$ in $\text{GMCC}(\mathcal{L})$, then $[\![\overline{a}]\!] = [\![a]\!]$.

Proof. By induction on $\Gamma \vdash a : A$. Note that $[\![\overline{A}]\!] = [\![A]\!]$.

- λ-calculus. By IH.
- Rule MC-Return. Have: $\Gamma \vdash \textbf{ret}\ a : S_1\ A$ where $\Gamma \vdash a : A$.
  By IH, $[\![\overline{a}]\!] = [\![a]\!]$.
  Now, $\overline{\textbf{ret}\ a} = \textbf{eta}^\perp\ \overline{a}$.
  Then, $[\![\textbf{eta}^\perp\ \overline{a}]\!] = \mathbf{S}^{\perp \sqsubseteq \perp}_{[\![A]\!]} \circ \eta_{[\![A]\!]} \circ [\![\overline{a}]\!] = \eta_{[\![A]\!]} \circ [\![a]\!] = [\![\textbf{ret}\ a]\!]$.

- Rule MC-Extract. Have: $\Gamma \vdash \textbf{extr}\ a : A$ where $\Gamma \vdash a : S_1\ A$.
  By IH, $[\![\overline{a}]\!] = [\![a]\!]$.
  Now, $\overline{\textbf{extr}\ a} = \textbf{bind}^{\perp}\ x = \overline{a}\ \textbf{in}\ x$.
  Then,

$$
\begin{aligned}
&[\![\textbf{bind}^{\perp}\ x = \overline{a}\ \textbf{in}\ x]\!] \\
=&k^{\perp}_{[\![A]\!]} \circ \mathbf{S}_{\perp}\pi_2 \circ t^{\mathbf{S}_{\perp}}_{[\![\Gamma]\!],[\![A]\!]} \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![\overline{a}]\!]\rangle \\
=&k^{\perp}_{[\![A]\!]} \circ \pi_2 \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![a]\!]\rangle\ [\because \mathbf{S}^{\perp} \text{ is strong}] \\
=&\epsilon_{[\![A]\!]} \circ [\![a]\!]\ [\text{By Lemma A.12}] \\
=&[\![\textbf{extr}\ a]\!]
\end{aligned}
$$

- Rule MC-Fmap. Have: $\Gamma \vdash \textbf{lift}^{\ell} f : S_{\ell}\ A \to S_{\ell}\ B$ where $\Gamma \vdash f : A \to B$.
  By IH, $[\![\overline{f}]\!] = [\![f]\!]$.
  Now, $\overline{\textbf{lift}^{\ell} f} = \lambda x : T_{\ell}\ \overline{A}.\textbf{bind}^{\ell}\ y = x\ \textbf{in}\ \textbf{eta}^{\ell}\ (\overline{f}\ y)$.
  Then,

$$
\begin{aligned}
&[\![\lambda x : T_{\ell}\ \overline{A}.\textbf{bind}^{\ell}\ y = x\ \textbf{in}\ \textbf{eta}^{\ell}\ (\overline{f}\ y)]\!] \\
=&\Lambda\Big(k^{\ell}_{\mathbf{S}_{\ell}[\![B]\!]} \circ \mathbf{S}_{\ell}[\![\textbf{eta}^{\ell}\ (\overline{f}\ y)]\!] \circ t^{\mathbf{S}_{\ell}}_{[\![\Gamma]\!]\times\mathbf{S}_{\ell}[\![A]\!],[\![A]\!]} \circ \langle \mathrm{id}_{[\![\Gamma]\!]\times\mathbf{S}_{\ell}[\![A]\!]}, \pi_2\rangle\Big) \\
=&\Lambda\Big(k^{\ell}_{\mathbf{S}_{\ell}[\![B]\!]} \circ \mathbf{S}_{\ell}(\mathbf{S}^{\perp\sqsubseteq\ell}_{[\![B]\!]} \circ \eta_{[\![B]\!]} \circ \mathrm{app} \circ ([\![f]\!] \times \mathrm{id}_{[\![A]\!]}) \circ \langle \pi_1 \circ \pi_1, \pi_2\rangle) \circ t^{\mathbf{S}_{\ell}}_{[\![\Gamma]\!]\times\mathbf{S}_{\ell}[\![A]\!],[\![A]\!]} \circ \langle \mathrm{id}_{[\![\Gamma]\!]\times\mathbf{S}_{\ell}[\![A]\!]}, \pi_2\rangle\Big) \\
=&\Lambda\Big(k^{\ell}_{\mathbf{S}_{\ell}[\![B]\!]} \circ \mathbf{S}_{\ell}\mathbf{S}^{\perp\sqsubseteq\ell}_{[\![B]\!]} \circ \mathbf{S}_{\ell}\eta_{[\![B]\!]} \circ \mathbf{S}_{\ell}(\Lambda^{-1}[\![f]\!]) \circ \mathbf{S}_{\ell}\langle \pi_1 \circ \pi_1, \pi_2\rangle \circ t^{\mathbf{S}_{\ell}}_{[\![\Gamma]\!]\times\mathbf{S}_{\ell}[\![A]\!],[\![A]\!]} \circ \langle \mathrm{id}_{[\![\Gamma]\!]\times\mathbf{S}_{\ell}[\![A]\!]}, \pi_2\rangle\Big) \\
=&\Lambda\Big(k^{\ell}_{\mathbf{S}_{\ell}[\![B]\!]} \circ \mathbf{S}^{\perp\sqsubseteq\ell}_{\mathbf{S}_{\ell}[\![B]\!]} \circ \eta_{\mathbf{S}_{\ell}[\![B]\!]} \circ \mathbf{S}_{\ell}(\Lambda^{-1}[\![f]\!]) \circ \mathbf{S}_{\ell}\langle \pi_1 \circ \pi_1, \pi_2\rangle \circ t^{\mathbf{S}_{\ell}}_{[\![\Gamma]\!]\times\mathbf{S}_{\ell}[\![A]\!],[\![A]\!]} \circ \langle \mathrm{id}_{[\![\Gamma]\!]\times\mathbf{S}_{\ell}[\![A]\!]}, \pi_2\rangle\Big) \\
&\qquad [\text{By idempotence}] \\
=&\Lambda\Big(\mathrm{id}_{\mathbf{S}_{\ell}[\![B]\!]} \circ \mathbf{S}_{\ell}(\Lambda^{-1}[\![f]\!]) \circ \mathbf{S}_{\ell}\langle \pi_1 \circ \pi_1, \pi_2\rangle \circ t^{\mathbf{S}_{\ell}}_{[\![\Gamma]\!]\times\mathbf{S}_{\ell}[\![A]\!],[\![A]\!]} \circ \langle \mathrm{id}_{[\![\Gamma]\!]\times\mathbf{S}_{\ell}[\![A]\!]}, \pi_2\rangle\Big) \\
&\qquad [\text{By Theorem A.12}] \\
=&\Lambda\Big(\mathbf{S}_{\ell}(\Lambda^{-1}[\![f]\!]) \circ t^{\mathbf{S}_{\ell}}_{[\![\Gamma]\!],[\![A]\!]}\Big)\ [\text{By commutative diagram A.16}] \\
=&[\![\textbf{lift}^{\ell} f]\!]
\end{aligned}
$$

  The upper rectangle in Figure A.16 commutes by strength and naturality.

- Rule MC-Join. Have: $\Gamma \vdash \textbf{join}^{\ell_1,\ell_2}\ a : S_{\ell_1 \sqcup \ell_2}\ A$ where $\Gamma \vdash a : S_{\ell_1}\ S_{\ell_2}\ A$.
  By IH, $[\![\overline{a}]\!] = [\![a]\!]$.
  Now, $\overline{\textbf{join}^{\ell_1,\ell_2}\ a} = \textbf{bind}^{\ell_1}\ x = \overline{a}\ \textbf{in}\ \textbf{bind}^{\ell_2}\ y = x\ \textbf{in}\ \textbf{eta}^{\ell_1 \sqcup \ell_2}\ y$.
  Then,

$$
\begin{aligned}
&[\![\textbf{bind}^{\ell_1}\ x = \overline{a}\ \textbf{in}\ \textbf{bind}^{\ell_2}\ y = x\ \textbf{in}\ \textbf{eta}^{\ell_1 \sqcup \ell_2}\ y]\!] \\
=&k^{\ell_1}_{\mathbf{S}_{\ell_1 \sqcup \ell_2}[\![A]\!]} \circ \mathbf{S}_{\ell_1}[\![\textbf{bind}^{\ell_2}\ y = x\ \textbf{in}\ \textbf{eta}^{\ell_1 \sqcup \ell_2}\ y]\!] \circ t^{\mathbf{S}_{\ell_1}}_{[\![\Gamma]\!],\mathbf{S}_{\ell_2}[\![A]\!]} \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![\overline{a}]\!]\rangle
\end{aligned}
$$

$$
\begin{array}{ccccc}
\llbracket\Gamma\rrbracket\times\mathbf{S}_\ell\llbracket A\rrbracket & & \xrightarrow{t^{\mathbf{S}_\ell}_{\llbracket\Gamma\rrbracket,\llbracket A\rrbracket}} & & \mathbf{S}_\ell(\llbracket\Gamma\rrbracket\times\llbracket A\rrbracket) \\
\downarrow{\scriptstyle\langle \mathrm{id},\pi_2\rangle} & \nwarrow{\scriptstyle \mathrm{id}\times\mathbf{S}_\ell\pi_2} & & & \uparrow{\scriptstyle \mathbf{S}_\ell\langle\pi_1\circ\pi_1,\pi_2\rangle} \\
(\llbracket\Gamma\rrbracket\times\mathbf{S}_\ell\llbracket A\rrbracket)\times\mathbf{S}_\ell\llbracket A\rrbracket & & \xrightarrow{t^{\mathbf{S}_\ell}_{\llbracket\Gamma\rrbracket\times\mathbf{S}_\ell\llbracket A\rrbracket,\llbracket A\rrbracket}} & & \mathbf{S}_\ell((\llbracket\Gamma\rrbracket\times\mathbf{S}_\ell\llbracket A\rrbracket)\times\llbracket A\rrbracket) \\
\downarrow{\scriptstyle\alpha^{-1}} & & & & \downarrow{\scriptstyle\mathbf{S}_\ell\alpha^{-1}} \quad {\scriptstyle \mathbf{S}_\ell(\mathrm{id}\times\pi_2)} \\
\llbracket\Gamma\rrbracket\times(\mathbf{S}_\ell\llbracket A\rrbracket\times\mathbf{S}_\ell\llbracket A\rrbracket) & \xrightarrow{\mathrm{id}\times t^{\mathbf{S}_\ell}_{\mathbf{S}_\ell\llbracket A\rrbracket,\llbracket A\rrbracket}} & \llbracket\Gamma\rrbracket\times\mathbf{S}_\ell(\mathbf{S}_\ell\llbracket A\rrbracket\times\llbracket A\rrbracket) & \xrightarrow{t^{\mathbf{S}_\ell}_{\llbracket\Gamma\rrbracket,\mathbf{S}_\ell\llbracket A\rrbracket\times\llbracket A\rrbracket}} & \mathbf{S}_\ell(\llbracket\Gamma\rrbracket\times(\mathbf{S}_\ell\llbracket A\rrbracket\times\llbracket A\rrbracket))
\end{array}
$$

Figure A.16: Commutative diagram

$$
\begin{array}{ccccc}
\mathbf{S}_{\ell_1}\mathbf{S}_{\ell_2}\llbracket A\rrbracket & \xrightarrow{\mathbf{S}_{\ell_1}\mathbf{S}_{\ell_2}\eta_{\llbracket A\rrbracket}} & \mathbf{S}_{\ell_1}\mathbf{S}_{\ell_2}\mathbf{S}_{\bot}\llbracket A\rrbracket & \xrightarrow{\mathbf{S}_{\ell_1}\mathbf{S}_{\ell_2}\mathbf{S}^{\bot\sqsubseteq\ell_1\sqcup\ell_2}_{\llbracket A\rrbracket}} & \mathbf{S}_{\ell_1}\mathbf{S}_{\ell_2}\mathbf{S}_{\ell_1\sqcup\ell_2}\llbracket A\rrbracket \\
& \searrow{\scriptstyle\mathrm{id}} & \downarrow{\scriptstyle \mathbf{S}_{\ell_1}\mu^{\ell_2,\bot}_{\llbracket A\rrbracket}} & & \downarrow{\scriptstyle \mathbf{S}_{\ell_1}\mu^{\ell_2,\ell_1\sqcup\ell_2}_{\llbracket A\rrbracket}} \\
& & \mathbf{S}_{\ell_1}\mathbf{S}_{\ell_2}\llbracket A\rrbracket & \xrightarrow{\mathbf{S}_{\ell_1}\mathbf{S}^{\ell_2\sqsubseteq\ell_1\sqcup\ell_2}_{\llbracket A\rrbracket}} & \mathbf{S}_{\ell_1}\mathbf{S}_{\ell_1\sqcup\ell_2}\llbracket A\rrbracket \\
& & \downarrow{\scriptstyle \mu^{\ell_1,\ell_2}_{\llbracket A\rrbracket}} & & \downarrow{\scriptstyle \mu^{\ell_1,\ell_1\sqcup\ell_2}_{\llbracket A\rrbracket}} \\
& & \mathbf{S}_{\ell_1\sqcup\ell_2}\llbracket A\rrbracket & \xrightarrow{\mathrm{id}} & \mathbf{S}_{\ell_1\sqcup\ell_2}\llbracket A\rrbracket
\end{array}
$$

Figure A.17: Commutative diagram

$$
\begin{aligned}
=& k^{\ell_1}_{\mathbf{S}_{\ell_1\sqcup\ell_2}\llbracket A\rrbracket}\circ\mathbf{S}_{\ell_1}(k^{\ell_2}_{\mathbf{S}_{\ell_1\sqcup\ell_2}\llbracket A\rrbracket}\circ\mathbf{S}_{\ell_2}\llbracket\mathbf{eta}^{\ell_1\sqcup\ell_2}\ y\rrbracket\circ t^{\mathbf{S}_{\ell_2}}_{\llbracket\Gamma\rrbracket\times\mathbf{S}_{\ell_2}\llbracket A\rrbracket,\llbracket A\rrbracket}\circ\langle\mathrm{id}_{\llbracket\Gamma\rrbracket\times\mathbf{S}_{\ell_2}\llbracket A\rrbracket},\pi_2\rangle) \\
&\quad\circ t^{\mathbf{S}_{\ell_1}}_{\llbracket\Gamma\rrbracket,\mathbf{S}_{\ell_2}\llbracket A\rrbracket}\circ\langle\mathrm{id}_{\llbracket\Gamma\rrbracket},\llbracket a\rrbracket\rangle \\
=& k^{\ell_1}_{\mathbf{S}_{\ell_1\sqcup\ell_2}\llbracket A\rrbracket}\circ\mathbf{S}_{\ell_1}(k^{\ell_2}_{\mathbf{S}_{\ell_1\sqcup\ell_2}\llbracket A\rrbracket}\circ\mathbf{S}_{\ell_2}(\mathbf{S}^{\bot\sqsubseteq\ell_1\sqcup\ell_2}_{\llbracket A\rrbracket}\circ\eta_{\llbracket A\rrbracket}\circ\pi_2)\circ t^{\mathbf{S}_{\ell_2}}_{\llbracket\Gamma\rrbracket\times\mathbf{S}_{\ell_2}\llbracket A\rrbracket,\llbracket A\rrbracket}\circ\langle\mathrm{id},\pi_2\rangle) \\
&\quad\circ t^{\mathbf{S}_{\ell_1}}_{\llbracket\Gamma\rrbracket,\mathbf{S}_{\ell_2}\llbracket A\rrbracket}\circ\langle\mathrm{id},\llbracket a\rrbracket\rangle \\
=& \mu^{\ell_1,\ell_1\sqcup\ell_2}_{\llbracket A\rrbracket}\circ\mathbf{S}_{\ell_1}\mu^{\ell_2,\ell_1\sqcup\ell_2}_{\llbracket A\rrbracket}\circ\mathbf{S}_{\ell_1}(\mathbf{S}_{\ell_2}(\mathbf{S}^{\bot\sqsubseteq\ell_1\sqcup\ell_2}_{\llbracket A\rrbracket}\circ\eta_{\llbracket A\rrbracket}\circ\pi_2)\circ t^{\mathbf{S}_{\ell_2}}_{\llbracket\Gamma\rrbracket\times\mathbf{S}_{\ell_2}\llbracket A\rrbracket,\llbracket A\rrbracket}\circ\langle\mathrm{id},\pi_2\rangle) \\
&\quad\circ t^{\mathbf{S}_{\ell_1}}_{\llbracket\Gamma\rrbracket,\mathbf{S}_{\ell_2}\llbracket A\rrbracket}\circ\langle\mathrm{id},\llbracket a\rrbracket\rangle\ \text{[By Lemma A.12]} \\
=& \mu^{\ell_1,\ell_1\sqcup\ell_2}_{\llbracket A\rrbracket}\circ\mathbf{S}_{\ell_1}\mu^{\ell_2,\ell_1\sqcup\ell_2}_{\llbracket A\rrbracket}\circ\mathbf{S}_{\ell_1}\mathbf{S}_{\ell_2}\mathbf{S}^{\bot\sqsubseteq\ell_1\sqcup\ell_2}_{\llbracket A\rrbracket}\circ\mathbf{S}_{\ell_1}\mathbf{S}_{\ell_2}\eta_{\llbracket A\rrbracket}\circ\mathbf{S}_{\ell_1}\pi_2\circ\mathbf{S}_{\ell_1}\langle\mathrm{id},\pi_2\rangle \\
&\quad\circ t^{\mathbf{S}_{\ell_1}}_{\llbracket\Gamma\rrbracket,\mathbf{S}_{\ell_2}\llbracket A\rrbracket}\circ\langle\mathrm{id},\llbracket a\rrbracket\rangle\ \text{[By strength]} \\
=& \mu^{\ell_1,\ell_1\sqcup\ell_2}_{\llbracket A\rrbracket}\circ\mathbf{S}_{\ell_1}\mu^{\ell_2,\ell_1\sqcup\ell_2}_{\llbracket A\rrbracket}\circ\mathbf{S}_{\ell_1}\mathbf{S}_{\ell_2}\mathbf{S}^{\bot\sqsubseteq\ell_1\sqcup\ell_2}_{\llbracket A\rrbracket}\circ\mathbf{S}_{\ell_1}\mathbf{S}_{\ell_2}\eta_{\llbracket A\rrbracket}\circ\pi_2\circ\langle\mathrm{id},\llbracket a\rrbracket\rangle\ \text{[By strength]} \\
=& \mu^{\ell_1,\ell_1\sqcup\ell_2}_{\llbracket A\rrbracket}\circ\mathbf{S}_{\ell_1}\mu^{\ell_2,\ell_1\sqcup\ell_2}_{\llbracket A\rrbracket}\circ\mathbf{S}_{\ell_1}\mathbf{S}_{\ell_2}\mathbf{S}^{\bot\sqsubseteq\ell_1\sqcup\ell_2}_{\llbracket A\rrbracket}\circ\mathbf{S}_{\ell_1}\mathbf{S}_{\ell_2}\eta_{\llbracket A\rrbracket}\circ\llbracket a\rrbracket \\
=& \mu^{\ell_1,\ell_2}_{\llbracket A\rrbracket}\circ\llbracket a\rrbracket\ \text{[By Figure A.17]} \\
=& \llbracket\mathbf{join}^{\ell_1,\ell_2}\ a\rrbracket
\end{aligned}
$$

The diagram in Figure A.17 commutes by lax monoidality and naturality.

- Rule MC-Fork. Have: $\Gamma \vdash \mathbf{fork}^{\ell_1,\ell_2}\, a : S_{\ell_1}\ S_{\ell_2}\ A$ where $\Gamma \vdash a : S_{\ell_1 \sqcup \ell_2}\ A$.
  By IH, $\llbracket \overline{a} \rrbracket = \llbracket a \rrbracket$.
  Now $\overline{\mathbf{fork}^{\ell_1,\ell_2}\, a} = \mathbf{bind}^{\ell_1 \sqcup \ell_2}\ x = \overline{a}\ \mathbf{in}\ \mathbf{eta}^{\ell_1}\ \mathbf{eta}^{\ell_2}\ x$.

  Then,

$$
\begin{aligned}
&\llbracket \mathbf{bind}^{\ell_1 \sqcup \ell_2}\ x = \overline{a}\ \mathbf{in}\ \mathbf{eta}^{\ell_1}\ \mathbf{eta}^{\ell_2}\ x \rrbracket \\
=& k^{\ell_1 \sqcup \ell_2}_{\mathbf{S}_{\ell_1}\mathbf{S}_{\ell_2}\llbracket A \rrbracket} \circ \mathbf{S}_{\ell_1 \sqcup \ell_2} \llbracket \mathbf{eta}^{\ell_1}\ \mathbf{eta}^{\ell_2}\ x \rrbracket \circ t^{\mathbf{S}_{\ell_1 \sqcup \ell_2}}_{\llbracket \Gamma \rrbracket, \llbracket A \rrbracket} \circ \langle \mathrm{id}_{\llbracket \Gamma \rrbracket}, \llbracket \overline{a} \rrbracket \rangle \\
=& k^{\ell_1 \sqcup \ell_2}_{\mathbf{S}_{\ell_1}\mathbf{S}_{\ell_2}\llbracket A \rrbracket} \circ \mathbf{S}_{\ell_1 \sqcup \ell_2} (\mathbf{S}^{\perp \sqsubseteq \ell_1}_{\mathbf{S}_{\ell_2}\llbracket A \rrbracket} \circ \eta_{\mathbf{S}_{\ell_2}\llbracket A \rrbracket} \circ \mathbf{S}^{\perp \sqsubseteq \ell_2}_{\llbracket A \rrbracket} \circ \eta_{\llbracket A \rrbracket} \circ \pi_2) \circ t^{\mathbf{S}_{\ell_1 \sqcup \ell_2}}_{\llbracket \Gamma \rrbracket, \llbracket A \rrbracket} \circ \langle \mathrm{id}_{\llbracket \Gamma \rrbracket}, \llbracket a \rrbracket \rangle \\
=& k^{\ell_1 \sqcup \ell_2}_{\mathbf{S}_{\ell_1}\mathbf{S}_{\ell_2}\llbracket A \rrbracket} \circ \mathbf{S}_{\ell_1 \sqcup \ell_2} (\mathbf{S}^{\perp \sqsubseteq \ell_1}_{\mathbf{S}_{\ell_2}\llbracket A \rrbracket} \circ \eta_{\mathbf{S}_{\ell_2}\llbracket A \rrbracket} \circ \mathbf{S}^{\perp \sqsubseteq \ell_2}_{\llbracket A \rrbracket} \circ \eta_{\llbracket A \rrbracket}) \circ \llbracket a \rrbracket \ [\because \mathbf{S}_{\ell_1 \sqcup \ell_2} \text{ is strong}] \\
=& k^{\ell_1}_{\mathbf{S}_{\ell_1}\mathbf{S}_{\ell_2}\llbracket A \rrbracket} \circ \mathbf{S}_{\ell_1} k^{\ell_2}_{\mathbf{S}_{\ell_1}\mathbf{S}_{\ell_2}\llbracket A \rrbracket} \circ \delta^{\ell_1,\ell_2}_{\mathbf{S}_{\ell_1}\mathbf{S}_{\ell_2}\llbracket A \rrbracket} \circ \mathbf{S}_{\ell_1 \sqcup \ell_2} (\mathbf{S}^{\perp \sqsubseteq \ell_1}_{\mathbf{S}_{\ell_2}\llbracket A \rrbracket} \circ \eta_{\mathbf{S}_{\ell_2}\llbracket A \rrbracket} \circ \mathbf{S}^{\perp \sqsubseteq \ell_2}_{\llbracket A \rrbracket} \circ \eta_{\llbracket A \rrbracket}) \circ \llbracket a \rrbracket \\
&\qquad [\text{By Lemma A.12}] \\
=& k^{\ell_1}_{\mathbf{S}_{\ell_1}\mathbf{S}_{\ell_2}\llbracket A \rrbracket} \circ \mathbf{S}_{\ell_1} (\mathbf{S}_{\ell_1} k^{\ell_2}_{\mathbf{S}_{\ell_2}\llbracket A \rrbracket} \circ \delta^{\ell_1,\ell_2}_{\mathbf{S}_{\ell_2}\llbracket A \rrbracket} \circ \mu^{\ell_2,\ell_1}_{\mathbf{S}_{\ell_2}\llbracket A \rrbracket}) \circ \delta^{\ell_1,\ell_2}_{\mathbf{S}_{\ell_1}\mathbf{S}_{\ell_2}\llbracket A \rrbracket} \\
&\qquad \circ \mathbf{S}_{\ell_1 \sqcup \ell_2} (\mathbf{S}^{\perp \sqsubseteq \ell_1}_{\mathbf{S}_{\ell_2}\llbracket A \rrbracket} \circ \eta_{\mathbf{S}_{\ell_2}\llbracket A \rrbracket} \circ \mathbf{S}^{\perp \sqsubseteq \ell_2}_{\llbracket A \rrbracket} \circ \eta_{\llbracket A \rrbracket}) \circ \llbracket a \rrbracket \ [\text{By Lemma A.12}] \\
=& \mu^{\ell_1,\ell_1}_{\mathbf{S}_{\ell_2}\llbracket A \rrbracket} \circ \mathbf{S}_{\ell_1} (\mathbf{S}_{\ell_1} \mu^{\ell_2,\ell_2}_{\llbracket A \rrbracket} \circ \delta^{\ell_1,\ell_2}_{\mathbf{S}_{\ell_2}\llbracket A \rrbracket} \circ \mu^{\ell_2,\ell_1}_{\mathbf{S}_{\ell_2}\llbracket A \rrbracket}) \circ \delta^{\ell_1,\ell_2}_{\mathbf{S}_{\ell_1}\mathbf{S}_{\ell_2}\llbracket A \rrbracket} \\
&\qquad \circ \mathbf{S}_{\ell_1 \sqcup \ell_2} (\mathbf{S}^{\perp \sqsubseteq \ell_1}_{\mathbf{S}_{\ell_2}\llbracket A \rrbracket} \circ \eta_{\mathbf{S}_{\ell_2}\llbracket A \rrbracket} \circ \mathbf{S}^{\perp \sqsubseteq \ell_2}_{\llbracket A \rrbracket} \circ \eta_{\llbracket A \rrbracket}) \circ \llbracket a \rrbracket \ [\text{By Lemma A.12}] \\
=& \mathbf{S}_{\ell_1} \mu^{\perp,\ell_2}_{\llbracket A \rrbracket} \circ \mathbf{S}_{\ell_1}\mathbf{S}_{\perp} \mu^{\ell_2,\ell_2}_{\llbracket A \rrbracket} \circ \mathbf{S}_{\ell_1} \delta^{\perp,\ell_2}_{\mathbf{S}_{\ell_2}\llbracket A \rrbracket} \circ \mathbf{S}_{\ell_1}\mathbf{S}_{\ell_2}\mathbf{S}^{\perp \sqsubseteq \ell_2}_{\llbracket A \rrbracket} \circ \mathbf{S}_{\ell_1}\mathbf{S}_{\ell_2}\eta_{\llbracket A \rrbracket} \circ \delta^{\ell_1,\ell_2}_{\llbracket A \rrbracket} \circ \llbracket a \rrbracket \ [\text{By Fig. A.18}] \\
=& \delta^{\ell_1,\ell_2}_{\llbracket A \rrbracket} \circ \llbracket a \rrbracket \ [\text{By Figure A.19}] \\
=& \llbracket \mathbf{fork}^{\ell_1,\ell_2}\, a \rrbracket
\end{aligned}
$$

  The diagram in Figure A.18 commutes: all the squares commute by naturality; the ellipses commute because $\mathbf{S}_{\ell_1} \mu^{\perp,\ell_2}_{\llbracket A \rrbracket} = \mu^{\ell_1,\perp}_{\mathbf{S}_{\ell_2}\llbracket A \rrbracket} = \mathbf{S}_{\ell_1} \epsilon_{\mathbf{S}_{\ell_2}\llbracket A \rrbracket}$; the circular segment commutes by lax monoidality. The diagram in Figure A.19 commutes because $\mathbf{S}$ is a strong monoidal functor.

- Rule MC-Up. Have: $\Gamma \vdash \mathbf{up}^{\ell_1,\ell_2}\, a : S_{\ell_2}\ A$ where $\Gamma \vdash a : S_{\ell_1}\ A$ and $\ell_1 \sqsubseteq \ell_2$.
  By IH, $\llbracket \overline{a} \rrbracket = \llbracket a \rrbracket$.
  Now, $\overline{\mathbf{up}^{\ell_1,\ell_2}\, a} = \mathbf{bind}^{\ell_1}\ x = \overline{a}\ \mathbf{in}\ \mathbf{eta}^{\ell_2}\ x$.
  Then,

$$
\begin{aligned}
&\llbracket \mathbf{bind}^{\ell_1}\ x = \overline{a}\ \mathbf{in}\ \mathbf{eta}^{\ell_2}\ x \rrbracket \\
=& k^{\ell_1}_{\mathbf{S}_{\ell_2}\llbracket A \rrbracket} \circ \mathbf{S}_{\ell_1} \llbracket \mathbf{eta}^{\ell_2}\ x \rrbracket \circ t^{\mathbf{S}_{\ell_1}}_{\llbracket \Gamma \rrbracket, \llbracket A \rrbracket} \circ \langle \mathrm{id}_{\llbracket \Gamma \rrbracket}, \llbracket a \rrbracket \rangle \\
=& k^{\ell_1}_{\mathbf{S}_{\ell_2}\llbracket A \rrbracket} \circ \mathbf{S}_{\ell_1} (\mathbf{S}^{\perp \sqsubseteq \ell_2}_{\llbracket A \rrbracket} \circ \eta_{\llbracket A \rrbracket} \circ \pi_2) \circ t^{\mathbf{S}_{\ell_1}}_{\llbracket \Gamma \rrbracket, \llbracket A \rrbracket} \circ \langle \mathrm{id}_{\llbracket \Gamma \rrbracket}, \llbracket a \rrbracket \rangle \\
=& \mu^{\ell_1,\ell_2}_{\llbracket A \rrbracket} \circ \mathbf{S}_{\ell_1} (\mathbf{S}^{\perp \sqsubseteq \ell_2}_{\llbracket A \rrbracket} \circ \eta_{\llbracket A \rrbracket} \circ \pi_2) \circ t^{\mathbf{S}_{\ell_1}}_{\llbracket \Gamma \rrbracket, \llbracket A \rrbracket} \circ \langle \mathrm{id}_{\llbracket \Gamma \rrbracket}, \llbracket a \rrbracket \rangle \ [\text{By Theorem A.12}] \\
=& \mu^{\ell_1,\ell_2}_{\llbracket A \rrbracket} \circ \mathbf{S}_{\ell_1} (\mathbf{S}^{\ell_1 \sqsubseteq \ell_2}_{\llbracket A \rrbracket} \circ \mathbf{S}^{\perp \sqsubseteq \ell_1}_{\llbracket A \rrbracket} \circ \eta_{\llbracket A \rrbracket}) \circ \mathbf{S}_{\ell_1} \pi_2 \circ t^{\mathbf{S}_{\ell_1}}_{\llbracket \Gamma \rrbracket, \llbracket A \rrbracket} \circ \langle \mathrm{id}_{\llbracket \Gamma \rrbracket}, \llbracket a \rrbracket \rangle \\
=& \mu^{\ell_1,\ell_2}_{\llbracket A \rrbracket} \circ \mathbf{S}_{\ell_1} \mathbf{S}^{\ell_1 \sqsubseteq \ell_2}_{\llbracket A \rrbracket} \circ \mathbf{S}_{\ell_1} (\mathbf{S}^{\perp \sqsubseteq \ell_1}_{\llbracket A \rrbracket} \circ \eta_{\llbracket A \rrbracket}) \circ \pi_2 \circ \langle \mathrm{id}_{\llbracket \Gamma \rrbracket}, \llbracket a \rrbracket \rangle \ [\because \mathbf{S}_{\ell_1} \text{ is strong}] \\
=& \mathbf{S}^{\ell_1 \sqsubseteq \ell_2}_{\llbracket A \rrbracket} \circ \mu^{\ell_1,\ell_1}_{\llbracket A \rrbracket} \circ \mathbf{S}_{\ell_1} (\mathbf{S}^{\perp \sqsubseteq \ell_1}_{\llbracket A \rrbracket} \circ \eta_{\llbracket A \rrbracket}) \circ \llbracket a \rrbracket \ [\text{By naturality}]
\end{aligned}
$$

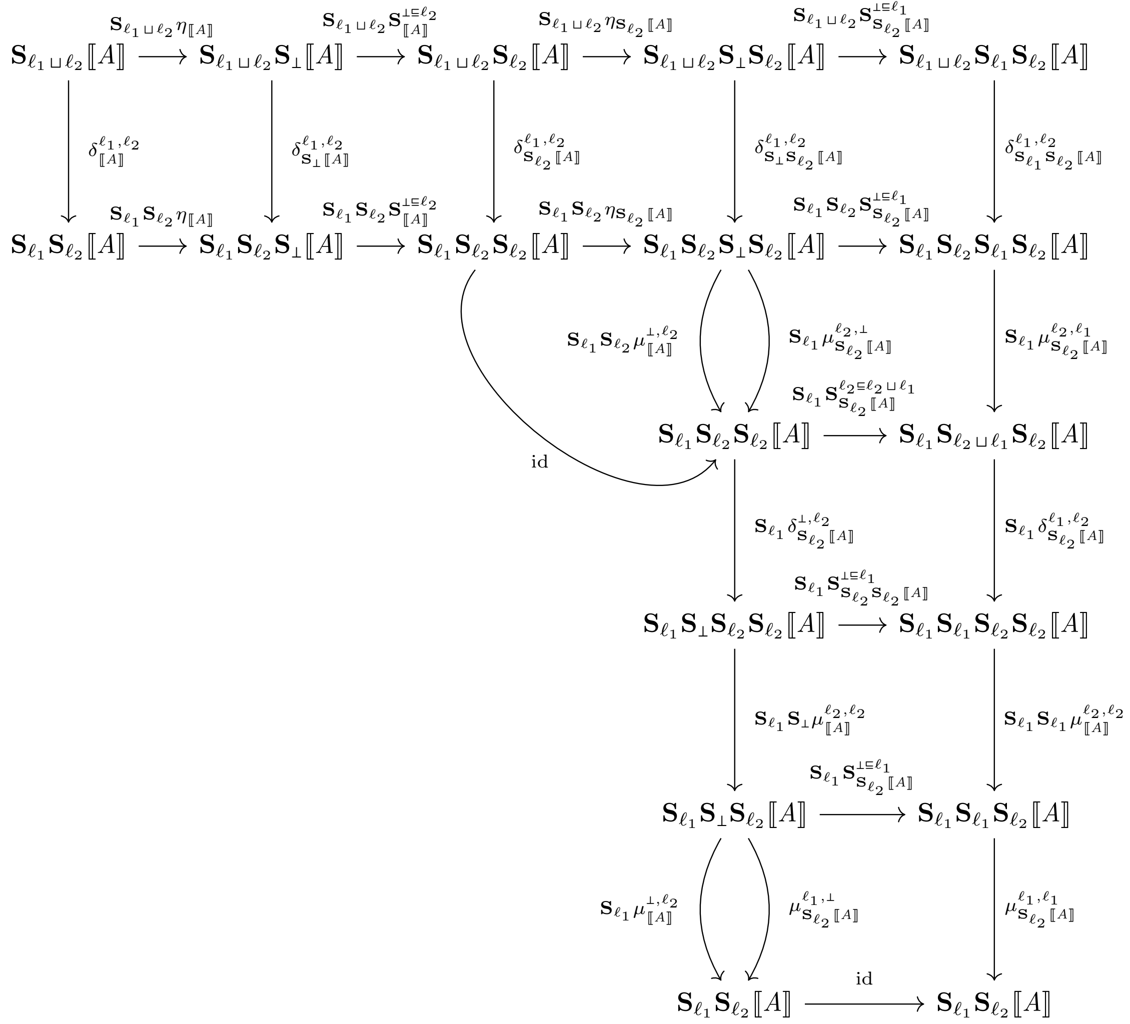


FIGURE A.18: Commutative diagram

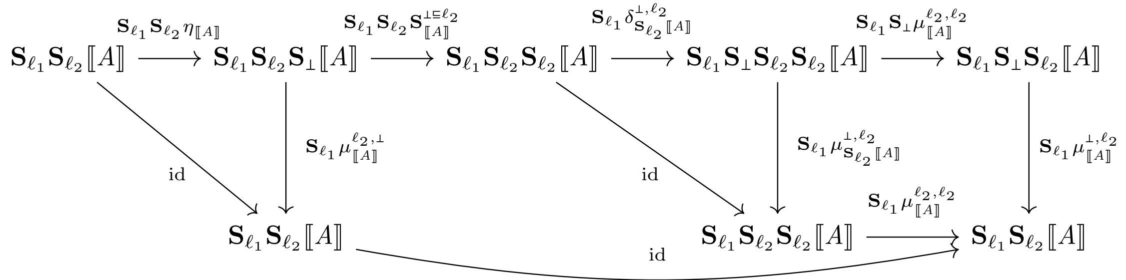


FIGURE A.19: Commutative diagram

$$=\mathbf{S}^{\ell_1 \sqsubseteq \ell_2}_{[\![A]\!]} \circ [\![a]\!] \; [\because (\mathbf{S}_{\ell_1}, \mathbf{S}^{\bot \sqsubseteq \ell_1} \circ \eta, \mu^{\ell_1,\ell_1}) \text{ is a monad}]$$
$$=[\![\mathbf{up}^{\ell_1,\ell_2}\, a]\!]$$

□

Theorem A.19 If $\Gamma \vdash a : A$ in $\mathrm{DCC}_e(\mathcal{L})$, then $[\![\underline{a}]\!] = [\![a]\!]$.

Proof. By induction on $\Gamma \vdash a : A$. Note that $[\![\underline{A}]\!] = [\![A]\!]$.

- λ-calculus. By IH.
- Rule DCC-Eta. Have: $\Gamma \vdash \mathbf{eta}^{\ell}\, a : T_{\ell}\, A$ where $\Gamma \vdash a : A$.
  By IH, $[\![\underline{a}]\!] = [\![a]\!]$.
  Now, $\underline{\mathbf{eta}^{\ell}\, a} = \mathbf{up}^{\bot,\ell}(\mathbf{ret}\ \underline{a})$.
  Then,

$$[\![\mathbf{up}^{\bot,\ell}(\mathbf{ret}\ \underline{a})]\!]$$
$$=\mathbf{S}^{\bot \sqsubseteq \ell}_{[\![A]\!]} \circ \eta_{[\![A]\!]} \circ [\![\underline{a}]\!]$$
$$=\mathbf{S}^{\bot \sqsubseteq \ell}_{[\![A]\!]} \circ \eta_{[\![A]\!]} \circ [\![a]\!] \text{ [By IH]}$$
$$=[\![\mathbf{eta}^{\ell}\, a]\!]$$

- Rule DCC-Bind. Have: $\Gamma \vdash \mathbf{bind}^{\ell}\, x = a\ \mathbf{in}\ b : B$ where $\Gamma \vdash a : T_{\ell}\, A$ and $\Gamma, x : A \vdash b : B$ and $\ell \sqsubseteq B$.
  By IH, $[\![\underline{a}]\!] = [\![a]\!]$ and $[\![\underline{b}]\!] = [\![b]\!]$.
  Now, $\underline{\mathbf{bind}^{\ell}\, x = a\ \mathbf{in}\ b} = j^{\ell}_{\underline{B}}\ ((\mathbf{lift}^{\ell}(\lambda x.\underline{b}))\ \underline{a})$.
  Then,

$$[\![j^{\ell}_{\underline{B}}\ ((\mathbf{lift}^{\ell}(\lambda x.\underline{b}))\ \underline{a})]\!]$$
$$=\mathrm{app} \circ \langle [\![j^{\ell}_{\underline{B}}]\!] \circ \langle\rangle, [\![(\mathbf{lift}^{\ell}(\lambda x.\underline{b}))\ \underline{a}]\!]\rangle$$
$$=\mathrm{app} \circ \langle [\![j^{\ell}_{\underline{B}}]\!] \circ \langle\rangle, \mathrm{app} \circ \langle [\![\mathbf{lift}^{\ell}(\lambda x.\underline{b})]\!], [\![\underline{a}]\!]\rangle\rangle$$
$$=\mathrm{app} \circ \langle [\![j^{\ell}_{\underline{B}}]\!] \circ \langle\rangle, \mathrm{app} \circ \langle \Lambda\Big(\mathbf{S}_{\ell}(\Lambda^{-1}[\![\lambda x.\underline{b}]\!]) \circ t^{\mathbf{S}_{\ell}}_{[\![\Gamma]\!],[\![A]\!]}\Big), [\![\underline{a}]\!]\rangle\rangle$$
$$=\mathrm{app} \circ \langle [\![j^{\ell}_{\underline{B}}]\!] \circ \langle\rangle, \mathrm{app} \circ \langle \Lambda\Big(\mathbf{S}_{\ell}[\![\underline{b}]\!] \circ t^{\mathbf{S}_{\ell}}_{[\![\Gamma]\!],[\![A]\!]}\Big), [\![\underline{a}]\!]\rangle\rangle$$
$$=\mathrm{app} \circ \langle [\![j^{\ell}_{\underline{B}}]\!] \circ \langle\rangle, \mathrm{app} \circ \langle \Lambda\Big(\mathbf{S}_{\ell}[\![b]\!] \circ t^{\mathbf{S}_{\ell}}_{[\![\Gamma]\!],[\![A]\!]}\Big), [\![a]\!]\rangle\rangle \text{ [By IH]}$$
$$=\mathrm{app} \circ \langle [\![j^{\ell}_{\underline{B}}]\!] \circ \langle\rangle, \mathbf{S}_{\ell}[\![b]\!] \circ t^{\mathbf{S}_{\ell}}_{[\![\Gamma]\!],[\![A]\!]} \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![a]\!]\rangle\rangle$$
$$=\Lambda^{-1}[\![j^{\ell}_{\underline{B}}]\!] \circ \langle\langle\rangle, \mathbf{S}_{\ell}[\![b]\!] \circ t^{\mathbf{S}_{\ell}}_{[\![\Gamma]\!],[\![A]\!]} \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![a]\!]\rangle\rangle$$
$$=\Lambda^{-1}[\![j^{\ell}_{\underline{B}}]\!] \circ \langle\langle\rangle, \mathrm{id}_{\mathbf{S}_{\ell}[\![B]\!]}\rangle \circ \mathbf{S}_{\ell}[\![b]\!] \circ t^{\mathbf{S}_{\ell}}_{[\![\Gamma]\!],[\![A]\!]} \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![a]\!]\rangle$$
$$=k^{\ell}_{[\![B]\!]} \circ \mathbf{S}_{\ell}[\![b]\!] \circ t^{\mathbf{S}_{\ell}}_{[\![\Gamma]\!],[\![A]\!]} \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![a]\!]\rangle \text{ [By Lemma A.20]}$$
$$=[\![\mathbf{bind}^{\ell}\, x = a\ \mathbf{in}\ b]\!]$$

$\square$

Lemma A.20 If $\ell \sqsubseteq A$ in DCC$_e$, then $k^\ell_{[\![A]\!]} = \Lambda^{-1}[\![j^\ell_{\underline{A}}]\!] \circ \langle\langle\rangle, \mathrm{id}_{\mathbf{S}_\ell [\![A]\!]}\rangle$.

Proof. By induction on $\ell \sqsubseteq A$.

- Rule Prot-Prod. Have $\ell \sqsubseteq A \times B$ where $\ell \sqsubseteq A$ and $\ell \sqsubseteq B$.
  By IH, $k^\ell_{[\![A]\!]} = \Lambda^{-1}[\![j^\ell_{\underline{A}}]\!] \circ \langle\langle\rangle, \mathrm{id}_{\mathbf{S}_\ell[\![A]\!]}\rangle$ and $k^\ell_{[\![B]\!]} = \Lambda^{-1}[\![j^\ell_{\underline{B}}]\!] \circ \langle\langle\rangle, \mathrm{id}_{\mathbf{S}_\ell[\![B]\!]}\rangle$.
  Now, $k^\ell_{[\![A]\!]\times[\![B]\!]} = k^\ell_{[\![A]\!]} \times k^\ell_{[\![B]\!]} \circ \langle \mathbf{S}_\ell \pi_1, \mathbf{S}_\ell \pi_2\rangle$.
  Next, $j^\ell_{\underline{A}\times\underline{B}} = \lambda z.(j^\ell_{\underline{A}}\ ((\mathbf{lift}^\ell(\lambda y.\mathbf{proj}_1\ y))\ z), j^\ell_{\underline{B}}\ ((\mathbf{lift}^\ell(\lambda y.\mathbf{proj}_2\ y))\ z))$.

  Then,

$$
\begin{aligned}
&\Lambda^{-1}[\![\lambda z.(j^\ell_{\underline{A}}\ ((\mathbf{lift}^\ell(\lambda y.\mathbf{proj}_1\ y))\ z), j^\ell_{\underline{B}}\ ((\mathbf{lift}^\ell(\lambda y.\mathbf{proj}_2\ y))\ z))]\!] \circ \langle\langle\rangle, \mathrm{id}\rangle\\
=&\langle [\![j^\ell_{\underline{A}}\ ((\mathbf{lift}^\ell(\lambda y.\mathbf{proj}_1\ y))\ z)]\!], [\![j^\ell_{\underline{B}}\ ((\mathbf{lift}^\ell(\lambda y.\mathbf{proj}_2\ y))\ z)]\!]\rangle \circ \langle\langle\rangle, \mathrm{id}\rangle\\
=&\langle \mathrm{app} \circ \langle [\![j^\ell_{\underline{A}}]\!] \circ \langle\rangle, [\![(\mathbf{lift}^\ell(\lambda y.\mathbf{proj}_1\ y))\ z]\!]\rangle, \mathrm{app} \circ \langle [\![j^\ell_{\underline{B}}]\!] \circ \langle\rangle, [\![(\mathbf{lift}^\ell(\lambda y.\mathbf{proj}_2\ y))\ z]\!]\rangle\rangle \circ \langle\langle\rangle, \mathrm{id}\rangle\\
=&\langle \Lambda^{-1}[\![j^\ell_{\underline{A}}]\!] \circ \langle\langle\rangle, [\![(\mathbf{lift}^\ell(\lambda y.\mathbf{proj}_1\ y))\ z]\!]\rangle, \Lambda^{-1}[\![j^\ell_{\underline{B}}]\!] \circ \langle\langle\rangle, [\![(\mathbf{lift}^\ell(\lambda y.\mathbf{proj}_2\ y))\ z]\!]\rangle\rangle \circ \langle\langle\rangle, \mathrm{id}\rangle\\
=&\langle k^\ell_{[\![A]\!]} \circ [\![(\mathbf{lift}^\ell(\lambda y.\mathbf{proj}_1\ y))\ z]\!], k^\ell_{[\![B]\!]} \circ [\![(\mathbf{lift}^\ell(\lambda y.\mathbf{proj}_2\ y))\ z]\!]\rangle \circ \langle\langle\rangle, \mathrm{id}\rangle \text{ [By IH]}\\
=&\langle k^\ell_{[\![A]\!]} \circ \mathrm{app} \circ \langle [\![\mathbf{lift}^\ell(\lambda y.\mathbf{proj}_1\ y)]\!], \pi_2\rangle, k^\ell_{[\![B]\!]} \circ \mathrm{app} \circ \langle [\![\mathbf{lift}^\ell(\lambda y.\mathbf{proj}_2\ y)]\!], \pi_2\rangle\rangle \circ \langle\langle\rangle, \mathrm{id}\rangle\\
=&\langle k^\ell_{[\![A]\!]} \circ \mathrm{app} \circ \langle \Lambda\Big(\mathbf{S}_\ell(\Lambda^{-1}[\![\lambda y.\mathbf{proj}_1\ y]\!]) \circ t^{\mathbf{S}_\ell}_{X,Y}\Big), \pi_2\rangle,\\
&\qquad k^\ell_{[\![B]\!]} \circ \mathrm{app} \circ \langle \Lambda\Big(\mathbf{S}_\ell(\Lambda^{-1}[\![\lambda y.\mathbf{proj}_2\ y]\!]) \circ t^{\mathbf{S}_\ell}_{X,Y}\Big), \pi_2\rangle\rangle \circ \langle\langle\rangle, \mathrm{id}\rangle\\
&\qquad [X := \top \times \mathbf{S}_\ell([\![A]\!] \times [\![B]\!]) \text{ and } Y := [\![A]\!] \times [\![B]\!]]\\
=&\langle k^\ell_{[\![A]\!]} \circ \mathrm{app} \circ \langle \Lambda\Big(\mathbf{S}_\ell(\pi_1 \circ \pi_2) \circ t^{\mathbf{S}_\ell}_{X,Y}\Big), \pi_2\rangle, k^\ell_{[\![B]\!]} \circ \mathrm{app} \circ \langle \Lambda\Big(\mathbf{S}_\ell(\pi_2 \circ \pi_2) \circ t^{\mathbf{S}_\ell}_{X,Y}\Big), \pi_2\rangle\rangle \circ \langle\langle\rangle, \mathrm{id}\rangle\\
=&\langle k^\ell_{[\![A]\!]} \circ \mathrm{app} \circ \langle \Lambda(\mathbf{S}_\ell \pi_1 \circ \pi_2), \pi_2\rangle, k^\ell_{[\![B]\!]} \circ \mathrm{app} \circ \langle \Lambda(\mathbf{S}_\ell \pi_2 \circ \pi_2), \pi_2\rangle\rangle \circ \langle\langle\rangle, \mathrm{id}\rangle\\
=&\langle k^\ell_{[\![A]\!]} \circ \mathrm{app} \circ \Lambda(\mathbf{S}_\ell \pi_1 \circ \pi_2) \times \mathrm{id} \circ \langle \mathrm{id}, \pi_2\rangle, k^\ell_{[\![B]\!]} \circ \mathrm{app} \circ \Lambda(\mathbf{S}_\ell \pi_2 \circ \pi_2) \times \mathrm{id} \circ \langle \mathrm{id}, \pi_2\rangle\rangle \circ \langle\langle\rangle, \mathrm{id}\rangle\\
=&\langle k^\ell_{[\![A]\!]} \circ \mathbf{S}_\ell \pi_1 \circ \pi_2 \circ \langle \mathrm{id}, \pi_2\rangle, k^\ell_{[\![B]\!]} \circ \mathbf{S}_\ell \pi_2 \circ \pi_2 \circ \langle \mathrm{id}, \pi_2\rangle\rangle \circ \langle\langle\rangle, \mathrm{id}\rangle\\
=&\langle k^\ell_{[\![A]\!]} \circ \mathbf{S}_\ell \pi_1, k^\ell_{[\![B]\!]} \circ \mathbf{S}_\ell \pi_2\rangle \circ \pi_2 \circ \langle \mathrm{id}, \pi_2\rangle \circ \langle\langle\rangle, \mathrm{id}\rangle\\
=&\langle k^\ell_{[\![A]\!]} \circ \mathbf{S}_\ell \pi_1, k^\ell_{[\![B]\!]} \circ \mathbf{S}_\ell \pi_2\rangle\\
=&k^\ell_{[\![A]\!]\times[\![B]\!]}
\end{aligned}
$$

- Rule Prot-Fun. Have: $\ell \sqsubseteq A \to B$ where $\ell \sqsubseteq B$.
  By IH, $k^\ell_{[\![B]\!]} = \Lambda^{-1}[\![j^\ell_{\underline{B}}]\!] \circ \langle\langle\rangle, \mathrm{id}_{\mathbf{S}_\ell[\![B]\!]}\rangle$.
  Now, $k^\ell_{[\![B]\!]^{[\![A]\!]}} = \Lambda(k^\ell_{[\![B]\!]} \circ \mathbf{S}_\ell \mathrm{app} \circ \mathbf{S}_\ell\langle \pi_2, \pi_1\rangle \circ t^{\mathbf{S}_\ell}_{[\![A]\!], [\![B]\!]^{[\![A]\!]}} \circ \langle \pi_2, \pi_1\rangle)$.
  And $j^\ell_{\underline{A}\to\underline{B}} = \lambda z.\lambda y.j^\ell_{\underline{B}}\ ((\mathbf{lift}^\ell(\lambda x.x\ y))\ z)$.

  Then,

$$\Lambda^{-1}[\![\lambda z.\lambda y.j^\ell_{\underline{B}}\ ((\mathbf{lift}^\ell(\lambda x.x\ y))\ z)]\!] \circ \langle\langle\rangle, \mathrm{id}\rangle$$

$$=[\![\lambda y.j^{\ell}_{\underline{B}}\ ((\mathbf{lift}^{\ell}(\lambda x.x\ y))\ z)]\!] \circ \langle\langle\rangle, \mathrm{id}\rangle$$

$$=\Lambda\Big(\mathrm{app} \circ \langle [\![j^{\ell}_{\underline{B}}]\!] \circ \langle\rangle, [\![(\mathbf{lift}^{\ell}(\lambda x.x\ y))\ z]\!]\rangle\Big) \circ \langle\langle\rangle, \mathrm{id}\rangle$$

$$=\Lambda\Big(\mathrm{app} \circ ([\![j^{\ell}_{\underline{B}}]\!] \times \mathrm{id}) \circ \langle\langle\rangle, [\![(\mathbf{lift}^{\ell}(\lambda x.x\ y))\ z]\!]\rangle\Big) \circ \langle\langle\rangle, \mathrm{id}\rangle$$

$$=\Lambda\Big(\Lambda^{-1}[\![j^{\ell}_{\underline{B}}]\!] \circ \langle\langle\rangle, \mathrm{app} \circ \langle [\![\mathbf{lift}^{\ell}(\lambda x.x\ y)]\!], \pi_2 \circ \pi_1\rangle\rangle\Big) \circ \langle\langle\rangle, \mathrm{id}\rangle$$

$$=\Lambda\Big(\Lambda^{-1}[\![j^{\ell}_{\underline{B}}]\!] \circ \langle\langle\rangle, \mathrm{app} \circ \langle \Lambda\Big(\mathbf{S}_{\ell}(\Lambda^{-1}[\![\lambda x.x\ y]\!]) \circ t^{\mathbf{S}_{\ell}}_{(\top\times\mathbf{S}_{\ell}[\![B]\!]^{[\![A]\!]})\times[\![A]\!],[\![B]\!]^{[\![A]\!]}}\Big), \pi_2 \circ \pi_1\rangle\rangle\Big) \circ \langle\langle\rangle, \mathrm{id}\rangle$$

$$=\Lambda\Big(\Lambda^{-1}[\![j^{\ell}_{\underline{B}}]\!] \circ \langle\langle\rangle, \mathrm{app} \circ \langle \Lambda\Big(\mathbf{S}_{\ell}(\mathrm{app} \circ \langle \pi_2, \pi_2 \circ \pi_1\rangle) \circ t^{\mathbf{S}_{\ell}}_{(\top\times\mathbf{S}_{\ell}[\![B]\!]^{[\![A]\!]})\times[\![A]\!],[\![B]\!]^{[\![A]\!]}}\Big), \pi_2 \circ \pi_1\rangle\rangle\Big) \circ \langle\langle\rangle, \mathrm{id}\rangle$$

$$=\Lambda\Big(\Lambda^{-1}[\![j^{\ell}_{\underline{B}}]\!] \circ \langle\langle\rangle, \mathbf{S}_{\ell}\mathrm{app} \circ \mathbf{S}_{\ell}\langle \pi_2, \pi_1\rangle \circ \mathbf{S}_{\ell}\langle \pi_2 \circ \pi_1, \pi_2\rangle \circ t^{\mathbf{S}_{\ell}}_{(\top\times\mathbf{S}_{\ell}[\![B]\!]^{[\![A]\!]})\times[\![A]\!],[\![B]\!]^{[\![A]\!]}} \circ \langle \mathrm{id}, \pi_2 \circ \pi_1\rangle\rangle\Big) \circ \langle\langle\rangle, \mathrm{id}\rangle$$

$$=\Lambda\Big(\Lambda^{-1}[\![j^{\ell}_{\underline{B}}]\!] \circ \langle\langle\rangle, \mathbf{S}_{\ell}\mathrm{app} \circ \mathbf{S}_{\ell}\langle \pi_2, \pi_1\rangle \circ t^{\mathbf{S}_{\ell}}_{[\![A]\!],[\![B]\!]^{[\![A]\!]}} \circ \langle \pi_2 \circ \pi_1, \pi_2\rangle \circ \langle \mathrm{id}, \pi_2 \circ \pi_1\rangle\rangle\Big) \circ \langle\langle\rangle, \mathrm{id}\rangle$$

[By naturality]

$$=\Lambda\Big(\Lambda^{-1}[\![j^{\ell}_{\underline{B}}]\!] \circ \langle\langle\rangle, \mathbf{S}_{\ell}\mathrm{app} \circ \mathbf{S}_{\ell}\langle \pi_2, \pi_1\rangle \circ t^{\mathbf{S}_{\ell}}_{[\![A]\!],[\![B]\!]^{[\![A]\!]}} \circ \langle \pi_2, \pi_2 \circ \pi_1\rangle\rangle \circ \langle\langle\langle\rangle, \pi_1\rangle, \pi_2\rangle\Big)$$

$$=\Lambda\Big(\Lambda^{-1}[\![j^{\ell}_{\underline{B}}]\!] \circ \langle\langle\rangle, \mathbf{S}_{\ell}\mathrm{app} \circ \mathbf{S}_{\ell}\langle \pi_2, \pi_1\rangle \circ t^{\mathbf{S}_{\ell}}_{[\![A]\!],[\![B]\!]^{[\![A]\!]}} \circ \langle \pi_2, \pi_1\rangle\rangle\Big)$$

$$=\Lambda\Big(\Lambda^{-1}[\![j^{\ell}_{\underline{B}}]\!] \circ \langle\langle\rangle, \mathrm{id}\rangle \circ \mathbf{S}_{\ell}\mathrm{app} \circ \mathbf{S}_{\ell}\langle \pi_2, \pi_1\rangle \circ t^{\mathbf{S}_{\ell}}_{[\![A]\!],[\![B]\!]^{[\![A]\!]}} \circ \langle \pi_2, \pi_1\rangle\Big)$$

$$=\Lambda\Big(k^{\ell}_{[\![B]\!]} \circ \mathbf{S}_{\ell}\mathrm{app} \circ \mathbf{S}_{\ell}\langle \pi_2, \pi_1\rangle \circ t^{\mathbf{S}_{\ell}}_{[\![A]\!],[\![B]\!]^{[\![A]\!]}} \circ \langle \pi_2, \pi_1\rangle\Big) \text{ [By IH]}$$

$$=k^{\ell}_{[\![B]\!]^{[\![A]\!]}}$$

- Rule Prot-Monad. Have: $\ell_1 \sqsubseteq T_{\ell_2}\ A$ where $\ell_1 \sqsubseteq \ell_2$.
  Now, $k^{\ell_1}_{\mathbf{S}_{\ell_2}[\![A]\!]} = \mu^{\ell_1,\ell_2}_{[\![A]\!]}$ and $j^{\ell}_{\underline{A}} = \lambda x.\mathbf{join}^{\ell_1,\ell_2} x$.
  Then, $\Lambda^{-1}[\![\lambda x.\mathbf{join}^{\ell_1,\ell_2} x]\!] \circ \langle\langle\rangle, \mathrm{id}_{\mathbf{S}_{\ell_1}\mathbf{S}_{\ell_2}[\![A]\!]}\rangle = \mu^{\ell_1,\ell_2}_{[\![A]\!]} \circ \pi_2 \circ \langle\langle\rangle, \mathrm{id}_{\mathbf{S}_{\ell_1}\mathbf{S}_{\ell_2}[\![A]\!]}\rangle = \mu^{\ell_1,\ell_2}_{[\![A]\!]}$.

- Rule Prot-Already. Have: $\ell \sqsubseteq T_{\ell'}\ A$ where $\ell \sqsubseteq A$.
  By IH, $k^{\ell}_{[\![A]\!]} = \Lambda^{-1}[\![j^{\ell}_{\underline{A}}]\!] \circ \langle\langle\rangle, \mathrm{id}_{\mathbf{S}_{\ell}[\![A]\!]}\rangle$.
  Now, $k^{\ell}_{\mathbf{S}_{\ell'}[\![A]\!]} = \mathbf{S}_{\ell'} k^{\ell}_{[\![A]\!]} \circ \delta^{\ell',\ell}_{[\![A]\!]} \circ \mu^{\ell,\ell'}_{[\![A]\!]}$ and $j^{\ell}_{S_{\ell'}\ \underline{A}} = \lambda x.(\mathbf{lift}^{\ell'}\ j^{\ell}_{\underline{A}})\ (\mathbf{fork}^{\ell',\ell}(\mathbf{join}^{\ell,\ell'} x))$.
  Then,

$$\Lambda^{-1}[\![\lambda x.(\mathbf{lift}^{\ell'}\ j^{\ell}_{\underline{A}})\ (\mathbf{fork}^{\ell',\ell}(\mathbf{join}^{\ell,\ell'} x))]\!] \circ \langle\langle\rangle, \mathrm{id}_{\mathbf{S}_{\ell}\mathbf{S}_{\ell'}[\![A]\!]}\rangle$$

$$=\mathrm{app} \circ \langle [\![\mathbf{lift}^{\ell'}\ j^{\ell}_{\underline{A}}]\!], [\![\mathbf{fork}^{\ell',\ell}(\mathbf{join}^{\ell,\ell'} x)]\!]\rangle \circ \langle\langle\rangle, \mathrm{id}_{\mathbf{S}_{\ell}\mathbf{S}_{\ell'}[\![A]\!]}\rangle$$

$$=\mathrm{app} \circ \langle \Lambda\Big(\mathbf{S}_{\ell'}(\Lambda^{-1}[\![j^{\ell}_{\underline{A}}]\!]) \circ t^{\mathbf{S}_{\ell'}}_{\top,\mathbf{S}_{\ell}[\![A]\!]}\Big) \circ \langle\rangle, \delta^{\ell',\ell}_{[\![A]\!]} \circ \mu^{\ell,\ell'}_{[\![A]\!]} \circ \pi_2\rangle \circ \langle\langle\rangle, \mathrm{id}_{\mathbf{S}_{\ell}\mathbf{S}_{\ell'}[\![A]\!]}\rangle$$

$$=\mathbf{S}_{\ell'}(\Lambda^{-1}[\![j^{\ell}_{\underline{A}}]\!] \circ \langle\langle\rangle, \mathrm{id}_{\mathbf{S}_{\ell}[\![A]\!]}\rangle \circ \pi_2) \circ t^{\mathbf{S}_{\ell'}}_{\top,\mathbf{S}_{\ell}[\![A]\!]} \circ \langle\langle\rangle, \delta^{\ell',\ell}_{[\![A]\!]} \circ \mu^{\ell,\ell'}_{[\![A]\!]} \circ \pi_2\rangle \circ \langle\langle\rangle, \mathrm{id}_{\mathbf{S}_{\ell}\mathbf{S}_{\ell'}[\![A]\!]}\rangle$$

$$[\because \langle\langle\rangle, \mathrm{id}\rangle \circ \pi_2 = \mathrm{id}_{\top\times\mathbf{S}_{\ell}[\![A]\!]}]$$

$$=\mathbf{S}_{\ell'}(\Lambda^{-1}[\![j^{\ell}_{\underline{A}}]\!] \circ \langle\langle\rangle, \mathrm{id}_{\mathbf{S}_{\ell}[\![A]\!]}\rangle) \circ \mathbf{S}_{\ell'}\pi_2 \circ t^{\mathbf{S}_{\ell'}}_{\top,\mathbf{S}_{\ell}[\![A]\!]} \circ \langle\langle\rangle, \delta^{\ell',\ell}_{[\![A]\!]} \circ \mu^{\ell,\ell'}_{[\![A]\!]} \circ \pi_2\rangle \circ \langle\langle\rangle, \mathrm{id}_{\mathbf{S}_{\ell}\mathbf{S}_{\ell'}[\![A]\!]}\rangle$$

$$=\mathbf{S}_{\ell'} k^{\ell}_{[\![A]\!]} \circ \mathbf{S}_{\ell'}\pi_2 \circ t^{\mathbf{S}_{\ell'}}_{\top,\mathbf{S}_{\ell}[\![A]\!]} \circ \langle\langle\rangle, \delta^{\ell',\ell}_{[\![A]\!]} \circ \mu^{\ell,\ell'}_{[\![A]\!]} \circ \pi_2\rangle \circ \langle\langle\rangle, \mathrm{id}_{\mathbf{S}_{\ell}\mathbf{S}_{\ell'}[\![A]\!]}\rangle \text{ [By IH]}$$

$$=\mathbf{S}_{\ell'} k^{\ell}_{[\![A]\!]} \circ \pi_2 \circ \langle\langle\rangle, \delta^{\ell',\ell}_{[\![A]\!]} \circ \mu^{\ell,\ell'}_{[\![A]\!]} \circ \pi_2\rangle \circ \langle\langle\rangle, \mathrm{id}_{\mathbf{S}_{\ell}\mathbf{S}_{\ell'}[\![A]\!]}\rangle \text{ [By strength]}$$

$$=\mathbf{S}_{\ell'} k^{\ell}_{[\![A]\!]} \circ \delta^{\ell',\ell}_{[\![A]\!]} \circ \mu^{\ell,\ell'}_{[\![A]\!]}$$

$$=k^{\ell}_{\mathbf{S}_{\ell'}[\![A]\!]}$$

- Rule Prot-Minimum. Have: $\bot \sqsubseteq A$.
  Now, $k^{\bot}_{[\![A]\!]} = \epsilon_{[\![A]\!]}$ and $j^{\bot}_{\underline{A}} = \lambda x.\mathbf{extr}\, x$.
  Then, $\Lambda^{-1}[\![\lambda x.\mathbf{extr}\, x]\!] \circ \langle\langle\rangle, \mathrm{id}_{\mathbf{S}_{\bot}[\![A]\!]}\rangle = \epsilon_{[\![A]\!]} \circ \pi_2 \circ \langle\langle\rangle, \mathrm{id}_{\mathbf{S}_{\bot}[\![A]\!]}\rangle = \epsilon_{[\![A]\!]} = k^{\bot}_{[\![A]\!]}$.
- Rule Prot-Combine. Have: $\ell \sqsubseteq A$ where $\ell_1 \sqsubseteq A$ and $\ell_2 \sqsubseteq A$ and $\ell \sqsubseteq \ell_1 \sqcup \ell_2$.
  By IH, $k^{\ell_1}_{[\![A]\!]} = \Lambda^{-1}[\![j^{\ell_1}_{\underline{A}}]\!] \circ \langle\langle\rangle, \mathrm{id}_{\mathbf{S}_{\ell_1}[\![A]\!]}\rangle$ and $k^{\ell_2}_{[\![A]\!]} = \Lambda^{-1}[\![j^{\ell_2}_{\underline{A}}]\!] \circ \langle\langle\rangle, \mathrm{id}_{\mathbf{S}_{\ell_2}[\![A]\!]}\rangle$.
  Now, $k^{\ell}_{[\![A]\!]} = k^{\ell_1}_{[\![A]\!]} \circ \mathbf{S}_{\ell_1} k^{\ell_2}_{[\![A]\!]} \circ \delta^{\ell_1,\ell_2}_{[\![A]\!]} \circ \mathbf{S}^{\ell \sqsubseteq \ell_1 \sqcup \ell_2}_{[\![A]\!]}$
  and $j^{\ell}_{\underline{A}} = \lambda x. j^{\ell_1}_{\underline{A}}\, ((\mathbf{lift}^{\ell_1} j^{\ell_2}_{\underline{A}})\, (\mathbf{fork}^{\ell_1,\ell_2}(\mathbf{up}^{\ell,\ell_1 \sqcup \ell_2} x)))$.
  Then,

$$
\begin{aligned}
&\Lambda^{-1}[\![\lambda x. j^{\ell_1}_{\underline{A}}\, ((\mathbf{lift}^{\ell_1} j^{\ell_2}_{\underline{A}})\, (\mathbf{fork}^{\ell_1,\ell_2}(\mathbf{up}^{\ell,\ell_1 \sqcup \ell_2} x)))]\!] \circ \langle\langle\rangle, \mathrm{id}_{\mathbf{S}_{\ell}[\![A]\!]}\rangle \\
=&[\![j^{\ell_1}_{\underline{A}}\, ((\mathbf{lift}^{\ell_1} j^{\ell_2}_{\underline{A}})\, (\mathbf{fork}^{\ell_1,\ell_2}(\mathbf{up}^{\ell,\ell_1 \sqcup \ell_2} x)))]\!] \circ \langle\langle\rangle, \mathrm{id}_{\mathbf{S}_{\ell}[\![A]\!]}\rangle \\
=&\mathrm{app} \circ \langle [\![j^{\ell_1}_{\underline{A}}]\!] \circ \langle\rangle, \mathrm{app} \circ \langle [\![\mathbf{lift}^{\ell_1} j^{\ell_2}_{\underline{A}}]\!] \circ \langle\rangle, [\![\mathbf{fork}^{\ell_1,\ell_2}(\mathbf{up}^{\ell,\ell_1 \sqcup \ell_2} x)]\!]\rangle\rangle \circ \langle\langle\rangle, \mathrm{id}_{\mathbf{S}_{\ell}[\![A]\!]}\rangle \\
=&\mathrm{app} \circ \langle [\![j^{\ell_1}_{\underline{A}}]\!] \circ \langle\rangle, \mathrm{app} \circ \langle \Lambda\Big(\mathbf{S}_{\ell_1}(\Lambda^{-1}[\![j^{\ell_2}_{\underline{A}}]\!]) \circ t^{\mathbf{S}_{\ell_1}}_{\top, \mathbf{S}_{\ell_2}[\![A]\!]}\Big) \circ \langle\rangle, \delta^{\ell_1,\ell_2}_{[\![A]\!]} \circ \mathbf{S}^{\ell \sqsubseteq \ell_1 \sqcup \ell_2}_{[\![A]\!]} \circ \pi_2\rangle\rangle \circ \langle\langle\rangle, \mathrm{id}_{\mathbf{S}_{\ell}[\![A]\!]}\rangle \\
=&\mathrm{app} \circ \langle [\![j^{\ell_1}_{\underline{A}}]\!] \circ \langle\rangle, \mathrm{app} \circ \langle \Lambda\Big(\mathbf{S}_{\ell_1}(\Lambda^{-1}[\![j^{\ell_2}_{\underline{A}}]\!] \circ \langle\langle\rangle, \mathrm{id}\rangle \circ \pi_2) \circ t^{\mathbf{S}_{\ell_1}}_{\top, \mathbf{S}_{\ell_2}[\![A]\!]}\Big) \circ \langle\rangle, \delta^{\ell_1,\ell_2}_{[\![A]\!]} \circ \mathbf{S}^{\ell \sqsubseteq \ell_1 \sqcup \ell_2}_{[\![A]\!]} \circ \pi_2\rangle\rangle \\
&\qquad \circ \langle\langle\rangle, \mathrm{id}_{\mathbf{S}_{\ell}[\![A]\!]}\rangle \\
=&\mathrm{app} \circ \langle [\![j^{\ell_1}_{\underline{A}}]\!] \circ \langle\rangle, \mathrm{app} \circ \langle \Lambda\Big(\mathbf{S}_{\ell_1} k^{\ell_2}_{[\![A]\!]} \circ \pi_2\Big) \circ \langle\rangle, \delta^{\ell_1,\ell_2}_{[\![A]\!]} \circ \mathbf{S}^{\ell \sqsubseteq \ell_1 \sqcup \ell_2}_{[\![A]\!]} \circ \pi_2\rangle\rangle \circ \langle\langle\rangle, \mathrm{id}_{\mathbf{S}_{\ell}[\![A]\!]}\rangle \\
&\qquad \text{[By IH and strength]} \\
=&\mathrm{app} \circ \langle [\![j^{\ell_1}_{\underline{A}}]\!] \circ \langle\rangle, \mathbf{S}_{\ell_1} k^{\ell_2}_{[\![A]\!]} \circ \pi_2 \circ (\mathrm{id} \times (\delta^{\ell_1,\ell_2}_{[\![A]\!]} \circ \mathbf{S}^{\ell \sqsubseteq \ell_1 \sqcup \ell_2}_{[\![A]\!]} \circ \pi_2)) \circ \langle\langle\rangle, \mathrm{id}\rangle\rangle \circ \langle\langle\rangle, \mathrm{id}\rangle \\
=&\mathrm{app} \circ \langle [\![j^{\ell_1}_{\underline{A}}]\!] \circ \langle\rangle, \mathbf{S}_{\ell_1} k^{\ell_2}_{[\![A]\!]} \circ \delta^{\ell_1,\ell_2}_{[\![A]\!]} \circ \mathbf{S}^{\ell \sqsubseteq \ell_1 \sqcup \ell_2}_{[\![A]\!]} \circ \pi_2 \circ \pi_2 \circ \langle\langle\rangle, \mathrm{id}\rangle\rangle \circ \langle\langle\rangle, \mathrm{id}\rangle \\
=&\mathrm{app} \circ \langle [\![j^{\ell_1}_{\underline{A}}]\!] \circ \langle\rangle, \mathbf{S}_{\ell_1} k^{\ell_2}_{[\![A]\!]} \circ \delta^{\ell_1,\ell_2}_{[\![A]\!]} \circ \mathbf{S}^{\ell \sqsubseteq \ell_1 \sqcup \ell_2}_{[\![A]\!]} \circ \pi_2\rangle \circ \langle\langle\rangle, \mathrm{id}\rangle \\
=&\mathrm{app} \circ ([\![j^{\ell_1}_{\underline{A}}]\!] \times \mathrm{id}_{\mathbf{S}_{\ell_1}[\![A]\!]}) \circ \langle\langle\rangle, \mathbf{S}_{\ell_1} k^{\ell_2}_{[\![A]\!]} \circ \delta^{\ell_1,\ell_2}_{[\![A]\!]} \circ \mathbf{S}^{\ell \sqsubseteq \ell_1 \sqcup \ell_2}_{[\![A]\!]} \circ \pi_2\rangle \circ \langle\langle\rangle, \mathrm{id}\rangle \\
=&\Lambda^{-1}[\![j^{\ell_1}_{\underline{A}}]\!] \circ \langle\langle\rangle, \mathrm{id}_{\mathbf{S}_{\ell_1}[\![A]\!]}\rangle \circ \pi_2 \circ \langle\langle\rangle, \mathbf{S}_{\ell_1} k^{\ell_2}_{[\![A]\!]} \circ \delta^{\ell_1,\ell_2}_{[\![A]\!]} \circ \mathbf{S}^{\ell \sqsubseteq \ell_1 \sqcup \ell_2}_{[\![A]\!]} \circ \pi_2\rangle \circ \langle\langle\rangle, \mathrm{id}\rangle \\
=&k^{\ell_1}_{[\![A]\!]} \circ \mathbf{S}_{\ell_1} k^{\ell_2}_{[\![A]\!]} \circ \delta^{\ell_1,\ell_2}_{[\![A]\!]} \circ \mathbf{S}^{\ell \sqsubseteq \ell_1 \sqcup \ell_2}_{[\![A]\!]} \text{ [By IH]} \\
=&k^{\ell}_{[\![A]\!]}
\end{aligned}
$$

□

Theorem A.21 (Theorem 2.16) Let $\mathcal{L}$ be any bounded join-semilattice, $\mathbb{C}$ be any bicartesian closed category and $\mathbf{S}$ be any strong monoidal functor from $\mathbf{C}(\mathcal{L})$ to $\mathbf{End}^{\mathrm{s}}_{\mathbb{C}}$. Now,

- If $\Gamma \vdash a : A$ in GMCC($\mathcal{L}$), then $[\![\overline{a}]\!]_{(\mathbb{C},\mathbf{S})} = [\![a]\!]_{(\mathbb{C},\mathbf{S})}$.
- If $\Gamma \vdash a : A$ in DCC$_e$($\mathcal{L}$), then $[\![\underline{a}]\!]_{(\mathbb{C},\mathbf{S})} = [\![a]\!]_{(\mathbb{C},\mathbf{S})}$.

Proof. Follows by theorems A.18 and A.19. □

Theorem A.22 (Theorem 2.17) If $\ell \sqsubseteq A$ in DCC_e, then $(\llbracket A \rrbracket, k^{\ell}_{\llbracket A \rrbracket})$ is an $\mathbf{S}_\ell$-algebra.
Further, if $\ell \sqsubseteq A$ and $\ell \sqsubseteq B$, then for any $f \in \text{Hom}_{\mathbb{C}}(\llbracket A \rrbracket, \llbracket B \rrbracket)$, f is an $\mathbf{S}_\ell$-algebra morphism.
Hence, the full subcategory of $\mathbb{C}$ with $\text{Obj} := \{\llbracket A \rrbracket \mid \ell \sqsubseteq A\}$ is also a full subcategory of the Eilenberg-Moore category, $\mathbb{C}^{\mathbf{S}_\ell}$.

Proof. Let $\ell \sqsubseteq A$. We show that $(\llbracket A \rrbracket, k^{\ell}_{\llbracket A \rrbracket})$ is an $(\mathbf{S}_\ell, \mathbf{S}^{\bot \sqsubseteq \ell} \circ \eta, \mu^{\ell,\ell})$-algebra.

We use the following shorthand: $\overline{\eta}_X := \mathbf{S}^{\bot \sqsubseteq \ell}_X \circ \eta_X$ and $\overline{\mu}_X := \mu^{\ell,\ell}_X$, where $X \in \text{Obj}(\mathbb{C})$.

We need to show: $k^{\ell}_{\llbracket A \rrbracket} \circ \overline{\eta}_{\llbracket A \rrbracket} = \text{id}_{\llbracket A \rrbracket}$. This follows by lemma A.12.

Next, we need to show: $k^{\ell}_{\llbracket A \rrbracket} \circ \mathbf{S}_\ell k^{\ell}_{\llbracket A \rrbracket} = k^{\ell}_{\llbracket A \rrbracket} \circ \overline{\mu}_{\llbracket A \rrbracket}$.

Now,

$$\begin{aligned}
k^{\ell}_{\llbracket A \rrbracket} \circ \overline{\eta}_{\llbracket A \rrbracket} &= \text{id}_{\llbracket A \rrbracket} \\
\text{or, } \mathbf{S}_\ell k^{\ell}_{\llbracket A \rrbracket} \circ \mathbf{S}_\ell \overline{\eta}_{\llbracket A \rrbracket} &= \text{id}_{\mathbf{S}_\ell \llbracket A \rrbracket} \\
\text{or, } k^{\ell}_{\llbracket A \rrbracket} \circ \mathbf{S}_\ell k^{\ell}_{\llbracket A \rrbracket} \circ \mathbf{S}_\ell \overline{\eta}_{\llbracket A \rrbracket} &= k^{\ell}_{\llbracket A \rrbracket} \\
\text{or, } k^{\ell}_{\llbracket A \rrbracket} \circ \mathbf{S}_\ell k^{\ell}_{\llbracket A \rrbracket} \circ \mathbf{S}_\ell \overline{\eta}_{\llbracket A \rrbracket} \circ \overline{\mu}_{\llbracket A \rrbracket} &= k^{\ell}_{\llbracket A \rrbracket} \circ \overline{\mu}_{\llbracket A \rrbracket} \\
\text{or, } k^{\ell}_{\llbracket A \rrbracket} \circ \mathbf{S}_\ell k^{\ell}_{\llbracket A \rrbracket} \circ \delta^{\ell,\ell}_{\llbracket A \rrbracket} \circ \mu^{\ell,\ell}_{\llbracket A \rrbracket} \circ \mathbf{S}_\ell \overline{\eta}_{\llbracket A \rrbracket} \circ \overline{\mu}_{\llbracket A \rrbracket} &= k^{\ell}_{\llbracket A \rrbracket} \circ \overline{\mu}_{\llbracket A \rrbracket} \\
\text{or, } k^{\ell}_{\llbracket A \rrbracket} \circ \mathbf{S}_\ell k^{\ell}_{\llbracket A \rrbracket} \circ \delta^{\ell,\ell}_{\llbracket A \rrbracket} \circ \overline{\mu}_{\llbracket A \rrbracket} \circ \mathbf{S}_\ell \overline{\eta}_{\llbracket A \rrbracket} \circ \overline{\mu}_{\llbracket A \rrbracket} &= k^{\ell}_{\llbracket A \rrbracket} \circ \overline{\mu}_{\llbracket A \rrbracket} \\
\text{or, } k^{\ell}_{\llbracket A \rrbracket} \circ \mathbf{S}_\ell k^{\ell}_{\llbracket A \rrbracket} \circ \delta^{\ell,\ell}_{\llbracket A \rrbracket} \circ \overline{\mu}_{\llbracket A \rrbracket} &= k^{\ell}_{\llbracket A \rrbracket} \circ \overline{\mu}_{\llbracket A \rrbracket} \; [\because (\mathbf{S}_\ell, \overline{\eta}, \overline{\mu}) \text{ is a monad}] \\
\text{or, } k^{\ell}_{\llbracket A \rrbracket} \circ \mathbf{S}_\ell k^{\ell}_{\llbracket A \rrbracket} \circ \delta^{\ell,\ell}_{\llbracket A \rrbracket} \circ \mu^{\ell,\ell}_{\llbracket A \rrbracket} &= k^{\ell}_{\llbracket A \rrbracket} \circ \overline{\mu}_{\llbracket A \rrbracket} \\
\text{or, } k^{\ell}_{\llbracket A \rrbracket} \circ \mathbf{S}_\ell k^{\ell}_{\llbracket A \rrbracket} &= k^{\ell}_{\llbracket A \rrbracket} \circ \overline{\mu}_{\llbracket A \rrbracket}
\end{aligned}$$

Hence, $(\llbracket A \rrbracket, k^{\ell}_{\llbracket A \rrbracket})$ is an $\mathbf{S}_\ell$-algebra.

Next, let $\ell \sqsubseteq A$ and $\ell \sqsubseteq B$ such that $f \in \text{Hom}_{\mathbb{C}}(\llbracket A \rrbracket, \llbracket B \rrbracket)$.

We show that f is an $\mathbf{S}_\ell$-algebra morphism.

Need to show: $f \circ k^{\ell}_{\llbracket A \rrbracket} = k^{\ell}_{\llbracket B \rrbracket} \circ \mathbf{S}_\ell f$.

Since $\overline{\eta}$ is a natural transformation, we have:

$$\begin{aligned}
\overline{\eta}_{\llbracket B \rrbracket} \circ f &= \mathbf{S}_\ell f \circ \overline{\eta}_{\llbracket A \rrbracket} \\
\text{or, } k^{\ell}_{\llbracket B \rrbracket} \circ \overline{\eta}_{\llbracket B \rrbracket} \circ f \circ k^{\ell}_{\llbracket A \rrbracket} &= k^{\ell}_{\llbracket B \rrbracket} \circ \mathbf{S}_\ell f \circ \overline{\eta}_{\llbracket A \rrbracket} \circ k^{\ell}_{\llbracket A \rrbracket} \\
\text{or, } \text{id}_{\llbracket B \rrbracket} \circ f \circ k^{\ell}_{\llbracket A \rrbracket} &= k^{\ell}_{\llbracket B \rrbracket} \circ \mathbf{S}_\ell f \circ \overline{\eta}_{\llbracket A \rrbracket} \circ k^{\ell}_{\llbracket A \rrbracket} \; [\text{By Lemma A.12}] \\
\text{or, } f \circ k^{\ell}_{\llbracket A \rrbracket} &= k^{\ell}_{\llbracket B \rrbracket} \circ \mathbf{S}_\ell f \circ \text{id}_{\mathbf{S}_\ell \llbracket A \rrbracket} \; [\text{By Lemma A.13}]
\end{aligned}$$

$$\text{or, } f \circ k^{\ell}_{[\![A]\!]} = k^{\ell}_{[\![B]\!]} \circ \mathbf{S}_{\ell} f$$

The final clause of the theorem follows. ☐

Theorem A.23 (Theorem 2.18) $[\![-]\!]_{(\mathbf{Set},\mathbf{S}^{\ell})}$, for any $\ell \in L$, is a computationally adequate interpretation of $\mathrm{DCC}_e(\mathcal{L})$.

Proof. For a bicartesian category $\mathbb{C}$, any strong monoidal functor from $\mathbf{C}(\mathcal{L})$ to $\mathbf{End}^{\mathrm{s}}_{\mathbb{C}}$ provides a computationally adequate interpretation of $\mathrm{DCC}_e(\mathcal{L})$, given the interpretation for ground types is injective. Now, with respect to $[\![-]\!]_{(\mathbf{Set},\mathbf{S}^{\ell})}$, the interpretation for ground types is injective. As such, to prove that $(\mathbf{Set}, \mathbf{S}^{\ell})$ is a computationally adequate interpretation of $\mathrm{DCC}_e(\mathcal{L})$, we just need to show that $\mathbf{S}^{\ell}$ is a strong monoidal functor from $\mathbf{C}(\mathcal{L})$ to $\mathbf{End}^{\mathrm{s}}_{\mathbb{C}}$.

Recall the definition of $\mathbf{S}^{\ell}$:

$$\mathbf{S}^{\ell}(\ell') = \begin{cases} \mathbf{Id}, & \text{if } \ell' \sqsubseteq \ell \\ *, & \text{otherwise} \end{cases} \qquad \mathbf{S}^{\ell}(\ell_1 \sqsubseteq \ell_2) = \begin{cases} \mathrm{id}, & \text{if } \ell_2 \sqsubseteq \ell \\ \langle\rangle, & \text{otherwise} \end{cases}$$

By this definition, $\mathbf{S}^{\ell}(\bot) = \mathbf{Id}$. Further, $\eta = \epsilon = \mathrm{id}_{\mathbf{Id}}$.

Now, for any $\ell_1, \ell_2 \in L$, there are two cases to consider:

- $\ell_1 \sqsubseteq \ell$ and $\ell_2 \sqsubseteq \ell$. Then, $\ell_1 \sqcup \ell_2 \sqsubseteq \ell$.
  So, $\mathbf{S}^{\ell}(\ell_1) = \mathbf{S}^{\ell}(\ell_2) = \mathbf{S}^{\ell}(\ell_1 \sqcup \ell_2) = \mathbf{Id}$.
  In this case, $\mu^{\ell_1,\ell_2} = \delta^{\ell_1,\ell_2} = \mathrm{id}_{\mathbf{Id}}$.
- $\neg(\ell_1 \sqsubseteq \ell)$ or $\neg(\ell_2 \sqsubseteq \ell)$. Then, $\neg(\ell_1 \sqcup \ell_2 \sqsubseteq \ell)$.
  So, $\mathbf{S}^{\ell}(\ell_1) = *$ or $\mathbf{S}^{\ell}(\ell_2) = *$. Hence, $\mathbf{S}^{\ell}(\ell_1) \circ \mathbf{S}^{\ell}(\ell_2) = *$. Further, $\mathbf{S}^{\ell}(\ell_1 \sqcup \ell_2) = *$.
  In this case, $\mu^{\ell_1,\ell_2} = \delta^{\ell_1,\ell_2} = \mathrm{id}_{*}$.

Hence, $\mathbf{S}^{\ell}$ is a strong (in fact a strict) monoidal functor from $\mathbf{C}(\mathcal{L})$ to $\mathbf{End}^{\mathrm{s}}_{\mathbb{C}}$. ☐

Lemma A.24 (Lemma 2.19) If $\ell'' \sqsubseteq A$ and $\neg(\ell'' \sqsubseteq \ell)$, then $[\![A]\!]_{(\mathbb{C},\mathbf{S}^{\ell})} \cong \top$.

Proof. By induction on $\ell'' \sqsubseteq A$.

- Rule Prot-Prod. Have: $\ell'' \sqsubseteq A \times B$, where $\ell'' \sqsubseteq A$ and $\ell'' \sqsubseteq B$. Further, $\neg(\ell'' \sqsubseteq \ell)$.
  Need to show: $[\![A \times B]\!]_{(\mathbb{C},\mathbf{S}^{\ell})} \cong \top$.
  By IH, $[\![A]\!]_{(\mathbb{C},\mathbf{S}^{\ell})} \cong \top$ and $[\![B]\!]_{(\mathbb{C},\mathbf{S}^{\ell})} \cong \top$.
  Therefore, $[\![A \times B]\!]_{(\mathbb{C},\mathbf{S}^{\ell})} = [\![A]\!]_{(\mathbb{C},\mathbf{S}^{\ell})} \times [\![B]\!]_{(\mathbb{C},\mathbf{S}^{\ell})} \cong \top \times \top \cong \top$.

- Rule Prot-Fun. Have: $\ell'' \sqsubseteq A \to B$, where $\ell'' \sqsubseteq B$. Further, $\neg(\ell'' \sqsubseteq \ell)$.
  Need to show: $[\![A \to B]\!]_{(\mathbb{C},\mathbf{S}^\ell)} \cong \top$.
  By IH, $[\![B]\!]_{(\mathbb{C},\mathbf{S}^\ell)} \cong \top$.
  Therefore, $[\![A \to B]\!]_{(\mathbb{C},\mathbf{S}^\ell)} = ([\![B]\!]_{(\mathbb{C},\mathbf{S}^\ell)})^{([\![A]\!]_{(\mathbb{C},\mathbf{S}^\ell)})} \cong \top^{([\![A]\!]_{(\mathbb{C},\mathbf{S}^\ell)})} \cong \top$.

- Rule Prot-Monad. Have: $\ell'' \sqsubseteq T_{\ell'}\ A$, where $\ell'' \sqsubseteq \ell'$. Further, $\neg(\ell'' \sqsubseteq \ell)$.
  Need to show: $[\![T_{\ell'}\ A]\!]_{(\mathbb{C},\mathbf{S}^\ell)} \cong \top$.
  Now, since $\ell'' \sqsubseteq \ell'$ and $\neg(\ell'' \sqsubseteq \ell)$, so $\neg(\ell' \sqsubseteq \ell)$.
  Then, $[\![T_{\ell'}\ A]\!]_{(\mathbb{C},\mathbf{S}^\ell)} = (\mathbf{S}^\ell(\ell'))[\![A]\!]_{(\mathbb{C},\mathbf{S}^\ell)} = (*)[\![A]\!]_{(\mathbb{C},\mathbf{S}^\ell)} = \top$.

- Rule Prot-Already. Have: $\ell'' \sqsubseteq T_{\ell'}\ A$, where $\ell'' \sqsubseteq A$. Further $\neg(\ell'' \sqsubseteq \ell)$.
  Need to show: $[\![T_{\ell'}\ A]\!]_{(\mathbb{C},\mathbf{S}^\ell)} \cong \top$.
  By IH, $[\![A]\!]_{(\mathbb{C},\mathbf{S}^\ell)} \cong \top$.
  Therefore, $[\![T_{\ell'}\ A]\!]_{(\mathbb{C},\mathbf{S}^\ell)} = (\mathbf{S}^\ell(\ell'))[\![A]\!]_{(\mathbb{C},\mathbf{S}^\ell)} \cong (\mathbf{S}^\ell(\ell'))(\top) \cong \top$.

- Rule Prot-Minimum. Have: $\bot \sqsubseteq A$.
  Further, $\neg(\bot \sqsubseteq \ell)$. But this is a contradiction because for any $\ell \in L$, $\bot \sqsubseteq \ell$.

- Rule Prot-Combine. Have: $\ell'' \sqsubseteq A$, where $\ell_1 \sqsubseteq A$ and $\ell_2 \sqsubseteq A$ and $\ell'' \sqsubseteq \ell_1 \sqcup \ell_2$. Further, $\neg(\ell'' \sqsubseteq \ell)$.
  Now, if both $\ell_1 \sqsubseteq \ell$ and $\ell_2 \sqsubseteq \ell$ hold, then $\ell'' \sqsubseteq \ell_1 \sqcup \ell_2 \sqsubseteq \ell$.
  Therefore, either $\neg(\ell_1 \sqsubseteq \ell)$ or $\neg(\ell_2 \sqsubseteq \ell)$.
  In either case, by IH, $[\![A]\!]_{(\mathbb{C},\mathbf{S}^\ell)} \cong \top$.

□

Theorem A.25 (Theorem 2.20) Let $\mathcal{L} = (L, \vee, \bot)$ be the parametrizing semilattice.

- Suppose $\ell \in L$ such that $\neg(\ell \sqsubseteq \bot)$. Let $\ell \sqsubseteq A$. Let $\varnothing \vdash f : A \to \mathbf{Bool}$ and $\varnothing \vdash a_1 : A$ and $\varnothing \vdash a_2 : A$. Then, $\vdash f\ a_1 \leadsto^* v$ if and only if $\vdash f\ a_2 \leadsto^* v$, where v is a value of type **Bool**.

- Suppose $\ell, \ell' \in L$ such that $\neg(\ell \sqsubseteq \ell')$. Let $\ell \sqsubseteq A$. Let $\varnothing \vdash f : A \to T_{\ell'}\,\mathbf{Bool}$ and $\varnothing \vdash a_1 : A$ and $\varnothing \vdash a_2 : A$. Then, $\vdash f\ a_1 \leadsto^* v$ if and only if $\vdash f\ a_2 \leadsto^* v$, where v is a value of type $T_{\ell'}\,\mathbf{Bool}$.

Proof. For the first part, note that $(\mathbf{Set}, \mathbf{S}^\bot)$ gives us a computationally adequate model of $\mathrm{DCC}_e(\mathcal{L})$ (by Theorem A.23).
Next, since $\ell \sqsubseteq A$ and $\neg(\ell \sqsubseteq \bot)$, by Lemma A.24, $[\![A]\!]_{(\mathbb{C},\mathbf{S}^\bot)} \cong \top$.
Therefore, $[\![f\ a_1]\!]_{(\mathbf{Set},\mathbf{S}^\bot)} = [\![f\ a_2]\!]_{(\mathbf{Set},\mathbf{S}^\bot)} = \mathrm{app} \circ \langle [\![f]\!]_{(\mathbf{Set},\mathbf{S}^\bot)}, \langle\rangle\rangle$.
Then, by Theorem A.17, $\vdash f\ a_1 \leadsto^* v$ if and only if $\vdash f\ a_2 \leadsto^* v$.

For the second part, note that $(\mathbf{Set}, \mathbf{S}^{\ell'})$ gives us a computationally adequate model of $\mathrm{DCC}_e(\mathcal{L})$ (by Theorem A.23). Further, note that $\overline{\eta} \triangleq (\mathbf{S}^{\ell'}(\bot \sqsubseteq \ell')) \circ \eta = \mathrm{id}_{\mathbf{Id}} \circ \mathrm{id}_{\mathbf{Id}} = \mathrm{id}_{\mathbf{Id}}$. So, the morphisms $\overline{\eta}_X$ are mono for any $X \in \mathrm{Obj}(\mathbb{C})$.

Next, since $\ell \sqsubseteq A$ and $\neg(\ell \sqsubseteq \ell')$, by Lemma A.24, $[\![A]\!]_{(\mathbb{C},\mathbf{S}^{\ell'})} \cong \top$.
Therefore, $[\![f\ a_1]\!]_{(\mathbf{Set},\mathbf{S}^{\ell'})} = [\![f\ a_2]\!]_{(\mathbf{Set},\mathbf{S}^{\ell'})} = \mathrm{app} \circ \langle [\![f]\!]_{(\mathbf{Set},\mathbf{S}^{\ell'})}, \langle\rangle \rangle$.
Then, by Theorem A.17, $\vdash f\ a_1 \leadsto^* v$ if and only if $\vdash f\ a_2 \leadsto^* v$. ☐

A.8 Proofs of Lemmas/Theorems Stated in Section 2.9

Theorem A.26 (Theorem 2.21) If $\Gamma \vdash a :^n A$ in λ°, then $[\![a]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], \mathbf{S}_n [\![A]\!])$. Further, if $\Gamma \vdash a_1 :^n A$ and $\Gamma \vdash a_2 :^n A$ such that $a_1 \equiv a_2$ in λ°, then $[\![a_1]\!] = [\![a_2]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], \mathbf{S}_n [\![A]\!])$.

Proof. The first part follows by induction on the typing derivation. Most of the cases are straightforward, given the detailed interpretation in Section 2.9.2. However, we elaborate on the case of **case**-expressions.

- Rule LC-Case. Have: $\Gamma \vdash \mathbf{case}\ a\ \mathbf{of}\ b_1\ ;\ b_2 :^n B$ where $\Gamma \vdash a :^n A_1 + A_2$ and $\Gamma \vdash b_1 :^n A_1 \to B$ and $\Gamma \vdash b_2 :^n A_2 \to B$.
  Need to show: $[\![\mathbf{case}\ a\ \mathbf{of}\ b_1\ ;\ b_2]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], \mathbf{S}_n [\![B]\!])$.
  By IH, $[\![a]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], \mathbf{S}_n([\![A_1]\!] + [\![A_2]\!]))$ and
  $[\![b_1]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], \mathbf{S}_n([\![B]\!]^{[\![A_1]\!]}))$ and $[\![b_2]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], \mathbf{S}_n([\![B]\!]^{[\![A_2]\!]}))$.
  Now,

  $$\begin{aligned}[\![\mathbf{case}\ a\ \mathbf{of}\ b_1\ ;\ b_2]\!] = [\![\Gamma]\!] &\xrightarrow{\langle\langle [\![b_1]\!], [\![b_2]\!]\rangle, [\![a]\!]\rangle} (\mathbf{S}_n([\![B]\!]^{[\![A_1]\!]}) \times \mathbf{S}_n([\![B]\!]^{[\![A_2]\!]})) \times \mathbf{S}_n([\![A_1]\!] + [\![A_2]\!]) \text{ [By IH]} \\ &\xrightarrow{p^{-1}_{[\![B]\!]^{[\![A_1]\!]}, [\![B]\!]^{[\![A_2]\!]}} \times \mathrm{id}} \mathbf{S}_n([\![B]\!]^{[\![A_1]\!]} \times [\![B]\!]^{[\![A_2]\!]}) \times \mathbf{S}_n([\![A_1]\!] + [\![A_2]\!]) \\ &\xrightarrow{p^{-1}_{[\![B]\!]^{[\![A_1]\!]} \times [\![B]\!]^{[\![A_2]\!]}, [\![A_1]\!] + [\![A_2]\!]}} \mathbf{S}_n(([\![B]\!]^{[\![A_1]\!]} \times [\![B]\!]^{[\![A_2]\!]}) \times ([\![A_1]\!] + [\![A_2]\!])) \\ &\xrightarrow{\mathbf{S}_n(h \times \mathrm{id})} \mathbf{S}_n([\![B]\!]^{[\![A_1]\!] + [\![A_2]\!]} \times ([\![A_1]\!] + [\![A_2]\!])) \xrightarrow{\mathbf{S}_n \mathrm{app}} \mathbf{S}_n [\![B]\!]\end{aligned}$$

  where $h = \Lambda\Big([\Lambda^{-1}\pi_1 \circ \langle \pi_2, \pi_1 \rangle, \Lambda^{-1}\pi_2 \circ \langle \pi_2, \pi_1 \rangle] \circ \Lambda^{-1}[\Lambda i_1, \Lambda i_2] \circ \langle \pi_2, \pi_1 \rangle\Big)$.

  Note that $h : [\![B]\!]^{[\![A_1]\!]} \times [\![B]\!]^{[\![A_2]\!]} \to [\![B]\!]^{[\![A_1]\!] + [\![A_2]\!]}$ is an isomorphism, where

  $$h^{-1} = \langle [\![B]\!]^{i_1}, [\![B]\!]^{i_2} \rangle : [\![B]\!]^{[\![A_1]\!] + [\![A_2]\!]} \to [\![B]\!]^{[\![A_1]\!]} \times [\![B]\!]^{[\![A_2]\!]}$$

  Recall that for $X, Y, Z \in \mathrm{Obj}(\mathbb{C})$ and $f \in \mathrm{Hom}_{\mathbb{C}}(Y, Z)$,

  $$X^f \triangleq \Lambda\Big(X^Z \times Y \xrightarrow{\mathrm{id} \times f} X^Z \times Z \xrightarrow{\mathrm{app}} X\Big) \in \mathrm{Hom}_{\mathbb{C}}(X^Z, X^Y)$$

For the second part, we invert the equality judgement.

- $(\lambda x : A.b)\ a \equiv b\{a/x\}$.
  Now,

$$\begin{aligned}
&[\![(\lambda x : A.b)\ a]\!] \\
=&\mathrm{app} \circ \langle q_{[\![A]\!],[\![B]\!]} \circ [\![\lambda x : A.b]\!], [\![a]\!]\rangle \\
=&\mathrm{app} \circ \langle q_{[\![A]\!],[\![B]\!]} \circ q^{-1}_{[\![A]\!],[\![B]\!]} \circ \Lambda[\![b]\!], [\![a]\!]\rangle \\
=&\mathrm{app} \circ \langle \Lambda[\![b]\!], [\![a]\!]\rangle = [\![b]\!] \circ \langle \mathrm{id}, [\![a]\!]\rangle = [\![b\{a/x\}]\!].
\end{aligned}$$

- $b \equiv \lambda x : A.b\ x$.
  Now,

$$\begin{aligned}
&[\![\lambda x : A.b\ x]\!] \\
=&q^{-1}_{[\![A]\!],[\![B]\!]} \circ \Lambda[\![b\ x]\!] \\
=&q^{-1}_{[\![A]\!],[\![B]\!]} \circ \Lambda\Big(\mathrm{app} \circ \langle q_{[\![A]\!],[\![B]\!]} \circ [\![b]\!] \circ \pi_1, \pi_2\rangle\Big) \\
=&q^{-1}_{[\![A]\!],[\![B]\!]} \circ q_{[\![A]\!],[\![B]\!]} \circ [\![b]\!] = [\![b]\!].
\end{aligned}$$

- $\mathbf{prev}\ (\mathbf{next}\ a) \equiv a$.
  Now, $[\![\mathbf{prev}\ (\mathbf{next}\ a)]\!] = \mu^{n, succ\, 0}_{[\![A]\!]} \circ \delta^{n, succ\, 0}_{[\![A]\!]} \circ [\![a]\!] = [\![a]\!]$.
- $\mathbf{next}\ (\mathbf{prev}\ a) \equiv a$.
  $[\![\mathbf{next}\ (\mathbf{prev}\ a)]\!] = \delta^{n, succ\, 0}_{[\![A]\!]} \circ \mu^{n, succ\, 0}_{[\![A]\!]} \circ [\![a]\!] = [\![a]\!]$.
- $\mathbf{case}\,\mathbf{inj}_1\ a_1\ \mathbf{of}\ b_1\ ;\ b_2 \equiv b_1\ a_1$.
  Now,

$$\begin{aligned}
&[\![\mathbf{case}\,\mathbf{inj}_1\ a_1\ \mathbf{of}\ b_1\ ;\ b_2]\!] \\
=&\mathbf{S}_n\mathrm{app} \circ \mathbf{S}_n(h \times \mathrm{id}) \circ p^{-1}_{[\![B]\!]^{[\![A_1]\!]} \times [\![B]\!]^{[\![A_2]\!]}, [\![A_1]\!]+[\![A_2]\!]} \circ p^{-1}_{[\![B]\!]^{[\![A_1]\!]}, [\![B]\!]^{[\![A_2]\!]}} \times \mathrm{id} \circ \langle\langle[\![b_1]\!], [\![b_2]\!]\rangle, \mathbf{S}_n i_1 \circ [\![a_1]\!]\rangle \\
=&\mathbf{S}_n([\Lambda^{-1}\pi_1 \circ \langle\pi_2, \pi_1\rangle, \Lambda^{-1}\pi_2 \circ \langle\pi_2, \pi_1\rangle] \circ \Lambda^{-1}[\Lambda i_1, \Lambda i_2] \circ \langle\pi_2, \pi_1\rangle) \circ p^{-1} \circ (p^{-1} \times \mathrm{id}) \circ \\
&\qquad \langle\langle[\![b_1]\!], [\![b_2]\!]\rangle, \mathbf{S}_n i_1 \circ [\![a_1]\!]\rangle \\
=&\mathbf{S}_n([\Lambda^{-1}\pi_1 \circ \langle\pi_2, \pi_1\rangle, \Lambda^{-1}\pi_2 \circ \langle\pi_2, \pi_1\rangle] \circ \Lambda^{-1}[\Lambda i_1, \Lambda i_2]) \circ p^{-1} \circ \langle\pi_2, \pi_1\rangle \circ (p^{-1} \times \mathrm{id}) \circ \\
&\qquad \langle\langle[\![b_1]\!], [\![b_2]\!]\rangle, \mathbf{S}_n i_1 \circ [\![a_1]\!]\rangle \\
=&\mathbf{S}_n([\Lambda^{-1}\pi_1 \circ \langle\pi_2, \pi_1\rangle, \Lambda^{-1}\pi_2 \circ \langle\pi_2, \pi_1\rangle] \circ \Lambda^{-1}[\Lambda i_1, \Lambda i_2]) \circ p^{-1} \circ \langle\mathbf{S}_n i_1 \circ [\![a_1]\!], p^{-1} \circ \langle[\![b_1]\!], [\![b_2]\!]\rangle\rangle \\
=&\mathbf{S}_n([\Lambda^{-1}\pi_1 \circ \langle\pi_2, \pi_1\rangle, \Lambda^{-1}\pi_2 \circ \langle\pi_2, \pi_1\rangle] \circ \Lambda^{-1}[\Lambda i_1, \Lambda i_2]) \circ p^{-1} \circ \mathbf{S}_n i_1 \times \mathrm{id} \circ \langle[\![a_1]\!], p^{-1} \circ \langle[\![b_1]\!], [\![b_2]\!]\rangle\rangle \\
=&\mathbf{S}_n([\Lambda^{-1}\pi_1 \circ \langle\pi_2, \pi_1\rangle, \Lambda^{-1}\pi_2 \circ \langle\pi_2, \pi_1\rangle] \circ \Lambda^{-1}[\Lambda i_1, \Lambda i_2]) \circ \mathbf{S}_n(i_1 \times \mathrm{id}) \circ p^{-1} \circ \\
&\qquad \langle[\![a_1]\!], p^{-1} \circ \langle[\![b_1]\!], [\![b_2]\!]\rangle\rangle \\
=&\mathbf{S}_n[\Lambda^{-1}\pi_1 \circ \langle\pi_2, \pi_1\rangle, \Lambda^{-1}\pi_2 \circ \langle\pi_2, \pi_1\rangle] \circ \mathbf{S}_n\Lambda^{-1}[\Lambda i_1, \Lambda i_2] \circ \mathbf{S}_n(i_1 \times \mathrm{id}) \circ p^{-1} \circ \\
&\qquad \langle[\![a_1]\!], p^{-1} \circ \langle[\![b_1]\!], [\![b_2]\!]\rangle\rangle \\
=&\mathbf{S}_n[\Lambda^{-1}\pi_1 \circ \langle\pi_2, \pi_1\rangle, \Lambda^{-1}\pi_2 \circ \langle\pi_2, \pi_1\rangle] \circ \mathbf{S}_n i_1 \circ p^{-1} \circ \langle[\![a_1]\!], p^{-1} \circ \langle[\![b_1]\!], [\![b_2]\!]\rangle\rangle
\end{aligned}$$

$$
\begin{aligned}
&=\mathbf{S}_n\Lambda^{-1}\pi_1 \circ \mathbf{S}_n\langle \pi_2,\pi_1\rangle \circ p^{-1} \circ \langle [\![a_1]\!], p^{-1}\circ\langle [\![b_1]\!],[\![b_2]\!]\rangle\rangle\\
&=\mathbf{S}_n\Lambda^{-1}\pi_1 \circ p^{-1} \circ \langle \pi_2,\pi_1\rangle \circ \langle [\![a_1]\!], p^{-1}\circ\langle [\![b_1]\!],[\![b_2]\!]\rangle\rangle\\
&=\mathbf{S}_n\mathrm{app}\circ \mathbf{S}_n(\pi_1\times \mathrm{id})\circ p^{-1}\circ\langle p^{-1}\circ\langle [\![b_1]\!],[\![b_2]\!]\rangle,[\![a_1]\!]\rangle\\
&=\mathbf{S}_n\mathrm{app}\circ p^{-1}\circ \mathbf{S}_n\pi_1\times \mathrm{id}\circ\langle p^{-1}\circ\langle [\![b_1]\!],[\![b_2]\!]\rangle,[\![a_1]\!]\rangle\\
&=\mathbf{S}_n\mathrm{app}\circ p^{-1}\circ \mathbf{S}_n\pi_1\times \mathrm{id}\circ p^{-1}\times\mathrm{id}\circ\langle \langle [\![b_1]\!],[\![b_2]\!]\rangle,[\![a_1]\!]\rangle\\
&=\mathbf{S}_n\mathrm{app}\circ p^{-1}\circ \pi_1\times \mathrm{id}\circ p\times\mathrm{id}\circ p^{-1}\times\mathrm{id}\circ\langle \langle [\![b_1]\!],[\![b_2]\!]\rangle,[\![a_1]\!]\rangle\\
&=\mathbf{S}_n\mathrm{app}\circ p^{-1}\circ\langle [\![b_1]\!],[\![a_1]\!]\rangle\\
&=\mathrm{app}\circ\Lambda(\mathbf{S}_n\mathrm{app}\circ p^{-1})\times\mathrm{id}\circ\langle [\![b_1]\!],[\![a_1]\!]\rangle=\mathrm{app}\circ\langle q\circ[\![b_1]\!],[\![a_1]\!]\rangle=[\![b_1\ a_1]\!]
\end{aligned}
$$

- $b\ a \equiv$ **case** a **of** $\lambda x_1.b\ (\mathbf{inj}_1\ x_1)\ ;\ \lambda x_2.b\ (\mathbf{inj}_2\ x_2)$.

  Note that

$$
\begin{aligned}
&[\![\lambda x_1.b\ (\mathbf{inj}_1\ x_1)]\!]\\
&=q^{-1}_{[\![A_1]\!],[\![B]\!]}\circ\Lambda\Big(\mathrm{app}\circ\langle q_{[\![A_1]\!]+[\![A_2]\!],[\![B]\!]}\circ[\![b]\!]\circ\pi_1,\mathbf{S}_n i_1\circ\pi_2\rangle\Big)\\
&=q^{-1}_{[\![A_1]\!],[\![B]\!]}\circ\Lambda\Big(\mathrm{app}\circ(\mathrm{id}\times\mathbf{S}_n i_1)\circ(q_{[\![A_1]\!]+[\![A_2]\!],[\![B]\!]}\circ[\![b]\!])\times\mathrm{id}\Big)\\
&=q^{-1}_{[\![A_1]\!],[\![B]\!]}\circ\Lambda(\mathrm{app}\circ(\mathrm{id}\times\mathbf{S}_n i_1))\circ q_{[\![A_1]\!]+[\![A_2]\!],[\![B]\!]}\circ[\![b]\!]\\
&=q^{-1}_{[\![A_1]\!],[\![B]\!]}\circ(\mathbf{S}_n[\![B]\!])^{\mathbf{S}_n i_1}\circ q_{[\![A_1]\!]+[\![A_2]\!],[\![B]\!]}\circ[\![b]\!]\\
&=\mathbf{S}_n[\![B]\!]^{i_1}\circ q^{-1}_{[\![A_1]\!]+[\![A_2]\!],[\![B]\!]}\circ q_{[\![A_1]\!]+[\![A_2]\!],[\![B]\!]}\circ[\![b]\!]=\mathbf{S}_n[\![B]\!]^{i_1}\circ[\![b]\!]
\end{aligned}
$$

  Similarly, $[\![\lambda x_2.b\ (\mathbf{inj}_2\ x_2)]\!]=\mathbf{S}_n[\![B]\!]^{i_2}\circ[\![b]\!]$.

  Now,

$$
\begin{aligned}
&[\![\mathbf{case}\ a\ \mathbf{of}\ \lambda x_1.b\ (\mathbf{inj}_1\ x_1)\ ;\ \lambda x_2.b\ (\mathbf{inj}_2\ x_2)]\!]\\
&=\mathbf{S}_n\mathrm{app}\circ\mathbf{S}_n(h\times\mathrm{id})\circ p^{-1}_{[\![B]\!]^{[\![A_1]\!]}\times[\![B]\!]^{[\![A_2]\!]},[\![A_1]\!]+[\![A_2]\!]}\circ(p^{-1}_{[\![B]\!]^{[\![A_1]\!]},[\![B]\!]^{[\![A_2]\!]}}\times\mathrm{id})\circ\\
&\qquad\qquad\langle\langle\mathbf{S}_n[\![B]\!]^{i_1},\mathbf{S}_n[\![B]\!]^{i_1}\rangle\circ[\![b]\!],[\![a]\!]\rangle\\
&=\mathbf{S}_n\mathrm{app}\circ\mathbf{S}_n(h\times\mathrm{id})\circ p^{-1}\circ p^{-1}\times\mathrm{id}\circ\langle\mathbf{S}_n[\![B]\!]^{i_1},\mathbf{S}_n[\![B]\!]^{i_1}\rangle\times\mathrm{id}\circ\langle[\![b]\!],[\![a]\!]\rangle\\
&=\mathbf{S}_n\mathrm{app}\circ\mathbf{S}_n(h\times\mathrm{id})\circ p^{-1}\circ\mathbf{S}_n\langle[\![B]\!]^{i_1},[\![B]\!]^{i_1}\rangle\times\mathrm{id}\circ\langle[\![b]\!],[\![a]\!]\rangle\\
&=\mathbf{S}_n\mathrm{app}\circ\mathbf{S}_n(h\times\mathrm{id})\circ\mathbf{S}_n(\langle[\![B]\!]^{i_1},[\![B]\!]^{i_1}\rangle\times\mathrm{id})\circ p^{-1}\circ\langle[\![b]\!],[\![a]\!]\rangle\\
&=\mathbf{S}_n(\mathrm{app}\circ(h\times\mathrm{id})\circ(h^{-1}\times\mathrm{id}))\circ p^{-1}\circ\langle[\![b]\!],[\![a]\!]\rangle\\
&=\mathbf{S}_n\mathrm{app}\circ p^{-1}\circ\langle[\![b]\!],[\![a]\!]\rangle=\mathrm{app}\circ\Lambda(\mathbf{S}_n\mathrm{app}\circ p^{-1})\times\mathrm{id}\circ\langle[\![b]\!],[\![a]\!]\rangle=[\![b\ a]\!].
\end{aligned}
$$

□

Theorem A.27 (Theorem 2.22) Let $\Gamma\vdash b :^0$ **Bool** and v be a value of type **Bool**. If $[\![b]\!]=[\![v]\!]$ then $\vdash b\longmapsto^* v$.

Proof. Let $\Gamma \vdash b :^0 \mathbf{Bool}$ and v be a boolean value such that $[\![b]\!]_{(\mathbb{C},\mathbf{S})} = [\![v]\!]_{(\mathbb{C},\mathbf{S})}$.
We show that $\vdash b \longmapsto^* v$.

First note that λ° is strongly normalizing with respect to the reduction relation, $\longmapsto^*$. Further, λ° is also type sound with respect to this reduction relation.

Therefore, given $\Gamma \vdash b :^0 \mathbf{Bool}$, we know that there exists a value v_0 such that $\Gamma \vdash v_0 :^0 \mathbf{Bool}$ and $\vdash b \longmapsto^* v_0$.

Next, by Theorem A.26, $[\![b]\!]_{(\mathbb{C},\mathbf{S})} = [\![v_0]\!]_{(\mathbb{C},\mathbf{S})}$.

Since $[\![b]\!]_{(\mathbb{C},\mathbf{S})} = [\![v]\!]_{(\mathbb{C},\mathbf{S})}$ (given) and $[\![b]\!]_{(\mathbb{C},\mathbf{S})} = [\![v_0]\!]_{(\mathbb{C},\mathbf{S})}$ (above), therefore, $[\![v]\!]_{(\mathbb{C},\mathbf{S})} = [\![v_0]\!]_{(\mathbb{C},\mathbf{S})}$.

By injectivity, $v = v_0$.

Thus, $\vdash b \longmapsto^* v$. ☐

Theorem A.28 (Theorem 2.23) $[\![\text{-}]\!]_{(\mathbf{Set},\mathbf{S}^0)}$ is a computationally adequate interpretation of λ°.

Proof. For a bicartesian category $\mathbb{C}$, any strong monoidal functor from $\mathbf{C}(\mathcal{N})$ to $\mathbf{End}_{\mathbb{C}}^{cc}$ provides a computationally adequate interpretation of λ°, given the interpretation for ground types is injective. Now, with respect to $[\![\text{-}]\!]_{(\mathbf{Set},\mathbf{S}^0)}$, the interpretation for ground types is injective. As such, to prove that $(\mathbf{Set}, \mathbf{S}^0)$ is a computationally adequate interpretation of λ°, we just need to show that $\mathbf{S}^0$ is a strong monoidal functor from $\mathbf{C}(\mathcal{N})$ to $\mathbf{End}_{\mathbb{C}}^{cc}$.

Recall the definition of $\mathbf{S}^0$:

$$\mathbf{S}^0(n) = \begin{cases} \mathbf{Id} & \text{if } n = 0 \\ * & \text{otherwise} \end{cases}$$

By this definition, $\mathbf{S}^0(0) = \mathbf{Id}$. Further, $\eta = \epsilon = \mathrm{id}_{\mathbf{Id}}$. Now for any $n_1, n_2 \in \mathbb{N}$, there are two cases to consider:

- $n_1 = n_2 = 0$. Then $n_1 + n_2 = 0$.
  So, $\mathbf{S}^0(n_1) = \mathbf{S}^0(n_2) = \mathbf{S}^0(n_1 + n_2) = \mathbf{Id}$.
  In this case, $\mu^{n_1,n_2} = \delta^{n_1,n_2} = \mathrm{id}_{\mathbf{Id}}$.

- $n_1 \neq 0$ or $n_2 \neq 0$. Then, $n_1 + n_2 \neq 0$.
  So, $\mathbf{S}^0(n_1) = *$ or $\mathbf{S}^0(n_2) = *$. Hence, $\mathbf{S}^0(n_1) \circ \mathbf{S}^0(n_2) = *$.
  Further, $\mathbf{S}^0(n_1 + n_2) = *$.
  In this case, $\mu^{n_1,n_2} = \delta^{n_1,n_2} = \mathrm{id}_*$.

Hence, $\mathbf{S}^0$ is a strong (in fact a strict) monoidal functor from $\mathbf{C}(\mathcal{N})$ to $\mathbf{End}_{\mathbb{C}}^{cc}$. ☐

Theorem A.29 (Theorem 2.24) Let $\varnothing \vdash f :^0 \bigcirc A \to \mathbf{Bool}$ and $\varnothing \vdash b_1 :^0 \bigcirc A$ and $\varnothing \vdash b_2 :^0 \bigcirc A$. Then, $\vdash f\ b_1 \longmapsto^* v$ if and only if $\vdash f\ b_2 \longmapsto^* v$, where v is a value of type $\mathbf{Bool}$.

Proof. First note that $(\mathbf{Set}, \mathbf{S}^0)$ gives us a computationally adequate model of λ° (by Theorem A.28).
Next, $[\![\bigcirc A]\!]_{(\mathbb{C},\mathbf{S}^0)} = (\mathbf{S}^0(succ\, 0))[\![A]\!]_{(\mathbb{C},\mathbf{S}^0)} = (*)[\![A]\!]_{(\mathbb{C},\mathbf{S}^0)} = \top$.
Therefore, $[\![f\ a_1]\!]_{(\mathbf{Set},\mathbf{S}^0)} = [\![f\ a_2]\!]_{(\mathbf{Set},\mathbf{S}^0)} = \mathrm{app} \circ \langle q \circ [\![f]\!]_{(\mathbf{Set},\mathbf{S}^0)}, \langle\rangle\rangle$.
Then, by Theorem A.27, $\vdash f\ a_1 \longmapsto^* v$ if and only if $\vdash f\ a_2 \longmapsto^* v$. □

A.9 Proofs of Lemmas/Theorems Stated in Section 2.10

Lemma A.30 (Lemma 2.25) Let $\mathcal{M}$ be any preordered monoid. Then, in $\mathrm{GMCC_e}(\mathcal{M})$,

- The types $S_m\ \mathbf{Unit}$ and $\mathbf{Unit}$ are isomorphic.
- The types $S_m\ (A_1 \times A_2)$ and $S_m\ A_1 \times S_m\ A_2$, for all types A_1 and A_2, are isomorphic.
- The types $S_m\ (A \to B)$ and $S_m\ A \to S_m\ B$, for all types A and B, are isomorphic.

Proof. For any two types, A and B, to show that $A \cong B$, we need to provide terms $\varnothing \vdash f :^{m'} A \to B$ and $\varnothing \vdash g :^{m'} B \to A$ such that $x :^{m'} A \vdash g\ (f\ x) \equiv x :^{m'} A$ and $y :^{m'} B \vdash f\ (g\ y) \equiv y :^{m'} B$, for any $m' \in M$. Note that the judgement $\Gamma \vdash a_1 \equiv a_2 :^m A$ is a shorthand for the judgements $\Gamma \vdash a_1 :^m A$ and $\Gamma \vdash a_2 :^m A$ and $a_1 \equiv a_2$.

- Need to show: $S_m\ \mathbf{Unit} \cong \mathbf{Unit}$.
  We have:

$$\frac{x :^{m'} S_m\ \mathbf{Unit} \vdash \mathbf{unit} :^{m'} \mathbf{Unit}}{\varnothing \vdash \lambda x.\mathbf{unit} :^{m'} S_m\ \mathbf{Unit} \to \mathbf{Unit}} \qquad \frac{\dfrac{y :^{m'} \mathbf{Unit} \vdash \mathbf{unit} :^{m' \cdot m} \mathbf{Unit}}{y :^{m'} \mathbf{Unit} \vdash \mathbf{split}^m \mathbf{unit} :^{m'} S_m\ \mathbf{Unit}}}{\varnothing \vdash \lambda y.\mathbf{split}^m \mathbf{unit} :^{m'} \mathbf{Unit} \to S_m\ \mathbf{Unit}}$$

  Say, $f_1 \triangleq \lambda x.\mathbf{unit}$ and $g_1 \triangleq \lambda y.\mathbf{split}^m \mathbf{unit}$.
  Next,

$$\frac{\dfrac{x :^{m'} S_m\ \mathbf{Unit} \vdash x :^{m'} S_m\ \mathbf{Unit}}{x :^{m'} S_m\ \mathbf{Unit} \vdash \mathbf{merge}^m x :^{m' \cdot m} \mathbf{Unit}}}{x :^{m'} S_m\ \mathbf{Unit} \vdash \mathbf{unit} \equiv \mathbf{merge}^m x :^{m' \cdot m} \mathbf{Unit}}$$

  So, given $x :^{m'} S_m\ \mathbf{Unit}$, we have, $g_1\ (f_1\ x) \equiv \mathbf{split}^m \mathbf{unit} \equiv \mathbf{split}^m (\mathbf{merge}^m x) \equiv x$.
  And, given $y :^{m'} \mathbf{Unit}$, we have, $f_1\ (g_1\ y) \equiv \mathbf{unit} \equiv y$.

- Need to show: $S_m\ (A_1 \times A_2) \cong S_m\ A_1 \times S_m\ A_2$.
  Let $C \triangleq S_m\ (A_1 \times A_2)$ and $D \triangleq S_m\ A_1 \times S_m\ A_2$.

Now, we have:

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{x :^{m'} C \vdash x :^{m'} S_m\ (A_1 \times A_2)}{x :^{m'} C \vdash \mathbf{merge}^m x :^{m' \cdot m} A_1 \times A_2}}{x :^{m'} C \vdash \mathbf{proj_1}\ (\mathbf{merge}^m x) :^{m' \cdot m} A_1}}{x :^{m'} C \vdash \mathbf{split}^m(\mathbf{proj_1}\ (\mathbf{merge}^m x)) :^{m'} S_m\ A_1}}{x :^{m'} C \vdash (\mathbf{split}^m(\mathbf{proj_1}\ (\mathbf{merge}^m x)), \mathbf{split}^m(\mathbf{proj_2}\ (\mathbf{merge}^m x))) :^{m'} S_m\ A_1 \times S_m\ A_2}}{\varnothing \vdash \lambda x.(\mathbf{split}^m(\mathbf{proj_1}\ (\mathbf{merge}^m x)), \mathbf{split}^m(\mathbf{proj_2}\ (\mathbf{merge}^m x))) :^{m'} C \to D}$$

and

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{y :^{m'} D \vdash y :^{m'} S_m\ A_1 \times S_m\ A_2}{y :^{m'} D \vdash \mathbf{proj_1}\ y :^{m'} S_m\ A_1}}{y :^{m'} D \vdash \mathbf{merge}^m(\mathbf{proj_1}\ y) :^{m' \cdot m} A_1}}{y :^{m'} D \vdash (\mathbf{merge}^m(\mathbf{proj_1}\ y), \mathbf{merge}^m(\mathbf{proj_2}\ y)) :^{m' \cdot m} A_1 \times A_2}}{y :^{m'} D \vdash \mathbf{split}^m(\mathbf{merge}^m(\mathbf{proj_1}\ y), \mathbf{merge}^m(\mathbf{proj_2}\ y)) :^{m'} S_m\ (A_1 \times A_2)}}{\varnothing \vdash \lambda y.\mathbf{split}^m(\mathbf{merge}^m(\mathbf{proj_1}\ y), \mathbf{merge}^m(\mathbf{proj_2}\ y)) :^{m'} D \to C}$$

Say,

$$\begin{aligned}
f_2 &\triangleq \lambda x.(\mathbf{split}^m(\mathbf{proj_1}\ (\mathbf{merge}^m x)), \mathbf{split}^m(\mathbf{proj_2}\ (\mathbf{merge}^m x))) \\
g_2 &\triangleq \lambda y.\mathbf{split}^m(\mathbf{merge}^m(\mathbf{proj_1}\ y), \mathbf{merge}^m(\mathbf{proj_2}\ y))
\end{aligned}$$

Next, given $x :^{m'} S_m\ (A_1 \times A_2)$, we have,

$$\begin{aligned}
& g_2\ (f_2\ x) \\
\equiv & (\lambda y.\mathbf{split}^m(\mathbf{merge}^m(\mathbf{proj_1}\ y), \mathbf{merge}^m(\mathbf{proj_2}\ y))) \\
& \qquad (\mathbf{split}^m(\mathbf{proj_1}\ (\mathbf{merge}^m x)), \mathbf{split}^m(\mathbf{proj_2}\ (\mathbf{merge}^m x))) \\
\equiv & \mathbf{split}^m(\mathbf{merge}^m(\mathbf{split}^m(\mathbf{proj_1}\ (\mathbf{merge}^m x))), \mathbf{merge}^m(\mathbf{split}^m(\mathbf{proj_2}\ (\mathbf{merge}^m x)))) \\
\equiv & \mathbf{split}^m(\mathbf{proj_1}\ (\mathbf{merge}^m x), \mathbf{proj_2}\ (\mathbf{merge}^m x)) \equiv \mathbf{split}^m(\mathbf{merge}^m x) \equiv x
\end{aligned}$$

And, given $y :^{m'} S_m\ A_1 \times S_m\ A_2$, we have,

$$\begin{aligned}
& f_2\ (g_2\ y) \\
\equiv & (\lambda x.(\mathbf{split}^m(\mathbf{proj_1}\ (\mathbf{merge}^m x)), \mathbf{split}^m(\mathbf{proj_2}\ (\mathbf{merge}^m x)))) \\
& \qquad (\mathbf{split}^m(\mathbf{merge}^m(\mathbf{proj_1}\ y), \mathbf{merge}^m(\mathbf{proj_2}\ y))) \\
\equiv & (\mathbf{split}^m(\mathbf{proj_1}\ (\mathbf{merge}^m(\mathbf{split}^m(\mathbf{merge}^m(\mathbf{proj_1}\ y), \mathbf{merge}^m(\mathbf{proj_2}\ y))))), \\
& \qquad \mathbf{split}^m(\mathbf{proj_2}\ (\mathbf{merge}^m(\mathbf{split}^m(\mathbf{merge}^m(\mathbf{proj_1}\ y), \mathbf{merge}^m(\mathbf{proj_2}\ y)))))) \\
\equiv & (\mathbf{split}^m(\mathbf{proj_1}\ (\mathbf{merge}^m(\mathbf{proj_1}\ y), \mathbf{merge}^m(\mathbf{proj_2}\ y))), \\
& \qquad \mathbf{split}^m(\mathbf{proj_2}\ (\mathbf{merge}^m(\mathbf{proj_1}\ y), \mathbf{merge}^m(\mathbf{proj_2}\ y))))
\end{aligned}$$

$$\equiv(\mathbf{split}^m(\mathbf{merge}^m(\mathbf{proj}_1\ y)), \mathbf{split}^m(\mathbf{merge}^m(\mathbf{proj}_2\ y))) \equiv (\mathbf{proj}_1\ y, \mathbf{proj}_2\ y) \equiv y$$

- Need to show: $S_m\ (A \to B) \cong S_m\ A \to S_m\ B$.
  Let $C \triangleq S_m\ (A \to B)$ and $D \triangleq S_m\ A \to S_m\ B$.
  Now, we have:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{x :^{m'} C, z :^{m'} S_m\ A \vdash x :^{m'} S_m\ (A \to B)}{x :^{m'} C, z :^{m'} S_m\ A \vdash \mathbf{merge}^m x :^{m' \cdot m} A \to B} \qquad \dfrac{x :^{m'} C, z :^{m'} S_m\ A \vdash z :^{m'} S_m\ A}{x :^{m'} C, z :^{m'} S_m\ A \vdash \mathbf{merge}^m z :^{m' \cdot m} A}}{x :^{m'} C, z :^{m'} S_m\ A \vdash (\mathbf{merge}^m x)\ (\mathbf{merge}^m z) :^{m' \cdot m} B}}{x :^{m'} C, z :^{m'} S_m\ A \vdash \mathbf{split}^m((\mathbf{merge}^m x)\ (\mathbf{merge}^m z)) :^{m'} S_m\ B}}{x :^{m'} C \vdash \lambda z.\mathbf{split}^m((\mathbf{merge}^m x)\ (\mathbf{merge}^m z)) :^{m'} S_m\ A \to S_m\ B}}{\varnothing \vdash \lambda x.\lambda z.\mathbf{split}^m((\mathbf{merge}^m x)\ (\mathbf{merge}^m z)) :^{m'} C \to D}$$

  and

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{y :^{m'} D, w :^{m' \cdot m} A \vdash y :^{m'} S_m\ A \to S_m\ B \qquad \dfrac{y :^{m'} D, w :^{m' \cdot m} A \vdash w :^{m' \cdot m} A}{y :^{m'} D, w :^{m' \cdot m} A \vdash \mathbf{split}^m w :^{m'} S_m\ A}}{y :^{m'} D, w :^{m' \cdot m} A \vdash y\ (\mathbf{split}^m w) :^{m'} S_m\ B}}{y :^{m'} D, w :^{m' \cdot m} A \vdash \mathbf{merge}^m(y\ (\mathbf{split}^m w)) :^{m' \cdot m} B}}{y :^{m'} D \vdash \lambda w.\mathbf{merge}^m(y\ (\mathbf{split}^m w)) :^{m' \cdot m} A \to B}}{\dfrac{y :^{m'} D \vdash \mathbf{split}^m(\lambda w.\mathbf{merge}^m(y\ (\mathbf{split}^m w))) :^{m'} S_m\ (A \to B)}{\varnothing \vdash \lambda y.\mathbf{split}^m(\lambda w.\mathbf{merge}^m(y\ (\mathbf{split}^m w))) :^{m'} D \to C}}$$

  Say,

$$f_3 \triangleq \lambda x.\lambda z.\mathbf{split}^m((\mathbf{merge}^m x)\ (\mathbf{merge}^m z))$$
$$g_3 \triangleq \lambda y.\mathbf{split}^m(\lambda w.\mathbf{merge}^m(y\ (\mathbf{split}^m w)))$$

  Next, given $x :^{m'} S_m\ (A \to B)$, we have,

$$\begin{aligned}
& g_3\ (f_3\ x) \\
\equiv & (\lambda y.\mathbf{split}^m(\lambda w.\mathbf{merge}^m(y\ (\mathbf{split}^m w))))\ (\lambda z.\mathbf{split}^m((\mathbf{merge}^m x)\ (\mathbf{merge}^m z))) \\
\equiv & \mathbf{split}^m(\lambda w.\mathbf{merge}^m((\lambda z.\mathbf{split}^m((\mathbf{merge}^m x)\ (\mathbf{merge}^m z)))\ (\mathbf{split}^m w))) \\
\equiv & \mathbf{split}^m(\lambda w.\mathbf{merge}^m(\mathbf{split}^m((\mathbf{merge}^m x)\ (\mathbf{merge}^m(\mathbf{split}^m w))))) \\
\equiv & \mathbf{split}^m(\lambda w.\mathbf{merge}^m(\mathbf{split}^m((\mathbf{merge}^m x)\ w))) \\
\equiv & \mathbf{split}^m(\lambda w.(\mathbf{merge}^m x)\ w) \equiv \mathbf{split}^m(\mathbf{merge}^m x) \equiv x
\end{aligned}$$

  And, given $y :^{m'} S_m\ A \to S_m\ B$, we have,

$$\begin{aligned}
& f_3\ (g_3\ y) \\
\equiv & (\lambda x.\lambda z.\mathbf{split}^m((\mathbf{merge}^m x)\ (\mathbf{merge}^m z)))\ (\mathbf{split}^m(\lambda w.\mathbf{merge}^m(y\ (\mathbf{split}^m w))))
\end{aligned}$$

$$\equiv \lambda z.\mathbf{split}^m((\mathbf{merge}^m(\mathbf{split}^m(\lambda w.\mathbf{merge}^m(y\ (\mathbf{split}^m w)))))\ (\mathbf{merge}^m z))$$
$$\equiv \lambda z.\mathbf{split}^m((\lambda w.\mathbf{merge}^m(y\ (\mathbf{split}^m w)))\ (\mathbf{merge}^m z))$$
$$\equiv \lambda z.\mathbf{split}^m(\mathbf{merge}^m(y\ (\mathbf{split}^m(\mathbf{merge}^m z))))$$
$$\equiv \lambda z.y\ (\mathbf{split}^m(\mathbf{merge}^m z)) \equiv \lambda z.y\ z \equiv y$$

□

Lemma A.31 (Lemma 2.26) Let $\Gamma \vdash a :^m A$ in $\text{GMCC}_\text{e}(\mathcal{M})$. Then, $m' \cdot \Gamma \vdash a :^{m' \cdot m} A$, for any $m' \in \mathcal{M}$.

Proof. By induction on $\Gamma \vdash a :^m A$. We present the interesting cases below.

- Rule E-Var. Have: $\Gamma_1, x :^m A, \Gamma_2 \vdash x :^m A$.
  Need to show: $m' \cdot \Gamma_1, x :^{m' \cdot m} A, m' \cdot \Gamma_2 \vdash x :^{m' \cdot m} A$.
  Follows by rule E-Var.
- Rule E-Lam. Have: $\Gamma \vdash \lambda x : A.b :^m A \to B$ where $\Gamma, x :^m A \vdash b :^m B$.
  Need to show: $m' \cdot \Gamma \vdash \lambda x : A.b :^{m' \cdot m} A \to B$.
  By IH, $m' \cdot \Gamma, x :^{m' \cdot m} A \vdash b :^{m' \cdot m} B$.
  This case, then, follows by rule E-Lam.
- Rule E-App. Have: $\Gamma \vdash b\ a :^m B$ where $\Gamma \vdash b :^m A \to B$ and $\Gamma \vdash a :^m A$.
  Need to show: $m' \cdot \Gamma \vdash b\ a :^{m' \cdot m} B$.
  By IH, $m' \cdot \Gamma \vdash b :^{m' \cdot m} A \to B$ and $m' \cdot \Gamma \vdash a :^{m' \cdot m} A$.
  This case, then, follows by rule E-App.
- Rule E-Split. Have: $\Gamma \vdash \mathbf{split}^{m_2} a :^{m_1} S_{m_2} A$ where $\Gamma \vdash a :^{m_1 \cdot m_2} A$.
  Need to show: $m' \cdot \Gamma \vdash \mathbf{split}^{m_2} a :^{m' \cdot m_1} A$.
  By IH, $m' \cdot \Gamma \vdash a :^{m' \cdot (m_1 \cdot m_2)} A$. By associativity, $m' \cdot (m_1 \cdot m_2) = (m' \cdot m_1) \cdot m_2$.
  This case, then, follows by rule E-Split.
- Rule E-Merge. Have: $\Gamma \vdash \mathbf{merge}^{m_2} a :^{m_1 \cdot m_2} A$ where $\Gamma \vdash a :^{m_1} S_{m_2} A$.
  Need to show: $m' \cdot \Gamma \vdash \mathbf{merge}^{m_2} a :^{m' \cdot (m_1 \cdot m_2)} A$.
  By IH, $m' \cdot \Gamma \vdash a :^{m' \cdot m_1} S_{m_2} A$.
  By rule E-Merge, $m' \cdot \Gamma \vdash \mathbf{merge}^{m_2} a :^{(m' \cdot m_1) \cdot m_2} A$.
  This case, then, follows by associativity.
- Rule E-Up. Have: $\Gamma \vdash a :^{m_2} A$ where $\Gamma \vdash a :^{m_1} A$ and $m_1 <: m_2$.
  Need to show: $m' \cdot \Gamma \vdash a :^{m' \cdot m_2} A$.
  By IH, $m' \cdot \Gamma \vdash a :^{m' \cdot m_1} A$. By monotonicity, $m' \cdot m_1 <: m' \cdot m_2$.
  This case, then, follows by rule E-Up.

□

Theorem A.32 (Theorem 2.27) If $\Gamma \vdash a :^m A$ in GMCC$_e$, then $[\![a]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], \mathbf{S}_m[\![A]\!])$. Further, if $\Gamma \vdash a_1 :^m A$ and $\Gamma \vdash a_2 :^m A$ such that $a_1 \equiv a_2$ in GMCC$_e$, then $[\![a_1]\!] = [\![a_2]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], \mathbf{S}_m[\![A]\!])$.

Proof. The first part follows by induction on the typing derivation. Most of the cases are similar to those in Theorem A.26. We present the differing ones below.

- Rule E-Pair. Have: $\Gamma \vdash (a_1, a_2) :^m A_1 \times A_2$ where $\Gamma \vdash a_1 :^m A_1$ and $\Gamma \vdash a_2 :^m A_2$.
  Need to show: $[\![(a_1, a_2)]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], \mathbf{S}_m([\![A_1]\!] \times [\![A_2]\!]))$.
  By IH, $[\![a_1]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], \mathbf{S}_m[\![A_1]\!])$ and $[\![a_2]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], \mathbf{S}_m[\![A_2]\!])$.
  Now, $[\![(a_1, a_2)]\!] = [\![\Gamma]\!] \xrightarrow{\langle [\![a_1]\!], [\![a_2]\!] \rangle} \mathbf{S}_m[\![A_1]\!] \times \mathbf{S}_m[\![A_2]\!] \xrightarrow{p^{-1}_{[\![A_1]\!],[\![A_2]\!]}} \mathbf{S}_m([\![A_1]\!] \times [\![A_2]\!])$.
- Rule E-Proj. Have: $\Gamma \vdash \mathbf{proj}_i\ a :^m A_i$ where $\Gamma \vdash a :^m A_1 \times A_2$.
  Need to show: $[\![\mathbf{proj}_i\ a]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], \mathbf{S}_m[\![A_i]\!])$.
  By IH, $[\![a]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], \mathbf{S}_m([\![A_1]\!] \times [\![A_2]\!]))$.
  Now, $[\![\mathbf{proj}_i\ a]\!] = [\![\Gamma]\!] \xrightarrow{[\![a]\!]} \mathbf{S}_m([\![A_1]\!] \times [\![A_2]\!]) \xrightarrow{\mathbf{S}_m \pi_i} \mathbf{S}_m[\![A_i]\!]$.
- Rule E-Split. Have: $\Gamma \vdash \mathbf{split}^{m_2}\, a :^{m_1} S_{m_2} A$ where $\Gamma \vdash a :^{m_1 \cdot m_2} A$.
  Need to show: $[\![\mathbf{split}^{m_2}\, a]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], \mathbf{S}_{m_1}\mathbf{S}_{m_2}[\![A]\!])$.
  By IH, $[\![a]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], \mathbf{S}_{m_1 \cdot m_2}[\![A]\!])$.
  Now, $[\![\mathbf{split}^{m_2}\, a]\!] = [\![\Gamma]\!] \xrightarrow{[\![a]\!]} \mathbf{S}_{m_1 \cdot m_2}[\![A]\!] \xrightarrow{\delta^{m_1, m_2}_{[\![A]\!]}} \mathbf{S}_{m_1}\mathbf{S}_{m_2}[\![A]\!]$.
- Rule E-Merge. Have: $\Gamma \vdash \mathbf{merge}^{m_2}\, a :^{m_1 \cdot m_2} A$ where $\Gamma \vdash a :^{m_1} S_{m_2} A$.
  Need to show: $[\![\mathbf{merge}^{m_2}\, a]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], \mathbf{S}_{m_1 \cdot m_2}[\![A]\!])$.
  By IH, $[\![a]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], \mathbf{S}_{m_1}\mathbf{S}_{m_2}[\![A]\!])$.
  Now, $[\![\mathbf{merge}^{m_2}\, a]\!] = [\![\Gamma]\!] \xrightarrow{[\![a]\!]} \mathbf{S}_{m_1}\mathbf{S}_{m_2}[\![A]\!] \xrightarrow{\mu^{m_1, m_2}_{[\![A]\!]}} \mathbf{S}_{m_1 \cdot m_2}[\![A]\!]$.
- Rule E-Up. Have: $\Gamma \vdash a :^{m_2} A$ where $\Gamma \vdash a :^{m_1} A$ and $m_1 <: m_2$.
  Need to show: $[\![\Gamma \vdash a :^{m_2} A]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], \mathbf{S}_{m_2}[\![A]\!])$.
  By IH, $[\![\Gamma \vdash a :^{m_1} A]\!] \in \mathrm{Hom}_{\mathbb{C}}([\![\Gamma]\!], \mathbf{S}_{m_1}[\![A]\!])$.
  Now, $[\![\Gamma \vdash a :^{m_2} A]\!] = [\![\Gamma]\!] \xrightarrow{[\![\Gamma \vdash a :^{m_1} A]\!]} \mathbf{S}_{m_1}[\![A]\!] \xrightarrow{\mathbf{S}^{m_1 <: m_2}_{[\![A]\!]}} \mathbf{S}_{m_2}[\![A]\!]$.

For the second part, invert the equality judgement, $a_1 \equiv a_2$. Most of the cases are similar to those in Theorem A.26. We present the differing ones below.

- $\mathbf{proj}_i\ (a_1, a_2) \equiv a_i$.
  Now,

$$
\begin{aligned}
&[\![\mathbf{proj}_i\ (a_1, a_2)]\!] \\
=&\mathbf{S}_m \pi_i \circ p^{-1}_{[\![A_1]\!],[\![A_2]\!]} \circ \langle [\![a_1]\!], [\![a_2]\!] \rangle \\
=&\pi_i \circ \langle \mathbf{S}_m \pi_1, \mathbf{S}_m \pi_2 \rangle \circ p^{-1}_{[\![A_1]\!],[\![A_2]\!]} \circ \langle [\![a_1]\!], [\![a_2]\!] \rangle \\
=&\pi_i \circ p_{[\![A_1]\!],[\![A_2]\!]} \circ p^{-1}_{[\![A_1]\!],[\![A_2]\!]} \circ \langle [\![a_1]\!], [\![a_2]\!] \rangle = \pi_i \circ \langle [\![a_1]\!], [\![a_2]\!] \rangle = [\![a_i]\!].
\end{aligned}
$$

- $a \equiv (\mathbf{proj}_1\ a, \mathbf{proj}_2\ a)$.
  Now,

$$\begin{aligned}
&[\![(\mathbf{proj}_1\ a, \mathbf{proj}_2\ a)]\!]\\
=&p^{-1}_{[\![A_1]\!],[\![A_2]\!]} \circ \langle [\![\mathbf{proj}_1\ a]\!], [\![\mathbf{proj}_2\ a]\!]\rangle\\
=&p^{-1}_{[\![A_1]\!],[\![A_2]\!]} \circ \langle \mathbf{S}_m \pi_1 \circ [\![a]\!], \mathbf{S}_m \pi_2 \circ [\![a]\!]\rangle\\
=&p^{-1}_{[\![A_1]\!],[\![A_2]\!]} \circ \langle \mathbf{S}_m \pi_1, \mathbf{S}_m \pi_2\rangle \circ [\![a]\!] = p^{-1}_{[\![A_1]\!],[\![A_2]\!]} \circ p_{[\![A_1]\!],[\![A_2]\!]} \circ [\![a]\!] = [\![a]\!]
\end{aligned}$$

- $\Gamma \vdash \mathbf{merge}^m(\mathbf{split}^m\, a) \equiv a :^{m'\cdot m} A$.
  Now,

$$\begin{aligned}
&[\![\Gamma \vdash \mathbf{merge}^m(\mathbf{split}^m\, a) :^{m'\cdot m} A]\!]\\
=&\mu^{m',m}_{[\![A]\!]} \circ [\![\Gamma \vdash \mathbf{split}^m\, a :^{m'} S_m\ A]\!]\\
=&\mu^{m',m}_{[\![A]\!]} \circ \delta^{m',m}_{[\![A]\!]} \circ [\![\Gamma \vdash a :^{m'\cdot m} A]\!] = [\![\Gamma \vdash a :^{m'\cdot m} A]\!]
\end{aligned}$$

- $\Gamma \vdash a \equiv \mathbf{split}^m(\mathbf{merge}^m\, a) :^{m'} S_m\ A$.
  Now,

$$\begin{aligned}
&[\![\Gamma \vdash \mathbf{split}^m(\mathbf{merge}^m\, a) :^{m'} S_m\ A]\!]\\
=&\delta^{m',m}_{[\![A]\!]} \circ [\![\Gamma \vdash \mathbf{merge}^m\, a :^{m'\cdot m} A]\!]\\
=&\delta^{m',m}_{[\![A]\!]} \circ \mu^{m',m}_{[\![A]\!]} \circ [\![\Gamma \vdash a :^{m'} S_m\ A]\!] = [\![\Gamma \vdash a :^{m'} S_m\ A]\!]
\end{aligned}$$

□

Theorem A.33 (Theorem 2.28) If $\Gamma \vdash a : A$ in GMCC($\mathcal{L}$), then $\Gamma^{\perp} \vdash \widetilde{a} :^{\perp} A$ in GMCC$_{\mathrm{e}}$($\mathcal{L}$). Further, if $\Gamma \vdash a_1 : A$ and $\Gamma \vdash a_2 : A$ such that $a_1 \equiv a_2$ in GMCC($\mathcal{L}$), then $\widetilde{a_1} \equiv \widetilde{a_2}$ in GMCC$_{\mathrm{e}}$($\mathcal{L}$).

Proof. Let $\Gamma \vdash a : A$ in GMCC($\mathcal{L}$). We show $\Gamma^{\perp} \vdash \widetilde{a} :^{\perp} A$ in GMCC$_{\mathrm{e}}$($\mathcal{L}$) by induction on the typing derivation.

- λ-calculus. By IH.
- Rule MC-Return. Have: $\Gamma \vdash \mathbf{ret}\ a : S_{\perp}\ A$ where $\Gamma \vdash a : A$.
  Need to show: $\Gamma^{\perp} \vdash \mathbf{split}^{\perp}\widetilde{a} :^{\perp} S_{\perp}\ A$.
  By IH, $\Gamma^{\perp} \vdash \widetilde{a} :^{\perp} A$.
  This case follows by rule E-Split.
- Rule MC-Extract. Have: $\Gamma \vdash \mathbf{extr}\ a : A$ where $\Gamma \vdash a : S_{\perp}\ A$.
  Need to show: $\Gamma^{\perp} \vdash \mathbf{merge}^{\perp}\widetilde{a} :^{\perp} A$.
  By IH, $\Gamma^{\perp} \vdash \widetilde{a} :^{\perp} S_{\perp}\ A$.
  This case follows by rule E-Merge.

- Rule MC-Join. Have: $\Gamma \vdash \mathbf{join}^{\ell_1,\ell_2}\, a : S_{\ell_1 \sqcup \ell_2}\ A$ where $\Gamma \vdash a : S_{\ell_1}\ S_{\ell_2}\ A$.
  Need to show: $\Gamma^{\perp} \vdash \mathbf{split}^{\ell_1 \sqcup \ell_2}(\mathbf{merge}^{\ell_2}(\mathbf{merge}^{\ell_1}\widetilde{a})) :^{\perp} S_{\ell_1 \sqcup \ell_2}\ A$.
  By IH, $\Gamma^{\perp} \vdash \widetilde{a} :^{\perp} S_{\ell_1}\ S_{\ell_2}\ A$.
  By rule E-Merge, $\Gamma^{\perp} \vdash \mathbf{merge}^{\ell_1}\widetilde{a} :^{\ell_1} S_{\ell_2}\ A$.
  Applying rule E-Merge again, $\Gamma^{\perp} \vdash \mathbf{merge}^{\ell_2}(\mathbf{merge}^{\ell_1}\widetilde{a}) :^{\ell_1 \sqcup \ell_2} A$.
  This case, then, follows by rule E-Split.

- Rule MC-Fork. Have: $\Gamma \vdash \mathbf{fork}^{\ell_1,\ell_2}\, a : S_{\ell_1}\ S_{\ell_2}\ A$ where $\Gamma \vdash a : S_{\ell_1 \sqcup \ell_2}\ A$.
  Need to show: $\Gamma^{\perp} \vdash \mathbf{split}^{\ell_1}(\mathbf{split}^{\ell_2}(\mathbf{merge}^{\ell_1 \sqcup \ell_2}\widetilde{a})) :^{\perp} S_{\ell_1}\ S_{\ell_2}\ A$.
  By IH, $\Gamma^{\perp} \vdash \widetilde{a} :^{\perp} S_{\ell_1 \sqcup \ell_2}\ A$.
  By rule E-Merge, $\Gamma^{\perp} \vdash \mathbf{merge}^{\ell_1 \sqcup \ell_2}\widetilde{a} :^{\ell_1 \sqcup \ell_2} A$.
  By rule E-Split, $\Gamma^{\perp} \vdash \mathbf{split}^{\ell_2}(\mathbf{merge}^{\ell_1 \sqcup \ell_2}\widetilde{a}) :^{\ell_1} S_{\ell_2}\ A$.
  Applying rule E-Split again, $\Gamma^{\perp} \vdash \mathbf{split}^{\ell_1}(\mathbf{split}^{\ell_2}(\mathbf{merge}^{\ell_1 \sqcup \ell_2}\widetilde{a})) :^{\perp} S_{\ell_1}\ S_{\ell_2}\ A$.

- Rule MC-Fmap. Have: $\Gamma \vdash \mathbf{lift}^{\ell} f : S_{\ell}\ A \to S_{\ell}\ B$ where $\Gamma \vdash f : A \to B$.
  Need to show: $\Gamma^{\perp} \vdash \lambda x.\mathbf{split}^{\ell}(\widetilde{f}\ (\mathbf{merge}^{\ell} x)) :^{\perp} S_{\ell}\ A \to S_{\ell}\ B$.
  By IH, $\Gamma^{\perp} \vdash \widetilde{f} :^{\perp} A \to B$.
  Now,

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma^{\perp}, x :^{\perp} S_{\ell}\ A \vdash \widetilde{f} :^{\perp} A \to B}{\Gamma^{\perp}, x :^{\perp} S_{\ell}\ A \vdash \widetilde{f} :^{\ell} A \to B}\ (\text{E-Up}) \qquad \dfrac{\Gamma^{\perp}, x :^{\perp} S_{\ell}\ A \vdash x :^{\perp} S_{\ell}\ A}{\Gamma^{\perp}, x :^{\perp} S_{\ell}\ A \vdash \mathbf{merge}^{\ell} x :^{\ell} A}}{\Gamma^{\perp}, x :^{\perp} S_{\ell}\ A \vdash \widetilde{f}\ (\mathbf{merge}^{\ell} x) :^{\ell} B}}{\Gamma^{\perp}, x :^{\perp} S_{\ell}\ A \vdash \mathbf{split}^{\ell}(\widetilde{f}\ (\mathbf{merge}^{\ell} x)) :^{\perp} S_{\ell}\ B}}{\Gamma^{\perp} \vdash \lambda x.\mathbf{split}^{\ell}(\widetilde{f}\ (\mathbf{merge}^{\ell} x)) :^{\perp} S_{\ell}\ A \to S_{\ell}\ B}$$

- Rule MC-Up. Have: $\Gamma \vdash \mathbf{up}^{\ell_1,\ell_2}\, a : S_{\ell_2}\ A$ where $\Gamma \vdash a : S_{\ell_1}\ A$ and $\ell_1 \sqsubseteq \ell_2$.
  Need to show: $\Gamma^{\perp} \vdash \mathbf{split}^{\ell_2}(\mathbf{merge}^{\ell_1}\widetilde{a}) :^{\perp} S_{\ell_2}\ A$.
  By IH, $\Gamma^{\perp} \vdash \widetilde{a} :^{\perp} S_{\ell_1}\ A$.
  By rule E-Merge, $\Gamma^{\perp} \vdash \mathbf{merge}^{\ell_1}\widetilde{a} :^{\ell_1} A$.
  By rule E-Up, $\Gamma^{\perp} \vdash \mathbf{merge}^{\ell_1}\widetilde{a} :^{\ell_2} A$.
  This case, then, follows by rule E-Split.

Next, we show that if $\Gamma \vdash a_1 : A$ and $\Gamma \vdash a_2 : A$ such that $a_1 \equiv a_2$ in GMCC($\mathcal{L}$), then $\widetilde{a_1} \equiv \widetilde{a_2}$ in GMCC$_{\text{e}}$($\mathcal{L}$). By inversion on $a_1 \equiv a_2$.

- λ-calculus. By IH.

- $\mathbf{lift}^{\ell}(\lambda x.x) \equiv \lambda x.x$.
  Now,

$$\begin{aligned}
&\widetilde{\mathbf{lift}^{\ell}(\lambda x.x)} \\
=&\lambda y.\mathbf{split}^{\ell}((\lambda x.x)\ (\mathbf{merge}^{\ell} y)) \\
\equiv&\lambda y.\mathbf{split}^{\ell}(\mathbf{merge}^{\ell} y) \equiv \lambda y.y
\end{aligned}$$

- $\mathbf{lift}^{\ell}(\lambda x.g\ (f\ x)) \equiv \lambda x.(\mathbf{lift}^{\ell} g)\ ((\mathbf{lift}^{\ell} f)\ x)$.
  Now,

$$\begin{aligned}
&\widetilde{\mathbf{lift}^{\ell}(\lambda x.g\ (f\ x))} \\
=&\lambda y.\mathbf{split}^{\ell}((\lambda x.\widetilde{g}\ (\widetilde{f}\ x))\ (\mathbf{merge}^{\ell} y)) \\
\equiv&\lambda y.\mathbf{split}^{\ell}(\widetilde{g}\ (\widetilde{f}\ (\mathbf{merge}^{\ell} y)))
\end{aligned}$$

  and

$$\begin{aligned}
&\widetilde{\lambda x.(\mathbf{lift}^{\ell} g)\ ((\mathbf{lift}^{\ell} f)\ x)} \\
\equiv&\lambda x.(\lambda y.\mathbf{split}^{\ell}(\widetilde{g}\ (\mathbf{merge}^{\ell} y)))\ (\mathbf{split}^{\ell}(\widetilde{f}\ (\mathbf{merge}^{\ell} x))) \\
\equiv&\lambda x.\mathbf{split}^{\ell}(\widetilde{g}\ (\mathbf{merge}^{\ell}(\mathbf{split}^{\ell}(\widetilde{f}\ (\mathbf{merge}^{\ell} x))))) \\
\equiv&\lambda x.\mathbf{split}^{\ell}(\widetilde{g}\ (\widetilde{f}\ (\mathbf{merge}^{\ell} x)))
\end{aligned}$$

- $\mathbf{up}^{\ell_1,\ell_1}\, a \equiv a$.
  Now, $\widetilde{\mathbf{up}^{\ell_1,\ell_1}\, a} = \mathbf{split}^{\ell_1}(\mathbf{merge}^{\ell_1} \widetilde{a}) \equiv \widetilde{a}$.
- $\mathbf{up}^{\ell_2,\ell_3}(\mathbf{up}^{\ell_1,\ell_2}\, a) \equiv \mathbf{up}^{\ell_1,\ell_3}\, a$.
  Now,

$$\begin{aligned}
&\widetilde{\mathbf{up}^{\ell_2,\ell_3}(\mathbf{up}^{\ell_1,\ell_2}\, a)} \\
=&\mathbf{split}^{\ell_3}(\mathbf{merge}^{\ell_2}(\mathbf{split}^{\ell_2}(\mathbf{merge}^{\ell_1} \widetilde{a}))) \\
\equiv&\mathbf{split}^{\ell_3}(\mathbf{merge}^{\ell_1} \widetilde{a}) = \widetilde{\mathbf{up}^{\ell_1,\ell_3}\, a}
\end{aligned}$$

- $(\mathbf{up}^{\ell_1,\ell_1'}\, a)\ {}^{\ell_1'}\!\!\gg\!=^{\ell_2} f \equiv \mathbf{up}^{\ell_1 \sqcup \ell_2, \ell_1' \sqcup \ell_2}(a\ {}^{\ell_1}\!\!\gg\!=^{\ell_2} f)$.
  First, note that:

$$\begin{aligned}
&\widetilde{a\ {}^{\ell_1}\!\!\gg\!=^{\ell_2} f} \\
=&\widetilde{\mathbf{join}^{\ell_1,\ell_2}((\mathbf{lift}^{\ell_1} f)\ a)} \\
=&\mathbf{split}^{\ell_1 \sqcup \ell_2}(\mathbf{merge}^{\ell_2}(\mathbf{merge}^{\ell_1}(\widetilde{\mathbf{lift}^{\ell_1} f}\ \widetilde{a}))) \\
\equiv&\mathbf{split}^{\ell_1 \sqcup \ell_2}(\mathbf{merge}^{\ell_2}(\mathbf{merge}^{\ell_1}(\mathbf{split}^{\ell_1}(\widetilde{f}\ (\mathbf{merge}^{\ell_1} \widetilde{a}))))) \\
\equiv&\mathbf{split}^{\ell_1 \sqcup \ell_2}(\mathbf{merge}^{\ell_2}(\widetilde{f}\ (\mathbf{merge}^{\ell_1} \widetilde{a})))
\end{aligned}$$

  Next,

$$\begin{aligned}
&\widetilde{(\mathbf{up}^{\ell_1,\ell_1'}\, a)\ {}^{\ell_1'}\!\!\gg\!=^{\ell_2} f} \\
\equiv&\mathbf{split}^{\ell_1' \sqcup \ell_2}(\mathbf{merge}^{\ell_2}(\widetilde{f}\ (\mathbf{merge}^{\ell_1'} \widetilde{\mathbf{up}^{\ell_1,\ell_1'}\, a}))) \\
\equiv&\mathbf{split}^{\ell_1' \sqcup \ell_2}(\mathbf{merge}^{\ell_2}(\widetilde{f}\ (\mathbf{merge}^{\ell_1'}(\mathbf{split}^{\ell_1'}(\mathbf{merge}^{\ell_1} \widetilde{a})))))
\end{aligned}$$

$$\equiv \mathbf{split}^{\ell_1' \sqcup \ell_2}(\mathbf{merge}^{\ell_2}(\widetilde{f}\ (\mathbf{merge}^{\ell_1}\widetilde{a})))$$

and

$$\widetilde{\mathbf{up}^{\ell_1 \sqcup \ell_2, \ell_1' \sqcup \ell_2}(a\ {}^{\ell_1}{\gg\!=}^{\ell_2}\ f)}$$
$$= \mathbf{split}^{\ell_1' \sqcup \ell_2}(\mathbf{merge}^{\ell_1 \sqcup \ell_2}\, \widetilde{a\ {}^{\ell_1}{\gg\!=}^{\ell_2}\ f})$$
$$\equiv \mathbf{split}^{\ell_1' \sqcup \ell_2}(\mathbf{merge}^{\ell_1 \sqcup \ell_2}(\mathbf{split}^{\ell_1 \sqcup \ell_2}(\mathbf{merge}^{\ell_2}(\widetilde{f}\ (\mathbf{merge}^{\ell_1}\widetilde{a})))))$$
$$\equiv \mathbf{split}^{\ell_1' \sqcup \ell_2}(\mathbf{merge}^{\ell_2}(\widetilde{f}\ (\mathbf{merge}^{\ell_1}\widetilde{a})))$$

- $a\ {}^{\ell_1}{\gg\!=}^{\ell_2'}\ (\lambda x.\mathbf{up}^{\ell_2, \ell_2'}\, b) \equiv \mathbf{up}^{\ell_1 \sqcup \ell_2, \ell_1 \sqcup \ell_2'}(a\ {}^{\ell_1}{\gg\!=}^{\ell_2}\ \lambda x.b)$.
  Now,

$$\widetilde{a\ {}^{\ell_1}{\gg\!=}^{\ell_2'}\ (\lambda x.\mathbf{up}^{\ell_2, \ell_2'}\, b)}$$
$$\equiv \mathbf{split}^{\ell_1 \sqcup \ell_2'}(\mathbf{merge}^{\ell_2'}(\widetilde{\lambda x.\mathbf{up}^{\ell_2, \ell_2'}\, b}\ (\mathbf{merge}^{\ell_1}\widetilde{a})))$$
$$= \mathbf{split}^{\ell_1 \sqcup \ell_2'}(\mathbf{merge}^{\ell_2'}((\lambda x.\mathbf{split}^{\ell_2'}(\mathbf{merge}^{\ell_2}\widetilde{b}))\ (\mathbf{merge}^{\ell_1}\widetilde{a})))$$
$$\equiv \mathbf{split}^{\ell_1 \sqcup \ell_2'}(\mathbf{merge}^{\ell_2'}(\mathbf{split}^{\ell_2'}(\mathbf{merge}^{\ell_2}\widetilde{b}\{\mathbf{merge}^{\ell_1}\widetilde{a}/x\})))$$
$$\equiv \mathbf{split}^{\ell_1 \sqcup \ell_2'}(\mathbf{merge}^{\ell_2}\widetilde{b}\{\mathbf{merge}^{\ell_1}\widetilde{a}/x\})$$

and

$$\widetilde{\mathbf{up}^{\ell_1 \sqcup \ell_2, \ell_1 \sqcup \ell_2'}(a\ {}^{\ell_1}{\gg\!=}^{\ell_2}\ \lambda x.b)}$$
$$= \mathbf{split}^{\ell_1 \sqcup \ell_2'}(\mathbf{merge}^{\ell_1 \sqcup \ell_2}\, \widetilde{a\ {}^{\ell_1}{\gg\!=}^{\ell_2}\ \lambda x.b})$$
$$\equiv \mathbf{split}^{\ell_1 \sqcup \ell_2'}(\mathbf{merge}^{\ell_1 \sqcup \ell_2}(\mathbf{split}^{\ell_1 \sqcup \ell_2}(\mathbf{merge}^{\ell_2}((\lambda x.\widetilde{b})\ (\mathbf{merge}^{\ell_1}\widetilde{a})))))$$
$$\equiv \mathbf{split}^{\ell_1 \sqcup \ell_2'}(\mathbf{merge}^{\ell_2}((\lambda x.\widetilde{b})\ (\mathbf{merge}^{\ell_1}\widetilde{a})))$$
$$\equiv \mathbf{split}^{\ell_1 \sqcup \ell_2'}(\mathbf{merge}^{\ell_2}\widetilde{b}\{\mathbf{merge}^{\ell_1}\widetilde{a}/x\})$$

- $(\mathbf{ret}\ a)\ {}^{\bot}{\gg\!=}^{\ell}\ f \equiv f\ a$.
  Now,

$$\widetilde{(\mathbf{ret}\ a)\ {}^{\bot}{\gg\!=}^{\ell}\ f}$$
$$\equiv \mathbf{split}^{\ell}(\mathbf{merge}^{\ell}(\widetilde{f}\ (\mathbf{merge}^{\bot}\widetilde{\mathbf{ret}\ a})))$$
$$= \mathbf{split}^{\ell}(\mathbf{merge}^{\ell}(\widetilde{f}\ (\mathbf{merge}^{\bot}(\mathbf{split}^{\bot}\widetilde{a}))))$$
$$\equiv \mathbf{split}^{\ell}(\mathbf{merge}^{\ell}(\widetilde{f}\ \widetilde{a})) \equiv \widetilde{f}\ \widetilde{a}$$

- $a \ {}^{\ell_1}\!\gg\!=^{\bot} (\lambda x.\mathbf{ret}\ x) \equiv a$.

  Now,

$$
\begin{aligned}
&\widetilde{a \ {}^{\ell_1}\!\gg\!=^{\bot} (\lambda x.\mathbf{ret}\ x)}\\
\equiv&\mathbf{split}^{\ell_1}(\mathbf{merge}^{\bot}(\widetilde{\lambda x.\mathbf{ret}}\ x\ (\mathbf{merge}^{\ell_1}\widetilde{a})))\\
=&\mathbf{split}^{\ell_1}(\mathbf{merge}^{\bot}((\lambda x.\mathbf{split}^{\bot} x)\ (\mathbf{merge}^{\ell_1}\widetilde{a})))\\
\equiv&\mathbf{split}^{\ell_1}(\mathbf{merge}^{\bot}(\mathbf{split}^{\bot}(\mathbf{merge}^{\ell_1}\widetilde{a})))\\
\equiv&\mathbf{split}^{\ell_1}(\mathbf{merge}^{\ell_1}\widetilde{a}) \equiv \widetilde{a}
\end{aligned}
$$

- $(a \ {}^{\ell_1}\!\gg\!=^{\ell_2} f) \ {}^{\ell_1 \sqcup \ell_2}\!\gg\!=^{\ell_3} g \equiv a \ {}^{\ell_1}\!\gg\!=^{\ell_2 \sqcup \ell_3} (\lambda x.(f\ x \ {}^{\ell_2}\!\gg\!=^{\ell_3} g))$.

  Now,

$$
\begin{aligned}
&\widetilde{(a \ {}^{\ell_1}\!\gg\!=^{\ell_2} f) \ {}^{\ell_1 \sqcup \ell_2}\!\gg\!=^{\ell_3} g}\\
\equiv&\mathbf{split}^{(\ell_1 \sqcup \ell_2) \sqcup \ell_3}(\mathbf{merge}^{\ell_3}(\widetilde{g}\ (\mathbf{merge}^{\ell_1 \sqcup \ell_2}\widetilde{a \ {}^{\ell_1}\!\gg\!=^{\ell_2} f})))\\
\equiv&\mathbf{split}^{(\ell_1 \sqcup \ell_2) \sqcup \ell_3}(\mathbf{merge}^{\ell_3}(\widetilde{g}\ (\mathbf{merge}^{\ell_1 \sqcup \ell_2}(\mathbf{split}^{\ell_1 \sqcup \ell_2}(\mathbf{merge}^{\ell_2}(\widetilde{f}\ (\mathbf{merge}^{\ell_1}\widetilde{a})))))))\\
\equiv&\mathbf{split}^{\ell_1 \sqcup \ell_2 \sqcup \ell_3}(\mathbf{merge}^{\ell_3}(\widetilde{g}\ (\mathbf{merge}^{\ell_2}(\widetilde{f}\ (\mathbf{merge}^{\ell_1}\widetilde{a})))))
\end{aligned}
$$

  and

$$
\begin{aligned}
&\widetilde{a \ {}^{\ell_1}\!\gg\!=^{\ell_2 \sqcup \ell_3} (\lambda x.(f\ x \ {}^{\ell_2}\!\gg\!=^{\ell_3} g))}\\
\equiv&\mathbf{split}^{\ell_1 \sqcup (\ell_2 \sqcup \ell_3)}(\mathbf{merge}^{\ell_2 \sqcup \ell_3}((\widetilde{\lambda x.(f\ x \ {}^{\ell_2}\!\gg\!=^{\ell_3} g)})\ (\mathbf{merge}^{\ell_1}\widetilde{a})))\\
\equiv&\mathbf{split}^{\ell_1 \sqcup (\ell_2 \sqcup \ell_3)}(\mathbf{merge}^{\ell_2 \sqcup \ell_3}((\lambda x.(\mathbf{split}^{\ell_2 \sqcup \ell_3}(\mathbf{merge}^{\ell_3}(\widetilde{g}\ (\mathbf{merge}^{\ell_2}(\widetilde{f}\ x))))))\ (\mathbf{merge}^{\ell_1}\widetilde{a})))\\
\equiv&\mathbf{split}^{\ell_1 \sqcup (\ell_2 \sqcup \ell_3)}(\mathbf{merge}^{\ell_2 \sqcup \ell_3}(\mathbf{split}^{\ell_2 \sqcup \ell_3}(\mathbf{merge}^{\ell_3}(\widetilde{g}\ (\mathbf{merge}^{\ell_2}(\widetilde{f}\ (\mathbf{merge}^{\ell_1}\widetilde{a})))))))\\
\equiv&\mathbf{split}^{\ell_1 \sqcup \ell_2 \sqcup \ell_3}(\mathbf{merge}^{\ell_3}(\widetilde{g}\ (\mathbf{merge}^{\ell_2}(\widetilde{f}\ (\mathbf{merge}^{\ell_1}\widetilde{a})))))
\end{aligned}
$$

- $\mathbf{ret}\ (\mathbf{extr}\ a) \equiv a$.

  Now, $\widetilde{\mathbf{ret}\ (\mathbf{extr}}\ a) = \mathbf{split}^{\bot}(\mathbf{merge}^{\bot}\widetilde{a}) \equiv \widetilde{a}$.

- $\mathbf{extr}\ (\mathbf{ret}\ a) \equiv a$.

  Now, $\widetilde{\mathbf{extr}\ (\mathbf{ret}}\ a) = \mathbf{merge}^{\bot}(\mathbf{split}^{\bot}\widetilde{a}) \equiv \widetilde{a}$.

- $\mathbf{join}^{\ell_1,\ell_2}(\mathbf{fork}^{\ell_1,\ell_2}\ a) \equiv a$.

  Now,

$$
\begin{aligned}
&\widetilde{\mathbf{join}^{\ell_1,\ell_2}(\mathbf{fork}^{\ell_1,\ell_2}\ a)}\\
=&\mathbf{split}^{\ell_1 \sqcup \ell_2}(\mathbf{merge}^{\ell_2}(\mathbf{merge}^{\ell_1}\widetilde{\mathbf{fork}^{\ell_1,\ell_2}}\ a))\\
=&\mathbf{split}^{\ell_1 \sqcup \ell_2}(\mathbf{merge}^{\ell_2}(\mathbf{merge}^{\ell_1}(\mathbf{split}^{\ell_1}(\mathbf{split}^{\ell_2}(\mathbf{merge}^{\ell_1 \sqcup \ell_2}\widetilde{a})))))\\
\equiv&\mathbf{split}^{\ell_1 \sqcup \ell_2}(\mathbf{merge}^{\ell_2}(\mathbf{split}^{\ell_2}(\mathbf{merge}^{\ell_1 \sqcup \ell_2}\widetilde{a})))
\end{aligned}
$$

$$\equiv \mathbf{split}^{\ell_1 \sqcup \ell_2}(\mathbf{merge}^{\ell_1 \sqcup \ell_2} \widetilde{a}) \equiv \widetilde{a}$$

- $\mathbf{fork}^{\ell_1,\ell_2}(\mathbf{join}^{\ell_1,\ell_2}\, a) \equiv a$.

  Now,

$$\begin{aligned}
&\widetilde{\mathbf{fork}^{\ell_1,\ell_2}(\mathbf{join}^{\ell_1,\ell_2}\, a)} \\
=&\mathbf{split}^{\ell_1}(\mathbf{split}^{\ell_2}(\mathbf{merge}^{\ell_1 \sqcup \ell_2}\widetilde{\mathbf{join}^{\ell_1,\ell_2}\, a})) \\
=&\mathbf{split}^{\ell_1}(\mathbf{split}^{\ell_2}(\mathbf{merge}^{\ell_1 \sqcup \ell_2}(\mathbf{split}^{\ell_1 \sqcup \ell_2}(\mathbf{merge}^{\ell_2}(\mathbf{merge}^{\ell_1}\widetilde{a}))))) \\
\equiv&\mathbf{split}^{\ell_1}(\mathbf{split}^{\ell_2}(\mathbf{merge}^{\ell_2}(\mathbf{merge}^{\ell_1}\widetilde{a}))) \\
\equiv&\mathbf{split}^{\ell_1}(\mathbf{merge}^{\ell_1}\widetilde{a}) \equiv \widetilde{a}
\end{aligned}$$

□

Theorem A.34 (Theorem 2.29) If $\Gamma \vdash a :^n A$ in λ°, then $\widehat{\Gamma} \vdash \widehat{a} :^n \widehat{A}$ in $\text{GMCC}_{\text{e}}(\mathcal{N})$. Further, if $\Gamma \vdash a_1 :^n A$ and $\Gamma \vdash a_2 :^n A$ such that $a_1 \equiv a_2$ in λ°, then $\widehat{a_1} \equiv \widehat{a_2}$ in $\text{GMCC}_{\text{e}}(\mathcal{N})$.

Proof. The first part follows by induction on the typing derivation. The second part follows by inversion on the equality judgment. □

Theorem A.35 (Theorem 2.30) If $\Gamma \vdash a :^n A$ in $\text{GMCC}_{\text{e}}(\mathcal{N})$, then $\underset{\smile}{\Gamma} \vdash \underset{\smile}{a} :^n \underset{\smile}{A}$ in λ°, extended with products. Further, if $\Gamma \vdash a_1 :^n A$ and $\Gamma \vdash a_2 :^n A$ such that $a_1 \equiv a_2$ in $\text{GMCC}_{\text{e}}(\mathcal{N})$, then $\underset{\smile}{a_1} \equiv \underset{\smile}{a_2}$ in λ°, extended with products.

Proof. The first part follows by induction on the typing derivation. The second part follows by inversion on the equality judgment. □

Theorem A.36 (Theorem 2.31) If $\Gamma \vdash a :^n A$ in λ°, then $\underset{\smile}{\hat{a}} = a$.

Proof. First note that for any type A, $\underset{\smile}{\hat{A}} = A$.
The proof then follows by induction on the typing derivation. □

A.10 Proof of a Proposition Stated in Section 2.11

Proposition A.37. The exponential object in $\mathcal{DC}$ does not satisfy the universal property, unless the relations in the definition of $\text{Obj}(\mathcal{DC})$ are restricted to reflexive ones only.

Proof. An object of $\mathcal{DC}$ is a pair, $X := (|X|, R_{X,\ell})$, where $|X|$ is a set and $R_{X,\ell}$ is a family of binary relations on $|X|$, indexed by elements, ℓ, of the parametrizing lattice, $\mathcal{L}$. (Strictly speaking, an object is a pair consisting of a cpo and a family of directed-complete relations. However, since we are not considering nonterminating

computations, we can simplify it to a set and a family of relations.) A morphism from X to Y is any function from $|X|$ to $|Y|$ that respects the binary relations, i.e. a function $h : |X| \to |Y|$ such that if $(x_1, x_2) \in R_{X,\ell}$, then $(h\,x_1, h\,x_2) \in R_{Y,\ell}$.
A product object is defined in $\mathcal{DC}$ as follows:

$$X \times Y := (|X| \times |Y|,\ \{((x_1, y_1), (x_2, y_2)) \mid (x_1, x_2) \in R_{X,\ell} \wedge (y_1, y_2) \in R_{Y,\ell}\}$$

An exponential object is defined as:

$$X \Rightarrow Y := (\mathrm{Hom}_{\mathcal{DC}}(X, Y),\ \{(f, g) \mid \forall (x_1, x_2) \in R_{X,\ell}, (f\,x_1, g\,x_2) \in R_{Y,\ell}\})$$

Now we present our counter-example. Let the parametrizing lattice $\mathcal{L}$ be $\mathbf{L} \sqsubset \mathbf{H}$. For a set A, let *id* and *true* denote the identity relation on A and the total relation on A respectively. Now define the following objects in $\mathcal{DC}$:

$$\begin{aligned} X &:= (\{x_1, x_2\},\ R_{X,\mathbf{L}} = R_{X,\mathbf{H}} = \{(x_1, x_1)\}) \\ Y &:= (\{y_1, y_2\},\ R_{Y,\mathbf{L}} = \mathit{true} \wedge R_{Y,\mathbf{H}} = \mathit{id}) \\ Z &:= (\{z_1, z_2\},\ R_{Z,\mathbf{L}} = R_{Z,\mathbf{H}} = \mathit{id}) \end{aligned}$$

Then, $\mathrm{Hom}_{\mathcal{DC}}(X \times Y, Z)$ has 8 elements but $\mathrm{Hom}_{\mathcal{DC}}(X, Y \Rightarrow Z)$ has only 4 elements. Therefore, $Y \Rightarrow Z$ does not satisfy the universal property. Hence, category $\mathcal{DC}$, as presented by Abadi et al. [1999], is not cartesian closed.

An analysis of the above counter-example shows that the problem stems from the fact that $R_{X,\ell}$ does not relate x_2 to itself. In fact, when we set $R_{X,\mathbf{L}} = R_{X,\mathbf{H}} = id$, both $\mathrm{Hom}_{\mathcal{DC}}(X \times Y, Z)$ and $\mathrm{Hom}_{\mathcal{DC}}(X, Y \Rightarrow Z)$ have 4 elements. Below, we show that if for every object X, the relation $R_{X,\ell}$ is required to be reflexive, then the exponential object indeed satisfies the universal property.

Let $X, Y, Z \in \mathrm{Obj}(\mathcal{DC})$. Define the exponential object, $X \Rightarrow Y$, as above. Need to check that for any $h \in \mathrm{Hom}_{\mathcal{DC}}(X, Y)$, we have, $(h, h) \in R_{X \Rightarrow Y,\ell}$.
Suppose, $(x_1, x_2) \in R_{X,\ell}$. Need to show: $(h\,x_1, h\,x_2) \in R_{Y,\ell}$.
But this is true because $h \in \mathrm{Hom}_{\mathcal{DC}}(X, Y)$.

Next, we define app as:

$$(X \Rightarrow Y) \times X \xrightarrow{\lambda w.(\pi_1\,w)\,(\pi_2\,w)} Y$$

Need to check that app is a $\mathcal{DC}$-morphism.
Suppose, $(w, w') \in R_{(X \Rightarrow Y) \times X,\ell}$. Then $(\pi_1\,w, \pi_1\,w') \in R_{X \Rightarrow Y,\ell}$ and $(\pi_2\,w, \pi_2\,w') \in R_{X,\ell}$.
Therefore, $((\pi_1\,w)\,(\pi_2\,w), (\pi_1\,w')\,(\pi_2\,w')) \in R_{Y,\ell}$.

Now, say $h \in \mathrm{Hom}_{\mathcal{DC}}(Z \times X, Y)$. We define $\Lambda h \in \mathrm{Hom}_{\mathcal{DC}}(Z, X \Rightarrow Y)$ as:

$$\Lambda h \triangleq \lambda z.\lambda x.h\ (z, x)$$

Need to check the following:

- First that Λh is well-defined.
  For any $z_0 \in |Z|$, need to show: $(\Lambda h)\ z_0 \in |X \Rightarrow Y|$, i.e. $\lambda x.h\ (z_0, x) \in \mathrm{Hom}_{\mathcal{DC}}(X, Y)$.
  Suppose $(x_1, x_2) \in R_{X,\ell}$. Need to show: $(h\ (z_0, x_1), h\ (z_0, x_2)) \in R_{Y,\ell}$.
  Since $R_{Z,\ell}$ is reflexive, $(z_0, z_0) \in R_{Z,\ell}$. Therefore, $((z_0, x_1), (z_0, x_2)) \in R_{Z\times X,\ell}$.
  This implies that $(h\ (z_0, x_1), h\ (z_0, x_2)) \in R_{Y,\ell}$ [$\because h \in \mathrm{Hom}_{\mathcal{DC}}(Z \times X, Y)$].
- Next, that Λh is a $\mathcal{DC}$-morphism.
  For $(z_1, z_2) \in R_{Z,\ell}$, need to show $(\lambda x.h\ (z_1, x), \lambda x.h\ (z_2, x)) \in R_{X\Rightarrow Y,\ell}$.
  Suppose $(x_1, x_2) \in R_{X,\ell}$. Need to show: $(h\ (z_1, x_1), h\ (z_2, x_2)) \in R_{Y,\ell}$.
  From what we are given, $((z_1, x_1), (z_2, x_2)) \in R_{Z\times X,\ell}$.
  As such, $(h\ (z_1, x_1), h\ (z_2, x_2)) \in R_{Y,\ell}$ [$\because h \in \mathrm{Hom}_{\mathcal{DC}}(Z \times X, Y)$].

Now we check that Λh satisfies the standard existence and uniqueness properties:

- Existence.

$$\begin{aligned}
&\mathrm{app} \circ (\Lambda h \times \mathrm{id})\\
=&\lambda v.\mathrm{app}\ (\Lambda h\ (\pi_1\ v), \pi_2\ v)\\
=&\lambda v.(\lambda w.(\pi_1\ w)\ (\pi_2\ w))\ ((\lambda z.\lambda x.h\ (z, x))\ (\pi_1\ v), \pi_2\ v)\\
=&\lambda v.(\lambda z.\lambda x.h\ (z, x))\ (\pi_1\ v)\ (\pi_2\ v)\\
=&\lambda v.h\ (\pi_1\ v, \pi_2\ v) = \lambda v.h\ v = h
\end{aligned}$$

- Uniqueness.
  Suppose, $h' \in \mathrm{Hom}_{\mathcal{DC}}(Z, X \Rightarrow Y)$ such that $\mathrm{app} \circ (h' \times \mathrm{id}) = h$.
  Then,
  $$h\ (z, x) = (\mathrm{app} \circ h' \times \mathrm{id})\ (z, x) = (\lambda w.(\pi_1\ w)\ (\pi_2\ w))\ (h'\ z, x) = h'\ z\ x$$
  But, $h\ (z, x) = \Lambda h\ z\ x$.
  So, by function extensionality, $h' = \Lambda h$.

□

Appendix B

Dependency Analysis in Pure Type Systems

Note: Many of the proofs presented in this appendix have also been mechanized in Coq: see Choudhury et al. [2022c].

B.1 Proofs of Lemmas/Theorems Stated in Section 3.2

Lemma B.1 (Lemma 3.1) If $\Omega' \vdash a :^{\ell} A$ and $\Omega \sqsubseteq \Omega'$, then $\Omega \vdash a :^{\ell} A$.

Proof. By induction on $\Omega' \vdash a :^{\ell} A$. The proof is straightforward. Still, we present some of the cases below.

- Rule SDC-Var. Have: $\Omega' \vdash x :^{\ell'} A$ where $x :^{\ell'_0} A \in \Omega'$ and $\ell'_0 \sqsubseteq \ell'$. Further, $\Omega \sqsubseteq \Omega'$.
  Need to show: $\Omega \vdash x :^{\ell'} A$.
  Since $\Omega \sqsubseteq \Omega'$, there exists ℓ_0 such that $x :^{\ell_0} A \in \Omega$ and $\ell_0 \sqsubseteq \ell'_0$. Then, by transitivity, $\ell_0 \sqsubseteq \ell'$.
  This case, then, follows by rule SDC-Var.
- Rule SDC-Lam. Have: $\Omega' \vdash \lambda x : A.b :^{\ell} A \to B$ where $\Omega', x :^{\ell} A \vdash b :^{\ell} B$. Further, $\Omega \sqsubseteq \Omega'$.
  Need to show: $\Omega \vdash \lambda x : A.b :^{\ell} A \to B$.
  Since $\Omega \sqsubseteq \Omega'$, we have, $\Omega, x :^{\ell} A \sqsubseteq \Omega', x :^{\ell} A$.
  By IH, $\Omega, x :^{\ell} A \vdash b :^{\ell} B$.
  This case, then, follows by rule SDC-Lam.
- Rule SDC-App. Have: $\Omega' \vdash b\ a :^{\ell} B$ where $\Omega' \vdash b :^{\ell} A \to B$ and $\Omega' \vdash a :^{\ell} A$. Further, $\Omega \sqsubseteq \Omega'$.
  Need to show: $\Omega \vdash b\ a :^{\ell} B$.
  By IH, $\Omega \vdash b :^{\ell} A \to B$ and $\Omega \vdash a :^{\ell} A$.
  This case, then, follows by rule SDC-App.

- Rule SDC-Lock. Have: $\Omega' \vdash \mathbf{lock}^{\ell_0}\ a :^{\ell} S_{\ell_0}\, A$ where $\Omega' \vdash a :^{\ell \sqcup \ell_0} A$. Further, $\Omega \sqsubseteq \Omega'$.
  Need to show: $\Omega \vdash \mathbf{lock}^{\ell_0}\ a :^{\ell} S_{\ell_0}\, A$.
  By IH, $\Omega \vdash a :^{\ell \sqcup \ell_0} A$.
  This case, then, follows by rule SDC-Lock.

- Rule SDC-Unlock. Have: $\Omega' \vdash \mathbf{unlock}^{\ell_0}\, x = a\, \mathbf{in}\, b :^{\ell} B$ where $\Omega' \vdash a :^{\ell} S_{\ell_0}\, A$ and $\Omega', x :^{\ell \sqcup \ell_0} A \vdash b :^{\ell} B$. Further, $\Omega \sqsubseteq \Omega'$.
  Need to show: $\Omega \vdash \mathbf{unlock}^{\ell_0}\, x = a\, \mathbf{in}\, b :^{\ell} B$.
  By IH, $\Omega \vdash a :^{\ell} S_{\ell_0}\, A$ and $\Omega, x :^{\ell \sqcup \ell_0} A \vdash b :^{\ell} B$.
  This case, then, follows by rule SDC-Unlock.

□

Lemma B.2 (Lemma 3.3) If $\Omega \vdash a :^{\ell} A$, then $m \sqcup \Omega \vdash a :^{m \sqcup \ell} A$.

Proof. By induction on $\Omega \vdash a :^{\ell} A$. The proof is straightforward. Still, we present some of the cases below.

- Rule SDC-Var. Have: $\Omega \vdash x :^{\ell} A$ where $x :^{\ell_0} A \in \Omega$ and $\ell_0 \sqsubseteq \ell$.
  Need to show $m \sqcup \Omega \vdash x :^{m \sqcup \ell} A$.
  Since $\ell_0 \sqsubseteq \ell$, so $m \sqcup \ell_0 \sqsubseteq m \sqcup \ell$.
  This case, then, follows by rule SDC-Var.

- Rule SDC-Lam. Have: $\Omega \vdash \lambda x{:}A.b :^{\ell} A \to B$ where $\Omega, x :^{\ell} A \vdash b :^{\ell} B$.
  Need to show: $m \sqcup \Omega \vdash \lambda x{:}A.b :^{m \sqcup \ell} B$.
  By IH, $m \sqcup \Omega, x :^{m \sqcup \ell} A \vdash b :^{m \sqcup \ell} B$.
  This case, then, follows by rule SDC-Lam.

- Rule SDC-App. Have: $\Omega \vdash b\ a :^{\ell} B$ where $\Omega \vdash b :^{\ell} A \to B$ and $\Omega \vdash a :^{\ell} A$.
  Need to show: $m \sqcup \Omega \vdash b\ a :^{m \sqcup \ell} B$.
  By IH, $m \sqcup \Omega \vdash b :^{m \sqcup \ell} A \to B$ and $m \sqcup \Omega \vdash a :^{m \sqcup \ell} A$.
  This case, then, follows by rule SDC-App.

- Rule SDC-Lock. Have: $\Omega \vdash \mathbf{lock}^{\ell_0}\ a :^{\ell} S_{\ell_0}\, A$ where $\Omega \vdash a :^{\ell \sqcup \ell_0} A$.
  Need to show: $m \sqcup \Omega \vdash \mathbf{lock}^{\ell_0}\ a :^{m \sqcup \ell} S_{\ell_0}\, A$.
  By IH, $m \sqcup \Omega \vdash a :^{m \sqcup (\ell \sqcup \ell_0)} A$. By associativity, $m \sqcup (\ell \sqcup \ell_0) = (m \sqcup \ell) \sqcup \ell_0$.
  This case, then, follows by rule SDC-Lock.

- Rule SDC-Unlock. Have: $\Omega \vdash \mathbf{unlock}^{\ell_0}\, x = a\, \mathbf{in}\, b :^{\ell} B$ where $\Omega \vdash a :^{\ell} S_{\ell_0}\, A$ and $\Omega, x :^{\ell \sqcup \ell_0} A \vdash b :^{\ell} B$.
  Need to show: $m \sqcup \Omega \vdash \mathbf{unlock}^{\ell_0}\, x = a\, \mathbf{in}\, b :^{m \sqcup \ell} B$.
  By IH, $m \sqcup \Omega \vdash a :^{m \sqcup \ell} S_{\ell_0}\, A$ and $m \sqcup \Omega, x :^{m \sqcup (\ell \sqcup \ell_0)} A \vdash b :^{m \sqcup \ell} B$.
  This case, then, follows by rule SDC-Unlock, using associativity of join.

□

Lemma B.3 (Lemma 3.2) If $\Omega_1, z :^{m_0} C, \Omega_2 \vdash a :^{\ell} A$ and $\ell_0 \sqsubseteq \ell$, then $\Omega_1, z :^{\ell_0 \sqcup m_0} C, \Omega_2 \vdash a :^{\ell} A$.

Proof. Let $\Omega_1, z :^{m_0} C, \Omega_2 \vdash a :^{\ell} A$ and $\ell_0 \sqsubseteq \ell$.
By Lemma B.2, $\ell_0 \sqcup \Omega_1, z :^{\ell_0 \sqcup m_0} C, \ell_0 \sqcup \Omega_2 \vdash a :^{\ell_0 \sqcup \ell} A$.
Now, since $\ell_0 \sqsubseteq \ell$, so $\ell_0 \sqcup \ell = \ell$.
Then, $\ell_0 \sqcup \Omega_1, z :^{\ell_0 \sqcup m_0} C, \ell_0 \sqcup \Omega_2 \vdash a :^{\ell} A$.
Next, $\Omega_1 \sqsubseteq \ell_0 \sqcup \Omega_1$ and $\Omega_2 \sqsubseteq \ell_0 \sqcup \Omega_2$.
So, by Lemma B.1, $\Omega_1, z :^{\ell_0 \sqcup m_0} C, \Omega_2 \vdash a :^{\ell} A$. □

Lemma B.4 (Lemma 3.4) If $\Omega \vdash a :^{\ell} A$ and $\ell \sqsubseteq m$, then $\Omega \vdash a :^{m} A$.

Proof. By induction on $\Omega \vdash a :^{\ell} A$. We present some of the interesting cases below.

- Rule SDC-Var. Have $\Omega \vdash x :^{\ell} A$ where $x :^{\ell_0} A \in \Omega$ and $\ell_0 \sqsubseteq \ell$. Further, $\ell \sqsubseteq m$.
  Need to show: $\Omega \vdash x :^{m} A$.
  This case follows by rule SDC-Var, via transitivity: $\ell_0 \sqsubseteq \ell \sqsubseteq m$.

- Rule SDC-Lam. Have: $\Omega \vdash \lambda x{:}A.b :^{\ell} A \to B$ where $\Omega, x :^{\ell} A \vdash b :^{\ell} B$. Further, $\ell \sqsubseteq m$.
  Need to show: $\Omega \vdash \lambda x{:}A.b :^{m} A \to B$.
  By IH, $\Omega, x :^{\ell} A \vdash b :^{m} B$.
  By Lemma B.3, $\Omega, x :^{m} A \vdash b :^{m} B$.
  This case, then, follows by rule SDC-Lam.

- Rule SDC-App. Have: $\Omega \vdash b\ a :^{\ell} B$ where $\Omega \vdash b :^{\ell} A \to B$ and $\Omega \vdash a :^{\ell} A$. Further, $\ell \sqsubseteq m$.
  Need to show: $\Omega \vdash b\ a :^{m} B$.
  By IH, $\Omega \vdash b :^{m} A \to B$ and $\Omega \vdash a :^{m} A$.
  This case, then, follows by rule SDC-App.

- Rule SDC-Lock. Have: $\Omega \vdash \mathbf{lock}^{\ell_0}\ a :^{\ell} S_{\ell_0} A$ where $\Omega \vdash a :^{\ell \sqcup \ell_0} A$. Further, $\ell \sqsubseteq m$.
  Need to show: $\Omega \vdash \mathbf{lock}^{\ell_0}\ a :^{m} S_{\ell_0} A$.
  Since $\ell \sqsubseteq m$, so $\ell \sqcup \ell_0 \sqsubseteq m \sqcup \ell_0$.
  By IH, $\Omega \vdash a :^{m \sqcup \ell_0} A$.
  This case, then, follows by rule SDC-Lock.

- Rule SDC-Unlock. Have: $\Omega \vdash \mathbf{unlock}^{\ell_0}\ x = a\ \mathbf{in}\ b :^{\ell} B$ where $\Omega \vdash a :^{\ell} S_{\ell_0} A$ and $\Omega, x :^{\ell \sqcup \ell_0} A \vdash b :^{\ell} B$.
  Further, $\ell \sqsubseteq m$.
  Need to show: $\Omega \vdash \mathbf{unlock}^{\ell_0}\ x = a\ \mathbf{in}\ b :^{m} B$.
  By IH, $\Omega \vdash a :^{m} S_{\ell_0} A$ and $\Omega, x :^{\ell \sqcup \ell_0} A \vdash b :^{m} B$.
  By Lemma B.3, $\Omega, x :^{m \sqcup \ell_0} A \vdash b :^{m} B$.
  This case, then, follows by rule SDC-Unlock.

□

Lemma B.5 (Lemma 3.5) If $\Omega_1, \Omega_2 \vdash a :^{\ell} A$, then $\Omega_1, z :^{m} C, \Omega_2 \vdash a :^{\ell} A$.

Proof. By straightforward induction on $\Omega_1, \Omega_2 \vdash a :^{\ell} A$. ☐

Lemma B.6 (Lemma 3.6) If $\Omega_1, z :^{m} C, \Omega_2 \vdash a :^{\ell} A$ and $\Omega_1 \vdash c :^{m} C$, then $\Omega_1, \Omega_2 \vdash a\{c/z\} :^{\ell} A$.

Proof. By induction on $\Omega_1, z :^{m} C, \Omega_2 \vdash a :^{\ell} A$. We present some of the interesting cases below.

- Rule SDC-Var. There are three cases to consider.
  - Have: $\Omega_{11}, z :^{m} C, \Omega_{12}, x :^{\ell_0} A, \Omega_2 \vdash x :^{\ell} A$ where $\ell_0 \sqsubseteq \ell$. Further, $\Omega_{11} \vdash c :^{m} C$.
    Need to show: $\Omega_{11}, \Omega_{12}, x :^{\ell_0} A, \Omega_2 \vdash x :^{\ell} A$.
    Follows by rule SDC-Var.
  - Have: $\Omega_1, x :^{\ell_0} A, \Omega_2 \vdash x :^{\ell} A$ where $\ell_0 \sqsubseteq \ell$. Further, $\Omega_1 \vdash a :^{\ell_0} A$.
    Need to show: $\Omega_1, \Omega_2 \vdash a :^{\ell} A$.
    By Lemma B.5, $\Omega_1, \Omega_2 \vdash a :^{\ell_0} A$.
    This case, then, follows by Lemma B.4.
  - Have: $\Omega_1, x :^{\ell_0} A, \Omega_{21}, z :^{m} C, \Omega_{22} \vdash x :^{\ell} A$ where $\ell_0 \sqsubseteq \ell$. Further, $\Omega_1, x :^{\ell_0} A, \Omega_{21} \vdash c :^{m} C$.
    Need to show: $\Omega_1, x :^{\ell_0} A, \Omega_{21}, \Omega_{22} \vdash x :^{\ell} A$.
    Follows by rule SDC-Var.

- Rule SDC-Lam. Have: $\Omega_1, z :^{m} C, \Omega_2 \vdash \lambda x : A.b :^{\ell} A \to B$ where $\Omega_1, z :^{m} C, \Omega_2, x :^{\ell} A \vdash b :^{\ell} B$. Further, $\Omega_1 \vdash c :^{m} C$.
  Need to show: $\Omega_1, \Omega_2 \vdash \lambda x : A.b\{c/z\} :^{\ell} A \to B$.
  By IH, $\Omega_1, \Omega_2, x :^{\ell} A \vdash b\{c/z\} :^{\ell} B$.
  This case, then, follows by rule ST-Lam.

- Rule SDC-App. Have $\Omega_1, z :^{m} C, \Omega_2 \vdash b\ a : B$ where $\Omega_1, z :^{m} C, \Omega_2 \vdash b :^{\ell} A \to B$ and $\Omega_1, z :^{m} C, \Omega_2 \vdash a :^{\ell} A$. Further, $\Omega_1 \vdash c :^{m} C$.
  Need to show: $\Omega_1, \Omega_2 \vdash b\{c/z\}\ a\{c/z\} :^{\ell} B$.
  By IH, $\Omega_1, \Omega_2 \vdash b\{c/z\} :^{\ell} A \to B$ and $\Omega_1, \Omega_2 \vdash a\{c/z\} :^{\ell} A$.
  This case, then, follows by rule SDC-App.

- Rule SDC-Lock. Have: $\Omega_1, z :^{m} C, \Omega_2 \vdash \mathbf{lock}^{\ell_0}\ a :^{\ell} S_{\ell_0} A$ where $\Omega_1, z :^{m} C, \Omega_2 \vdash a :^{\ell \sqcup \ell_0} A$. Further, $\Omega_1 \vdash c :^{m} C$.
  Need to show: $\Omega_1, \Omega_2 \vdash \mathbf{lock}^{\ell_0}\ a\{c/z\} :^{\ell} S_{\ell_0} A$.
  By IH, $\Omega_1, \Omega_2 \vdash a\{c/z\} :^{\ell \sqcup \ell_0} A$.
  This case, then, follows by rule SDC-Lock.

- Rule SDC-Unlock. Have: $\Omega_1, z :^{m} C, \Omega_2 \vdash \mathbf{unlock}^{\ell_0}\ x = a\ \mathbf{in}\ b :^{\ell} B$ where $\Omega_1, z :^{m} C, \Omega_2 \vdash a :^{\ell} S_{\ell_0} A$ and $\Omega_1, z :^{m} C, \Omega_2, x :^{\ell \sqcup \ell_0} A \vdash b :^{\ell} B$. Further, $\Omega_1 \vdash c :^{m} C$.
  Need to show: $\Omega_1, \Omega_2 \vdash \mathbf{unlock}^{\ell_0}\ x = a\{c/z\}\ \mathbf{in}\ b\{c/z\} :^{\ell} B$.

By IH, $\Omega_1, \Omega_2 \vdash a\{c/z\} :^{\ell} S_{\ell_0} A$ and $\Omega_1, \Omega_2, x :^{\ell \sqcup \ell_0} A \vdash b\{c/z\} :^{\ell} B$.
This case, then, follows by rule SDC-Unlock.

□

Theorem B.7 (Theorem 3.7) If $\Omega \vdash a :^{\ell} A$ and $\vdash a \leadsto a'$, then $\Omega \vdash a' :^{\ell} A$.

Proof. By induction on $\Omega \vdash a :^{\ell} A$ and inversion on $\vdash a \leadsto a'$. We present some of the interesting cases below.

- Rule SDC-App. Have: $\Omega \vdash b\ a :^{\ell} B$ where $\Omega \vdash b :^{\ell} A \to B$ and $\Omega \vdash a :^{\ell} A$. By inversion on $\vdash b\ a \leadsto c$:
  - $\vdash b\ a \leadsto b'\ a$, when $\vdash b \leadsto b'$.
    Need to show $\Omega \vdash b'\ a :^{\ell} B$.
    By IH, $\Omega \vdash b' :^{\ell} A \to B$.
    This case, then, follows by rule SDC-App.
  - $b = \lambda x{:}A'.b'$ and $\vdash (\lambda x{:}A'.b')\ a \leadsto b'\{a/x\}$.
    Need to show: $\Omega \vdash b'\{a/x\} :^{\ell} B$.
    By inversion on $\Omega \vdash \lambda x{:}A'.b' :^{\ell} A \to B$, we get, $A' = A$ and $\Omega, x :^{\ell} A \vdash b :^{\ell} B$.
    This case, then, follows by Lemma B.6.
- Rule SDC-Unlock. Have: $\Omega \vdash \textbf{unlock}^{\ell_0}\ x = a\ \textbf{in}\ b :^{\ell} B$ where $\Omega \vdash a :^{\ell} S_{\ell_0} A$ and $\Omega, x :^{\ell \sqcup \ell_0} A \vdash b :^{\ell} B$.
  By inversion on $\vdash \textbf{unlock}^{\ell_0}\ x = a\ \textbf{in}\ b \leadsto c$:
  - $\vdash \textbf{unlock}^{\ell_0}\ x = a\ \textbf{in}\ b \leadsto \textbf{unlock}^{\ell_0}\ x = a'\ \textbf{in}\ b$, when $\vdash a \leadsto a'$.
    Need to show: $\Omega \vdash \textbf{unlock}^{\ell_0}\ x = a'\ \textbf{in}\ b :^{\ell} B$.
    By IH, $\Omega \vdash a' :^{\ell} S_{\ell_0} A$.
    This case, then, follows by rule SDC-Unlock.
  - $a = \textbf{lock}^{\ell_0}\ a'$ and $\vdash \textbf{unlock}^{\ell_0}\ x = \textbf{lock}^{\ell_0}\ a'\ \textbf{in}\ b \leadsto b\{a'/x\}$.
    Need to show: $\Omega \vdash b\{a'/x\} :^{\ell} B$.
    By inversion on $\Omega \vdash \textbf{lock}^{\ell_0}\ a' :^{\ell} S_{\ell_0} A$, we get, $\Omega \vdash a' :^{\ell \sqcup \ell_0} A$.
    This case, then, follows by Lemma B.6.

□

Theorem B.8 (Theorem 3.8) If $\varnothing \vdash a :^{\ell} A$, then either a is a value or there exists a' such that $\vdash a \leadsto a'$.

Proof. By induction on $\varnothing \vdash a :^{\ell} A$. We present some of the interesting cases below.

- Rule SDC-App. Have: $\varnothing \vdash b\ a :^{\ell} B$ where $\varnothing \vdash b :^{\ell} A \to B$ and $\varnothing \vdash a :^{\ell} A$.
  Need to show: $\exists c, \vdash b\ a \leadsto c$.
  By IH, b is either a value or $\vdash b \leadsto b'$.

If b is a value, then $b = \lambda x{:}A'.b'$. Therefore, $\vdash b\ a \leadsto b'\{a/x\}$.
Otherwise, $\vdash b\ a \leadsto b'\ a$.

- Rule SDC-Unlock. Have: $\varnothing \vdash \mathbf{unlock}^{\ell_0}\ x = a\ \mathbf{in}\ b :^{\ell} B$ where $\varnothing \vdash a :^{\ell} S_{\ell_0}\ A$ and $x :^{\ell \sqcup \ell_0} A \vdash b :^{\ell} B$.
  Need to show: $\exists c, \vdash \mathbf{unlock}^{\ell_0}\ x = a\ \mathbf{in}\ b \leadsto c$.
  By IH, a is either a value or $\vdash a \leadsto a'$.
  If a is a value, then $a = \mathbf{lock}^{\ell_0}\ a'$. Therefore, $\vdash \mathbf{unlock}^{\ell_0}\ x = a\ \mathbf{in}\ b \leadsto b\{a'/x\}$.
  Otherwise, $\vdash \mathbf{unlock}^{\ell_0}\ x = a\ \mathbf{in}\ b \leadsto \mathbf{unlock}^{\ell_0}\ x = a'\ \mathbf{in}\ b$.

□

Lemma B.9 (Lemma 3.9) If $\Omega \vdash a :^{\ell} A$ then $\lceil \Omega \rceil \vdash a \sim_{\ell} a$.

Proof. By induction on $\Omega \vdash a :^{\ell} A$. All the cases are straightforward other than that of rule SDC-Lock.
In case of rule SDC-Lock, we have: $\Omega \vdash \mathbf{lock}^{\ell_0}\ a :^{\ell} S_{\ell_0}\ A$ where $\Omega \vdash a :^{\ell \sqcup \ell_0} A$.
We need to show: $\lceil \Omega \rceil \vdash \mathbf{lock}^{\ell_0}\ a \sim_{\ell} \mathbf{lock}^{\ell_0}\ a$.
First, we show that $\lceil \Omega \rceil \vdash^{\ell_0}_{\ell} a \sim a$.
If $\ell_0 \sqsubseteq \ell$, then by IH, $\lceil \Omega \rceil \vdash a \sim_{\ell} a$. Therefore, by rule EEq-Leq, $\lceil \Omega \rceil \vdash^{\ell_0}_{\ell} a \sim a$.
Otherwise, by rule EEq-NLeq, $\lceil \Omega \rceil \vdash^{\ell_0}_{\ell} a \sim a$.
Hence, $\lceil \Omega \rceil \vdash^{\ell_0}_{\ell} a \sim a$.
This case, then, follows by rule Ind-Lock. □

Lemma B.10 (Lemma 3.10) If $\Phi \vdash a_1 \sim_{\ell} a_2$, then $\Phi \vdash a_1 : \ell$ and $\Phi \vdash a_2 : \ell$.

Proof. By induction on $\Phi \vdash a_1 \sim_{\ell} a_2$. We present some of the interesting cases below.

- Rule Ind-Lam. Have: $\Phi \vdash \lambda x{:}A.b_1 \sim_{\ell} \lambda x{:}A.b_2$ where $\Phi, x{:}\ \ell \vdash b_1 \sim_{\ell} b_2$.
  Need to show: $\Phi \vdash \lambda x{:}A.b_1 : \ell$ and $\Phi \vdash \lambda x{:}A.b_2 : \ell$.
  By IH, $\Phi, x{:}\ \ell \vdash b_1 : \ell$ and $\Phi, x{:}\ \ell \vdash b_2 : \ell$.
  This case, then, follows by rule Ind-Lam.

- Rule Ind-App. Have: $\Phi \vdash b_1\ a_1 \sim_{\ell} b_2\ a_2$ where $\Phi \vdash b_1 \sim_{\ell} b_2$ and $\Phi \vdash a_1 \sim_{\ell} a_2$.
  Need to show: $\Phi \vdash b_1\ a_1 : \ell$ and $\Phi \vdash b_2\ a_2 : \ell$.
  By IH, $\Phi \vdash b_1 : \ell$ and $\Phi \vdash b_2 : \ell$.
  Again, by IH, $\Phi \vdash a_1 : \ell$ and $\Phi \vdash a_2 : \ell$.
  This case, then, follows by rule Ind-App.

- Rule Ind-Lock. Have: $\Phi \vdash \mathbf{lock}^{\ell_0}\ a_1 \sim_{\ell} \mathbf{lock}^{\ell_0}\ a_2$ where $\Phi \vdash^{\ell_0}_{\ell} a_1 \sim a_2$.
  Need to show: $\Phi \vdash \mathbf{lock}^{\ell_0}\ a_1 : \ell$ and $\Phi \vdash \mathbf{lock}^{\ell_0}\ a_2 : \ell$.
  Now, if $\ell_0 \sqsubseteq \ell$, then we have, $\Phi \vdash a_1 \sim_{\ell} a_2$. By IH, $\Phi \vdash a_1 : \ell$ and $\Phi \vdash a_2 : \ell$. This case, then, follows by rule CInd-Leq and rule Ind-Lock.
  Otherwise, by rule CInd-NLeq, $\Phi \vdash^{\ell_0}_{\ell} a_1 \sim a_1$ and $\Phi \vdash^{\ell_0}_{\ell} a_2 \sim a_2$. This case, then, follows by rule Ind-Lock.

- Rule $\textsc{Ind-Unlock}$. Have: $\Phi \vdash \textbf{unlock}^{\ell_0}\ x = a_1 \textbf{ in } b_1 \sim_\ell \textbf{unlock}^{\ell_0}\ x = a_2 \textbf{ in } b_2$ where $\Phi \vdash a_1 \sim_\ell a_2$ and $\Phi, x\!:\ell \sqcup \ell_0 \vdash b_1 \sim_\ell b_2$.
  Need to show: $\Phi \vdash \textbf{unlock}^{\ell_0}\ x = a_1 \textbf{ in } b_1 : \ell$ and $\Phi \vdash \textbf{unlock}^{\ell_0}\ x = a_2 \textbf{ in } b_2 : \ell$.
  By IH, $\Phi \vdash a_1 : \ell$ and $\Phi \vdash a_2 : \ell$.
  Again, by IH, $\Phi, x\!:\ell \sqcup \ell_0 \vdash b_1 : \ell$ and $\Phi, x\!:\ell \sqcup \ell_0 \vdash b_2 : \ell$.
  This case, then, follows by rule $\textsc{Ind-Unlock}$.

□

Lemma B.11 (Lemma 3.11) Indexed indistinguishability at ℓ is an equivalence relation on well-graded terms at ℓ.

Proof. Need to show that $\sim_\ell$ is an equivalence relation on well-graded terms at ℓ.
A term a is said to be well-graded if and only if $\Phi \vdash a \sim_\ell a$, for some Φ.
Therefore, by definition, the relation $\sim_\ell$ is reflexive on well-graded terms at ℓ.

Next, we show that $\sim_\ell$ is symmetric: if $\Phi \vdash a \sim_\ell b$, then $\Phi \vdash b \sim_\ell a$.
The proof follows by straightforward induction on $\Phi \vdash a \sim_\ell b$.

Finally, we show that $\sim_\ell$ is transitive: if $\Phi \vdash a \sim_\ell b$ and $\Phi \vdash b \sim_\ell c$, then $\Phi \vdash a \sim_\ell c$.
The proof is by induction on $\Phi \vdash a \sim_\ell b$ and subsequent inversion on $\Phi \vdash b \sim_\ell c$. We present some of the interesting cases below.

- Rule $\textsc{Ind-Var}$. Have: $\Phi \vdash x \sim_\ell x$ where $x\!:\!\ell_0 \in \Phi$ and $\ell_0 \sqsubseteq \ell$. Further $\Phi \vdash x \sim_\ell c$.
  Need to show: $\Phi \vdash x \sim_\ell c$.
  Follows from premises.

- Rule $\textsc{Ind-Lam}$. Have: $\Phi \vdash \lambda x\!:\!A.b_1 \sim_\ell \lambda x\!:\!A.b_2$ where $\Phi, x\!:\ell \vdash b_1 \sim_\ell b_2$. Further, $\Phi \vdash \lambda x\!:\!A.b_2 \sim_\ell c$.
  Need to show: $\Phi \vdash \lambda x\!:\!A.b_1 \sim_\ell c$.
  By inversion on $\Phi \vdash \lambda x\!:\!A.b_2 \sim_\ell c$, we get, $c = \lambda x\!:\!A.b_3$ and $\Phi, x\!:\ell \vdash b_2 \sim_\ell b_3$.
  By IH, $\Phi, x\!:\ell \vdash b_1 \sim_\ell b_3$.
  This case, then, follows by rule $\textsc{Ind-Lam}$.

- Rule $\textsc{Ind-App}$. Have: $\Phi \vdash b_1\ a_1 \sim_\ell b_2\ a_2$ where $\Phi \vdash b_1 \sim_\ell b_2$ and $\Phi \vdash a_1 \sim_\ell a_2$. Further, $\Phi \vdash b_2\ a_2 \sim_\ell c$.
  Need to show: $\Phi \vdash b_1\ a_1 \sim_\ell c$.
  By inversion on $\Phi \vdash b_2\ a_2 \sim_\ell c$, we get, $c = b_3\ a_3$ and $\Phi \vdash b_2 \sim_\ell b_3$ and $\Phi \vdash a_2 \sim_\ell a_3$.
  By IH, $\Phi \vdash b_1 \sim_\ell b_3$ and $\Phi \vdash a_1 \sim_\ell a_3$.
  This case, then, follows by rule SDC-$\textsc{App}$.

- Rule $\textsc{Ind-Lock}$. Have: $\Phi \vdash \textbf{lock}^{\ell_0}\ a_1 \sim_\ell \textbf{lock}^{\ell_0}\ a_2$ where $\Phi \vdash^{\ell_0}_{\ell} a_1 \sim a_2$. Further, $\Phi \vdash \textbf{lock}^{\ell_0}\ a_2 \sim_\ell c$.
  Need to show: $\Phi \vdash \textbf{lock}^{\ell_0}\ a_1 \sim_\ell c$.
  By inversion on $\Phi \vdash \textbf{lock}^{\ell_0}\ a_2 \sim_\ell c$, we get $c = \textbf{lock}^{\ell_0}\ a_3$ and $\Phi \vdash^{\ell_0}_{\ell} a_2 \sim a_3$.
  Now, if $\ell_0 \sqsubseteq \ell$, then $\Phi \vdash a_1 \sim_\ell a_2$ and $\Phi \vdash a_2 \sim_\ell a_3$. Then, by IH, $\Phi \vdash a_1 \sim_\ell a_3$. And by rule $\textsc{Eeq-Leq}$,

$\Phi \vdash^{\ell_0}_{\ell} a_1 \sim a_3$.
Otherwise, by rule Eeq-Nleq, $\Phi \vdash^{\ell_0}_{\ell} a_1 \sim a_3$.
Therefore, this case follows by rule Ind-Lock.

- Rule Ind-Unlock. Have: $\Phi \vdash \mathbf{unlock}^{\ell_0}\, x = a_1\, \mathbf{in}\, b_1 \sim_\ell \mathbf{unlock}^{\ell_0}\, x = a_2\, \mathbf{in}\, b_2$ where $\Phi \vdash a_1 \sim_\ell a_2$ and $\Phi, x\colon \ell \sqcup \ell_0 \vdash b_1 \sim_\ell b_2$. Further, $\Phi \vdash \mathbf{unlock}^{\ell_0}\, x = a_2\, \mathbf{in}\, b_2 \sim_\ell c$.
Need to show: $\Phi \vdash \mathbf{unlock}^{\ell_0}\, x = a_1\, \mathbf{in}\, b_1 \sim_\ell c$.
By inversion on $\Phi \vdash \mathbf{unlock}^{\ell_0}\, x = a_2\, \mathbf{in}\, b_2 \sim_\ell c$, we get, $c = \mathbf{unlock}^{\ell_0}\, x = a_3\, \mathbf{in}\, b_3$ and $\Phi \vdash a_2 \sim_\ell a_3$ and $\Phi, x\colon \ell \sqcup \ell_0 \vdash b_2 \sim_\ell b_3$.
By IH, $\Phi \vdash a_1 \sim_\ell a_3$ and $\Phi, x\colon \ell \sqcup \ell_0 \vdash b_1 \sim_\ell b_3$.
This case, then, follows by rule Ind-Unlock.

□

Lemma B.12 (Narrowing) If $\Phi_1, z\colon m, \Phi_2 \vdash a_1 \sim_\ell a_2$ and $m_0 \sqsubseteq m$, then $\Phi_1, z\colon m_0, \Phi_2 \vdash a_1 \sim_\ell a_2$.

Proof. By induction on $\Phi_1, z\colon m, \Phi_2 \vdash a_1 \sim_\ell a_2$. We present some of the interesting cases below.

- Rule Ind-Var. There are three cases to consider.
  - Have: $\Phi_{11}, z\colon m, \Phi_{12}, x\colon \ell_0, \Phi_2 \vdash x \sim_\ell x$ where $\ell_0 \sqsubseteq \ell$. Further, $m_0 \sqsubseteq m$.
    Need to show: $\Phi_{11}, z\colon m_0, \Phi_{12}, x\colon \ell_0, \Phi_2 \vdash x \sim_\ell x$.
    Follows by rule Ind-Var.
  - Have: $\Phi_1, x\colon \ell_0, \Phi_2 \vdash x \sim_\ell x$ where $\ell_0 \sqsubseteq \ell$. Further, $m_0 \sqsubseteq \ell_0$.
    Need to show: $\Phi_1, x\colon m_0, \Phi_2 \vdash x \sim_\ell x$.
    Since $m_0 \sqsubseteq \ell_0$ and $\ell_0 \sqsubseteq \ell$, we have, $m_0 \sqsubseteq \ell$.
    This case, then, follows by rule Ind-Var.
  - Have: $\Phi_1, x\colon \ell_0, \Phi_{21}, z\colon m, \Phi_{22} \vdash x \sim_\ell x$ where $\ell_0 \sqsubseteq \ell$. Further, $m_0 \sqsubseteq m$.
    Need to show: $\Phi_1, x\colon \ell_0, \Phi_{21}, z\colon m_0, \Phi_{22} \vdash x \sim_\ell x$.
    Follows by rule Ind-Var.

- Rule Ind-Lock. Have: $\Phi_1, z\colon m, \Phi_2 \vdash \mathbf{lock}^{\ell_0}\ a_1 \sim_\ell \mathbf{lock}^{\ell_0}\ a_2$ where $\Phi_1, z\colon m, \Phi_2 \vdash^{\ell_0}_{\ell} a_1 \sim a_2$. Further, $m_0 \sqsubseteq m$.
Need to show: $\Phi_1, z\colon m_0, \Phi_2 \vdash \mathbf{lock}^{\ell_0}\ a_1 \sim_\ell \mathbf{lock}^{\ell_0}\ a_2$.
If $\ell_0 \sqsubseteq \ell$, then we have, $\Phi_1, z\colon m, \Phi_2 \vdash a_1 \sim_\ell a_2$. By IH, $\Phi_1, z\colon m_0, \Phi_2 \vdash a_1 \sim_\ell a_2$. Then, by rule CInd-Leq, $\Phi_1, z\colon m_0, \Phi_2 \vdash^{\ell_0}_{\ell} a_1 \sim a_2$.
Otherwise, by rule CInd-Nleq, $\Phi_1, z\colon m_0, \Phi_2 \vdash^{\ell_0}_{\ell} a_1 \sim a_2$.
This case, then, follows by rule Ind-Lock.

- Rule Ind-Unlock. Have: $\Phi_1, z : m, \Phi_2 \vdash \mathbf{unlock}^{\ell_0}\, x = a_1\, \mathbf{in}\, b_1 \sim_\ell \mathbf{unlock}^{\ell_0}\, x = a_2\, \mathbf{in}\, b_2$ where $\Phi_1, z : m, \Phi_2 \vdash a_1 \sim_\ell a_2$ and $\Phi_1, z\colon m, \Phi_2, x\colon \ell \sqcup \ell_0 \vdash b_1 \sim_\ell b_2$. Further, $m_0 \sqsubseteq m$.
Need to show: $\Phi_1, z\colon m_0, \Phi_2 \vdash \mathbf{unlock}^{\ell_0}\, x = a_1\, \mathbf{in}\, b_1 \sim_\ell \mathbf{unlock}^{\ell_0}\, x = a_2\, \mathbf{in}\, b_2$.

By IH, $\Phi_1, z\!: m_0, \Phi_2 \vdash a_1 \sim_\ell a_2$.
Again by IH, $\Phi_1, z\!: m_0, \Phi_2, x\!: \ell \sqcup \ell_0 \vdash b_1 \sim_\ell b_2$.
This case, then, follows by rule IND-UNLOCK.

□

Lemma B.13 (Weakening) If $\Phi_1, \Phi_2 \vdash a_1 \sim_\ell a_2$, then $\Phi_1, z\!: m, \Phi_2 \vdash a_1 \sim_\ell a_2$.

Proof. By straightforward induction on $\Phi_1, \Phi_2 \vdash a_1 \sim_\ell a_2$. □

Lemma B.14 (Lemma 3.12) If $\Phi_1, z : m, \Phi_2 \vdash a_1 \sim_\ell a_2$ and $\Phi_1 \vdash^m_\ell c_1 \sim c_2$, then $\Phi_1, \Phi_2 \vdash a_1\{c_1/z\} \sim_\ell a_2\{c_2/z\}$.

Proof. By induction on $\Phi_1, z\!: m, \Phi_2 \vdash a_1 \sim_\ell a_2$. We present some of the interesting cases below.

- Rule IND-VAR. There are three cases to consider.
  - Have: $\Phi_{11}, z\!: m, \Phi_{12}, x\!: \ell_0, \Phi_2 \vdash x \sim_\ell x$ where $\ell_0 \sqsubseteq \ell$. Further, $\Phi_{11} \vdash^m_\ell c_1 \sim c_2$.
    Need to show: $\Phi_{11}, \Phi_{12}, x\!: \ell_0, \Phi_2 \vdash x \sim_\ell x$.
    Follows by rule IND-VAR.
  - Have: $\Phi_1, x\!: \ell_0, \Phi_2 \vdash x \sim_\ell x$ where $\ell_0 \sqsubseteq \ell$. Further, $\Phi_1 \vdash^{\ell_0}_\ell c_1 \sim c_2$.
    Need to show: $\Phi_1, \Phi_2 \vdash c_1 \sim_\ell c_2$.
    By inversion on $\Phi_1 \vdash^{\ell_0}_\ell c_1 \sim c_2$, we get $\Phi_1 \vdash c_1 \sim_\ell c_2$.
    This case, then, follows by Lemma B.13.
  - Have: $\Phi_1, x\!: \ell_0, \Phi_{21}, z\!: m, \Phi_{22} \vdash x \sim_\ell x$ where $\ell_0 \sqsubseteq \ell$. Further, $\Phi_1, x\!: \ell_0, \Phi_{21} \vdash^m_\ell c_1 \sim c_2$.
    Need to show: $\Phi_1, x\!: \ell_0, \Phi_{21}, \Phi_{22} \vdash x \sim_\ell x$.
    Follows by rule IND-VAR.
- Rule IND-LAM. Have: $\Phi_1, z\!: m, \Phi_2 \vdash \lambda x\!:\!A.b_1 \sim_\ell \lambda x\!:\!A.b_2$ where $\Phi_1, z\!: m, \Phi_2, x\!: \ell \vdash b_1 \sim_\ell b_2$. Further, $\Phi_1 \vdash^m_\ell c_1 \sim c_2$.
  Need to show: $\Phi_1, \Phi_2 \vdash \lambda x\!:\!A.b_1\{c_1/z\} \sim_\ell \lambda x\!:\!A.b_2\{c_2/z\}$.
  By IH, $\Phi_1, \Phi_2, x\!: \ell \vdash b_1\{c_1/z\} \sim_\ell b_2\{c_2/z\}$.
  This case, then, follows by rule IND-LAM.
- Rule IND-APP. Have: $\Phi_1, z : m, \Phi_2 \vdash b_1\ a_1 \sim_\ell b_2\ a_2$ where $\Phi_1, z : m, \Phi_2 \vdash b_1 \sim_\ell b_2$ and $\Phi_1, z : m, \Phi_2 \vdash a_1 \sim_\ell a_2$. Further, $\Phi_1 \vdash^m_\ell c_1 \sim c_2$.
  Need to show: $\Phi_1, \Phi_2 \vdash b_1\{c_1/z\}\ a_1\{c_1/z\} \sim_\ell b_2\{c_2/z\}\ a_2\{c_2/z\}$.
  By IH, $\Phi_1, \Phi_2 \vdash b_1\{c_1/z\} \sim_\ell b_2\{c_2/z\}$ and $\Phi_1, \Phi_2 \vdash a_1\{c_1/z\} \sim_\ell a_2\{c_2/z\}$.
  This case, then, follows by rule IND-APP.

- Rule Ind-Lock. Have: $\Phi_1, z : m, \Phi_2 \vdash \textbf{lock}^{\ell_0}\ a_1 \sim_\ell \textbf{lock}^{\ell_0}\ a_2$ where $\Phi_1, z : m, \Phi_2 \vdash^{\ell_0}_{\ell} a_1 \sim a_2$. Further, $\Phi_1 \vdash^{m}_{\ell} c_1 \sim c_2$.
  Need to show: $\Phi_1, \Phi_2 \vdash \textbf{lock}^{\ell_0}\ a_1\{c_1/z\} \sim_\ell \textbf{lock}^{\ell_0}\ a_2\{c_2/z\}$.
  If $\ell_0 \sqsubseteq \ell$, then $\Phi_1, z : m, \Phi_2 \vdash a_1 \sim_\ell a_2$. Then, by IH, $\Phi_1, \Phi_2 \vdash a_1\{c_1/z\} \sim_\ell a_2\{c_2/z\}$. By rule Eeq-Leq, $\Phi_1, \Phi_2 \vdash^{\ell_0}_{\ell} a_1\{c_1/z\} \sim a_2\{c_2/z\}$. This case, then, follows by rule Ind-Lock.
  If $\neg(\ell_0 \sqsubseteq \ell)$, by rule Eeq-Nleq, $\Phi_1, \Phi_2 \vdash^{\ell_0}_{\ell} a_1\{c_1/z\} \sim a_2\{c_2/z\}$. This case, then, follows by rule Ind-Lock.

- Rule Ind-Unlock. Have: $\Phi_1, z : m, \Phi_2 \vdash \textbf{unlock}^{\ell_0}\ x = a_1\ \textbf{in}\ b_1 \sim_\ell \textbf{unlock}^{\ell_0}\ x = a_2\ \textbf{in}\ b_2$ where $\Phi_1, z : m, \Phi_2 \vdash a_1 \sim_\ell a_2$ and $\Phi_1, z : m, \Phi_2, x : \ell \sqcup \ell_0 \vdash b_1 \sim_\ell b_2$.
  Need to show: $\Phi_1, \Phi_2 \vdash \textbf{unlock}^{\ell_0}\ x = a_1\{c_1/z\}\ \textbf{in}\ b_1\{c_1/z\} \sim_\ell \textbf{unlock}^{\ell_0}\ x = a_2\{c_2/z\}\ \textbf{in}\ b_2\{c_2/z\}$.
  By IH, $\Phi_1, \Phi_2 \vdash a_1\{c_1/z\} \sim_\ell a_2\{c_2/z\}$ and $\Phi_1, \Phi_2, x : \ell \sqcup \ell_0 \vdash b_1\{c_1/z\} \sim_\ell b_2\{c_2/z\}$.
  This case, then, follows by rule Ind-Unlock.

□

Theorem B.15 (Theorem 3.13) If $\Phi \vdash a_1 \sim_\ell a_2$ and $\vdash a_1 \leadsto a_1'$, then there exists a_2' such that $\vdash a_2 \leadsto a_2'$ and $\Phi \vdash a_1' \sim_\ell a_2'$.

Proof. By induction on $\Phi \vdash a_1 \sim_\ell a_2$ and subsequent inversion on $\vdash a_1 \leadsto a_1'$. We present some of the interesting cases below.

- Rule Ind-App. Have: $\Phi \vdash b_1\ a_1 \sim_\ell b_2\ a_2$ where $\Phi \vdash b_1 \sim_\ell b_2$ and $\Phi \vdash a_1 \sim_\ell a_2$. By inversion on $\vdash b_1\ a_1 \leadsto c_1'$:
  - $\vdash b_1\ a_1 \leadsto b_1'\ a_1$, when $\vdash b_1 \leadsto b_1'$.
    Need to show: $\exists c_2'$ such that $\vdash b_2\ a_2 \leadsto c_2'$ and $\Phi \vdash b_1'\ a_1 \sim_\ell c_2'$.
    By IH, $\exists b_2'$ such that $\vdash b_2 \leadsto b_2'$ and $\Phi \vdash b_1' \sim_\ell b_2'$.
    Therefore, $\vdash b_2\ a_2 \leadsto b_2'\ a_2$.
    And by rule Ind-App, $\Phi \vdash b_1'\ a_1 \sim_\ell b_2'\ a_2$.
  - $b_1 = \lambda x{:}A.b_1'$ and $\vdash b_1\ a_1 \leadsto b_1'\{a_1/x\}$.
    Need to show: $\exists c_2'$ such that $\vdash b_2\ a_2 \leadsto c_2'$ and $\Phi \vdash b_1'\{a_1/x\} \sim_\ell c_2'$.
    By inversion on $\Phi \vdash \lambda x{:}A.b_1' \sim_\ell b_2$, we get, $b_2 = \lambda x{:}A.b_2'$ and $\Phi, x : \ell \vdash b_1' \sim_\ell b_2'$.
    Then, $\vdash b_2\ a_2 \leadsto b_2'\{a_2/x\}$.
    Next, since $\Phi \vdash a_1 \sim_\ell a_2$, so $\Phi \vdash^{\ell}_{\ell} a_1 \sim a_2$.
    Therefore, by Lemma B.14, $\Phi \vdash b_1'\{a_1/x\} \sim_\ell b_2'\{a_2/x\}$.

- Rule Ind-Unlock. Have: $\Phi \vdash \textbf{unlock}^{\ell_0}\ x = a_1\ \textbf{in}\ b_1 \sim_\ell \textbf{unlock}^{\ell_0}\ x = a_2\ \textbf{in}\ b_2$ where $\Phi \vdash a_1 \sim_\ell a_2$ and $\Phi, x : \ell \sqcup \ell_0 \vdash b_1 \sim_\ell b_2$. By inversion on $\vdash \textbf{unlock}^{\ell_0}\ x = a_1\ \textbf{in}\ b_1 \leadsto c_1'$:
  - $\vdash \textbf{unlock}^{\ell_0}\ x = a_1\ \textbf{in}\ b_1 \leadsto \textbf{unlock}^{\ell_0}\ x = a_1'\ \textbf{in}\ b_1$, when $\vdash a_1 \leadsto a_1'$.
    Need to show: $\exists c_2'$ such that $\vdash \textbf{unlock}^{\ell_0}\ x = a_2\ \textbf{in}\ b_2 \leadsto c_2'$ and $\Phi \vdash \textbf{unlock}^{\ell_0}\ x = a_1'\ \textbf{in}\ b_1 \sim_\ell c_2'$.

By IH, $\exists a_2'$ such that $\vdash a_2 \rightsquigarrow a_2'$ and $\Phi \vdash a_1' \sim_\ell a_2'$.
Therefore, $\vdash \textbf{unlock}^{\ell_0}\ x = a_2 \textbf{ in } b_2 \rightsquigarrow \textbf{unlock}^{\ell_0}\ x = a_2' \textbf{ in } b_2$.
And by rule Ind-Unlock, $\Phi \vdash \textbf{unlock}^{\ell_0}\ x = a_1' \textbf{ in } b_1 \sim_\ell \textbf{unlock}^{\ell_0}\ x = a_2' \textbf{ in } b_2$.

– $a_1 = \textbf{lock}^{\ell_0}\ a_1'$ and $\vdash \textbf{unlock}^{\ell_0}\ x = \textbf{lock}^{\ell_0}\ a_1' \textbf{ in } b_1 \rightsquigarrow b_1\{a_1'/x\}$.
Need to show: $\exists c_2'$ such that $\vdash \textbf{unlock}^{\ell_0}\ x = a_2 \textbf{ in } b_2 \rightsquigarrow c_2'$ and $\Phi \vdash b_1\{a_1'/x\} \sim_\ell c_2'$.
By inversion on $\Phi \vdash \textbf{lock}^{\ell_0}\ a_1' \sim_\ell a_2$, we get $a_2 = \textbf{lock}^{\ell_0}\ a_2'$ and $\Phi \vdash_\ell^{\ell_0} a_1' \sim a_2'$.
Then, $\vdash \textbf{unlock}^{\ell_0}\ x = a_2 \textbf{ in } b_2 \rightsquigarrow b_2\{a_2'/x\}$.
Next, since $\Phi, x\!: \ell \sqcup \ell_0 \vdash b_1 \sim_\ell b_2$, by Lemma B.12, $\Phi, x\!: \ell_0 \vdash b_1 \sim_\ell b_2$.
Therefore, by Lemma B.14, $\Phi \vdash b_1\{a_1'/x\} \sim_\ell b_2\{a_2'/x\}$.

□

Theorem B.16 (Theorem 3.14) If $\vdash a \longmapsto a'$ in $\lambda^{[]}$, then $\vdash \overline{a} \longmapsto \overline{a'}$ in SDC.

Proof. By induction on $\vdash a \longmapsto a'$. We present some of the interesting cases below.

- $\vdash b\ a \longmapsto b'\ a$, when $\vdash b \longmapsto b'$.
  Need to show: $\vdash \overline{b}\ \overline{a} \longmapsto \overline{b'}\ \overline{a}$.
  By IH, $\vdash \overline{b} \longmapsto \overline{b'}$.
  Therefore, $\vdash \overline{b}\ \overline{a} \longmapsto \overline{b'}\ \overline{a}$.

- $\vdash b\ a \longmapsto b\ a'$, when $\vdash a \longmapsto a'$.
  Need to show: $\vdash \overline{b}\ \overline{a} \longmapsto \overline{b}\ \overline{a'}$.
  By IH, $\vdash \overline{a} \longmapsto \overline{a'}$.
  Therefore, $\vdash \overline{b}\ \overline{a} \longmapsto \overline{b}\ \overline{a'}$.

- $\vdash (\lambda x\!:\!A.b)\ a \longmapsto b\{a/x\}$.
  Need to show: $\vdash (\lambda x\!:\!A.\overline{b})\ \overline{a} \longmapsto \overline{b\{a/x\}}$.
  By β-rule, $\vdash (\lambda x\!:\!A.\overline{b})\ \overline{a} \longmapsto \overline{b}\{\overline{a}/x\}$.
  But $\overline{b\{a/x\}} = \overline{b}\{\overline{a}/x\}$ (can be shown by straightforward induction).

- $\vdash \textbf{unseal}^{\ell_0} a \longmapsto \textbf{unseal}^{\ell_0} a'$, when $\vdash a \longmapsto a'$.
  Need to show: $\vdash \textbf{unlock}^{\ell_0}\ x = \overline{a} \textbf{ in } x \longmapsto \textbf{unlock}^{\ell_0}\ x = \overline{a'} \textbf{ in } x$.
  Follows using IH.

- $\vdash \textbf{unseal}^{\ell_0}\textbf{seal}^{\ell_0}\ a \longmapsto a$.
  Need to show: $\vdash \textbf{unlock}^{\ell_0}\ x = \textbf{lock}^{\ell_0}\ \overline{a} \textbf{ in } x \longmapsto \overline{a}$.
  Follows by β-rule.

□

Theorem B.17 (Theorem 3.15) For any term a in $\lambda^{[]}$, if $\vdash \overline{a} \longmapsto b$ in SDC, then there exists a' in $\lambda^{[]}$ such that $\vdash a \longmapsto a'$ and $\overline{a'} = b$.

Proof. By induction on a and subsequent inversion on $\vdash \overline{a} \longmapsto b$. We present some of the interesting cases below.

- $a = a_2\ a_1$. By inversion on $\vdash \overline{a_2}\ \overline{a_1} \longmapsto b$:
  - $\vdash \overline{a_2}\ \overline{a_1} \longmapsto b_2\ \overline{a_1}$, when $\vdash \overline{a_2} \longmapsto b_2$.
    Need to show: $\exists a'$ such that $\vdash a_2\ a_1 \longmapsto a'$ and $\overline{a'} = b_2\ \overline{a_1}$.
    By IH, $\exists a_2'$ such that $\vdash a_2 \longmapsto a_2'$ and $\overline{a_2'} = b_2$.
    Therefore, $\vdash a_2\ a_1 \longmapsto a_2'\ a_1$ and $\overline{a_2'\ a_1} = \overline{a_2'}\ \overline{a_1} = b_2\ \overline{a_1}$.
  - $\vdash \overline{a_2}\ \overline{a_1} \longmapsto \overline{a_2}\ b_1$, when $\vdash \overline{a_1} \longmapsto b_1$.
    Need to show: $\exists a'$ such that $\vdash a_2\ a_1 \longmapsto a'$ and $\overline{a'} = \overline{a_2}\ b_1$.
    By IH, $\exists a_1'$ such that $\vdash a_1 \longmapsto a_1'$ and $\overline{a_1'} = b_1$.
    Therefore, $\vdash a_2\ a_1 \longmapsto a_2\ a_1'$ and $\overline{a_2\ a_1'} = \overline{a_2}\ \overline{a_1'} = \overline{a_2}\ b_1$.
  - $\overline{a_2} = \lambda x\!:\!A.b_2$ and $\vdash \overline{a_2}\ \overline{a_1} \longmapsto b_2\{\overline{a_1}/x\}$.
    Need to show: $\exists a'$ such that $\vdash a_2\ a_1 \longmapsto a'$ and $\overline{a'} = b_2\{\overline{a_1}/x\}$.
    Since $\overline{a_2} = \lambda x\!:\!A.b_2$, so $a_2 = \lambda x\!:\!A.a_2'$ (for some a_2') and $b_2 = \overline{a_2'}$.
    Therefore, $\vdash a_2\ a_1 \longmapsto a_2'\{a_1/x\}$ and $\overline{a_2'\{a_1/x\}} = \overline{a_2'}\{\overline{a_1}/x\} = b_2\{\overline{a_1}/x\}$.
- $a = \mathbf{unseal}^{\ell_0}\, a_0$. By inversion on $\vdash \mathbf{unlock}^{\ell_0}\ x = \overline{a_0}\,\mathbf{in}\,x \longmapsto b$:
  - $\vdash \mathbf{unlock}^{\ell_0}\ x = \overline{a_0}\,\mathbf{in}\,x \longmapsto \mathbf{unlock}^{\ell_0}\ x = b_0\,\mathbf{in}\,x$, when $\vdash \overline{a_0} \longmapsto b_0$.
    Need to show, $\exists a'$ such that $\vdash \mathbf{unseal}^{\ell_0}\, a_0 \longmapsto a'$ and $\overline{a'} = \mathbf{unlock}^{\ell_0}\ x = b_0\,\mathbf{in}\,x$.
    By IH, $\exists a_0'$ such that $\vdash a_0 \longmapsto a_0'$ and $\overline{a_0'} = b_0$.
    Therefore, $\vdash \mathbf{unseal}^{\ell_0}\, a_0 \longmapsto \mathbf{unseal}^{\ell_0}\, a_0'$ and $\overline{\mathbf{unseal}^{\ell_0}\, a_0'} = \mathbf{unlock}^{\ell_0}\ x = \overline{a_0'}\,\mathbf{in}\,x = \mathbf{unlock}^{\ell_0}\ x = b_0\,\mathbf{in}\,x$.
  - $\overline{a_0} = \mathbf{lock}^{\ell_0}\ b_0$ and $\vdash \mathbf{unlock}^{\ell_0}\ x = \mathbf{lock}^{\ell_0}\ b_0\,\mathbf{in}\,x \longmapsto b_0$.
    Need to show: $\exists a'$ such that $\vdash \mathbf{unseal}^{\ell_0}\, a_0 \longmapsto a'$ and $\overline{a'} = b_0$.
    Since $\overline{a_0} = \mathbf{lock}^{\ell_0}\ b_0$, so $a_0 = \mathbf{seal}^{\ell_0}\ a_0'$ (for some a_0') and $b_0 = \overline{a_0'}$.
    Therefore, $\vdash \mathbf{unseal}^{\ell_0}\, a_0 \longmapsto a_0'$ and $\overline{a_0'} = b_0$.

□

Theorem B.18 (Theorem 3.16) If $\Gamma \vdash a :^{\ell} A$, then $\Gamma^{\ell} \vdash \overline{a} :^{\ell} A$.

Proof. By induction on $\Gamma \vdash a :^{\ell} A$. We present some of the interesting cases below.

- Rule SC-Var. Have: $\Gamma_1, x : A, \Gamma_2 \vdash x :^{\ell} A$.
  Need to show: $\Gamma_1^{\ell}, x :^{\ell} A, \Gamma_2^{\ell} \vdash x :^{\ell} A$.
  Follows by rule SDC-Var.

- Rule SC-Lam. Have: $\Gamma \vdash \lambda x{:}A.b :^{\ell} A \to B$ where $\Gamma, x : A \vdash b :^{\ell} B$.
  Need to show: $\Gamma^{\ell} \vdash \lambda x{:}A.\overline{b} :^{\ell} A \to B$.
  By IH, $\Gamma^{\ell}, x{:}^{\ell} A \vdash \overline{b} :^{\ell} B$.
  This case, then, follows by rule SDC-Lam.

- Rule SC-App. Have: $\Gamma \vdash b\ a :^{\ell} B$ where $\Gamma \vdash b :^{\ell} A \to B$ and $\Gamma \vdash a :^{\ell} A$.
  Need to show: $\Gamma^{\ell} \vdash \overline{b}\ \overline{a} :^{\ell} B$.
  By IH, $\Gamma^{\ell} \vdash \overline{b} :^{\ell} A \to B$ and $\Gamma^{\ell} \vdash \overline{a} :^{\ell} A$.
  This case, then, follows by rule SDC-App.

- Rule SC-Seal. Have: $\Gamma \vdash \mathbf{seal}^{\ell_0}\ a :^{\ell} S_{\ell_0} A$ where $\Gamma \vdash a :^{\ell \sqcup \ell_0} A$.
  Need to show: $\Gamma^{\ell} \vdash \mathbf{lock}^{\ell_0}\ a :^{\ell} S_{\ell_0} A$.
  By IH, $\Gamma^{\ell \sqcup \ell_0} \vdash \overline{a} :^{\ell \sqcup \ell_0} A$.
  Now, $\Gamma^{\ell} \sqsubseteq \Gamma^{\ell \sqcup \ell_0}$. Therefore, by Lemma B.1, $\Gamma^{\ell} \vdash \overline{a} :^{\ell \sqcup \ell_0} A$.
  This case, then, follows by rule SDC-Lock.

- Rule SC-Unseal. Have: $\Gamma \vdash \mathbf{unseal}^{\ell_0} a :^{\ell} A$ where $\Gamma \vdash a :^{\ell} S_{\ell_0} A$ and $\ell_0 \sqsubseteq \ell$.
  Need to show: $\Gamma^{\ell} \vdash \mathbf{unlock}^{\ell_0}\ x = \overline{a}\, \mathbf{in}\, x :^{\ell} A$.
  By IH, $\Gamma^{\ell} \vdash \overline{a} :^{\ell} S_{\ell_0} A$.
  By rule SDC-Var, $\Gamma^{\ell}, x{:}^{\ell \sqcup \ell_0} A \vdash x :^{\ell} A$, since $\ell \sqcup \ell_0 \sqsubseteq \ell$.
  This case, then, follows by rule SDC-Unlock.

$\square$

B.2 Proofs of Lemmas/Theorems Stated in Section 3.3

Lemma B.19 (Lemma 3.17) If $\Omega \vdash\ a :^{\ell}\ A$, then $atm(cod(\Omega) \cup \{A\})\ \vec{:}^{\top} \ast, \overline{\overline{\Omega}} \vdash \overline{\overline{a}} :^{\ell} \overline{\overline{A}}$.

Proof. By induction on $\Omega \vdash a :^{\ell} A$. We present some of the interesting cases below.

- Rule SDC-Var. Have: $\Omega \vdash x :^{\ell} A$ where $x{:}^{\ell_0} A \in \Omega$ and $\ell_0 \sqsubseteq \ell$.
  Need to show: $atm(cod(\Omega) \cup \{A\})\ \vec{:}^{\top} \ast, \overline{\overline{\Omega}} \vdash x :^{\ell} \overline{\overline{A}}$.
  Follows by rule DCT-Var and weakening lemma.

- Rule SDC-Lam. Have: $\Omega \vdash \lambda x{:}A.b :^{\ell} A \to B$ where $\Omega, x{:}^{\ell} A \vdash b :^{\ell} B$.
  Need to show: $atm(cod(\Omega) \cup \{A \to B\})\ \vec{:}^{\top} \ast, \overline{\overline{\Omega}} \vdash \lambda x{:}^{\perp}\overline{\overline{A}}.\overline{\overline{b}} :^{\ell}\ {}^{\perp}\overline{\overline{A}} \to \overline{\overline{B}}$.
  By IH, $atm(cod(\Omega) \cup \{A\} \cup \{B\})\ \vec{:}^{\top} \ast, \overline{\overline{\Omega}}, x{:}^{\ell}\overline{\overline{A}} \vdash \overline{\overline{b}} :^{\ell} \overline{\overline{B}}$.
  This case, then, follows by rule DCT-Lam.

- Rule SDC-App. Have: $\Omega \vdash b\ a :^{\ell} B$ where $\Omega \vdash b :^{\ell} A \to B$ and $\Omega \vdash a :^{\ell} A$.
  Need to show: $atm(cod(\Omega) \cup \{B\})\ \vec{:}^{\top} \ast, \overline{\overline{\Omega}} \vdash \overline{\overline{b}}\ \overline{\overline{a}}^{\perp} :^{\ell} \overline{\overline{B}}$.

By IH, $atm(cod(\Omega) \cup \{A\} \cup \{B\}) \vdots^{\top} *, \overline{\overline{\Omega}} \vdash \overline{\overline{b}} :^{\ell} {}^{\bot}\overline{\overline{A}} \to \overline{\overline{B}}$ and $atm(cod(\Omega) \cup \{A\}) \vdots^{\top} *, \overline{\overline{\Omega}} \vdash \overline{\overline{a}} :^{\ell} \overline{\overline{A}}$.
By rule DCT-App, $atm(cod(\Omega) \cup \{A\} \cup \{B\}) \vdots^{\top} *, \overline{\overline{\Omega}} \vdash \overline{\overline{b}}\, \overline{\overline{a}}^{\bot} :^{\ell} \overline{\overline{B}}$.
Now, since $atm(cod(\Omega) \cup \{B\}) \vdots^{\top} *$ assigns sorts to all atomic types in Ω and B, so $atm(cod(\Omega) \cup \{B\}) \vdots^{\top} *, \overline{\overline{\Omega}} \vdash \overline{\overline{b}}\, \overline{\overline{a}}^{\bot} :^{\ell} \overline{\overline{B}}$.

- Rule SDC-Lock. Have: $\Omega \vdash \mathbf{lock}^{\ell_0}\ a :^{\ell} S_{\ell_0}\ A$ where $\Omega \vdash a :^{\ell \sqcup \ell_0} A$.
  Need to show: $atm(cod(\Omega) \cup \{A\}) \vdots^{\top} *, \overline{\overline{\Omega}} \vdash (\overline{\overline{a}}^{\ell_0}, \mathbf{unit}) :^{\ell} \Sigma x{:}^{\ell_0}\overline{\overline{A}}.\mathbf{Unit}$.
  By IH, $atm(cod(\Omega) \cup \{A\}) \vdots^{\top} *, \overline{\overline{\Omega}} \vdash \overline{\overline{a}} :^{\ell \sqcup \ell_0} \overline{\overline{A}}$.
  This case, then, follows by rule DCT-Pair.

- Rule SDC-Unlock. Have: $\Omega \vdash \mathbf{unlock}^{\ell_0}\ x = a\ \mathbf{in}\ b :^{\ell} B$ where $\Omega \vdash a :^{\ell} S_{\ell_0}\ A$ and $\Omega, x :^{\ell \sqcup \ell_0} A \vdash b :^{\ell} B$.
  Need to show: $atm(cod(\Omega) \cup \{B\}) \vdots^{\top} *, \overline{\overline{\Omega}} \vdash \mathbf{let}\ (x^{\ell_0}, y) = \overline{\overline{a}}\ \mathbf{in}\ \overline{\overline{b}} :^{\ell} \overline{\overline{B}}$.
  By IH, $atm(cod(\Omega) \cup \{A\}) \vdots^{\top} *, \overline{\overline{\Omega}} \vdash \overline{\overline{a}} :^{\ell} \Sigma x{:}^{\ell_0}\overline{\overline{A}}.\mathbf{Unit}$.
  Also, by IH, $atm(cod(\Omega) \cup \{A\} \cup \{B\}) \vdots^{\top} *, \overline{\overline{\Omega}}, x :^{\ell \sqcup \ell_0} \overline{\overline{A}} \vdash \overline{\overline{b}} :^{\ell} \overline{\overline{B}}$.
  By weakening, $atm(cod(\Omega) \cup \{A\} \cup \{B\}) \vdots^{\top} *, \overline{\overline{\Omega}}, x :^{\ell \sqcup \ell_0} \overline{\overline{A}}, y :^{\ell} \mathbf{Unit} \vdash \overline{\overline{b}} :^{\ell} \overline{\overline{B}}$.
  By rule DCT-LetPair, $atm(cod(\Omega) \cup \{A\} \cup \{B\}) \vdots^{\top} *, \overline{\overline{\Omega}} \vdash \mathbf{let}\ (x^{\ell_0}, y) = \overline{\overline{a}}\ \mathbf{in}\ \overline{\overline{b}} :^{\ell} \overline{\overline{B}}$.
  Now, since $atm(cod(\Omega) \cup \{B\}) \vdots^{\top} *$ assigns sorts to all atomic types in Ω and B, so $atm(cod(\Omega) \cup \{B\}) \vdots^{\top} *, \overline{\overline{\Omega}} \vdash \mathbf{let}\ (x^{\ell_0}, y) = \overline{\overline{a}}\ \mathbf{in}\ \overline{\overline{b}} :^{\ell} \overline{\overline{B}}$.

□

Lemma B.20 (Lemma 3.18) If $\vdash a \leadsto a'$ in SDC, then $\vdash \overline{\overline{a}} \leadsto \overline{\overline{a'}}$ in DDC$^{\top}$.

Proof. By induction on $\vdash a \leadsto a'$. We present the interesting cases below.

- Rule SDCStep-UnlockLeft. Have: $\vdash \mathbf{unlock}^{\ell_0}\ x = a\ \mathbf{in}\ b \leadsto \mathbf{unlock}^{\ell_0}\ x = a'\ \mathbf{in}\ b$ when $\vdash a \leadsto a'$.
  Need to show: $\vdash \mathbf{let}\ (x^{\ell_0}, y) = \overline{\overline{a}}\ \mathbf{in}\ \overline{\overline{b}} \leadsto \mathbf{let}\ (x^{\ell_0}, y) = \overline{\overline{a'}}\ \mathbf{in}\ \overline{\overline{b}}$.
  Follows by IH and left step rule for **let**.

- Rule SDCStep-UnlockBeta. Have: $\vdash \mathbf{unlock}^{\ell_0}\ x = \mathbf{lock}^{\ell_0}\ a\ \mathbf{in}\ b \leadsto b\{a/x\}$.
  Need to show: $\vdash \mathbf{let}\ (x^{\ell_0}, y) = (\overline{\overline{a}}^{\ell_0}, \mathbf{unit})\ \mathbf{in}\ \overline{\overline{b}} \leadsto \overline{\overline{b}}\{\overline{\overline{a}}/x\}$.
  Follows by beta step rule for **let** and freshness of y.

□

Lemma B.21 (Lemma 3.19) For any term a in SDC, if $\vdash \overline{\overline{a}} \leadsto b$ in DDC$^{\top}$, then there exists a' in SDC such that $\vdash a \leadsto a'$ and $\overline{\overline{a'}} = b$.

Proof. By induction on a and subsequent inversion on $\vdash \overline{\overline{a}} \leadsto b$. We present the **unlock**-case below.
We have: $a = \mathbf{unlock}^{\ell_0}\ x = a_1\ \mathbf{in}\ a_2$. Further, $\vdash \mathbf{let}\ (x^{\ell_0}, y) = \overline{\overline{a_1}}\ \mathbf{in}\ \overline{\overline{a_2}} \leadsto b$.
By inversion on $\vdash \mathbf{let}\ (x^{\ell_0}, y) = \overline{\overline{a_1}}\ \mathbf{in}\ \overline{\overline{a_2}} \leadsto b$:

- $\vdash \textbf{let}\ (x^{\ell_0}, y)\ =\ \overline{\overline{a_1}}\ \textbf{in}\ \overline{\overline{a_2}} \rightsquigarrow \textbf{let}\ (x^{\ell_0}, y)\ =\ b_1\ \textbf{in}\ \overline{\overline{a_2}}$ when $\vdash \overline{\overline{a_1}} \rightsquigarrow b_1$.
  Need to show: $\exists a'$ such that $\vdash \textbf{unlock}^{\ell_0}\ x = a_1\ \textbf{in}\ a_2 \rightsquigarrow a'$ and $\overline{\overline{a'}} = \textbf{let}\ (x^{\ell_0}, y)\ =\ b_1\ \textbf{in}\ \overline{\overline{a_2}}$.
  By IH, $\exists a'_1$ such that $\vdash a_1 \rightsquigarrow a'_1$ and $\overline{\overline{a'_1}} = b_1$.
  Then, $\vdash \textbf{unlock}^{\ell_0}\ x = a_1\ \textbf{in}\ a_2 \rightsquigarrow \textbf{unlock}^{\ell_0}\ x = a'_1\ \textbf{in}\ a_2$.
  And $\overline{\overline{\textbf{unlock}^{\ell_0}\ x = a'_1\ \textbf{in}\ a_2}} = \textbf{let}\ (x^{\ell_0}, y)\ =\ \overline{\overline{a'_1}}\ \textbf{in}\ \overline{\overline{a_2}} = \textbf{let}\ (x^{\ell_0}, y)\ =\ b_1\ \textbf{in}\ \overline{\overline{a_2}}$.

- $\overline{\overline{a_1}} = ({a_{11}}^{\ell_0}, a_{12})$ and $\vdash \textbf{let}\ (x^{\ell_0}, y)\ =\ ({a_{11}}^{\ell_0}, a_{12})\ \textbf{in}\ \overline{\overline{a_2}} \rightsquigarrow \overline{\overline{a_2}}\{a_{11}/x\}\{a_{12}/y\}$.
  Need to show: $\exists a'$ such that $\vdash \textbf{unlock}^{\ell_0}\ x = a_1\ \textbf{in}\ a_2 \rightsquigarrow a'$ and $\overline{\overline{a'}} = \overline{\overline{a_2}}\{a_{11}/x\}\{a_{12}/y\}$.
  Since y is fresh, $\overline{\overline{a_2}}\{a_{11}/x\}\{a_{12}/y\} = \overline{\overline{a_2}}\{a_{11}/x\}$.
  Next, since $\overline{\overline{a_1}} = ({a_{11}}^{\ell_0}, a_{12})$, so $a_1 = \textbf{lock}^{\ell_0}\ a'_1$ (for some a'_1) and $\overline{\overline{a'_1}} = a_{11}$.
  Then, $\vdash \textbf{unlock}^{\ell_0}\ x = a_1\ \textbf{in}\ a_2 \rightsquigarrow a_2\{a'_1/x\}$.
  And $\overline{\overline{a_2\{a'_1/x\}}} = \overline{\overline{a_2}}\{\overline{\overline{a'_1}}/x\} = \overline{\overline{a_2}}\{a_{11}/x\} = \overline{\overline{a_2}}\{a_{11}/x\}\{a_{12}/y\}$.

□

Lemma B.22 (Narrowing for indistinguishability relation) If $\Phi_1, z : m, \Phi_2 \vdash a_1 \sim_\ell a_2$ and $m_0 \sqsubseteq m$, then $\Phi_1, z\!: m_0, \Phi_2 \vdash a_1 \sim_\ell a_2$.

Proof. By induction on $\Phi_1, z\!: m, \Phi_2 \vdash a_1 \sim_\ell a_2$. Follow proof of lemma B.12. □

Lemma B.23 (Substitution for indistinguishability relation) If $\Phi_1, z\!: m, \Phi_2 \vdash a_1 \sim_\ell a_2$ and $\Phi_1 \vdash^m_\ell c_1 \sim c_2$, then $\Phi_1, \Phi_2 \vdash a_1\{c_1/z\} \sim_\ell a_2\{c_2/z\}$.

Proof. By induction on $\Phi_1, z\!: m, \Phi_2 \vdash a_1 \sim_\ell a_2$. We present some of the interesting cases below.

- Rule IndD-Var. Similar to rule Ind-Var case of Lemma B.14.

- Rule IndD-Lam. Have: $\Phi_1, z\!: m, \Phi_2 \vdash \lambda x\!:^{\ell_0} A_1.b_1 \sim_\ell \lambda x\!:^{\ell_0} A_2.b_2$ where $\Phi_1, z\!: m, \Phi_2, x\!: \ell \sqcup \ell_0 \vdash b_1 \sim_\ell b_2$ and $\Phi_1, z\!: m, \Phi_2 \vdash^\top_\ell A_1 \sim A_2$. Further, $\Phi_1 \vdash^m_\ell c_1 \sim c_2$.
  Need to show: $\Phi_1, \Phi_2 \vdash \lambda x\!:^{\ell_0} A\{c_1/z\}.b_1\{c_1/z\} \sim_\ell \lambda x\!:^{\ell_0} A_2\{c/z\}.b_2\{c_2/z\}$.
  By IH, $\Phi_1, \Phi_2, x\!: \ell \sqcup \ell_0 \vdash b_1\{c_1/z\} \sim_\ell b_2\{c_2/z\}$.
  Next, if $\top \sqsubseteq \ell$, we have: $\Phi_1, z\!: m, \Phi_2 \vdash A_1 \sim_\ell A_2$. Then, by IH, $\Phi_1, \Phi_2 \vdash A_1\{c_1/z\} \sim_\ell A_2\{c_2/z\}$. And by rule CIndD-Leq, $\Phi_1, \Phi_2 \vdash^\top_\ell A_1\{c_1/z\} \sim A_2\{c_2/z\}$.
  Otherwise, by rule CIndD-Nleq, $\Phi_1, \Phi_2 \vdash^\top_\ell A_1\{c_1/z\} \sim A_2\{c_2/z\}$.
  This case, then, follows by rule IndD-Lam.

- Rule IndD-App. Have: $\Phi_1, z : m, \Phi_2 \vdash b_1\ {a_1}^{\ell_0} \sim_\ell b_2\ {a_2}^{\ell_0}$ where $\Phi_1, z : m, \Phi_2 \vdash b_1 \sim_\ell b_2$ and $\Phi_1, z : m, \Phi_2 \vdash^{\ell_0}_\ell a_1 \sim a_2$. Further, $\Phi_1 \vdash^m_\ell c_1 \sim c_2$.
  Need to show: $\Phi_1, \Phi_2 \vdash b_1\{c_1/z\}\ (a_1\{c_1/z\})^{\ell_0} \sim_\ell b_2\{c_2/z\}\ (a_2\{c_2/z\})^{\ell_0}$.
  By IH, $\Phi_1, \Phi_2 \vdash b_1\{c_1/z\} \sim_\ell b_2\{c_2/z\}$.
  Now, if $\ell_0 \sqsubseteq \ell$, we have: $\Phi_1, z\!: m, \Phi_2 \vdash a_1 \sim_\ell a_2$. Then, by IH, $\Phi_1, \Phi_2 \vdash a_1\{c_1/z\} \sim_\ell a_2\{c_2/z\}$. And by rule CIndD-Leq, $\Phi_1, \Phi_2 \vdash^{\ell_0}_\ell a_1\{c_1/z\} \sim a_2\{c_2/z\}$.

Otherwise, by rule CIndD-Nleq, $\Phi_1, \Phi_2 \vdash a_1\{c_1/z\} \sim_\ell a_2\{c_2/z\}$.
This case, then, follows by rule IndD-App.

□

Lemma B.24 (Lemma 3.20) If $\Phi \vdash a_1 \sim_\ell a_2$, then $\lfloor a_1 \rfloor_\ell = \lfloor a_2 \rfloor_\ell$.

Proof. By induction on $\Phi \vdash a_1 \sim_\ell a_2$. We present some of the interesting cases below.

- Rule IndD-Lam. Have: $\Phi \vdash \lambda x :^{\ell_0} A_1 . b_1 \sim_\ell \lambda x :^{\ell_0} A_2 . b_2$ where $\Phi, x : \ell \sqcup \ell_0 \vdash b_1 \sim_\ell b_2$ and $\Phi \vdash_\ell^\top A_1 \sim A_2$.
  Need to show: $\lfloor \lambda x :^{\ell_0} A_1 . b_1 \rfloor_\ell = \lfloor \lambda x :^{\ell_0} A_2 . b_2 \rfloor_\ell$.
  There are two possibilities.
  - $\ell = \top$. Then, we have, $\Phi \vdash A_1 \sim_\ell A_2$.
    By IH, $\lfloor b_1 \rfloor_\ell = \lfloor b_2 \rfloor_\ell$ and $\lfloor A_1 \rfloor_\ell = \lfloor A_2 \rfloor_\ell$.
    Then, $\lfloor \lambda x :^{\ell_0} A_1 . b_1 \rfloor_\ell = \lambda x :^{\ell_0} \lfloor A_1 \rfloor_\ell . \lfloor b_1 \rfloor_\ell = \lambda x :^{\ell_0} \lfloor A_2 \rfloor_\ell . \lfloor b_2 \rfloor_\ell = \lfloor \lambda x :^{\ell_0} A_2 . b_2 \rfloor_\ell$.
  - $\ell \neq \top$. Then, $\lfloor \lambda x :^{\ell_0} A_1 . b_1 \rfloor_\ell = \lambda x :^{\ell_0} \mathbf{Unit} . \lfloor b_1 \rfloor_\ell$ and $\lfloor \lambda x :^{\ell_0} A_2 . b_2 \rfloor_\ell = \lambda x :^{\ell_0} \mathbf{Unit} . \lfloor b_2 \rfloor_\ell$.
    This case, then, follows by IH.
- Rule IndD-App. Have: $\Phi \vdash b_1 \; {a_1}^{\ell_0} \sim_\ell b_2 \; {a_2}^{\ell_0}$ where $\Phi \vdash b_1 \sim_\ell b_2$ and $\Phi \vdash_\ell^{\ell_0} a_1 \sim a_2$.
  Need to show: $\lfloor b_1 \; {a_1}^{\ell_0} \rfloor_\ell = \lfloor b_2 \; {a_2}^{\ell_0} \rfloor_\ell$.
  There are two possibilities.
  - $\ell_0 \sqsubseteq \ell$. Then, we have, $\Phi \vdash a_1 \sim_\ell a_2$.
    By IH, $\lfloor b_1 \rfloor_\ell = \lfloor b_2 \rfloor_\ell$ and $\lfloor a_1 \rfloor_\ell = \lfloor a_2 \rfloor_\ell$.
    Then, $\lfloor b_1 \; {a_1}^{\ell_0} \rfloor_\ell = \lfloor b_1 \rfloor_\ell \; (\lfloor a_1 \rfloor_\ell)^{\ell_0} = \lfloor b_2 \rfloor_\ell \; (\lfloor a_2 \rfloor_\ell)^{\ell_0} = \lfloor b_2 \; {a_2}^{\ell_0} \rfloor_\ell$.
  - $\neg(\ell_0 \sqsubseteq \ell)$. Then, $\lfloor b_1 \; {a_1}^{\ell_0} \rfloor_\ell = \lfloor b_1 \rfloor_\ell \; \mathbf{unit}^{\ell_0}$ and $\lfloor b_2 \; {a_2}^{\ell_0} \rfloor_\ell = \lfloor b_2 \rfloor_\ell \; \mathbf{unit}^{\ell_0}$.
    This case, then, follows by IH.

□

Lemma B.25 (Lemma 3.21) If $\Phi \vdash a : \ell$, then $\Phi \vdash a \sim_\ell \lfloor a \rfloor_\ell$.

Proof. By induction on $\Phi \vdash a : \ell$. We present some of the interesting cases below.

- Rule IndD-Lam. Have: $\Phi \vdash \lambda x :^{\ell_0} A . b : \ell$ where $\Phi, x : \ell \sqcup \ell_0 \vdash b : \ell$ and $\Phi \vdash_\ell^\top A \sim A$.
  Need to show: $\Phi \vdash \lambda x :^{\ell_0} A . b \sim_\ell \lfloor \lambda x :^{\ell_0} A . b \rfloor_\ell$.
  There are two possibilities.
  - $\ell = \top$. Then, we have, $\Phi \vdash A : \ell$.
    By IH, $\Phi \vdash A \sim_\ell \lfloor A \rfloor_\ell$ and $\Phi, x : \ell \sqcup \ell_0 \vdash b \sim_\ell \lfloor b \rfloor_\ell$.
    By rule CIndD-Leq, $\Phi \vdash_\ell^\top A \sim \lfloor A \rfloor_\ell$.
    This case, then, follows by rule IndD-Lam.

- $\ell_0 \neq \top$. Then, $\lfloor \lambda x :^{\ell_0} A.b \rfloor_\ell = \lambda x :^{\ell_0} \mathbf{Unit}.\lfloor b \rfloor_\ell$.
  And by rule CIndD-Nleq, $\Phi \vdash^{\top}_{\ell} A \sim \mathbf{Unit}$.
  Next, by IH, $\Phi, x : \ell \sqcup \ell_0 \vdash b \sim_\ell \lfloor b \rfloor_\ell$.
  This case, then, follows by rule IndD-Lam.

- Rule IndD-App. Have: $\Phi \vdash b\ a^{\ell_0} : \ell$ where $\Phi \vdash b : \ell$ and $\Phi \vdash^{\ell_0}_{\ell} a \sim a$.
  Need to show: $\Phi \vdash b\ a^{\ell_0} \sim_\ell \lfloor b\ a^{\ell_0} \rfloor_\ell$.
  There are two possibilities:
  - $\ell_0 \sqsubseteq \ell$. Then, we have, $\Phi \vdash a : \ell$.
    By IH, $\Phi \vdash b \sim_\ell \lfloor b \rfloor_\ell$ and $\Phi \vdash a \sim_\ell \lfloor a \rfloor_\ell$.
    By rule CIndD-Leq, $\Phi \vdash^{\ell_0}_{\ell} a \sim \lfloor a \rfloor_\ell$.
    This case, then, follows by rule IndD-App.
  - $\neg(\ell_0 \sqsubseteq \ell)$. Then, $\lfloor b\ a^{\ell_0} \rfloor_\ell = \lfloor b \rfloor_\ell\ \mathbf{unit}^{\ell_0}$.
    And by rule CIndD-Nleq, $\Phi \vdash^{\ell_0}_{\ell} a \sim \mathbf{unit}$.
    Next, by IH, $\Phi \vdash b \sim_\ell \lfloor b \rfloor_\ell$.
    This case, then, follows by rule IndD-App.

□

Lemma B.26 (Lemma 3.22) If $\Phi \vdash a : \ell$ and $\vdash a \rightsquigarrow b$, then $\vdash \lfloor a \rfloor_\ell \rightsquigarrow \lfloor b \rfloor_\ell$. Otherwise, if a is a value, then so is $\lfloor a \rfloor_\ell$.

Proof. By induction on $\Phi \vdash a : \ell$ and subsequent inversion on $\vdash a \rightsquigarrow b$. We present some of the interesting cases below.

- Rule IndD-App. Have: $\Phi \vdash b\ a^{\ell_0} : \ell$ where $\Phi \vdash b : \ell$ and $\Phi \vdash^{\ell_0}_{\ell} a \sim a$. Further, $\vdash b\ a^{\ell_0} \rightsquigarrow c$.
  By inversion on $\vdash b\ a^{\ell_0} \rightsquigarrow c$:
  - $\vdash b\ a^{\ell_0} \rightsquigarrow b'\ a^{\ell_0}$ when $\vdash b \rightsquigarrow b'$.
    If $\ell_0 \sqsubseteq \ell$, we need to show, $\vdash \lfloor b \rfloor_\ell\ \lfloor a \rfloor^{\ell_0}_\ell \rightsquigarrow \lfloor b' \rfloor_\ell\ \lfloor a \rfloor^{\ell_0}_\ell$, which follows by IH.
    Otherwise, we need to show, $\vdash \lfloor b \rfloor_\ell\ \mathbf{unit}^{\ell_0} \rightsquigarrow \lfloor b' \rfloor_\ell\ \mathbf{unit}^{\ell_0}$, which also follows by IH.
  - $b = \lambda x :^{\ell_0} A.b'$ and $\vdash (\lambda x :^{\ell_0} A.b')\ a^{\ell_0} \rightsquigarrow b'\{a/x\}$.
    Now, there are three possibilities:
    * $\ell_0 \sqsubseteq \ell$ and $\ell = \top$. Need to show: $\vdash (\lambda x :^{\ell_0} \lfloor A \rfloor_\ell.\lfloor b' \rfloor_\ell)\ \lfloor a \rfloor^{\ell_0}_\ell \rightsquigarrow \lfloor b' \rfloor_\ell\{\lfloor a \rfloor_\ell/x\}$.
      Follows by β-rule for application.
    * $\ell_0 \sqsubseteq \ell$ and $\ell \neq \top$. Need to show: $\vdash (\lambda x :^{\ell_0} \mathbf{Unit}.\lfloor b' \rfloor_\ell)\ \lfloor a \rfloor^{\ell_0}_\ell \rightsquigarrow \lfloor b' \rfloor_\ell\{\lfloor a \rfloor_\ell/x\}$.
      Again, follows by β-rule for application.
    * $\neg(\ell_0 \sqsubseteq \ell)$. Need to show: $\vdash (\lambda x :^{\ell_0} \mathbf{Unit}.\lfloor b' \rfloor_\ell)\ \mathbf{unit}^{\ell_0} \rightsquigarrow \lfloor b' \rfloor_\ell\{\lfloor a \rfloor_\ell/x\}$.
      Now, $\vdash (\lambda x :^{\ell_0} \mathbf{Unit}.\lfloor b' \rfloor_\ell)\ \mathbf{unit}^{\ell_0} \rightsquigarrow \lfloor b' \rfloor_\ell\{\mathbf{unit}/x\}$.

Next, we show: $\lfloor b' \rfloor_\ell\{\textbf{unit}/x\} = \lfloor b' \rfloor_\ell\{\lfloor a \rfloor_\ell/x\}$.

By inversion on $\Phi \vdash \lambda x{:}^{\ell_0} A.b' : \ell$, we have, $\Phi, x : \ell \sqcup \ell_0 \vdash b' : \ell$.
By Lemma B.22, $\Phi, x : \ell_0 \vdash b' : \ell$.
Next, by rule CIndD-Nleq, $\Phi \vdash_\ell^{\ell_0} \textbf{unit} \sim a$.
Then, by Lemma B.23, $\Phi \vdash b'\{\textbf{unit}/x\} \sim_\ell b'\{a/x\}$.
Finally, by Lemma B.24, $\lfloor b' \rfloor_\ell\{\textbf{unit}/x\} = \lfloor b' \rfloor_\ell\{\lfloor a \rfloor_\ell/x\}$.

- Rule IndD-LetPair. Have: $\Phi \vdash \textbf{let}\ (x^{\ell_0}, y) = a\ \textbf{in}\ b : \ell$ where $\Phi \vdash a : \ell$ and $\Phi, x : \ell \sqcup \ell_0, y : \ell \vdash b : \ell$.
Further, $\vdash \textbf{let}\ (x^{\ell_0}, y) = a\ \textbf{in}\ b \leadsto c$.
By inversion on $\vdash \textbf{let}\ (x^{\ell_0}, y) = a\ \textbf{in}\ b \leadsto c$:
  - $\vdash \textbf{let}\ (x^{\ell_0}, y) = a\ \textbf{in}\ b \leadsto \textbf{let}\ (x^{\ell_0}, y) = a'\ \textbf{in}\ b$ when $\vdash a \leadsto a'$.
  Need to show: $\vdash \textbf{let}\ (x^{\ell_0}, y) = \lfloor a \rfloor_\ell\ \textbf{in}\ \lfloor b \rfloor_\ell \leadsto \textbf{let}\ (x^{\ell_0}, y) = \lfloor a' \rfloor_\ell\ \textbf{in}\ \lfloor b \rfloor_\ell$.
  Follows by IH and left step rule for **let**.
  - $a = ({a_1}^{\ell_0}, a_2)$ and $\vdash \textbf{let}\ (x^{\ell_0}, y) = ({a_1}^{\ell_0}, a_2)\ \textbf{in}\ b \leadsto b\{a_1/x\}\{a_2/y\}$.
  Now, there are two possibilities.
    * $\ell_0 \sqsubseteq \ell$.
    Need to show: $\vdash \textbf{let}\ (x^{\ell_0}, y) = (\lfloor a_1 \rfloor_\ell^{\ell_0}, \lfloor a_2 \rfloor_\ell)\ \textbf{in}\ \lfloor b \rfloor_\ell \leadsto \lfloor b \rfloor_\ell\{\lfloor a_1 \rfloor_\ell/x\}\{\lfloor a_2 \rfloor_\ell/y\}$.
    Follows by β-rule for **let**.
    * $\neg(\ell_0 \sqsubseteq \ell)$.
    Need to show: $\vdash \textbf{let}\ (x^{\ell_0}, y) = (\textbf{unit}^{\ell_0}, \lfloor a_2 \rfloor_\ell)\ \textbf{in}\ \lfloor b \rfloor_\ell \leadsto \lfloor b \rfloor_\ell\{\lfloor a_1 \rfloor_\ell/x\}\{\lfloor a_2 \rfloor_\ell/y\}$.
    Now, $\vdash \textbf{let}\ (x^{\ell_0}, y) = (\textbf{unit}^{\ell_0}, \lfloor a_2 \rfloor_\ell)\ \textbf{in}\ \lfloor b \rfloor_\ell \leadsto \lfloor b \rfloor_\ell\{\textbf{unit}/x\}\{\lfloor a_2 \rfloor_\ell/y\}$.
    Next, we show: $\lfloor b \rfloor_\ell\{\textbf{unit}/x\}\{\lfloor a_2 \rfloor_\ell/y\} = \lfloor b \rfloor_\ell\{\lfloor a_1 \rfloor_\ell/x\}\{\lfloor a_2 \rfloor_\ell/y\}$.

    By inversion on $\Phi \vdash ({a_1}^{\ell_0}, a_2) : \ell$, we have, $\Phi \vdash a_2 : \ell$.
    Next, we also have: $\Phi, x : \ell \sqcup \ell_0, y : \ell \vdash b : \ell$.
    By Lemma B.22, $\Phi, x : \ell_0, y : \ell \vdash b : \ell$.
    Now, by rule CIndD-Nleq, $\Phi \vdash_\ell^{\ell_0} \textbf{unit} \sim a_1$.
    Then, by Lemma B.23, $\Phi, y : \ell \vdash b\{\textbf{unit}/x\} \sim_\ell b\{a_1/x\}$.
    Applying Lemma B.23 again, $\Phi \vdash b\{\textbf{unit}/x\}\{a_2/y\} \sim_\ell b\{a_1/x\}\{a_2/y\}$.
    Finally, by Lemma B.24, $\lfloor b \rfloor_\ell\{\textbf{unit}/x\}\{\lfloor a_2 \rfloor_\ell/y\} = \lfloor b \rfloor_\ell\{\lfloor a_1 \rfloor_\ell/x\}\{\lfloor a_2 \rfloor_\ell/y\}$.

□

B.3 Proofs of Lemmas/Theorems Stated in Section 3.4

Lemma B.27 (Weakening) If $\Phi_1, \Phi_2 \vdash a_1 \sim_\ell a_2$, then $\Phi_1, z : m, \Phi_2 \vdash a_1 \sim_\ell a_2$.

Proof. By straightforward induction on $\Phi_1, \Phi_2 \vdash a_1 \sim_\ell a_2$. □

Lemma B.28 (Lemma 3.23) If $\Omega \vdash a :^{\ell} A$ then $\lceil\Omega\rceil \vdash a : \ell$.

Proof. By induction on $\Omega \vdash a :^{\ell} A$. We present some of the interesting cases below.

- Rule DDC-WEAK. Have: $\Omega, y :^{\ell_0} B \vdash a :^{\ell} A$ where $\Omega \vdash a :^{\ell} A$ and $\mathcal{C} \sqcap \Omega \vdash B :^{\mathcal{C}} s$.
  Need to show: $\lceil\Omega\rceil, y : \ell_0 \vdash a \sim_{\ell} a$.
  By IH, $\lceil\Omega\rceil \vdash a \sim_{\ell} a$.
  This case, then, follows by Lemma B.27.

- Rule DDC-LAM. Have: $\Omega \vdash \lambda x :^{\ell_0} A.b :^{\ell} \Pi x :^{\ell_0} A.B$ where $\Omega, x :^{\ell \sqcup \ell_0} A \vdash b :^{\ell} B$ and $\mathcal{C} \sqcap \Omega \vdash \Pi x :^{\ell_0} A.B :^{\mathcal{C}} s$.
  Need to show: $\lceil\Omega\rceil \vdash \lambda x :^{\ell_0} A.b \sim_{\ell} \lambda x :^{\ell_0} A.b$.
  By IH, $\lceil\Omega\rceil, x : \ell \sqcup \ell_0 \vdash b \sim_{\ell} b$.
  Next, since $\ell \sqsubseteq \mathcal{C}$, by rule CINDD-NLEQ, we have: $\lceil\Omega\rceil \vdash^{\top}_{\ell} A \sim A$.
  This case, then, follows by rule INDD-LAM.

- Rule DDC-APP. Have: $\Omega \vdash b\ a^{\ell_0} :^{\ell} B\{a/x\}$ where $\Omega \vdash b :^{\ell} \Pi x :^{\ell_0} A.B$ and $\Omega \Vdash a :^{\ell \sqcup \ell_0} A$.
  Need to show: $\lceil\Omega\rceil \vdash b\ a^{\ell_0} \sim_{\ell} b\ a^{\ell_0}$.
  By IH, $\lceil\Omega\rceil \vdash b \sim_{\ell} b$.
  Next, we show that $\lceil\Omega\rceil \vdash^{\ell_0}_{\ell} a \sim a$.
  If $\ell_0 \sqsubseteq \ell$, then $\ell \sqcup \ell_0 = \ell$. Further, $\ell \sqsubseteq \mathcal{C}$. So, $\Omega \vdash a :^{\ell} A$. Then, by IH, $\lceil\Omega\rceil \vdash a \sim_{\ell} a$. And, by rule CINDD-LEQ, $\lceil\Omega\rceil \vdash^{\ell_0}_{\ell} a \sim a$.
  Otherwise, by rule CINDD-NLEQ, $\lceil\Omega\rceil \vdash^{\ell_0}_{\ell} a \sim a$.
  Hence, $\lceil\Omega\rceil \vdash^{\ell_0}_{\ell} a \sim a$.
  This case, then, follows by rule INDD-APP.

□

Lemma B.29 (Lemma 3.24) If $\Phi \vdash a_1 \sim_{\ell} a_2$, then $\Phi \vdash a_1 : \ell$ and $\Phi \vdash a_2 : \ell$.

Proof. Follow proof of lemma B.10. □

Lemma B.30 (Lemma 3.25) Indexed indistinguishability at ℓ is an equivalence relation on well-graded terms at ℓ.

Proof. Follow proof of Lemma B.11. □

Lemma B.31 (Lemma 3.26) If $\Phi_1, z : m, \Phi_2 \vdash a_1 \sim_{\ell} a_2$ and $\Phi_1 \vdash^{m}_{\ell} c_1 \sim c_2$, then $\Phi_1, \Phi_2 \vdash a_1\{c_1/z\} \sim_{\ell} a_2\{c_2/z\}$.

Proof. Already proved as Lemma B.23. □

Theorem B.32 (Theorem 3.27) If $\Phi \vdash a_1 \sim_{\ell} a_2$ and $\vdash a_1 \leadsto a_1'$, then there exists a_2' such that $\vdash a_2 \leadsto a_2'$ and $\Phi \vdash a_1' \sim_{\ell} a_2'$.

Proof. By induction on $\Phi \vdash a_1 \sim_\ell a_2$ and subsequent inversion on $\vdash a_1 \rightsquigarrow a_1'$. We present some of the interesting cases below.

- Rule IndD-App. Have: $\Phi \vdash b_1\ {a_1}^{\ell_0} \sim_\ell b_2\ {a_2}^{\ell_0}$ where $\Phi \vdash b_1 \sim_\ell b_2$ and $\Phi \vdash_\ell^{\ell_0} a_1 \sim a_2$. By inversion on $\vdash b_1\ {a_1}^{\ell_0} \rightsquigarrow c_1'$:
  - $\vdash b_1\ {a_1}^{\ell_0} \rightsquigarrow b_1'\ {a_1}^{\ell_0}$ when $\vdash b_1 \rightsquigarrow b_1'$.
    Need to show: $\exists c_2'$ such that $\vdash b_2\ {a_2}^{\ell_0} \rightsquigarrow c_2'$ and $\Phi \vdash b_1'\ {a_1}^{\ell_0} \sim_\ell c_2'$.
    By IH, $\exists b_2'$ such that $\vdash b_2 \rightsquigarrow b_2'$ and $\Phi \vdash b_1' \sim_\ell b_2'$.
    Therefore, $\vdash b_2\ {a_2}^{\ell_0} \rightsquigarrow b_2'\ {a_2}^{\ell_0}$.
    And by rule IndD-App, $\Phi \vdash b_1'\ {a_1}^{\ell_0} \sim_\ell b_2'\ {a_2}^{\ell_0}$.
  - $b_1 = \lambda x{:}^{\ell_0} A_1.b_1'$ and $\vdash b_1\ {a_1}^{\ell_0} \rightsquigarrow b_1'\{a_1/x\}$.
    Need to show: $\exists c_2'$ such that $\vdash b_2\ {a_2}^{\ell_0} \rightsquigarrow c_2'$ and $\Phi \vdash b_1'\{a_1/x\} \sim_\ell c_2'$.
    By inversion on $\Phi \vdash \lambda x{:}^{\ell_0} A_1.b_1' \sim_\ell b_2$, we get, $b_2 = \lambda x{:}^{\ell_0} A_2.b_2'$ and $\Phi, x{:}\,\ell \sqcup \ell_0 \vdash b_1' \sim_\ell b_2'$.
    Then, $\vdash b_2\ {a_2}^{\ell_0} \rightsquigarrow b_2'\{a_2/x\}$.
    Next, since $\Phi, x{:}\,\ell \sqcup \ell_0 \vdash b_1' \sim_\ell b_2'$, by Lemma B.22, $\Phi, x{:}\,\ell_0 \vdash b_1' \sim_\ell b_2'$.
    So, by Lemma B.31, $\Phi \vdash b_1'\{a_1/x\} \sim_\ell b_2'\{a_2/x\}$.

- Rule IndD-LetPair. Have: $\Phi \vdash \textbf{let}\ (x^{\ell_0}, y)\ =\ a_1\ \textbf{in}\ b_1 \sim_\ell \textbf{let}\ (x^{\ell_0}, y)\ =\ a_2\ \textbf{in}\ b_2$ where $\Phi \vdash a_1 \sim_\ell a_2$ and $\Phi, x{:}\,\ell \sqcup \ell_0, y{:}\,\ell \vdash b_1 \sim_\ell b_2$. By inversion on $\vdash \textbf{let}\ (x^{\ell_0}, y)\ =\ a_1\ \textbf{in}\ b_1 \rightsquigarrow c_1'$:
  - $\vdash \textbf{let}\ (x^{\ell_0}, y)\ =\ a_1\ \textbf{in}\ b_1 \rightsquigarrow \textbf{let}\ (x^{\ell_0}, y)\ =\ a_1'\ \textbf{in}\ b_1$ when $\vdash a_1 \rightsquigarrow a_1'$.
    Need to show: $\exists c_2'$ such that $\vdash \textbf{let}\ (x^{\ell_0}, y)\ =\ a_2\ \textbf{in}\ b_2 \rightsquigarrow c_2'$ and $\Phi \vdash \textbf{let}\ (x^{\ell_0}, y)\ =\ a_1'\ \textbf{in}\ b_1 \sim_\ell c_2'$.
    By IH, $\exists a_2'$ such that $\vdash a_2 \rightsquigarrow a_2'$ and $\Phi \vdash a_1' \sim_\ell a_2'$.
    Therefore, $\vdash \textbf{let}\ (x^{\ell_0}, y)\ =\ a_2\ \textbf{in}\ b_2 \rightsquigarrow \textbf{let}\ (x^{\ell_0}, y)\ =\ a_2'\ \textbf{in}\ b_2$.
    And by rule IndD-LetPair, $\Phi \vdash \textbf{let}\ (x^{\ell_0}, y)\ =\ a_1'\ \textbf{in}\ b_1 \sim_\ell \textbf{let}\ (x^{\ell_0}, y)\ =\ a_2'\ \textbf{in}\ b_2$.
  - $a_1 = ({a_{11}}^{\ell_0}, a_{12})$ and $\vdash \textbf{let}\ (x^{\ell_0}, y)\ =\ a_1\ \textbf{in}\ b_1 \rightsquigarrow b_1\{a_{11}/x\}\{a_{12}/y\}$.
    Need to show: $\exists c_2'$ such that $\vdash \textbf{let}\ (x^{\ell_0}, y)\ =\ a_2\ \textbf{in}\ b_2 \rightsquigarrow c_2'$ and $\Phi \vdash b_1\{a_{11}/x\}\{a_{12}/y\} \sim_\ell c_2'$.
    By inversion on $\Phi \vdash ({a_{11}}^{\ell_0}, a_{12}) \sim_\ell a_2$, we get, $a_2 = ({a_{21}}^{\ell_0}, a_{22})$ and $\Phi \vdash_\ell^{\ell_0} a_{11} \sim a_{21}$ and $\Phi \vdash a_{12} \sim_\ell a_{22}$.
    Then, $\vdash \textbf{let}\ (x^{\ell_0}, y)\ =\ a_2\ \textbf{in}\ b_2 \rightsquigarrow b_2\{a_{21}/x\}\{a_{22}/y\}$.
    Next, since $\Phi, x : \ell \sqcup \ell_0, y : \ell \vdash b_1 \sim_\ell b_2$, by Lemma B.22, $\Phi, x : \ell_0, y : \ell \vdash b_1 \sim_\ell b_2$. Further, since $\Phi \vdash a_{12} \sim_\ell a_{22}$, so $\Phi \vdash_\ell^\ell a_{12} \sim a_{22}$.
    By applying Lemma B.31 twice, we get, $\Phi \vdash b_1\{a_{11}/x\}\{a_{12}/y\} \sim_\ell b_2\{a_{21}/x\}\{a_{22}/y\}$.

□

Lemma B.33 (Weakening) If $\Phi_1, \Phi_2 \vdash a \Rightarrow_\ell b$, then $\Phi_1, z{:}\,m, \Phi_2 \vdash a \Rightarrow_\ell b$.

Proof. By induction on $\Phi_1, \Phi_2 \vdash a \Rightarrow_\ell b$. □

Lemma B.34 (Well-graded) If $\Phi \vdash a_1 \Rightarrow_\ell a_2$, then $\Phi \vdash a_1 : \ell$ and $\Phi \vdash a_2 : \ell$.

Proof. By induction on $\Phi \vdash a_1 \Rightarrow_\ell a_2$. We present some of the interesting cases below.

- Rule Par-Refl. Have: $\Phi \vdash a \Rightarrow_\ell a$ where $\Phi \vdash a : \ell$.
  Need to show: $\Phi \vdash a : \ell$.
  Follows from premises.

- Rule Par-Abs. Have: $\Phi \vdash \lambda x{:}^{\ell_0} A_1.b_1 \Rightarrow_\ell \lambda x{:}^{\ell_0} A_2.b_2$ where $\Phi, x{:}\,\ell \sqcup \ell_0 \vdash b_1 \Rightarrow_\ell b_2$ and $\Phi \vdash^{\top}_{\ell} A_1 \Rightarrow A_2$.
  Need to show: $\Phi \vdash \lambda x{:}^{\ell_0} A_1.b_1 : \ell$ and $\Phi \vdash \lambda x{:}^{\ell_0} A_2.b_2 : \ell$.
  By IH, $\Phi, x{:}\,\ell \sqcup \ell_0 \vdash b_1 : \ell$ and $\Phi, x{:}\,\ell \sqcup \ell_0 \vdash b_2 : \ell$.
  Now, if $\top \sqsubseteq \ell$, then we have, $\Phi \vdash A_1 \Rightarrow_\ell A_2$. By IH, $\Phi \vdash A_1 \sim_\ell A_1$ and $\Phi \vdash A_2 \sim_\ell A_2$. Then, by rule CIndD-Leq, $\Phi \vdash^{\top}_{\ell} A_1 \sim A_1$ and $\Phi \vdash^{\top}_{\ell} A_2 \sim A_2$.
  Otherwise, by rule CIndD-Nleq, $\Phi \vdash^{\top}_{\ell} A_1 \sim A_1$ and $\Phi \vdash^{\top}_{\ell} A_2 \sim A_2$.
  This case, then, follows by rule IndD-Lam.

- Rule Par-App. Have: $\Phi \vdash b_1\ {a_1}^{\ell_0} \Rightarrow_\ell b_2\ {a_2}^{\ell_0}$ where $\Phi \vdash b_1 \Rightarrow_\ell b_2$ and $\Phi \vdash^{\ell_0}_{\ell} a_1 \Rightarrow a_2$.
  Need to show: $\Phi \vdash b_1\ {a_1}^{\ell_0} : \ell$ and $\Phi \vdash b_2\ {a_2}^{\ell_0} : \ell$.
  By IH, $\Phi \vdash b_1 : \ell$ and $\Phi \vdash b_2 : \ell$.
  Now, if $\ell_0 \sqsubseteq \ell$, then $\Phi \vdash a_1 \Rightarrow_\ell a_2$. By IH, $\Phi \vdash a_1 \sim_\ell a_1$ and $\Phi \vdash a_2 \sim_\ell a_2$. Then, by rule CIndD-Leq, $\Phi \vdash^{\ell_0}_{\ell} a_1 \sim a_1$ and $\Phi \vdash^{\ell_0}_{\ell} a_2 \sim a_2$.
  Otherwise, by rule CIndD-Nleq, $\Phi \vdash^{\ell_0}_{\ell} a_1 \sim a_1$ and $\Phi \vdash^{\ell_0}_{\ell} a_2 \sim a_2$.
  This case, then, follows by rule IndD-App.

- Rule Par-AppBeta. Have: $\Phi \vdash b\ a^{\ell_0} \Rightarrow_\ell b'\{a'/x\}$ where $\Phi \vdash b \Rightarrow_\ell \lambda x{:}^{\ell_0} A.b'$ and $\Phi \vdash a \Rightarrow_\ell a'$.
  Need to show: $\Phi \vdash b\ a^{\ell_0} : \ell$ and $\Phi \vdash b'\{a'/x\} : \ell$.
  By IH, $\Phi \vdash b : \ell$ and $\Phi \vdash \lambda x{:}^{\ell_0} A.b' : \ell$.
  Again by IH, $\Phi \vdash a : \ell$ and $\Phi \vdash a' : \ell$.
  So, $\Phi \vdash^{\ell_0}_{\ell} a \sim a$ and $\Phi \vdash^{\ell_0}_{\ell} a' \sim a'$.
  Therefore, by rule IndD-App, $\Phi \vdash b\ a^{\ell_0} : \ell$.
  Next, by inversion on $\Phi \vdash \lambda x :^{\ell_0} A.b' : \ell$, we get, $\Phi, x : \ell \sqcup \ell_0 \vdash b' : \ell$. By Lemma B.22, we have, $\Phi, x{:}\,\ell_0 \vdash b' : \ell$.
  Then, by Lemma B.31, $\Phi \vdash b'\{a'/x\} : \ell$.

□

Lemma B.35 (Substitution) If $\Phi_1, z{:}\,m, \Phi_2 \vdash a : \ell$ and $\Phi_1 \vdash^{m}_{\ell} c_1 \Rightarrow c_2$, then $\Phi_1, \Phi_2 \vdash a\{c_1/z\} \Rightarrow_\ell a\{c_2/z\}$.

Proof. By induction on $\Phi_1, z{:}\,m, \Phi_2 \vdash a : \ell$. We present some of the interesting cases below.

- Rule IndD-Var. There are three cases to consider.
  - Have: $\Phi_{11}, z{:}\,m, \Phi_{12}, x{:}\,\ell_0, \Phi_2 \vdash x : \ell$ where $\ell_0 \sqsubseteq \ell$. Further, $\Phi_{11} \vdash^{m}_{\ell} c_1 \Rightarrow c_2$.
    Need to show: $\Phi_{11}, \Phi_{12}, x{:}\,\ell_0, \Phi_2 \vdash x \Rightarrow_\ell x$.

By rule IndD-Var, $\Phi_{11}, \Phi_{12}, x\!: \ell_0, \Phi_2 \vdash x : \ell$.
This case, then, follows by rule Par-Refl.

- Have: $\Phi_1, x\!: \ell_0, \Phi_2 \vdash x : \ell$ where $\ell_0 \sqsubseteq \ell$. Further, $\Phi_1 \vdash^{\ell_0}_{\ell} c_1 \Rightarrow c_2$.
  Need to show: $\Phi_1, \Phi_2 \vdash c_1 \Rightarrow_{\ell} c_2$.
  Since $\ell_0 \sqsubseteq \ell$, we have, $\Phi_1 \vdash c_1 \Rightarrow_{\ell} c_2$.
  This case, then, follows by Lemma B.33.

- Have: $\Phi_1, x\!: \ell_0, \Phi_{21}, z\!: m, \Phi_{22} \vdash x : \ell$ where $\ell_0 \sqsubseteq \ell$. Further, $\Phi_1, x\!: \ell_0, \Phi_{21} \vdash^{m}_{\ell} c_1 \Rightarrow c_2$.
  Need to show: $\Phi_1, x\!: \ell_0, \Phi_{21}, \Phi_{22} \vdash x \Rightarrow_{\ell} x$.
  By rule IndD-Var, $\Phi_1, x\!: \ell_0, \Phi_{21}, \Phi_{22} \vdash x : \ell$.
  This case, then, follows by rule Par-Refl.

- Rule IndD-Lam. Have: $\Phi_1, z : m, \Phi_2 \vdash \lambda x :^{\ell_0} A.b : \ell$ where $\Phi_1, z : m, \Phi_2, x : \ell \sqcup \ell_0 \vdash b : \ell$ and $\Phi_1, z\!: m, \Phi_2 \vdash^{\top}_{\ell} A \sim A$. Further, $\Phi_1 \vdash^{m}_{\ell} c_1 \Rightarrow c_2$.
  Need to show: $\Phi_1, \Phi_2 \vdash \lambda x\!:^{\ell_0} A\{c_1/z\}.b\{c_1/z\} \Rightarrow_{\ell} \lambda x\!:^{\ell_0} A\{c_2/z\}.b\{c_2/z\}$.
  By IH, $\Phi_1, \Phi_2, x\!: \ell \sqcup \ell_0 \vdash b\{c_1/z\} \Rightarrow_{\ell} b\{c_2/z\}$.
  Now, if $\top \sqsubseteq \ell$, then we have, $\Phi_1, z : m, \Phi_2 \vdash A \sim_{\ell} A$. By IH, $\Phi_1, \Phi_2 \vdash A\{c_1/z\} \Rightarrow_{\ell} A\{c_2/z\}$. Then, by rule CPar-Leq, $\Phi_1, \Phi_2 \vdash^{\top}_{\ell} A\{c_1/z\} \Rightarrow A\{c_2/z\}$.
  Otherwise, by rule CPar-Nleq, $\Phi_1, \Phi_2 \vdash^{\top}_{\ell} A\{c_1/z\} \Rightarrow A\{c_2/z\}$.
  This case, then, follows by rule Par-Lam.

- Rule IndD-App. Have: $\Phi_1, z\!: m, \Phi_2 \vdash b\ a^{\ell_0} : \ell$ where $\Phi_1, z\!: m, \Phi_2 \vdash b : \ell$ and $\Phi_1, z\!: m, \Phi_2 \vdash^{\ell_0}_{\ell} a \sim a$. Further, $\Phi_1 \vdash^{m}_{\ell} c_1 \Rightarrow c_2$.
  Need to show: $\Phi_1, \Phi_2 \vdash b\{c_1/z\}\ (a\{c_1/z\})^{\ell_0} \Rightarrow_{\ell} b\{c_2/z\}\ (a\{c_2/z\})^{\ell_0}$.
  By IH, $\Phi_1, \Phi_2 \vdash b\{c_1/z\} \Rightarrow_{\ell} b\{c_2/z\}$.
  Now, if $\ell_0 \sqsubseteq \ell$, then we have, $\Phi_1, z\!: m, \Phi_2 \vdash a \sim_{\ell} a$. By IH, $\Phi_1, \Phi_2 \vdash a\{c_1/z\} \Rightarrow_{\ell} a\{c_2/z\}$. Then, by rule CPar-Leq, $\Phi_1, \Phi_2 \vdash^{\ell_0}_{\ell} a\{c_1/z\} \Rightarrow a\{c_2/z\}$.
  Otherwise, by rule CPar-Nleq, $\Phi_1, \Phi_2 \vdash^{\ell_0}_{\ell} a\{c_1/z\} \Rightarrow a\{c_2/z\}$.
  This case, then, follows by rule Par-App.

□

Lemma B.36 (Substitution) If $\Phi_1, z : m, \Phi_2 \vdash a_1 \Rightarrow_{\ell} a_2$ and $\Phi_1 \vdash^{m}_{\ell} c_1 \Rightarrow c_2$, then $\Phi_1, \Phi_2 \vdash a_1\{c_1/z\} \Rightarrow_{\ell} a_2\{c_2/z\}$.

Proof. By induction on $\Phi_1, z\!: m, \Phi_2 \vdash a_1 \Rightarrow_{\ell} a_2$. We present some of the interesting cases below.

- Rule Par-Refl. Have: $\Phi_1, z\!: m, \Phi_2 \vdash a \Rightarrow_{\ell} a$ where $\Phi_1, z\!: m, \Phi_2 \vdash a : \ell$. Further, $\Phi_1 \vdash^{m}_{\ell} c_1 \Rightarrow c_2$.
  Need to show: $\Phi_1, \Phi_2 \vdash a\{c_1/z\} \Rightarrow_{\ell} a\{c_2/z\}$.
  Follows by Lemma B.35.

- Rule Par-Lam. Have: $\Phi_1, z\colon m, \Phi_2 \vdash \lambda x\!:^{\ell_0} A_1.b_1 \Rightarrow_\ell \lambda x\!:^{\ell_0} A_2.b_2$ where $\Phi_1, z\colon m, \Phi_2, x\colon \ell \sqcup \ell_0 \vdash b_1 \Rightarrow_\ell b_2$ and $\Phi_1, z\colon m, \Phi_2 \vdash^{\top}_{\ell} A_1 \Rightarrow A_2$. Further, $\Phi_1 \vdash^{m}_{\ell} c_1 \Rightarrow c_2$.
  Need to show: $\Phi_1, \Phi_2 \vdash \lambda x\!:^{\ell_0} A_1\{c_1/z\}.b_1\{c_1/z\} \Rightarrow_\ell \lambda x\!:^{\ell_0} A_2\{c_2/z\}.b_2\{c_2/z\}$.
  By IH, $\Phi_1, \Phi_2, x\colon \ell \sqcup \ell_0 \vdash b_1\{c_1/z\} \Rightarrow_\ell b_2\{c_2/z\}$.
  Now, if $\top \sqsubseteq \ell$, then we have, $\Phi_1, z\colon m, \Phi_2 \vdash A_1 \Rightarrow_\ell A_2$. By IH, $\Phi_1, \Phi_2 \vdash A_1\{c_1/z\} \Rightarrow_\ell A_2\{c_2/z\}$. Then, by rule CPar-Leq, $\Phi_1, \Phi_2 \vdash^{\top}_{\ell} A_1\{c_1/z\} \Rightarrow A_2\{c_2/z\}$.
  Otherwise, by rule CPar-Nleq, $\Phi_1, \Phi_2 \vdash^{\top}_{\ell} A_1\{c_1/z\} \Rightarrow A_2\{c_2/z\}$.
  This case, then, follows by rule Par-Lam.

- Rule Par-App. Have: $\Phi_1, z : m, \Phi_2 \vdash b_1\ {a_1}^{\ell_0} \Rightarrow_\ell b_2\ {a_2}^{\ell_0}$ where $\Phi_1, z : m, \Phi_2 \vdash b_1 \Rightarrow_\ell b_2$ and $\Phi_1, z : m, \Phi_2 \vdash^{\ell_0}_{\ell} a_1 \Rightarrow a_2$. Further, $\Phi_1 \vdash^{m}_{\ell} c_1 \Rightarrow c_2$.
  Need to show: $\Phi_1, \Phi_2 \vdash b_1\{c_1/z\}\ (a_1\{c_1/z\})^{\ell_0} \Rightarrow_\ell b_2\{c_2/z\}\ (a_2\{c_2/z\})^{\ell_0}$.
  By IH, $\Phi_1, \Phi_2 \vdash b_1\{c_1/z\} \Rightarrow_\ell b_2\{c_2/z\}$.
  Now, if $\ell_0 \sqsubseteq \ell$, then we have, $\Phi_1, z : m, \Phi_2 \vdash a_1 \Rightarrow_\ell a_2$. By IH, $\Phi_1, \Phi_2 \vdash a_1\{c_1/z\} \Rightarrow_\ell a_2\{c_2/z\}$. Then, by rule CPar-Leq, $\Phi_1, \Phi_2 \vdash^{\ell_0}_{\ell} a_1\{c_1/z\} \Rightarrow a_2\{c_2/z\}$.
  Otherwise, by rule CPar-Nleq, $\Phi_1, \Phi_2 \vdash^{\ell_0}_{\ell} a_1\{c_1/z\} \Rightarrow a_2\{c_2/z\}$.
  This case, then, follows by rule Par-App.

- Rule Par-AppBeta. Have: $\Phi_1, z\colon m, \Phi_2 \vdash b\ a^{\ell_0} \Rightarrow_\ell b'\{a'/x\}$ where $\Phi_1, z\colon m, \Phi_2 \vdash b \Rightarrow_\ell \lambda x\!:^{\ell_0} A.b'$ and $\Phi_1, z\colon m, \Phi_2 \vdash^{\ell_0}_{\ell} a \Rightarrow a'$. Further, $\Phi_1 \vdash^{m}_{\ell} c_1 \Rightarrow c_2$.
  Need to show: $\Phi_1, \Phi_2 \vdash b\{c_1/z\}\ (a\{c_1/z\})^{\ell_0} \Rightarrow_\ell b'\{c_2/z\}\{a'\{c_2/z\}/x\}$.
  By IH, $\Phi_1, \Phi_2 \vdash b\{c_1/z\} \Rightarrow_\ell \lambda x\!:^{\ell_0} A\{c_2/z\}.b'\{c_2/z\}$.
  Next, by reasoning along the lines of the previous case, we have, $\Phi_1, \Phi_2 \vdash a\{c_1/z\} \Rightarrow_\ell a'\{c_2/z\}$.
  This case, then, follows by rule Par-AppBeta.

□

Lemma B.37 (Confluence) If $\Phi \vdash a \Rightarrow_\ell c_1$ and $\Phi \vdash a \Rightarrow_\ell c_2$, then there exists c such that $\Phi \vdash c_1 \Rightarrow_\ell c$ and $\Phi \vdash c_2 \Rightarrow_\ell c$.

Proof. By induction on $\Phi \vdash a \Rightarrow_\ell c_1$. We present some of the interesting cases below.

- Rule Par-Refl. Have: $\Phi \vdash a \Rightarrow_\ell a$ where $\Phi \vdash a : \ell$. Further, $\Phi \vdash a \Rightarrow_\ell c_2$.
  Need to show: $\exists c$ such that $\Phi \vdash a \Rightarrow_\ell c$ and $\Phi \vdash c_2 \Rightarrow_\ell c$.
  Since $\Phi \vdash a \Rightarrow_\ell c_2$, by Lemma B.34, $\Phi \vdash c_2 : \ell$. By rule Par-Refl, $\Phi \vdash c_2 \Rightarrow_\ell c_2$.
  This case, then, follows by setting $c := c_2$.

- Rule Par-Lam. Have: $\Phi \vdash \lambda x\!:^{\ell_0} A.b \Rightarrow_\ell \lambda x\!:^{\ell_0} A_1.b_1$ where $\Phi, x\colon \ell \sqcup \ell_0 \vdash b \Rightarrow_\ell b_1$ and $\Phi \vdash^{\top}_{\ell} A \Rightarrow A_1$. Further, $\Phi \vdash \lambda x\!:^{\ell_0} A.b \Rightarrow_\ell c_2$.
  By inversion on $\Phi \vdash \lambda x\!:^{\ell_0} A.b \Rightarrow_\ell c_2$:

- Rule Par-Refl. Have: $\Phi \vdash \lambda x :^{\ell_0} A.b \Rightarrow_\ell \lambda x :^{\ell_0} A.b$.
  Need to show: $\exists c$ such that $\Phi \vdash \lambda x :^{\ell_0} A_1.b_1 \Rightarrow_\ell c$ and $\Phi \vdash \lambda x :^{\ell_0} A.b \Rightarrow_\ell c$.
  Follows by setting $c := \lambda x :^{\ell_0} A_1.b_1$.
- Rule Par-Lam. Have: $\Phi \vdash \lambda x :^{\ell_0} A.b \Rightarrow_\ell \lambda x :^{\ell_0} A_2.b_2$ where $\Phi, x : \ell \sqcup \ell_0 \vdash b \Rightarrow_\ell b_2$ and $\Phi \vdash^{\top}_{\ell} A \Rightarrow A_2$.
  Need to show: $\exists c$ such that $\Phi \vdash \lambda x :^{\ell_0} A_1.b_1 \Rightarrow_\ell c$ and $\Phi \vdash \lambda x :^{\ell_0} A_2.b_2 \Rightarrow_\ell c$.
  By IH, $\exists b'$ such that $\Phi, x : \ell \sqcup \ell_0 \vdash b_1 \Rightarrow_\ell b'$ and $\Phi, x : \ell \sqcup \ell_0 \vdash b_2 \Rightarrow_\ell b'$.
  Now, if $\top \sqsubseteq \ell$, then we have, $\Phi \vdash A \Rightarrow_\ell A_1$ and $\Phi \vdash A \Rightarrow_\ell A_2$. By IH, $\exists A'$ such that $\Phi \vdash A_1 \Rightarrow_\ell A'$ and $\Phi \vdash A_2 \Rightarrow_\ell A'$. Then, by rule CPar-Leq, $\Phi \vdash^{\top}_{\ell} A_1 \Rightarrow A'$ and $\Phi \vdash^{\top}_{\ell} A_2 \Rightarrow A'$. This case, then, follows by setting $c := \lambda x :^{\ell_0} A'.b'$.
  Otherwise, by rule CPar-Nleq, $\Phi \vdash^{\top}_{\ell} A_1 \Rightarrow A$ and $\Phi \vdash^{\top}_{\ell} A_2 \Rightarrow A$. This case, then, follows by setting $c := \lambda x :^{\ell_0} A.b'$.

- Rule Par-App. Have: $\Phi \vdash b\ a^{\ell_0} \Rightarrow_\ell b_1\ {a_1}^{\ell_0}$ where $\Phi \vdash b \Rightarrow_\ell b_1$ and $\Phi \vdash^{\ell_0}_{\ell} a \Rightarrow a_1$. Further, $\Phi \vdash b\ a^{\ell_0} \Rightarrow_\ell c_2$.
  By inversion on $\Phi \vdash b\ a^{\ell_0} \Rightarrow_\ell c_2$:

  - Rule Par-Refl. Have: $\Phi \vdash b\ a^{\ell_0} \Rightarrow_\ell b\ a^{\ell_0}$.
    Need to show: $\exists c$ such that $\Phi \vdash b_1\ {a_1}^{\ell_0} \Rightarrow_\ell c$ and $\Phi \vdash b\ a^{\ell_0} \Rightarrow_\ell c$.
    Follows by setting $c := b_1\ {a_1}^{\ell_0}$.
  - Rule Par-App. Have: $\Phi \vdash b\ a^{\ell_0} \Rightarrow_\ell b_2\ {a_2}^{\ell_0}$ where $\Phi \vdash b \Rightarrow_\ell b_2$ and $\Phi \vdash^{\ell_0}_{\ell} a \Rightarrow a_2$.
    Need to show: $\exists c$ such that $\Phi \vdash b_1\ {a_1}^{\ell_0} \Rightarrow_\ell c$ and $\Phi \vdash b_2\ {a_2}^{\ell_0} \Rightarrow_\ell c$.
    By IH, $\exists b'$ such that $\Phi \vdash b_1 \Rightarrow_\ell b'$ and $\Phi \vdash b_2 \Rightarrow_\ell b'$.
    Now, if $\ell_0 \sqsubseteq \ell$, then we have, $\Phi \vdash a \Rightarrow_\ell a_1$ and $\Phi \vdash a \Rightarrow_\ell a_2$. By IH, $\exists a'$ such that $\Phi \vdash a_1 \Rightarrow_\ell a'$ and $\Phi \vdash a_2 \Rightarrow_\ell a'$. Then by rule CPar-Leq, $\Phi \vdash^{\ell_0}_{\ell} a_1 \Rightarrow a'$ and $\Phi \vdash^{\ell_0}_{\ell} a_2 \Rightarrow a'$. This case, then, follows by setting $c := b'\ {a'}^{\ell_0}$.
    Otherwise, by rule CPar-Nleq, $\Phi \vdash^{\ell_0}_{\ell} a_1 \Rightarrow a$ and $\Phi \vdash^{\ell_0}_{\ell} a_2 \Rightarrow a$. This case, then, follows by setting $c := b'\ a^{\ell_0}$.
  - Rule Par-AppBeta. Have: $\Phi \vdash b\ a^{\ell_0} \Rightarrow_\ell b_2\{a_2/x\}$ where $\Phi \vdash b \Rightarrow_\ell \lambda x : A_2.b_2$ and $\Phi \vdash^{\ell_0}_{\ell} a \Rightarrow a_2$.
    Need to show: $\exists c$ such that $\Phi \vdash b_1\ {a_1}^{\ell_0} \Rightarrow_\ell c$ and $\Phi \vdash b_2\{a_2/x\} \Rightarrow_\ell c$.
    By IH, $\exists c'$ such that $\Phi \vdash b_1 \Rightarrow_\ell c'$ and $\Phi \vdash \lambda x :^{\ell_0} A_2.b_2 \Rightarrow_\ell c'$. By inversion on $\Phi \vdash \lambda x :^{\ell_0} A_2.b_2 \Rightarrow_\ell c'$, we get, $c' = \lambda x :^{\ell_0} A'.b'$ (for some A' and b') and $\Phi, x : \ell \sqcup \ell_0 \vdash b_2 \Rightarrow_\ell b'$.
    Now, there are two possibilities:

    * $\ell_0 \sqsubseteq \ell$. Then, we have, $\Phi \vdash a \Rightarrow_\ell a_1$ and $\Phi \vdash a \Rightarrow_\ell a_2$.
      By IH, $\exists a'$ such that $\Phi \vdash a_1 \Rightarrow_\ell a'$ and $\Phi \vdash a_2 \Rightarrow_\ell a'$.
      Now, since $\Phi \vdash b_1 \Rightarrow_\ell \lambda x :^{\ell_0} A'.b'$ and $\Phi \vdash a_1 \Rightarrow_\ell a'$, so by rule Par-AppBeta, $\Phi \vdash b_1\ {a_1}^{\ell_0} \Rightarrow_\ell b'\{a'/x\}$.
      Next, since $\Phi, x : \ell \sqcup \ell_0 \vdash b_2 \Rightarrow_\ell b'$, by Lemma B.22, $\Phi, x : \ell_0 \vdash b_2 \Rightarrow_\ell b'$. Further, since $\Phi \vdash a_2 \Rightarrow_\ell a'$, by rule CIndD-leq, $\Phi \vdash^{\ell_0}_{\ell} a_2 \Rightarrow a'$. Therefore, by Lemma B.36, $\Phi \vdash b_2\{a_2/x\} \Rightarrow_\ell b'\{a'/x\}$.
      This case, then, follows by setting $c := b'\{a'/x\}$.

* $\neg(\ell_0 \sqsubseteq \ell)$. By rule CIndD-Nleq, $\Phi \vdash^{\ell_0}_{\ell} a_1 \Rightarrow a$ and $\Phi \vdash^{\ell_0}_{\ell} a_2 \Rightarrow a$.
    Now, since $\Phi \vdash b_1 \Rightarrow_\ell \lambda x :^{\ell_0} A'.b'$ and $\Phi \vdash^{\ell_0}_{\ell} a_1 \Rightarrow a$, so by rule Par-AppBeta, $\Phi \vdash b_1\ {a_1}^{\ell_0} \Rightarrow_\ell b'\{a/x\}$.
    Next, since $\Phi, x : \ell \sqcup \ell_0 \vdash b_2 \Rightarrow_\ell b'$, by Lemma B.22, $\Phi, x : \ell_0 \vdash b_2 \Rightarrow_\ell b'$. Therefore, by Lemma B.36, $\Phi \vdash b_2\{a_2/x\} \Rightarrow_\ell b'\{a/x\}$.
    This case, then, follows by setting $c := b'\{a/x\}$.

- Rule Par-AppBeta. Have: $\Phi \vdash b\ a^{\ell_0} \Rightarrow_\ell b_1\{a_1/x\}$ where $\Phi \vdash b \Rightarrow_\ell \lambda x :^{\ell_0} A_1.b_1$ and $\Phi \vdash^{\ell_0}_{\ell} a \Rightarrow a_1$. Further, $\Phi \vdash b\ a^{\ell_0} \Rightarrow_\ell c_2$.
  By inversion on $\Phi \vdash b\ a^{\ell_0} \Rightarrow_\ell c_2$:
  - Rule Par-Refl. Have: $\Phi \vdash b\ a^{\ell_0} \Rightarrow_\ell b\ a^{\ell_0}$.
    Need to show: $\exists c$ such that $\Phi \vdash b_1\{a_1/x\} \Rightarrow_\ell c$ and $\Phi \vdash b\ a^{\ell_0} \Rightarrow_\ell c$.
    Since $\Phi \vdash b \Rightarrow_\ell \lambda x :^{\ell_0} A_1.b_1$, by Lemma B.34, $\Phi \vdash \lambda x :^{\ell_0} A_1.b_1 : \ell$. Then, by inversion, $\Phi, x : \ell \sqcup \ell_0 \vdash b_1 : \ell$. And by Lemma B.22, $\Phi, x : \ell_0 \vdash b_1 : \ell$.
    Now, we show, $\Phi \vdash^{\ell_0}_{\ell} a_1 \sim a_1$. If $\ell_0 \sqsubseteq \ell$, then we have, $\Phi \vdash a \Rightarrow_\ell a_1$. By Lemma B.34, $\Phi \vdash a_1 : \ell$. By rule CIndD-Leq, $\Phi \vdash^{\ell_0}_{\ell} a_1 \sim a_1$. On the other hand, if $\neg(\ell_0 \sqsubseteq \ell)$, by rule CIndD-Nleq, $\Phi \vdash^{\ell_0}_{\ell} a_1 \sim a_1$.
    Next, since $\Phi, x : \ell_0 \vdash b_1 : \ell$ and $\Phi \vdash^{\ell_0}_{\ell} a_1 \sim a_1$, by Lemma B.31, $\Phi \vdash b_1\{a_1/x\} : \ell$. Then, by rule Par-Refl, $\Phi \vdash b_1\{a_1/x\} \Rightarrow_\ell b_1\{a_1/x\}$.
    Therefore, this case follows by setting $c := b_1\{a_1/x\}$.
  - Rule Par-App. Have: $\Phi \vdash b\ a^{\ell_0} \Rightarrow_\ell b_2\ {a_2}^{\ell_0}$ where $\Phi \vdash b \Rightarrow_\ell b_2$ and $\Phi \vdash^{\ell_0}_{\ell} a \Rightarrow a_2$.
    Need to show: $\exists c$ such that $\Phi \vdash b_1\{a_1/x\} \Rightarrow_\ell c$ and $\Phi \vdash b_2\ {a_2}^{\ell_0} \Rightarrow_\ell c$.
    Symmetric to the Par-AppBeta sub-case of Par-App case, presented above.
  - Rule Par-AppBeta. Have: $\Phi \vdash b\ a^{\ell_0} \Rightarrow_\ell b_2\{a_2/x\}$ where $\Phi \vdash b \Rightarrow_\ell \lambda x :^{\ell_0} A_2.b_2$ and $\Phi \vdash^{\ell_0}_{\ell} a \Rightarrow a_2$.
    Need to show: $\exists c$ such that $\Phi \vdash b_1\{a_1/x\} \Rightarrow_\ell c$ and $\Phi \vdash b_2\{a_2/x\} \Rightarrow_\ell c$.
    Since $\Phi \vdash b \Rightarrow_\ell \lambda x :^{\ell_0} A_1.b_1$ and $\Phi \vdash b \Rightarrow_\ell \lambda x :^{\ell_0} A_2.b_2$, there exists c' such that $\Phi \vdash \lambda x :^{\ell_0} A_1.b_1 \Rightarrow_\ell c'$ and $\Phi \vdash \lambda x :^{\ell_0} A_2.b_2 \Rightarrow_\ell c'$.
    By inversion on $\Phi \vdash \lambda x :^{\ell_0} A_1.b_1 \Rightarrow_\ell c'$, we get, $c' = \lambda x : A'.b'$ (for some A' and b') and $\Phi, x : \ell \sqcup \ell_0 \vdash b_1 \Rightarrow_\ell b'$. Then, by Lemma B.22, $\Phi, x : \ell_0 \vdash b_1 \Rightarrow_\ell b'$.
    Again, by inversion on $\Phi \vdash \lambda x :^{\ell_0} A_2.b_2 \Rightarrow_\ell \lambda x : A'.b'$, we get, $\Phi, x : \ell \sqcup \ell_0 \vdash b_2 \Rightarrow_\ell b'$. And by Lemma B.22, $\Phi, x : \ell_0 \vdash b_2 \Rightarrow_\ell b'$.
    Now, there are two possibilities:
    * $\ell_0 \sqsubseteq \ell$. Then, we have, $\Phi \vdash a \Rightarrow_\ell a_1$ and and $\Phi \vdash a \Rightarrow_\ell a_2$.
      By IH, $\exists a'$ such that $\Phi \vdash a_1 \Rightarrow_\ell a'$ and $\Phi \vdash a_2 \Rightarrow_\ell a'$.
      By rule CPar-Leq, $\Phi \vdash^{\ell_0}_{\ell} a_1 \Rightarrow a'$ and $\Phi \vdash^{\ell_0}_{\ell} a_2 \Rightarrow a'$.
      Since $\Phi, x : \ell_0 \vdash b_1 \Rightarrow_\ell b'$ and $\Phi, x : \ell_0 \vdash b_2 \Rightarrow_\ell b'$, therefore, by Lemma B.36, $\Phi \vdash b_1\{a_1/x\} \Rightarrow_\ell b'\{a'/x\}$ and $\Phi \vdash b_2\{a_2/x\} \Rightarrow_\ell b'\{a'/x\}$.
      Therefore, this case follows by setting $c := b'\{a'/x\}$.
    * $\neg(\ell_0 \sqsubseteq \ell)$. By rule CPar-Nleq, $\Phi \vdash^{\ell_0}_{\ell} a_1 \Rightarrow a$ and $\Phi \vdash^{\ell_0}_{\ell} a_2 \Rightarrow a$.
      Since $\Phi, x : \ell_0 \vdash b_1 \Rightarrow_\ell b'$ and $\Phi, x : \ell_0 \vdash b_2 \Rightarrow_\ell b'$, therefore, by Lemma B.36, $\Phi \vdash b_1\{a_1/x\} \Rightarrow_\ell$

$b'\{a/x\}$ and $\Phi \vdash b_2\{a_2/x\} \Rightarrow_\ell b'\{a/x\}$.

Therefore, this case follows by setting $c \coloneqq b'\{a/x\}$.

$\square$

Lemma B.38 (Confluence) If $\Phi \vdash a \Rightarrow_\ell^* c_1$ and $\Phi \vdash a \Rightarrow_\ell^* c_2$, then there exists c such that $\Phi \vdash c_1 \Rightarrow_\ell^* c$ and $\Phi \vdash c_2 \Rightarrow_\ell^* c$.

Proof. To prove this lemma, we need an auxiliary lemma: if $\Phi \vdash a \Rightarrow_\ell^* c_1$ and $\Phi \vdash a \Rightarrow_\ell c_2$, then there exists c such that $\Phi \vdash c_1 \Rightarrow_\ell c$ and $\Phi \vdash c_2 \Rightarrow_\ell^* c$.
The proof of this auxiliary lemma is by induction on $\Phi \vdash a \Rightarrow_\ell^* c_1$.

- Rule MultiPar-One. Have: $\Phi \vdash a \Rightarrow_\ell^* c_1$ where $\Phi \vdash a \Rightarrow_\ell c_1$. Further, $\Phi \vdash a \Rightarrow_\ell c_2$.
  Need to show: $\exists c$ such that $\Phi \vdash c_1 \Rightarrow_\ell c$ and $\Phi \vdash c_2 \Rightarrow_\ell^* c$.
  Follows by Lemma B.37.
- Rule MultiPar-Many. Have: $\Phi \vdash a \Rightarrow_\ell^* c_1$ where $\Phi \vdash a \Rightarrow_\ell^* b_1$ and $\Phi \vdash b_1 \Rightarrow_\ell^* c_1$. Further, $\Phi \vdash a \Rightarrow_\ell c_2$.
  Need to show: $\exists c$ such that $\Phi \vdash c_1 \Rightarrow_\ell c$ and $\Phi \vdash c_2 \Rightarrow_\ell^* c$.
  Since $\Phi \vdash a \Rightarrow_\ell^* b_1$ and $\Phi \vdash a \Rightarrow_\ell c_2$, by IH, $\exists b$ such that $\Phi \vdash b_1 \Rightarrow_\ell b$ and $\Phi \vdash c_2 \Rightarrow_\ell^* b$.
  Again, since $\Phi \vdash b_1 \Rightarrow_\ell^* c_1$ and $\Phi \vdash b_1 \Rightarrow_\ell b$, by IH, $\exists c$ such that $\Phi \vdash c_1 \Rightarrow_\ell c$ and $\Phi \vdash b \Rightarrow_\ell^* c$.
  Next, since $\Phi \vdash c_2 \Rightarrow_\ell^* b$ and $\Phi \vdash b \Rightarrow_\ell^* c$, so by rule MultiPar-Many, $\Phi \vdash c_2 \Rightarrow_\ell^* c$.
  This case, then, follows by setting $c \coloneqq c$.

Now, we prove the main lemma. The proof is again by induction on $\Phi \vdash a \Rightarrow_\ell^* c_1$.

- Rule MultiPar-One. Have: $\Phi \vdash a \Rightarrow_\ell^* c_1$ where $\Phi \vdash a \Rightarrow_\ell c_1$. Further, $\Phi \vdash a \Rightarrow_\ell^* c_2$.
  Need to show: $\exists c$ such that $\Phi \vdash c_1 \Rightarrow_\ell^* c$ and $\Phi \vdash c_2 \Rightarrow_\ell^* c$.
  This case follows by the auxiliary lemma.
- Rule MultiPar-Many. Have: $\Phi \vdash a \Rightarrow_\ell^* c_1$ where $\Phi \vdash a \Rightarrow_\ell^* b_1$ and $\Phi \vdash b_1 \Rightarrow_\ell^* c_1$. Further, $\Phi \vdash a \Rightarrow_\ell^* c_2$.
  Need to show: $\exists c$ such that $\Phi \vdash c_1 \Rightarrow_\ell^* c$ and $\Phi \vdash c_2 \Rightarrow_\ell^* c$.
  Since $\Phi \vdash a \Rightarrow_\ell^* b_1$ and $\Phi \vdash a \Rightarrow_\ell^* c_2$, by IH, $\exists b$ such that $\Phi \vdash b_1 \Rightarrow_\ell^* b$ and $\Phi \vdash c_2 \Rightarrow_\ell^* b$.
  Again, since $\Phi \vdash b_1 \Rightarrow_\ell^* c_1$ and $\Phi \vdash b_1 \Rightarrow_\ell^* b$, by IH, $\exists c$ such that $\Phi \vdash c_1 \Rightarrow_\ell^* c$ and $\Phi \vdash b \Rightarrow_\ell^* c$.
  Next, since $\Phi \vdash c_2 \Rightarrow_\ell^* b$ and $\Phi \vdash b \Rightarrow_\ell^* c$, so by rule MultiPar-Many, $\Phi \vdash c_2 \Rightarrow_\ell^* c$.
  This case, then, follows by setting $c \coloneqq c$.

$\square$

Lemma B.39 (Lemma 3.28) If $\Phi \vdash a \Rightarrow_\ell a_1$ and $\Phi \vdash a \Rightarrow_\ell a_2$, then there exists b such that $\Phi \vdash a_1 \Rightarrow_\ell b$ and $\Phi \vdash a_2 \Rightarrow_\ell b$. Similarly, if $\Phi \vdash a \Rightarrow_\ell^* a_1$ and $\Phi \vdash a \Rightarrow_\ell^* a_2$, then there exists b such that $\Phi \vdash a_1 \Rightarrow_\ell^* b$ and $\Phi \vdash a_2 \Rightarrow_\ell^* b$.

Proof. Follows by Lemmas B.37 and B.38. □

Lemma B.40 (Small-step and Parallel Reduction) If $\Phi \vdash a : \ell$ and $\vdash a \leadsto a'$, then $\Phi \vdash a \Rightarrow_\ell a'$.

Proof. By induction on $\Phi \vdash a : \ell$ and subsequent inversion on $\vdash a \leadsto a'$. We present one interesting case below.

- Rule IndD-App. Have: $\Phi \vdash b\ a^{\ell_0} : \ell$ where $\Phi \vdash b : \ell$ and $\Phi \vdash^{\ell_0}_{\ell} a \sim a$. By inversion on $\vdash b\ a^{\ell_0} \leadsto c$:
  - $\vdash b\ a^{\ell_0} \leadsto b'\ a^{\ell_0}$ when $\vdash b \leadsto b'$.
    Need to show: $\Phi \vdash b\ a^{\ell_0} \Rightarrow_\ell b'\ a^{\ell_0}$.
    By IH, $\Phi \vdash b \Rightarrow_\ell b'$.
    Now, if $\ell_0 \sqsubseteq \ell$, then we have, $\Phi \vdash a : \ell$. By rule Par-Refl, $\Phi \vdash a \Rightarrow_\ell a$. And by rule CPar-Leq, $\Phi \vdash^{\ell_0}_{\ell} a \Rightarrow a$.
    Otherwise, by rule CPar-Nleq, $\Phi \vdash^{\ell_0}_{\ell} a \Rightarrow a$.
    This case, then, follows by rule Par-App.
  - $b = \lambda x{:}^{\ell_0} A.b'$ and $\vdash b\ a^{\ell_0} \leadsto b'\{a/x\}$.
    Need to show: $\Phi \vdash (\lambda x{:}^{\ell_0} A.b')\ a \Rightarrow_\ell b'\{a/x\}$.
    Since $\Phi \vdash b : \ell$, by rule Par-Refl, $\Phi \vdash b \Rightarrow_\ell b$, i.e., $\Phi \vdash \lambda x{:}^{\ell_0} A.b' \Rightarrow_\ell \lambda x{:}^{\ell_0} A.b'$.
    Next, by reasoning as in the previous sub-case, $\Phi \vdash^{\ell_0}_{\ell} a \Rightarrow a$.
    This case, then, follows by rule Par-AppBeta.

□

Theorem B.41 (Theorem 3.29) $\Phi \vdash a \Leftrightarrow_\ell b$ if and only if $\Phi \vdash a \equiv_\ell b$.

Proof. First we show: if $\Phi \vdash a \equiv_\ell b$, then $\Phi \vdash a \Leftrightarrow_\ell b$.
By induction on $\Phi \vdash a \equiv_\ell b$. We present some of the interesting cases below.

- Rule Eq-Refl. Have: $\Phi \vdash a \equiv_\ell a$ where $\Phi \vdash a : \ell$.
  Need to show: $\exists c$ such that $\Phi \vdash a \Rightarrow^*_\ell c$ and $\Phi \vdash a \Rightarrow^*_\ell c$.
  Since $\Phi \vdash a : \ell$, by rule Par-Refl, $\Phi \vdash a \Rightarrow_\ell a$. Then, by rule MultiPar-One, $\Phi \vdash a \Rightarrow^*_\ell a$.
  This case follows by setting $c := a$.

- Rule Eq-Sym. Have: $\Phi \vdash b \equiv_\ell a$ where $\Phi \vdash a \equiv_\ell b$.
  Need to show: $\exists c$ such that $\Phi \vdash b \Rightarrow^*_\ell c$ and $\Phi \vdash a \Rightarrow^*_\ell c$.
  Follows by IH.

- Rule Eq-Trans. Have: $\Phi \vdash a_1 \equiv_\ell a_3$ where $\Phi \vdash a_1 \equiv_\ell a_2$ and $\Phi \vdash a_2 \equiv_\ell a_3$.
  Need to show: $\exists c$ such that $\Phi \vdash a_1 \Rightarrow^*_\ell c$ and $\Phi \vdash a_3 \Rightarrow^*_\ell c$.
  Since $\Phi \vdash a_1 \equiv_\ell a_2$, by IH, $\exists c_{12}$ such that $\Phi \vdash a_1 \Rightarrow^*_\ell c_{12}$ and $\Phi \vdash a_2 \Rightarrow^*_\ell c_{12}$.
  Again since $\Phi \vdash a_2 \equiv_\ell a_3$, by IH, $\exists c_{23}$ such that $\Phi \vdash a_2 \Rightarrow^*_\ell c_{23}$ and $\Phi \vdash a_3 \Rightarrow^*_\ell c_{23}$.

Now, since $\Phi \vdash a_2 \Rightarrow_\ell^* c_{12}$ and $\Phi \vdash a_2 \Rightarrow_\ell^* c_{23}$, by Lemma B.39, $\exists c_{13}$ such that $\Phi \vdash c_{12} \Rightarrow_\ell^* c_{13}$ and $\Phi \vdash c_{23} \Rightarrow_\ell^* c_{13}$.
Next, since $\Phi \vdash a_1 \Rightarrow_\ell^* c_{12}$ and $\Phi \vdash c_{12} \Rightarrow_\ell^* c_{13}$, so by rule MultiPar-Many, $\Phi \vdash a_1 \Rightarrow_\ell^* c_{13}$.
Again, since $\Phi \vdash a_3 \Rightarrow_\ell^* c_{23}$ and $\Phi \vdash c_{23} \Rightarrow_\ell^* c_{13}$, so by rule MultiPar-Many, $\Phi \vdash a_3 \Rightarrow_\ell^* c_{13}$.
This case, then, follows by setting $c \coloneqq c_{13}$.

- Rule Eq-Beta. Have: $\Phi \vdash a \equiv_\ell b$ where $\Phi \vdash a : \ell$ and $\vdash a \rightsquigarrow b$.
  Need to show: $\exists c$ such that $\Phi \vdash a \Rightarrow_\ell^* c$ and $\Phi \vdash b \Rightarrow_\ell^* c$.
  By Lemma rule smallpar, $\Phi \vdash a \Rightarrow_\ell b$. By rule MultiPar-One, $\Phi \vdash a \Rightarrow_\ell^* b$.
  Next, by Lemma B.34, $\Phi \vdash b : \ell$. Then, by rule Par-Refl, $\Phi \vdash b \Rightarrow_\ell b$. And by rule MultiPar-One, $\Phi \vdash b \Rightarrow_\ell^* b$.
  This case, then, follows by setting $c \coloneqq b$.

- Rule Eq-Lam. Have: $\Phi \vdash \lambda x{:}^{\ell_0} A_1.b_1 \equiv_\ell \lambda x{:}^{\ell_0} A_2.b_2$ where $\Phi, x{:}\,\ell \sqcup \ell_0 \vdash b_1 \equiv_\ell b_2$ and $\Phi \vdash^\top A_1 \equiv_\ell A_2$.
  Need to show: $\exists c$ such that $\Phi \vdash \lambda x{:}^{\ell_0} A_1.b_1 \Rightarrow_\ell^* c$ and $\Phi \vdash \lambda x{:}^{\ell_0} A_2.b_2 \Rightarrow_\ell^* c$.
  By IH, $\exists b'$ such that $\Phi, x{:}\,\ell \sqcup \ell_0 \vdash b_1 \Rightarrow_\ell^* b'$ and $\Phi, x{:}\,\ell \sqcup \ell_0 \vdash b_2 \Rightarrow_\ell^* b'$.
  Now, there are two possibilities:

  - $\top \sqsubseteq \ell$. Then, we have, $\Phi \vdash A_1 \equiv_\ell A_2$.
    By IH, $\exists A'$ such that $\Phi \vdash A_1 \Rightarrow_\ell^* A'$ and $\Phi \vdash A_2 \Rightarrow_\ell^* A'$.
    This case, then, follows by setting $c \coloneqq \lambda x{:}^{\ell_0} A'.b'$.
  - $\neg(\top \sqsubseteq \ell)$. By rule CPar-Nleq, $\Phi \vdash_\ell^\top A_1 \Rightarrow A_1$ and $\Phi \vdash_\ell^\top A_2 \Rightarrow A_1$.
    This case, then, follows by setting $c \coloneqq \lambda x{:}^{\ell_0} A_1.b'$.

- Rule Eq-App. Have: $\Phi \vdash b_1\ {a_1}^{\ell_0} \equiv_\ell b_2\ {a_2}^{\ell_0}$ where $\Phi \vdash b_1 \equiv_\ell b_2$ and $\Phi \vdash^{\ell_0} a_1 \equiv_\ell a_2$.
  Need to show: $\exists c$ such that $\Phi \vdash b_1\ {a_1}^{\ell_0} \Rightarrow_\ell^* c$ and $\Phi \vdash b_2\ {a_2}^{\ell_0} \Rightarrow_\ell^* c$.
  By IH, $\exists b'$ such that $\Phi \vdash b_1 \Rightarrow_\ell^* b'$ and $\Phi \vdash b_2 \Rightarrow_\ell^* b'$.
  Now, there are two possibilities:

  - $\ell_0 \sqsubseteq \ell$. Then, we have, $\Phi \vdash a_1 \equiv_\ell a_2$.
    By IH, $\exists a'$ such that $\Phi \vdash a_1 \Rightarrow_\ell^* a'$ and $\Phi \vdash a_2 \Rightarrow_\ell^* a'$.
    This case, then, follows by setting $c \coloneqq b'\ {a'}^{\ell_0}$.
  - $\neg(\ell_0 \sqsubseteq \ell)$. By rule CPar-Nleq, $\Phi \vdash_\ell^{\ell_0} a_1 \Rightarrow a_1$ and $\Phi \vdash_\ell^{\ell_0} a_2 \Rightarrow a_1$.
    This case, then, follows by setting $c \coloneqq b'\ {a_1}^{\ell_0}$.

To prove the 'only-if' part of the theorem, we need an auxiliary lemma: if $\Phi \vdash a \Rightarrow_\ell b$, then $\Phi \vdash a \equiv_\ell b$. The proof of this auxiliary lemma is by induction on $\Phi \vdash a \Rightarrow_\ell b$. We present some of the interesting cases below.

- Rule Par-Refl. Have: $\Phi \vdash a \Rightarrow_\ell a$ where $\Phi \vdash a : \ell$.
  Need to show: $\Phi \vdash a \equiv_\ell a$.
  Follows by rule Eq-Refl.

- Rule Par-Lam. Have: $\Phi \vdash \lambda x :^{\ell_0} A_1 . b_1 \Rightarrow_\ell \lambda x :^{\ell_0} A_2 . b_2$ where $\Phi, x : \ell \sqcup \ell_0 \vdash b_1 \Rightarrow_\ell b_2$ and $\Phi \vdash^{\top}_{\ell} A_1 \Rightarrow A_2$.
  Need to show: $\Phi \vdash \lambda x :^{\ell_0} A_1 . b_1 \equiv_\ell \lambda x :^{\ell_0} A_2 . b_2$.
  By IH, $\Phi, x : \ell \sqcup \ell_0 \vdash b_1 \equiv_\ell b_2$.
  Now, if $\top \sqsubseteq \ell$, then we have, $\Phi \vdash A_1 \Rightarrow_\ell A_2$. By IH, $\Phi \vdash A_1 \equiv_\ell A_2$. Then, by rule CEq-Leq, $\Phi \vdash^{\top} A_1 \equiv_\ell A_2$.
  Otherwise, by rule CEq-Nleq, $\Phi \vdash^{\top} A_1 \equiv_\ell A_2$.
  This case, then, follows by rule Eq-Lam.

- Rule Par-App. Have: $\Phi \vdash b_1 \; {a_1}^{\ell_0} \Rightarrow_\ell b_2 \; {a_2}^{\ell_0}$ where $\Phi \vdash b_1 \Rightarrow_\ell b_2$ and $\Phi \vdash^{\ell_0}_{\ell} a_1 \Rightarrow a_2$.
  Need to show: $\Phi \vdash b_1 \; {a_1}^{\ell_0} \equiv_\ell b_2 \; {a_2}^{\ell_0}$.
  By IH, $\Phi \vdash b_1 \equiv_\ell b_2$.
  Now if $\ell_0 \sqsubseteq \ell$, then we have, $\Phi \vdash a_1 \Rightarrow_\ell a_2$. By IH, $\Phi \vdash a_1 \equiv_\ell a_2$. Then, by rule CEq-Leq, $\Phi \vdash^{\ell_0} a_1 \equiv_\ell a_2$.
  Otherwise, by rule CEq-Nleq, $\Phi \vdash^{\ell_0} a_1 \equiv_\ell a_2$.
  This case, then, follows by rule Eq-App.

- Rule Par-AppBeta. Have: $\Phi \vdash b \; a^{\ell_0} \Rightarrow_\ell b'\{a'/x\}$ where $\Phi \vdash b \Rightarrow_\ell \lambda x :^{\ell_0} A . b'$ and $\Phi \vdash^{\ell_0}_{\ell} a \Rightarrow a'$.
  Need to show: $\Phi \vdash b \; a^{\ell_0} \equiv_\ell b'\{a'/x\}$.
  By IH, $\Phi \vdash b \equiv_\ell \lambda x :^{\ell_0} A . b'$.
  Now, if $\ell_0 \sqsubseteq \ell$, then we have, $\Phi \vdash a \Rightarrow_\ell a'$. By IH, $\Phi \vdash a \equiv_\ell a'$. Then, by rule CEq-Leq, $\Phi \vdash^{\ell_0} a \equiv_\ell a'$.
  Otherwise, by rule CEq-Nleq, $\Phi \vdash^{\ell_0} a \equiv_\ell a'$.
  So, in either case, $\Phi \vdash^{\ell_0} a \equiv_\ell a'$.
  Therefore, by rule Eq-App, $\Phi \vdash b \; a^{\ell_0} \equiv_\ell (\lambda x :^{\ell_0} A . b') \; a'$.
  But by rule Eq-Beta, $\Phi \vdash (\lambda x :^{\ell_0} A . b') \; a' \Rightarrow_\ell b'\{a'/x\}$.
  This case, then, follows by rule Eq-Trans.

Note that a corollary of this auxiliary lemma is: if $\Phi \vdash a \Rightarrow^{*}_{\ell} b$, then $\Phi \vdash a \equiv_\ell b$.

Now we show the 'only-if' part of the theorem: if $\Phi \vdash a \Leftrightarrow_\ell b$, then $\Phi \vdash a \equiv_\ell b$.
Since $\Phi \vdash a \Leftrightarrow_\ell b$, there exists c such that $\Phi \vdash a \Rightarrow^{*}_{\ell} c$ and $\Phi \vdash b \Rightarrow^{*}_{\ell} c$.
Now, since $\Phi \vdash a \Rightarrow^{*}_{\ell} c$, so by the above corollary, $\Phi \vdash a \equiv_\ell c$.
Again, since $\Phi \vdash b \Rightarrow^{*}_{\ell} c$, so by the above corollary, $\Phi \vdash b \equiv_\ell c$.
Then, by symmetry and transitivity, $\Phi \vdash a \equiv_\ell b$. □

Theorem B.42 (Theorem 3.30) If $\Phi \vdash a \Leftrightarrow_\ell b$, then $\mathsf{Ct} \; a \; b$.

Proof. By case analysis on a and b.
First, since $\Phi \vdash a \Leftrightarrow_\ell b$, there exists c such that $\Phi \vdash a \Rightarrow^{*}_{\ell} c$ and $\Phi \vdash b \Rightarrow^{*}_{\ell} c$.
Next, if either a or b is not a value type, then $\mathsf{Ct} \; a \; b$.
So, we are left with the cases where both a and b are value types.

To prove these cases, we begin by defining a function, *hd*, on value types that returns their head. Recall that a value type can be a sort (in $\mathcal{S}$) or **Unit** or a Π or a Σ or a sum. Here, Πs and Σs also have grade

annotations on them. So, the set of head forms of value types may be represented using:

$$\mathit{Heads} = \mathcal{S} \cup \{\,\mathit{Unit}\} \cup \{(\Pi, \ell) \mid \ell \in \mathcal{L}\} \cup \{(\Sigma, \ell) \mid \ell \in \mathcal{L}\} \cup \{\mathit{Sum}\}$$

The function hd maps value types to Heads as follows:

$$hd(s) = s \qquad hd(\mathbf{Unit}) = \mathit{Unit} \qquad hd(\Pi x{:}^{\ell} A.B) = (\Pi, \ell)$$
$$hd(\Sigma x{:}^{\ell} A.B) = (\Sigma, \ell) \qquad hd(A + B) = \mathit{Sum}$$

Now, we prove an auxiliary lemma: if $\mathsf{VType}\ a$ and $\Phi \vdash a \Rightarrow_{\ell} b$, then $hd(a) = hd(b)$.
The proof of this auxiliary lemma is by inversion first on $\Phi \vdash a \Rightarrow_{\ell} b$ and subsequently on $\mathsf{VType}\ a$. We present some of the interesting cases below.

- Rule PAR-REFL. Have: $\Phi \vdash a \Rightarrow_{\ell} a$. Further, $\mathsf{VType}\ a$.
  Need to show: $hd(a) = hd(a)$.
  Follows from the fact that hd is a well-defined function.
- Rule PAR-PI. Have: $\Phi \vdash \Pi x{:}^{\ell_0} A_1.B_1 \Rightarrow_{\ell} \Pi x{:}^{\ell_0} A_2.B_2$.
  Need to show: $hd(\Pi x{:}^{\ell_0} A_1.B_1) = hd(\Pi x{:}^{\ell_0} A_2.B_2)$.
  Follows by definition of hd.
- Rule PAR-LAM. Have: $\Phi \vdash \lambda x{:}^{\ell_0} A_1.b_1 \Rightarrow_{\ell} \lambda x{:}^{\ell_0} A_2.b_2$. Further, $\mathsf{VType}\ \lambda x{:}^{\ell_0} A_1.b_1$.
  A contradiction.
- Rule PAR-APP. Have: $\Phi \vdash b_1\ {a_1}^{\ell_0} \Rightarrow_{\ell} b_2\ {a_2}^{\ell_0}$. Further, $\mathsf{VType}\ b_1\ {a_1}^{\ell_0}$.
  A contradiction.
- Rule PAR-APPBETA. Have: $\Phi \vdash b\ a^{\ell_0} \Rightarrow_{\ell} b'\{a'/x\}$. Further, $\mathsf{VType}\ b\ a^{\ell_0}$.
  A contradiction.

A corollary of the above lemma is that: if $\mathsf{VType}\ a$ and $\Phi \vdash a \Rightarrow^{*}_{\ell} b$, then $hd(a) = hd(b)$.

Now, we show another auxiliary lemma: if $\mathsf{VType}\ a$ and $\mathsf{VType}\ b$ and $hd(a) = hd(b)$, then $\mathsf{Ct}\ a\ b$.
The proof of this auxiliary lemma is by inversion first on $\mathsf{VType}\ a$ and subsequently on $\mathsf{VType}\ b$.

Finally, we get back to the main proof.
We have: $\mathsf{VType}\ a$ and $\mathsf{VType}\ b$ and $\Phi \vdash a \Rightarrow^{*}_{\ell} c$ and $\Phi \vdash b \Rightarrow^{*}_{\ell} c$.
Need to show: $\mathsf{Ct}\ a\ b$.
Since $\mathsf{VType}\ a$ and $\Phi \vdash a \Rightarrow^{*}_{\ell} c$, so $hd(a) = hd(c)$.
Again since $\mathsf{VType}\ b$ and $\Phi \vdash b \Rightarrow^{*}_{\ell} c$, so $hd(b) = hd(c)$.
Therefore, $hd(a) = hd(b)$.
Further, $\mathsf{VType}\ a$ and $\mathsf{VType}\ b$.
So, we conclude that $\mathsf{Ct}\ a\ b$. $\square$

Lemma B.43 (Lemma 3.31) If $\Omega' \vdash a :^{\ell} A$ and $\Omega \sqsubseteq \Omega'$, then $\Omega \vdash a :^{\ell} A$

Proof. By induction on $\Omega' \vdash a :^{\ell} A$. We present some of the interesting cases below.

- Rule DDC-VAR. Have: $\Omega', x :^{\ell_0'} A \vdash x :^{\ell'} A$ where $\mathcal{C} \sqcap \Omega' \vdash A :^{\mathcal{C}} s$ and $\ell_0' \sqsubseteq \ell'$ and $\ell' \sqsubseteq \mathcal{C}$. Further, $\Omega_1 \sqsubseteq \Omega', x :^{\ell_0'} A$.
  Need to show: $\Omega_1 \vdash x :^{\ell'} A$.
  Since $\Omega_1 \sqsubseteq \Omega', x :^{\ell_0'} A$, so $\Omega_1 = \Omega, x :^{\ell_0} A$ where $\Omega \sqsubseteq \Omega'$ and $\ell_0 \sqsubseteq \ell_0'$.
  As such, $\mathcal{C} \sqcap \Omega \sqsubseteq \mathcal{C} \sqcap \Omega'$ and $\ell_0 \sqsubseteq \ell'$.
  By IH, $\mathcal{C} \sqcap \Omega \vdash A :^{\mathcal{C}} s$.
  Then, by rule DDC-VAR, $\Omega, x :^{\ell_0} A \vdash x :^{\ell'} A$.

- Rule DDC-APP. Have: $\Omega' \vdash b\ a^{\ell_0} :^{\ell} B\{a/x\}$ where $\Omega' \vdash b :^{\ell} \Pi x :^{\ell_0} A.B$ and $\Omega' \Vdash a :^{\ell \sqcup \ell_0} A$. Further, $\Omega \sqsubseteq \Omega'$.
  Need to show: $\Omega \vdash b\ a^{\ell_0} :^{\ell} B\{a/x\}$.
  By IH, $\Omega \vdash b :^{\ell} \Pi x :^{\ell_0} A.B$.
  Now, if $\ell \sqcup \ell_0 \sqsubseteq \mathcal{C}$, then we have, $\Omega' \vdash a :^{\ell \sqcup \ell_0} A$. By IH, $\Omega \vdash a :^{\ell \sqcup \ell_0} A$. And by rule TC-LEQ, $\Omega \Vdash a :^{\ell \sqcup \ell_0} A$.
  Otherwise, we have, $\mathcal{C} \sqcap \Omega' \vdash a :^{\mathcal{C}} A$. Since $\Omega \sqsubseteq \Omega'$, so $\mathcal{C} \sqcap \Omega \sqsubseteq \mathcal{C} \sqcap \Omega'$. Then, by IH, $\mathcal{C} \sqcap \Omega \vdash a :^{\mathcal{C}} A$. And by rule TC-TOP, $\Omega \Vdash a :^{\ell \sqcup \ell_0} A$.
  This case, then, follows by rule DDC-APP.

- Rule DDC-CONV. Have: $\Omega' \vdash a :^{\ell} B$ where $\Omega' \vdash a :^{\ell} A$ and $\lceil \mathcal{C} \sqcap \Omega' \rceil \vdash A \equiv_{\mathcal{C}} B$ and $\mathcal{C} \sqcap \Omega' \vdash B :^{\mathcal{C}} s$. Further, $\Omega \sqsubseteq \Omega'$.
  Need to show: $\Omega \vdash a :^{\ell} B$.
  To show this case, we use the following auxiliary lemma: if $\Phi_1, x : \ell_0', \Phi_2 \vdash a \equiv_{\ell} b$ and $\ell_0 \sqsubseteq \ell_0'$, then $\Phi_1, x : \ell_0, \Phi_2 \vdash a \equiv_{\ell} b$.
  The proof of this auxiliary lemma is by a straightforward induction on $\Phi_1, x : \ell_0', \Phi_2 \vdash a \equiv_{\ell} b$.
  Now, using this auxiliary lemma, we have, $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash A \equiv_{\mathcal{C}} B$.
  And by IH, we have, $\Omega \vdash a :^{\ell} A$ and $\mathcal{C} \sqcap \Omega \vdash B :^{\mathcal{C}} s$.
  This case, then, follows by rule DDC-CONV.

□

Lemma B.44 (Lemma 3.33) If $\Omega \vdash a :^{\ell} A$ and $m \sqsubseteq \mathcal{C}$, then $m \sqcup \Omega \vdash a :^{m \sqcup \ell} A$.

Proof. By induction on $\Omega \vdash a :^{\ell} A$. We present some of the interesting cases below.

- Rule DDC-VAR. Have: $\Omega, x :^{\ell_0} A \vdash x :^{\ell} A$ where $\mathcal{C} \sqcap \Omega \vdash A :^{\mathcal{C}} s$ and $\ell_0 \sqsubseteq \ell$ and $\ell \sqsubseteq \mathcal{C}$. Further, $m \sqsubseteq \mathcal{C}$.
  Need to show: $m \sqcup \Omega, x :^{m \sqcup \ell_0} A \vdash x :^{m \sqcup \ell} A$.
  By IH, $m \sqcup (\mathcal{C} \sqcap \Omega) \vdash A :^{\mathcal{C}} s$.

Now, for any $\ell' \in \mathcal{L}$, we have, $m \sqcup (\mathcal{C} \sqcap \ell') = \mathcal{C} \sqcap (m \sqcup \ell')$. To see why, we do a case analysis on ℓ'. If $\ell' = \top$, then $m \sqcup (\mathcal{C} \sqcap \ell') = \mathcal{C} \sqcap (m \sqcup \ell') = \mathcal{C}$. Otherwise, $\ell' \sqsubseteq \mathcal{C}$ and $m \sqcup (\mathcal{C} \sqcap \ell') = \mathcal{C} \sqcap (m \sqcup \ell') = m \sqcup \ell'$.
Therefore, for any Ω, we have, $m \sqcup (\mathcal{C} \sqcap \Omega) = \mathcal{C} \sqcap (m \sqcup \Omega)$.
Then, $\mathcal{C} \sqcap (m \sqcup \Omega) \vdash A :^{\mathcal{C}} s$.
Next, $m \sqcup \ell_0 \sqsubseteq m \sqcup \ell$ and $m \sqcup \ell \sqsubseteq \mathcal{C}$.
This case, then, follows by rule DDC-VAR.

- Rule DDC-APP. Have: $\Omega \vdash b\ a^{\ell_0} :^{\ell} B\{a/x\}$ where $\Omega \vdash b :^{\ell} \Pi x :^{\ell_0} A.B$ and $\Omega \Vdash a :^{\ell \sqcup \ell_0} A$. Further, $m \sqsubseteq \mathcal{C}$.
  Need to show: $m \sqcup \Omega \vdash b\ a^{\ell_0} :^{m \sqcup \ell} B\{a/x\}$.
  By IH, $m \sqcup \Omega \vdash b :^{m \sqcup \ell} \Pi x :^{\ell_0} A.B$.
  Now, if $\ell \sqcup \ell_0 \sqsubseteq \mathcal{C}$, then we have, $\Omega \vdash a :^{\ell \sqcup \ell_0} A$. By IH, $m \sqcup \Omega \vdash a :^{m \sqcup (\ell \sqcup \ell_0)} A$. Using associativity of $\sqcup$ and rule TC-LEQ, we have, $m \sqcup \Omega \Vdash a :^{(m \sqcup \ell) \sqcup \ell_0} A$.
  Otherwise, we have, $\ell \sqcup \ell_0 = \top$ and $\mathcal{C} \sqcap \Omega \vdash a :^{\mathcal{C}} A$. Then, reasoning as in the previous case, we have, $\mathcal{C} \sqcap (m \sqcup \Omega) \vdash a :^{\mathcal{C}} A$. So, by rule TC-TOP, $m \sqcup \Omega \Vdash a :^{\top} A$, which is same as $m \sqcup \Omega \Vdash a :^{(m \sqcup \ell) \sqcup \ell_0} A$.
  This case, then, follows by rule DDC-APP.

- Rule DDC-CONV. Have: $\Omega \vdash a :^{\ell} B$ where $\Omega \vdash a :^{\ell} A$ and $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash A \equiv_{\mathcal{C}} B$ and $\mathcal{C} \sqcap \Omega \vdash B :^{\mathcal{C}} s$. Further, $m \sqsubseteq \mathcal{C}$.
  Need to show: $m \sqcup \Omega \vdash a :^{m \sqcup \ell} B$.
  To show this case, we use the following auxiliary lemma: if $\Phi_1, x : \ell_1, \Phi_2 \vdash a \equiv_{\ell_2} b$ and $\ell_0 \sqsubseteq \ell_2$, then $\Phi_1, x : \ell_0 \sqcup \ell_1, \Phi_2 \vdash a \equiv_{\ell_2} b$.
  The proof of this auxiliary lemma is by a straightforward induction on $\Phi_1, x : \ell_1, \Phi_2 \vdash a \equiv_{\ell_2} b$.
  Now, using this auxiliary lemma, we have, $\lceil m \sqcup (\mathcal{C} \sqcap \Omega) \rceil \vdash A \equiv_{\mathcal{C}} B$.
  But $m \sqcup (\mathcal{C} \sqcap \Omega) = \mathcal{C} \sqcap (m \sqcup \Omega)$, as shown in the DDC-VAR case.
  Therefore, we have, $\lceil \mathcal{C} \sqcap (m \sqcup \Omega) \rceil \vdash A \equiv_{\mathcal{C}} B$.
  Next, by IH, we have, $m \sqcup (\mathcal{C} \sqcap \Omega) \vdash B :^{\mathcal{C}} s$, which is same as $\mathcal{C} \sqcap (m \sqcup \Omega) \vdash B :^{\mathcal{C}} s$.
  Again, by IH, we have, $m \sqcup \Omega \vdash a :^{m \sqcup \ell} A$.
  This case, then, follows by rule DDC-CONV.

□

Lemma B.45 (Lemma 3.32) If $\Omega_1, y :^{m_0} B, \Omega_2 \vdash a :^{\ell} A$ and $\ell_0 \sqsubseteq \ell$, then $\Omega_1, y :^{\ell_0 \sqcup m_0} B, \Omega_2 \vdash a :^{\ell} A$.

Proof. Let $\Omega_1, y :^{m_0} B, \Omega_2 \vdash a :^{\ell} A$ and $\ell_0 \sqsubseteq \ell$.
Since $\ell \sqsubseteq \mathcal{C}$, so $\ell_0 \sqsubseteq \mathcal{C}$.
Then, by Lemma B.44, $\ell_0 \sqcup \Omega_1, y :^{\ell_0 \sqcup m_0} B, \ell_0 \sqcup \Omega_2 \vdash a :^{\ell_0 \sqcup \ell} A$.
Now, since $\ell_0 \sqsubseteq \ell$, so $\ell_0 \sqcup \ell = \ell$.
Then, $\ell_0 \sqcup \Omega_1, y :^{\ell_0 \sqcup m_0} B, \ell_0 \sqcup \Omega_2 \vdash a :^{\ell} A$.
Next, $\Omega_1 \sqsubseteq \ell_0 \sqcup \Omega_1$ and $\Omega_2 \sqsubseteq \ell_0 \sqcup \Omega_2$.
So, by Lemma B.43, $\Omega_1, y :^{\ell_0 \sqcup m_0} B, \Omega_2 \vdash a :^{\ell} A$. □

Lemma B.46 (Lemma 3.34) If $\Omega \vdash a :^{\ell} A$ and $\ell \sqsubseteq m$ and $m \sqsubseteq \mathcal{C}$, then $\Omega \vdash a :^{m} A$

Proof. By induction on $\Omega \vdash a :^{\ell} A$. We present some of the interesting cases below.

- Rule DDC-VAR. Have: $\Omega, x :^{\ell_0} A \vdash x :^{\ell} A$ where $\Omega \Vdash A :^{\top} s$ and $\ell_0 \sqsubseteq \ell$. Further, $\ell \sqsubseteq m$ and $m \sqsubseteq \mathcal{C}$.
  Need to show: $\Omega, x :^{\ell_0} A \vdash x :^{m} A$.
  Since $\ell_0 \sqsubseteq \ell$ and $\ell \sqsubseteq m$, so $\ell_0 \sqsubseteq m$.
  This case, then, follows by rule DDC-VAR.
- Rule DDC-APP. Have: $\Omega \vdash b\ a^{\ell_0} :^{\ell} B\{a/x\}$ where $\Omega \vdash b :^{\ell} \Pi x :^{\ell_0} A.B$ and $\Omega \Vdash a :^{\ell \sqcup \ell_0} A$. Further, $\ell \sqsubseteq m$ and $m \sqsubseteq \mathcal{C}$.
  Need to show: $\Omega \vdash b\ a^{\ell_0} :^{m} B\{a/x\}$.
  By IH, $\Omega \vdash b :^{m} \Pi x :^{\ell_0} A.B$.
  Now, if $\ell \sqcup \ell_0 \sqsubseteq \mathcal{C}$, then we have, $\Omega \vdash a :^{\ell \sqcup \ell_0} A$. Now, $\ell \sqcup \ell_0 \sqsubseteq m \sqcup \ell_0$ and $m \sqcup \ell_0 \sqsubseteq \mathcal{C}$. Then, by IH, $\Omega \vdash a :^{m \sqcup \ell_0} A$. And by rule TC-LEQ, $\Omega \Vdash a :^{m \sqcup \ell_0} A$.
  Otherwise, $\ell \sqcup \ell_0 = \top$ and $\mathcal{C} \sqcap \Omega \vdash a :^{\mathcal{C}} A$. Then, $m \sqcup \ell_0 = \top$ and $\Omega \Vdash a :^{m \sqcup \ell_0} A$.
  This case, then, follows by rule DDC-APP.
- Rule DDC-CONV. Have: $\Omega \vdash a :^{\ell} B$ where $\Omega \vdash a :^{\ell} A$ and $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash A \equiv_{\mathcal{C}} B$ and $\Omega \Vdash B :^{\top} s$. Further, $\ell \sqsubseteq m$ and $m \sqsubseteq \mathcal{C}$.
  Need to show: $\Omega \vdash a :^{m} B$.
  By IH, $\Omega \vdash a :^{m} A$.
  This case, then, follows by rule DDC-CONV.

□

Lemma B.47 (Weakening for definitional equality) If $\Phi_1, \Phi_2 \vdash a_1 \equiv_{\ell} a_2$, then $\Phi_1, z : m, \Phi_2 \vdash a_1 \equiv_{\ell} a_2$.

Proof. By induction on $\Phi_1, \Phi_2 \vdash a_1 \equiv_{\ell} a_2$. □

Lemma B.48 (Lemma 3.35) If $\Omega_1, \Omega_2 \vdash a :^{\ell} A$ and $\Omega_1 \Vdash B :^{\top} s$, then $\Omega_1, y :^{m} B, \Omega_2 \vdash a :^{\ell} A$.

Proof. By induction on $\Omega_1, \Omega_2 \vdash a :^{\ell} A$. □

Lemma B.49 (Substitution for definitional equality) If $\Phi_1, z : m, \Phi_2 \vdash a_1 \equiv_{\ell} a_2$ and $\Phi_1 \vdash^{m} c_1 \equiv_{\ell} c_2$, then $\Phi_1, \Phi_2 \vdash a_1\{c_1/z\} \equiv_{\ell} a_2\{c_2/z\}$.

Proof. By induction on $\Phi_1, z : m, \Phi_2 \vdash a_1 \equiv_{\ell} a_2$. We present some of the interesting cases below.

- Rule EQ-REFL. Have: $\Phi_1, z : m, \Phi_2 \vdash a \equiv_{\ell} a$ where $\Phi_1, z : m, \Phi_2 \vdash a : \ell$. Further, $\Phi_1 \vdash^{m} c_1 \equiv_{\ell} c_2$.
  Need to show: $\Phi_1, \Phi_2 \vdash a\{c_1/z\} \equiv_{\ell} a\{c_2/z\}$.
  The proof for this case is by induction on $\Phi_1, z : m, \Phi_2 \vdash a : \ell$. We present some of the interesting cases below.

- Rule IndD-Var. There are three cases to consider.
  - Have: $\Phi_{11}, x{:}\,\ell_0, \Phi_{12}, z{:}\,m, \Phi_2 \vdash x : \ell$ where $\ell_0 \sqsubseteq \ell$. Further, $\Phi_{11}, x{:}\,\ell_0, \Phi_{12} \vdash^m c_1 \equiv_\ell c_2$.
    Need to show: $\Phi_1, x{:}\,\ell_0, \Phi_{12}, \Phi_2 \vdash x \equiv_\ell x$.
    By rule IndD-Var, $\Phi_1, x{:}\,\ell_0, \Phi_{12}, \Phi_2 \vdash x : \ell$.
    This case, then, follows by rule Eq-Refl.
  - Have: $\Phi_1, x{:}\,\ell_0, \Phi_2 \vdash x : \ell$ where $\ell_0 \sqsubseteq \ell$. Further, $\Phi_1 \vdash^{\ell_0} c_1 \equiv_\ell c_2$.
    Need to show: $\Phi_1, \Phi_2 \vdash c_1 \equiv_\ell c_2$.
    Since $\ell_0 \sqsubseteq \ell$, we have, $\Phi_1 \vdash c_1 \equiv_\ell c_2$.
    This case, then, follows by weakening.
  - Have: $\Phi_1, z{:}\,m, \Phi_{21}, x{:}\,\ell_0, \Phi_{22} \vdash x : \ell$ where $\ell_0 \sqsubseteq \ell$. Further, $\Phi_1 \vdash^m c_1 \equiv_\ell c_2$.
    Need to show: $\Phi_1, \Phi_{21}, x{:}\,\ell_0, \Phi_{22} \vdash x \equiv_\ell x$.
    By rule IndD-Var, $\Phi_1, \Phi_{21}, x{:}\,\ell_0, \Phi_{22} \vdash x : \ell$.
    This case, then, follows by rule Eq-Refl.
- Rule IndD-Lam. Have: $\Phi_1, z : m, \Phi_2 \vdash \lambda x :^{\ell_0} A.b : \ell$ where $\Phi_1, z : m, \Phi_2, x : \ell \sqcup \ell_0 \vdash b : \ell$ and $\Phi_1, z{:}\,m, \Phi_2 \vdash^{\top}_{\ell} A \sim A$. Further, $\Phi_1 \vdash^m c_1 \equiv_\ell c_2$.
  Need to show: $\Phi_1, \Phi_2 \vdash \lambda x{:}^{\ell_0} A\{c_1/z\}.b\{c_1/z\} \equiv_\ell \lambda x{:}^{\ell_0} A\{c_2/z\}.b\{c_2/z\}$.
  By IH, $\Phi_1, \Phi_2, x{:}\,\ell \sqcup \ell_0 \vdash b\{c_1/z\} \equiv_\ell b\{c_2/z\}$.
  Now, if $\top \sqsubseteq \ell$, then we have, $\Phi_1, z{:}\,m, \Phi_2 \vdash A : \ell$. By IH, $\Phi_1, \Phi_2 \vdash A\{c_1/z\} \equiv_\ell A\{c_2/z\}$. Then, by rule CEq-Leq, $\Phi_1, \Phi_2 \vdash^{\top} A\{c_1/z\} \equiv_\ell A\{c_2/z\}$.
  Otherwise, by rule CEq-Nleq, $\Phi_1, \Phi_2 \vdash^{\top} A\{c_1/z\} \equiv_\ell A\{c_2/z\}$.
  This case, then, follows by rule Eq-Lam.
- Rule IndD-App. Have: $\Phi_1, z{:}\,m, \Phi_2 \vdash b\ a^{\ell_0} : \ell$ where $\Phi_1, z{:}\,m, \Phi_2 \vdash b : \ell$ and $\Phi_1, z{:}\,m, \Phi_2 \vdash^{\ell_0}_{\ell} a \sim a$. Further, $\Phi_1 \vdash^m c_1 \equiv_\ell c_2$.
  Need to show: $\Phi_1, \Phi_2 \vdash b\{c_1/z\}\ (a\{c_1/z\})^{\ell_0} \equiv_\ell b\{c_2/z\}\ (a\{c_2/z\})^{\ell_0}$.
  By IH, $\Phi_1, \Phi_2 \vdash b\{c_1/z\} \equiv_\ell b\{c_2/z\}$.
  Now, if $\ell_0 \sqsubseteq \ell$, then we have, $\Phi_1, z : m, \Phi_2 \vdash a : \ell$. By IH, $\Phi_1, \Phi_2 \vdash a\{c_1/z\} \equiv_\ell a\{c_2/z\}$. By rule CEq-Leq, $\Phi_1, \Phi_2 \vdash^{\ell_0} a\{c_1/z\} \equiv_\ell a\{c_2/z\}$.
  Otherwise, by rule CEq-Nleq, $\Phi_1, \Phi_2 \vdash^{\ell_0} a\{c_1/z\} \equiv_\ell a\{c_2/z\}$.
  This case, then, follows by rule Eq-App.

- Rule Eq-Sym. Have: $\Phi_1, z{:}\,m, \Phi_2 \vdash a_1 \equiv_\ell a_2$ where $\Phi_1, z{:}\,m, \Phi_2 \vdash a_2 \equiv_\ell a_1$. Further, $\Phi_1 \vdash^m c_1 \equiv_\ell c_2$.
  Need to show: $\Phi_1, \Phi_2 \vdash a_1\{c_1/z\} \equiv_\ell a_2\{c_2/z\}$.
  There are two possibilities.
  - $m \sqsubseteq \ell$. Then, we have, $\Phi_1 \vdash c_1 \equiv_\ell c_2$.
    By rule Eq-Sym, $\Phi_1 \vdash c_2 \equiv_\ell c_1$. And by rule CEq-Leq, $\Phi_1 \vdash^m c_2 \equiv_\ell c_1$.
    By IH, $\Phi_1, \Phi_2 \vdash a_2\{c_2/z\} \equiv_\ell a_1\{c_1/z\}$.
    This case, then, follows by rule Eq-Sym.
  - $\neg(m \sqsubseteq \ell)$. Then, by rule CEq-Nleq, $\Phi \vdash^m c_2 \equiv_\ell c_1$.
    By IH, $\Phi_1, \Phi_2 \vdash a_2\{c_2/z\} \equiv_\ell a_1\{c_1/z\}$.
    This case, then, follows by rule Eq-Sym.

- Rule Eq-Trans. Have: $\Phi_1, z\!:m, \Phi_2 \vdash a_1 \equiv_\ell a_2$ where $\Phi_1, z\!:m, \Phi_2 \vdash a_1 \equiv_\ell a_0$ and $\Phi_1, z\!:m, \Phi_2 \vdash a_0 \equiv_\ell a_2$. Further, $\Phi_1 \vdash^m c_1 \equiv_\ell c_2$.
  Need to show: $\Phi_1, \Phi_2 \vdash a_1\{c_1/z\} \equiv_\ell a_2\{c_2/z\}$.
  By IH, $\Phi_1, \Phi_2 \vdash a_1\{c_1/z\} \equiv_\ell a_0\{c_2/z\}$.
  Now, since $\Phi_1 \vdash^m c_2 \equiv_\ell c_2$, so by IH, $\Phi_1, \Phi_2 \vdash a_0\{c_2/z\} \equiv_\ell a_2\{c_2/z\}$.
  This case, then, follows by rule Eq-Trans.

- Rule Eq-Beta. Have: $\Phi_1, z : m, \Phi_2 \vdash a \equiv_\ell b$ where $\Phi_1, z : m, \Phi_2 \vdash a : \ell$ and $\vdash a \leadsto b$. Further, $\Phi_1 \vdash^m c_1 \equiv_\ell c_2$.
  Need to show: $\Phi_1, \Phi_2 \vdash a\{c_1/z\} \equiv_\ell b\{c_2/z\}$.
  Reasoning as in the Eq-Refl case, we have, $\Phi_1, \Phi_2 \vdash a\{c_1/z\} \equiv_\ell a\{c_2/z\}$.
  Now, by substitution property of the reduction relation, we have, $\vdash a\{c_2/z\} \leadsto b\{c_2/z\}$.
  Since $\Phi_1, \Phi_2 \vdash a\{c_2/z\} : \ell$ and $\vdash a\{c_2/z\} \leadsto b\{c_2/z\}$, so by rule Eq-Beta, $\Phi_1, \Phi_2 \vdash a\{c_2/z\} \equiv_\ell b\{c_2/z\}$.
  This case, then, follows by rule Eq-Trans.

- Rule Eq-Lam. Have: $\Phi_1, z\!:m, \Phi_2 \vdash \lambda x\!:^{\ell_0} A_1.b_1 \equiv_\ell \lambda x\!:^{\ell_0} A_2.b_2$ where $\Phi_1, z\!:m, \Phi_2, x\!:\ell \sqcup \ell_0 \vdash b_1 \equiv_\ell b_2$ and $\Phi_1, z\!:m, \Phi_2 \vdash^\top A_1 \equiv_\ell A_2$. Further, $\Phi_1 \vdash^m c_1 \equiv_\ell c_2$.
  Need to show: $\Phi_1, \Phi_2 \vdash \lambda x\!:^{\ell_0} A_1\{c_1/z\}.b_1\{c_1/z\} \equiv_\ell \lambda x\!:^{\ell_0} A_2\{c_2/z\}.b_2\{c_2/z\}$.
  By IH, $\Phi_1, \Phi_2, x\!:\ell \sqcup \ell_0 \vdash b_1\{c_1/z\} \equiv_\ell b_2\{c_2/z\}$.
  Now, if $\top \sqsubseteq \ell$, then we have, $\Phi_1, z\!:m, \Phi_2 \vdash A_1 \equiv_\ell A_2$. By IH, $\Phi_1, \Phi_2 \vdash A_1\{c_1/z\} \equiv_\ell A_2\{c_2/z\}$. Then, by rule CEq-Leq, $\Phi_1, \Phi_2 \vdash^\top A_1\{c_1/z\} \equiv_\ell A_2\{c_2/z\}$.
  Otherwise, by rule CEq-Nleq, $\Phi_1, \Phi_2 \vdash^\top A_1\{c_1/z\} \equiv_\ell A_2\{c_2/z\}$.
  This case, then, follows by rule Eq-Lam.

- Rule Eq-App. Have: $\Phi_1, z\!:m, \Phi_2 \vdash b_1\ {a_1}^{\ell_0} \equiv_\ell b_2\ {a_2}^{\ell_0}$ where $\Phi_1, z\!:m, \Phi_2 \vdash b_1 \equiv_\ell b_2$ and $\Phi_1, z\!:m, \Phi_2 \vdash^{\ell_0} a_1 \equiv_\ell a_2$. Further, $\Phi_1 \vdash^m c_1 \equiv_\ell c_2$.
  Need to show: $\Phi_1, \Phi_2 \vdash b_1\{c_1/z\}\ (a_1\{c_1/z\})^{\ell_0} \equiv_\ell b_2\{c_2/z\}\ (a_2\{c_2/z\})^{\ell_0}$.
  By IH, $\Phi_1, \Phi_2 \vdash b_1\{c_1/z\} \equiv_\ell b_2\{c_2/z\}$.
  Now, if $\ell_0 \sqsubseteq \ell$, then we have, $\Phi_1, z\!:m, \Phi_2 \vdash a_1 \equiv_\ell a_2$. By IH, $\Phi_1, \Phi_2 \vdash a_1\{c_1/z\} \equiv_\ell a_2\{c_2/z\}$. Then, by rule CEq-Leq, $\Phi_1, \Phi_2 \vdash^{\ell_0} a_1\{c_1/z\} \equiv_\ell a_2\{c_2/z\}$.
  Otherwise, by rule CEq-Nleq, $\Phi_1, \Phi_2 \vdash^{\ell_0} a_1\{c_1/z\} \equiv_\ell a_2\{c_2/z\}$.
  This case, then, follows by rule Eq-App.

□

Lemma B.50 (Lemma 3.36) If $\Omega_1, y\!:^m B, \Omega_2 \vdash a :^\ell A$ and $\Omega_1 \Vdash b :^m B$, then $\Omega_1, \Omega_2\{b/y\} \vdash a\{b/y\} :^\ell A\{b/y\}$

Proof. By induction on $\Omega_1, y\!:^m B, \Omega_2 \vdash a :^\ell A$. We present some of the interesting cases below.

- Rule Ddc-Var. There are two cases to consider.
  - Have: $\Omega_1, y\!:^m B, \Omega_2, x\!:^{\ell_0} A \vdash x :^\ell A$ where $\mathcal{C} \sqcap \Omega_1, y\!:^{\mathcal{C} \sqcap m} B, \mathcal{C} \sqcap \Omega_2 \vdash A :^{\mathcal{C}} s$ and $\ell_0 \sqsubseteq \ell$ and $\ell \sqsubseteq \mathcal{C}$. Further, $\Omega_1 \Vdash b :^m B$.

Need to show: $\Omega_1, \Omega_2\{b/y\}, x :^{\ell_0} A\{b/y\} \vdash x :^{\ell} A\{b/y\}$.
There are two possibilities:

* $m \sqsubseteq \mathcal{C}$. Then, we have, $\Omega_1 \vdash b :^{m} B$. By Lemma B.43, $\mathcal{C} \sqcap \Omega_1 \vdash b :^{m} B$. And by rule TC-Leq, $\mathcal{C} \sqcap \Omega_1 \Vdash b :^{m} B$.
  We also have, $\mathcal{C} \sqcap \Omega_1, y :^{m} B, \mathcal{C} \sqcap \Omega_2 \vdash A :^{\mathcal{C}} s$.
  Then, by IH, $\mathcal{C} \sqcap \Omega_1, \mathcal{C} \sqcap \Omega_2\{b/y\} \vdash A\{b/y\} :^{\mathcal{C}} s$.
  This case, then, follows by rule Ddc-Var.
* $m = \top$. Then, we have, $\mathcal{C} \sqcap \Omega_1 \vdash b :^{\mathcal{C}} B$. By rule TC-Leq, $\mathcal{C} \sqcap \Omega_1 \Vdash b :^{\mathcal{C}} B$.
  We also have, $\mathcal{C} \sqcap \Omega_1, y :^{\mathcal{C}} B, \mathcal{C} \sqcap \Omega_2 \vdash A :^{\mathcal{C}} s$.
  Then, by IH, $\mathcal{C} \sqcap \Omega_1, \mathcal{C} \sqcap \Omega_2\{b/y\} \vdash A\{b/y\} :^{\mathcal{C}} s$.
  This case, then, follows by rule Ddc-Var.

– Have: $\Omega, x :^{\ell_0} A \vdash x :^{\ell} A$ where $\mathcal{C} \sqcap \Omega \vdash A :^{\mathcal{C}} s$ and $\ell_0 \sqsubseteq \ell$ and $\ell \sqsubseteq \mathcal{C}$. Further, $\Omega \Vdash a :^{\ell_0} A$.
  Need to show: $\Omega \vdash a :^{\ell} A$.
  Since $\ell_0 \sqsubseteq \ell$ and $\ell \sqsubseteq \mathcal{C}$, so $\ell_0 \sqsubseteq \mathcal{C}$.
  As such, we have, $\Omega \vdash a :^{\ell_0} A$.
  Then, since $\ell_0 \sqsubseteq \ell$ and $\ell \sqsubseteq \mathcal{C}$, by Lemma B.46, $\Omega \vdash a :^{\ell} A$.

- Rule Ddc-App. Have: $\Omega_1, y :^{m} B, \Omega_2 \vdash a_2\ {a_1}^{\ell_0} :^{\ell} A_2\{a_1/x\}$ where $\Omega_1, y :^{m} B, \Omega_2 \vdash a_2 :^{\ell} \Pi x :^{\ell_0} A_1.A_2$ and $\Omega_1, y :^{m} B, \Omega_2 \Vdash a_1 :^{\ell \sqcup \ell_0} A_1$. Further, $\Omega_1 \Vdash b :^{m} B$.
  Need to show: $\Omega_1, \Omega_2\{b/y\} \vdash a_2\{b/y\}\ (a_1\{b/y\})^{\ell_0} :^{\ell} A_2\{b/y\}\{a_1\{b/y\}/x\}$.
  By IH, $\Omega_1, \Omega_2\{b/y\} \vdash a_2\{b/y\} :^{\ell} \Pi x : A_1\{b/y\}.A_2\{b/y\}$.
  Now, there are two possibilities:

  – $\ell \sqcup \ell_0 \sqsubseteq \mathcal{C}$. Then, we have, $\Omega_1, y :^{m} B, \Omega_2 \vdash a_1 :^{\ell \sqcup \ell_0} A_1$.
    By IH, $\Omega_1, \Omega_2\{b/y\} \vdash a_1\{b/y\} :^{\ell \sqcup \ell_0} A_1\{b/y\}$.
    And by rule TC-Leq, $\Omega_1, \Omega_2\{b/y\} \Vdash a_1\{b/y\} :^{\ell \sqcup \ell_0} A_1\{b/y\}$.
    This case, then, follows by rule Ddc-App.
  – $\ell \sqcup \ell_0 = \top$. Then, we have, $\mathcal{C} \sqcap \Omega_1, y :^{\mathcal{C} \sqcap m} B, \mathcal{C} \sqcap \Omega_2 \vdash a_1 :^{\mathcal{C}} A_1$.
    Reasoning as in Ddc-Var case, we have, $\mathcal{C} \sqcap \Omega_1, \mathcal{C} \sqcap \Omega_2\{b/y\} \vdash a_1\{b/y\} :^{\mathcal{C}} A_1\{b/y\}$.
    Then, by rule TC-Top, we have, $\Omega_1, \Omega_2\{b/y\} \Vdash a_1\{b/y\} :^{\ell \sqcup \ell_0} A_1\{b/y\}$.
    This case, then, follows by rule Ddc-App.

- Rule Ddc-Conv. Have: $\Omega_1, y :^{m} B, \Omega_2 \vdash a :^{\ell} A_2$ where $\Omega_1, y :^{m} B, \Omega_2 \vdash a :^{\ell} A_1$ and $\lceil \mathcal{C} \sqcap \Omega_1, y :^{\mathcal{C} \sqcap m} B, \mathcal{C} \sqcap \Omega_2 \rceil \vdash A_1 \equiv_{\mathcal{C}} A_2$ and $\mathcal{C} \sqcap \Omega_1, y :^{\mathcal{C} \sqcap m} B, \mathcal{C} \sqcap \Omega_2 \vdash A_2 :^{\mathcal{C}} s$. Further, $\Omega_1 \Vdash b :^{m} B$.
  Need to show: $\Omega_1, \Omega_2\{b/y\} \vdash a\{b/y\} :^{\ell} A_2\{b/y\}$.
  By IH, $\Omega_1, \Omega_2\{b/y\} \vdash a\{b/y\} :^{\ell} A_1\{b/y\}$.
  Next, reasoning as in Ddc-Var case, we have, $\mathcal{C} \sqcap \Omega_1, \mathcal{C} \sqcap \Omega_2\{b/y\} \vdash A_2\{b/y\} :^{\mathcal{C}} s$.
  Now, there are two possibilities:

  – $m \sqsubseteq \mathcal{C}$. Then , we have, $\Omega_1 \vdash b :^{m} B$.
    By Lemma B.43, $\mathcal{C} \sqcap \Omega_1 \vdash b :^{m} B$.

By Lemma B.46, $\mathcal{C} \sqcap \Omega_1 \vdash b :^{\mathcal{C}} B$.
By Lemma B.28, $\lceil \mathcal{C} \sqcap \Omega_1 \rceil \vdash b : \mathcal{C}$.
By rule Eq-Refl, $\lceil \mathcal{C} \sqcap \Omega_1 \rceil \vdash b \equiv_{\mathcal{C}} b$.
By rule CEq-Leq, $\lceil \mathcal{C} \sqcap \Omega_1 \rceil \vdash^{m} b \equiv_{\mathcal{C}} b$.
Then, since $\lceil \mathcal{C} \sqcap \Omega_1, y :^{m} B, \mathcal{C} \sqcap \Omega_2 \rceil \vdash A_1 \equiv_{\mathcal{C}} A_2$, so by Lemma B.49, $\lceil \mathcal{C} \sqcap \Omega_1, \mathcal{C} \sqcap \Omega_2\{b/y\} \rceil \vdash A_1\{b/y\} \equiv_{\mathcal{C}} A_2\{b/y\}$.
This case, then, follows by rule Ddc-Conv.

- $m = \top$. Then, we have, $\mathcal{C} \sqcap \Omega_1 \vdash b :^{\mathcal{C}} B$.
  By Lemma B.28, $\lceil \mathcal{C} \sqcap \Omega_1 \rceil \vdash b : \mathcal{C}$.
  By rule Eq-Refl, $\lceil \mathcal{C} \sqcap \Omega_1 \rceil \vdash b \equiv_{\mathcal{C}} b$.
  By rule CEq-Leq, $\lceil \mathcal{C} \sqcap \Omega_1 \rceil \vdash^{\mathcal{C}} b \equiv_{\mathcal{C}} b$.
  Then, since $\lceil \mathcal{C} \sqcap \Omega_1, y :^{\mathcal{C}} B, \mathcal{C} \sqcap \Omega_2 \rceil \vdash A_1 \equiv_{\mathcal{C}} A_2$, so by Lemma B.49, $\lceil \mathcal{C} \sqcap \Omega_1, \mathcal{C} \sqcap \Omega_2\{b/y\} \rceil \vdash A_1\{b/y\} \equiv_{\mathcal{C}} A_2\{b/y\}$.
  This case, then, follows by rule Ddc-Conv.

□

Lemma B.51 (Regularity) If $\Omega \vdash a :^{\ell} A$, then $\exists s$ such that $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash A \equiv_{\mathcal{C}} s$ or $\mathcal{C} \sqcap \Omega \vdash A :^{\mathcal{C}} s$.

Proof. By induction on $\Omega \vdash a :^{\ell} A$. We present some of the interesting cases below.

- Rule Ddc-Var. Have: $\Omega, x :^{\ell_0} A \vdash x :^{\ell} A$ where $\mathcal{C} \sqcap \Omega \vdash A :^{\mathcal{C}} s$.
  By weakening lemma B.48, $\mathcal{C} \sqcap \Omega, x :^{\mathcal{C} \sqcap \ell_0} A \vdash A :^{\mathcal{C}} s$.
  This case, then, follows by setting $s \coloneqq s$.

- Rule Ddc-Weak. Have: $\Omega, y :^{\ell_0} B \vdash a :^{\ell} A$ where $\Omega \vdash a :^{\ell} A$ and $\mathcal{C} \sqcap \Omega \vdash B :^{\mathcal{C}} s$.
  This case follows by IH and weakening lemmas.

- Rule Ddc-Axiom. Have: $\varnothing \vdash s_1 : s_2$ where $(s_1, s_2) \in \mathcal{A}$.
  By rule IndD-Sort, we have, $\varnothing \vdash s_2 \sim_{\mathcal{C}} s_2$.
  Then, by rule Eq-Refl, $\varnothing \vdash s_2 \equiv_{\mathcal{C}} s_2$.
  This case, then, follows by setting $s \coloneqq s_2$.

- Rule Ddc-Pi. Have: $\Omega \vdash \Pi x :^{\ell_0} A.B : s_3$ where $\Omega \vdash A :^{\ell} s_1$ and $\Omega, x :^{\ell} A \vdash B :^{\ell} s_2$ and $(s_1, s_2, s_3) \in \mathcal{R}$.
  This case follows by setting $s \coloneqq s_3$.

- Rule Ddc-Lam. Have: $\Omega \vdash \lambda x :^{\ell_0} A.b :^{\ell} \Pi x :^{\ell_0} A.B$ where $\Omega, x :^{\ell \sqcup \ell_0} A \vdash b :^{\ell} B$ and $\mathcal{C} \sqcap \Omega \vdash \Pi x :^{\ell_0} A.B :^{\mathcal{C}} s$.
  This case follows by setting $s \coloneqq s$.

- Rule Ddc-App. Have: $\Omega \vdash b\ a^{\ell_0} :^{\ell} B\{a/x\}$ where $\Omega \vdash b :^{\ell} \Pi x :^{\ell_0} A.B$ and $\Omega \Vdash a :^{\ell \sqcup \ell_0} A$.
  By IH, there exists s such that $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash \Pi x :^{\ell_0} A.B \equiv_{\mathcal{C}} s$ or $\mathcal{C} \sqcap \Omega \vdash \Pi x :^{\ell_0} A.B :^{\mathcal{C}} s$.
  Now, since definitional equality is consistent, $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash \Pi x :^{\ell_0} A.B \equiv_{\mathcal{C}} s$ cannot hold.

As such, $\mathcal{C} \sqcap \Omega \vdash \Pi x{:}^{\ell_0} A.B :^{\mathcal{C}} s$.

Then, by inversion, there exists s_1 and s_2 such that $\mathcal{C} \sqcap \Omega \vdash A :^{\mathcal{C}} s_1$ and $\mathcal{C} \sqcap \Omega, x{:}^{\mathcal{C}} A \vdash B :^{\mathcal{C}} s_2$.

Now, we show, $\mathcal{C} \sqcap \Omega \vdash a :^{\mathcal{C}} A$.

We have: $\Omega \Vdash a :^{\ell \sqcup \ell_0} A$.

If $\ell \sqcup \ell_0 \sqsubseteq \mathcal{C}$, then, $\Omega \vdash a :^{\ell \sqcup \ell_0} A$. By narrowing and subsumption, $\mathcal{C} \sqcap \Omega \vdash a :^{\mathcal{C}} A$.

Otherwise, $\ell \sqcup \ell_0 = \top$ and $\mathcal{C} \sqcap \Omega \vdash a :^{\mathcal{C}} A$.

So in either case, $\mathcal{C} \sqcap \Omega \vdash a :^{\mathcal{C}} A$.

Now, by substitution lemma B.50, $\mathcal{C} \sqcap \Omega \vdash B\{a/x\} :^{\mathcal{C}} s_2$.

This case, then, follows by setting $s := s_2$.

- Rule Ddc-Conv. Have: $\Omega \vdash a :^{\ell} B$ where $\Omega \vdash a :^{\ell} A$ and $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash A \equiv_{\mathcal{C}} B$ and $\mathcal{C} \sqcap \Omega \vdash B :^{\mathcal{C}} s$.
  This case follows by setting $s := s$.

$\square$

Lemma B.52 The projection rules Ddc-Proj1 and Ddc-Proj2 are derivable.

Proof. First, we derive rule Ddc-Proj1.

Have: $\Omega \vdash a :^{\ell} \Sigma x{:}^{\ell_0} A.B$ where $\ell_0 \sqsubseteq \ell$.

Need to show: $\Omega \vdash \mathbf{proj}_1^{\ell_0}\ a :^{\ell} A$ where $\mathbf{proj}_1^{\ell_0}\ a \triangleq \mathbf{let}\ (x^{\ell_0}, y) = a\ \mathbf{in}\ x$.

Since $\ell_0 \sqsubseteq \ell$, so $\ell \sqcup \ell_0 \sqsubseteq \ell$.

Since $\Omega \vdash a :^{\ell} \Sigma x{:}^{\ell_0} A.B$, so $\ell \sqsubseteq \mathcal{C}$.

Again, since $\Omega \vdash a :^{\ell} \Sigma x{:}^{\ell_0} A.B$, by regularity and consistency, $\mathcal{C} \sqcap \Omega \vdash \Sigma x{:}^{\ell_0} A.B :^{\mathcal{C}} s$, for some s.

Then, by inversion, we have, $\mathcal{C} \sqcap \Omega \vdash A :^{\mathcal{C}} s_1$ and $\mathcal{C} \sqcap \Omega, x{:}^{\mathcal{C}} A \vdash B :^{\mathcal{C}} s_2$, for some s_1 and s_2.

Next, since $\mathcal{C} \sqcap \Omega \vdash A :^{\mathcal{C}} s_1$ and $\ell \sqcup \ell_0 \sqsubseteq \ell$ and $\ell \sqsubseteq \mathcal{C}$, so by rule Ddc-Var, $\Omega, x{:}^{\ell \sqcup \ell_0} A \vdash x :^{\ell} A$.

Then, by weakening, $\Omega, x{:}^{\ell \sqcup \ell_0} A, y{:}^{\ell} B \vdash x :^{\ell} A$.

Finally, by rule Ddc-LetPair, we have, $\Omega \vdash \mathbf{let}\ (x^{\ell_0}, y) = a\ \mathbf{in}\ x :^{\ell} A$, as required.

Now, we derive rule Ddc-Proj2.

Have: $\Omega \vdash a :^{\ell} \Sigma x{:}^{\ell_0} A.B$ where $\ell_0 \sqsubseteq \mathcal{C}$.

Need to show: $\Omega \vdash \mathbf{proj}_2^{\ell_0} a :^{\ell} B\{\mathbf{proj}_1^{\ell_0}\ a/x\}$ where $\mathbf{proj}_2^{\ell_0} a \triangleq \mathbf{let}\ (x^{\ell_0}, y) = a\ \mathbf{in}\ y$.

Since $\Omega \vdash a :^{\ell} \Sigma x{:}^{\ell_0} A.B$, so $\ell \sqsubseteq \mathcal{C}$.

Again, since $\Omega \vdash a :^{\ell} \Sigma x{:}^{\ell_0} A.B$, by regularity and consistency, $\mathcal{C} \sqcap \Omega \vdash \Sigma x{:}^{\ell_0} A.B :^{\mathcal{C}} s$, for some s.

Then, by inversion, we have, $\mathcal{C} \sqcap \Omega \vdash A :^{\mathcal{C}} s_1$ and $\mathcal{C} \sqcap \Omega, x{:}^{\mathcal{C}} A \vdash B :^{\mathcal{C}} s_2$, for some s_1 and s_2.

Next, by rule Ddc-Var, we have, $\mathcal{C} \sqcap \Omega, z{:}^{\mathcal{C}} \Sigma x{:}^{\ell_0} A.B \vdash z :^{\mathcal{C}} \Sigma x{:}^{\ell_0} A.B$.

Since $\ell_0 \sqsubseteq \mathcal{C}$, by the argument above, $\mathcal{C} \sqcap \Omega, z{:}^{\mathcal{C}} \Sigma x{:}^{\ell_0} A.B \vdash \mathbf{proj}_1^{\ell_0}\ z :^{\mathcal{C}} A$.

Next, since $\mathcal{C} \sqcap \Omega, x{:}^{\mathcal{C}} A \vdash B :^{\mathcal{C}} s_2$, by weakening, $\mathcal{C} \sqcap \Omega, z{:}^{\mathcal{C}} \Sigma x{:}^{\ell_0} A.B, x{:}^{\mathcal{C}} A \vdash B :^{\mathcal{C}} s_2$.

Then, by substitution, $\mathcal{C} \sqcap \Omega, z{:}^{\mathcal{C}} \Sigma x{:}^{\ell_0} A.B \vdash B\{\mathbf{proj}_1^{\ell_0}\ z/x\} :^{\mathcal{C}} s_2$.

Next, since $\ell \sqsubseteq \mathcal{C}$, by rule Ddc-Var, $\Omega, x{:}^{\ell \sqcup \ell_0} A, y{:}^{\ell} B \vdash y :^{\ell} B$.

But $\lceil \mathcal{C} \sqcap (\Omega, x{:}^{\ell \sqcup \ell_0} A, y{:}^{\ell} B) \rceil \vdash B \equiv_{\mathcal{C}} (B\{\mathbf{proj}_1^{\ell_0}\ z/x\})\{(x^{\ell_0}, y)/z\}$.

So, by rule Ddc-Conv, $\Omega, x :^{\ell \sqcup \ell_0} A, y :^{\ell} B \vdash y :^{\ell} (B\{\mathbf{proj}_1^{\ell_0}\ z/x\})\{(x^{\ell_0}, y)/z\}$.
Finally, by rule Ddc-LetPair, $\Omega \vdash \mathbf{let}\ (x^{\ell_0}, y) = a\ \mathbf{in}\ y :^{\ell} (B\{\mathbf{proj}_1^{\ell_0}\ z/x\})\{a/z\}$, or $\Omega \vdash \mathbf{let}\ (x^{\ell_0}, y) = a\ \mathbf{in}\ y :^{\ell} B\{\mathbf{proj}_1^{\ell_0}\ a/x\}$, as required.

□

Lemma B.53 (Equality inversion) The definitional equality relation satisfies the following inversion lemmas:

- If $\Phi \vdash \Pi x :^{\ell_1} A_1.B_1 \equiv_{\ell} \Pi x :^{\ell_2} A_2.B_2$, then $\ell_1 = \ell_2$ and $\Phi \vdash A_1 \equiv_{\ell} A_2$. Further, if $\Phi \vdash a_1 \equiv_{\ell} a_2$, then $\Phi \vdash B_1\{a_1/x\} \equiv_{\ell} B_2\{a_2/x\}$.
- If $\Phi \vdash \Sigma x :^{\ell_1} A_1.B_1 \equiv_{\ell} \Sigma x :^{\ell_2} A_2.B_2$, then $\ell_1 = \ell_2$ and $\Phi \vdash A_1 \equiv_{\ell} A_2$. Further, if $\Phi \vdash a_1 \equiv_{\ell} a_2$, then $\Phi \vdash B_1\{a_1/x\} \equiv_{\ell} B_2\{a_2/x\}$.

Proof. We present the proof for the Π-case below. The proof for the Σ-case is similar.
Given: $\Phi \vdash \Pi x :^{\ell_1} A_1.B_1 \equiv_{\ell} \Pi x :^{\ell_2} A_2.B_2$.
Note that since the definitional equality relation is consistent (by Theorems B.41 and B.42), we have, $\mathsf{Ct}\ (\Pi x :^{\ell_1} A_1.B_1)\ (\Pi x :^{\ell_2} A_2.B_2)$.
Therefore, $\ell_1 = \ell_2$.
Say, $\ell_0 := \ell_1$.
Now, since definitional equality implies joinability, we have, $\Phi \vdash \Pi x :^{\ell_0} A_1.B_1 \Leftrightarrow_{\ell} \Pi x :^{\ell_0} A_2.B_2$.
As such, $\exists C$ such that $\Phi \vdash \Pi x :^{\ell_0} A_1.B_1 \Rightarrow_{\ell}^{*} C$ and $\Phi \vdash \Pi x :^{\ell_0} A_2.B_2 \Rightarrow_{\ell}^{*} C$.
Since $\Phi \vdash \Pi x :^{\ell_0} A_1.B_1 \Rightarrow_{\ell}^{*} C$, so $C = \Pi x :^{\ell_0} A_0.B_0$ (for some A_0 and B_0). Further, $\Phi \vdash A_1 \Rightarrow_{\ell}^{*} A_0$ and $\Phi, x : \ell \vdash B_1 \Rightarrow_{\ell}^{*} B_0$.
Again, since $\Phi \vdash \Pi x :^{\ell_0} A_2.B_2 \Rightarrow_{\ell}^{*} C$, so $\Phi \vdash A_2 \Rightarrow_{\ell}^{*} A_0$ and $\Phi, x : \ell \vdash B_2 \Rightarrow_{\ell}^{*} B_0$.
Then, $\Phi \vdash A_1 \Leftrightarrow_{\ell} A_2$ and $\Phi, x : \ell \vdash B_1 \Leftrightarrow_{\ell} B_2$.
Since joinability implies definitional equality, we have, $\Phi \vdash A_1 \equiv_{\ell} A_2$ and $\Phi, x : \ell \vdash B_1 \equiv_{\ell} B_2$.
Next, since $\Phi \vdash a_1 \equiv_{\ell} a_2$ (given), we have, $\Phi \vdash^{\ell} a_1 \equiv_{\ell} a_2$.
Therefore, by Lemma B.49, $\Phi \vdash B_1\{a_1/x\} \equiv_{\ell} B_2\{a_2/x\}$. □

Lemma B.54 (Typing inversion) DDC satisfies the following inversion lemmas:

- If $\Omega \vdash v :^{\ell} A$ and $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash A \equiv_{\mathcal{C}} \Pi x :^{\ell_0} B_1.B_2$ and v is a value, then exists A_1, A_2 and a_2 such that
  - $v = \lambda x :^{\ell_0} A_1.a_2$
  - $\Omega, x :^{\ell \sqcup \ell_0} A_1 \vdash a_2 :^{\ell} A_2$
  - $\mathcal{C} \sqcap \Omega \vdash \Pi x :^{\ell_0} A_1.A_2 :^{\mathcal{C}} s$
  - $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash A \equiv_{\mathcal{C}} \Pi x :^{\ell_0} A_1.A_2$
- If $\Omega \vdash v :^{\ell} A$ and $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash A \equiv_{\mathcal{C}} \Sigma x :^{\ell_0} B_1.B_2$ and v is a value, then exists A_1, A_2, a_1 and a_2 such that
  - $v = ({a_1}^{\ell_0}, a_2)$
  - $\Omega \Vdash a_1 :^{\ell \sqcup \ell_0} A_1$ and $\Omega \vdash a_2 :^{\ell} A_2\{a_1/x\}$

– $\mathcal{C} \sqcap \Omega \vdash \Sigma x{:}^{\ell_0} A_1.A_2 :^{\mathcal{C}} s$

– $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash A \equiv_{\mathcal{C}} \Sigma x{:}^{\ell_0} A_1.A_2$

Proof. We present the proof for the Π-case below. The proof for the Σ-case is similar. The proof is by induction on $\Omega \vdash v :^{\ell} A$. We present some of the interesting cases below.

- Rule Ddc-Var. Have: $\Omega, x{:}^{\ell_0} A \vdash x :^{\ell} A$ where $\mathcal{C} \sqcap \Omega \vdash A :^{\mathcal{C}} s$ and $\ell_0 \sqsubseteq \ell$.
  The lemma statement does not apply because x is not a value.

- Rule Ddc-Weak. Have: $\Omega, y{:}^{m} B \vdash v :^{\ell} A$ where $\Omega \vdash v :^{\ell} A$ and $\mathcal{C} \sqcap \Omega \vdash B :^{\mathcal{C}} s$. Further, v is a value and $\lceil \mathcal{C} \sqcap \Omega, y{:}^{\mathcal{C} \sqcap m} B \rceil \vdash A \equiv_{\mathcal{C}} \Pi x{:}^{\ell_0} B_1.B_2$.
  Now, $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash \mathbf{unit} : \mathcal{C}$. Then, $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash \mathbf{unit} \equiv_{\mathcal{C}} \mathbf{unit}$.
  By lemma B.49, we have, $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash A\{\mathbf{unit}/y\} \equiv_{\mathcal{C}} \Pi x{:}^{\ell_0} B_1\{\mathbf{unit}/y\}.B_2\{\mathbf{unit}/y\}$.
  But since $y \notin \mathsf{fv}\, A$, so $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash A \equiv_{\mathcal{C}} \Pi x{:}^{\ell_0} B_1\{\mathbf{unit}/y\}.B_2\{\mathbf{unit}/y\}$.
  Using IH, we have:

  – $v = \lambda x{:}^{\ell_0} A_1.a_2$.

  – $\Omega, x{:}^{\ell \sqcup \ell_0} A_1 \vdash a_2 :^{\ell} A_2$.
  By weakening, $\Omega, y{:}^{m} B, x{:}^{\ell \sqcup \ell_0} A_1 \vdash a_2 :^{\ell} A_2$.

  – $\mathcal{C} \sqcap \Omega \vdash \Pi x{:}^{\ell_0} A_1.A_2 :^{\mathcal{C}} s$.
  By weakening, $\mathcal{C} \sqcap \Omega, y{:}^{\mathcal{C} \sqcap m} B \vdash \Pi x{:}^{\ell_0} A_1.A_2 :^{\mathcal{C}} s$.

  – $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash A \equiv_{\mathcal{C}} \Pi x{:}^{\ell_0} A_1.A_2$.
  By weakening, $\lceil \mathcal{C} \sqcap \Omega, y{:}^{\mathcal{C} \sqcap m} B \rceil \vdash A \equiv_{\mathcal{C}} \Pi x{:}^{\ell_0} A_1.A_2$.

- Rule Ddc-Pi. Have: $\Omega \vdash \Pi x :^{m_0} A_1.A_2 :^{\ell} s_3$ where $\Omega \vdash A_1 :^{\ell} s_1$ and $\Omega, x :^{\ell} A_1 \vdash A_2 :^{\ell} s_2$ and $(s_1, s_2, s_3) \in \mathcal{R}$. Further, $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash s_3 \equiv_{\mathcal{C}} \Pi x{:}^{\ell_0} B_1.B_2$.
  But by consistency of definitional equality, $\mathsf{Ct}\ s_3\ (\Pi x{:}^{\ell_0} B_1.B_2)$, a contradiction.

- Rule Ddc-Lam. Have: $\Omega \vdash \lambda x :^{m_0} A_1.a_2 :^{\ell} \Pi x :^{m_0} A_1.A_2$ where $\Omega, x :^{\ell \sqcup m_0} A_1 \vdash a_2 :^{\ell} A_2$ and $\mathcal{C} \sqcap \Omega \vdash \Pi x{:}^{m_0} A_1.A_2 :^{\mathcal{C}} s$. Further, $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash \Pi x{:}^{m_0} A_1.A_2 \equiv_{\mathcal{C}} \Pi x{:}^{\ell_0} B_1.B_2$.
  By consistency of definitional equality, $\mathsf{Ct}\ (\Pi x{:}^{m_0} A_1.A_2)\ (\Pi x{:}^{\ell_0} B_1.B_2)$.
  Therefore, $m_0 = \ell_0$.
  So, we have:

  – $v = \lambda x{:}^{\ell_0} A_1.a_2$.

  – $\Omega, x{:}^{\ell \sqcup \ell_0} A_1 \vdash a_2 :^{\ell} A_2$.

  – $\mathcal{C} \sqcap \Omega \vdash \Pi x{:}^{\ell_0} A_1.A_2 :^{\mathcal{C}} s$.

  – $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash \Pi x{:}^{\ell_0} A_1.A_2 \equiv_{\mathcal{C}} \Pi x{:}^{\ell_0} A_1.A_2$ (by Lemma B.28 and rule Eq-Refl).

- Rule Ddc-App. Have: $\Omega \vdash b\ a^{m_0} :^{\ell} B\{a/x\}$ where $\Omega \vdash b :^{\ell} \Pi x{:}^{m_0} A.B$ and $\Omega \Vdash a :^{\ell \sqcup m_0} A$.
  The lemma statement does not apply because $b\ a^{m_0}$ is not a value.

- Rule Ddc-Conv. Have: $\Omega \vdash v :^{\ell} B$ where $\Omega \vdash v :^{\ell} A$ and $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash A \equiv_{\mathcal{C}} B$ and $\mathcal{C} \sqcap \Omega \vdash B :^{\mathcal{C}} s$. Further, v is a value and $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash B \equiv_{\mathcal{C}} \Pi x :^{\ell_0} B_1 . B_2$.
  By rule Eq-Trans, $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash A \equiv_{\mathcal{C}} \Pi x :^{\ell_0} B_1 . B_2$.
  Using IH, we have:
  - $v = \lambda x :^{\ell_0} A_1 . a_2$.
  - $\Omega, x :^{\ell \sqcup \ell_0} A_1 \vdash a_2 :^{\ell} A_2$.
  - $\mathcal{C} \sqcap \Omega \vdash \Pi x :^{\ell_0} A_1 . A_2 :^{\mathcal{C}} s$.
  - $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash A \equiv_{\mathcal{C}} \Pi x :^{\ell_0} A_1 . A_2$.
    By rule Eq-Trans, $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash B \equiv_{\mathcal{C}} \Pi x :^{\ell_0} A_1 . A_2$.

□

Theorem B.55 (Theorem 3.37) If $\Omega \vdash a :^{\ell} A$ and $\vdash a \rightsquigarrow a'$, then $\Omega \vdash a' :^{\ell} A$.

Proof. By induction on $\Omega \vdash a :^{\ell} A$ and subsequent inversion on $\vdash a \rightsquigarrow a'$. We present some of the interesting cases below.

- Rule Ddc-App. Have: $\Omega \vdash b\ a^{\ell_0} :^{\ell} B\{a/x\}$ where $\Omega \vdash b :^{\ell} \Pi x :^{\ell_0} A . B$ and $\Omega \Vdash a :^{\ell \sqcup \ell_0} A$. By inversion on $\vdash b\ a^{\ell_0} \rightsquigarrow c$:
  - $\vdash b\ a^{\ell_0} \rightsquigarrow b'\ a^{\ell_0}$ when $\vdash b \rightsquigarrow b'$.
    Need to show: $\Omega \vdash b'\ a^{\ell_0} :^{\ell} B\{a/x\}$.
    By IH, $\Omega \vdash b' :^{\ell} \Pi x :^{\ell_0} A . B$.
    This case, then, follows by rule Ddc-App.
  - $b = \lambda x :^{\ell_0} A_0 . b'$ and $\vdash (\lambda x :^{\ell_0} A_0 . b')\ a^{\ell_0} \rightsquigarrow b'\{a/x\}$.
    Need to show: $\Omega \vdash b'\{a/x\} :^{\ell} B\{a/x\}$.
    Using typing inversion lemma B.54 on $\Omega \vdash \lambda x :^{\ell_0} A_0 . b' :^{\ell} \Pi x :^{\ell_0} A . B$, we have:
    * $\Omega, x :^{\ell \sqcup \ell_0} A' \vdash b' :^{\ell} B'$
    * $\mathcal{C} \sqcap \Omega \vdash \Pi x :^{\ell_0} A' . B' :^{\mathcal{C}} s$. Therefore, there exists s_1 and s_2 such that $\mathcal{C} \sqcap \Omega \vdash A' :^{\mathcal{C}} s_1$ and $\mathcal{C} \sqcap \Omega, x :^{\mathcal{C}} A' \vdash B' :^{\mathcal{C}} s_2$.
    * $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash \Pi x :^{\ell_0} A . B \equiv_{\mathcal{C}} \Pi x :^{\ell_0} A' . B'$. Therefore, by equality inversion lemma B.53, $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash A \equiv_{\mathcal{C}} A'$ and $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash B\{a/x\} \equiv_{\mathcal{C}} B'\{a/x\}$.

    So, by rule Ddc-Conv, $\Omega \Vdash a :^{\ell \sqcup \ell_0} A'$.
    Then, by substitution lemma B.50, $\Omega \vdash b'\{a/x\} :^{\ell} B'\{a/x\}$.
    Using rule Ddc-Conv again, we have, $\Omega \vdash b'\{a/x\} :^{\ell} B\{a/x\}$.
- Rule Ddc-LetPair. Have: $\Omega \vdash \textbf{let}\ (x^{\ell_0}, y)\ =\ a\ \textbf{in}\ b :^{\ell} B\{a/z\}$ where $\Omega \vdash a :^{\ell} \Sigma x :^{\ell_0} A_1 . A_2$ and $\Omega, x :^{\ell \sqcup \ell_0} A_1, y :^{\ell} A_2 \vdash b :^{\ell} B\{(x^{\ell_0}, y)/z\}$ and $\mathcal{C} \sqcap \Omega, z :^{\mathcal{C}} \Sigma x :^{\ell_0} A_1 . A_2 \vdash B :^{\mathcal{C}} s$. By inversion on $\vdash \textbf{let}\ (x^{\ell_0}, y)\ =\ a\ \textbf{in}\ b \rightsquigarrow c$:

– $\vdash \textbf{let}\ (x^{\ell_0}, y) = a\ \textbf{in}\ b \leadsto \textbf{let}\ (x^{\ell_0}, y) = a'\ \textbf{in}\ b$ when $\vdash a \leadsto a'$.
  Need to show: $\Omega \vdash \textbf{let}\ (x^{\ell_0}, y) = a'\ \textbf{in}\ b :^{\ell} B\{a/z\}$.
  By IH, $\Omega \vdash a' :^{\ell} \Sigma x{:}^{\ell_0} A_1.A_2$.
  Then, by rule DDC-LETPAIR, $\Omega \vdash \textbf{let}\ (x^{\ell_0}, y) = a'\ \textbf{in}\ b :^{\ell} B\{a'/z\}$.
  Now, we show that $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash B\{a'/z\} \equiv_{\mathcal{C}} B\{a/z\}$.
  Since $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash a : \mathcal{C}$ and $\vdash a \leadsto a'$, by rule EQ-BETA, $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash a \equiv_{\mathcal{C}} a'$.
  Since $\mathcal{C} \sqcap \Omega, z{:}^{\mathcal{C}} \Sigma x{:}^{\ell_0} A_1.A_2 \vdash B :^{\mathcal{C}} s$, we have, $\lceil \mathcal{C} \sqcap \Omega, z{:}^{\mathcal{C}} \Sigma x{:}^{\ell_0} A_1.A_2 \rceil \vdash B \equiv_{\mathcal{C}} B$.
  So by substitution lemma B.49, $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash B\{a/z\} \equiv_{\mathcal{C}} B\{a'/z\}$.
  This case, then, follows by rule DDC-CONV.

– $a = ({a_1}^{\ell_0}, a_2)$ and $\vdash \textbf{let}\ (x^{\ell_0}, y) = ({a_1}^{\ell_0}, a_2)\ \textbf{in}\ b \leadsto b\{a_1/x\}\{a_2/y\}$.
  Need to show: $\Omega \vdash b\{a_1/x\}\{a_2/y\} :^{\ell} B\{({a_1}^{\ell_0}, a_2)/z\}$.
  First, note that since $\Omega \vdash ({a_1}^{\ell_0}, a_2) :^{\ell} \Sigma x{:}^{\ell_0} A_1.A_2$, by regularity lemma B.51, $\mathcal{C} \sqcap \Omega \vdash \Sigma x{:}^{\ell_0} A_1.A_2 :^{\mathcal{C}} s$, for some s. Then, by inversion, there exists s_1 and s_2 such that $\mathcal{C} \sqcap \Omega \vdash A_1 :^{\mathcal{C}} s_1$ and $\mathcal{C} \sqcap \Omega, x{:}^{\mathcal{C}} A_1 \vdash A_2 :^{\mathcal{C}} s_2$.
  Next, using typing inversion lemma B.54 on $\Omega \vdash ({a_1}^{\ell_0}, a_2) :^{\ell} \Sigma x{:}^{\ell_0} A_1.A_2$, we have:
  * $\Omega \Vdash a_1 :^{\ell \sqcup \ell_0} A_1'$ and $\Omega \vdash a_2 :^{\ell} A_2'\{a_1/x\}$
  * $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash \Sigma x{:}^{\ell_0} A_1.A_2 \equiv_{\mathcal{C}} \Sigma x{:}^{\ell_0} A_1'.A_2'$. Therefore, by equality inversion lemma B.53, $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash A_1 \equiv_{\mathcal{C}} A_1'$ and $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash A_2\{a_1/x\} \equiv_{\mathcal{C}} A_2'\{a_1/x\}$.

  So, by rule DDC-CONV, $\Omega \Vdash a_1 :^{\ell \sqcup \ell_0} A_1$ and $\Omega \vdash a_2 :^{\ell} A_2\{a_1/x\}$.
  Then, by applying substitution lemma B.50 twice, we have, $\Omega \vdash b\{a_1/x\}\{a_2/y\} :^{\ell} B\{({a_1}^{\ell_0}, a_2)/z\}$.

□

Theorem B.56 (Theorem 3.38) If $\varnothing \vdash a :^{\ell} A$, then either a is a value or there exists some a' such that $\vdash a \leadsto a'$.

Proof. By induction on $\varnothing \vdash a :^{\ell} A$. We present some of the interesting cases below.

- Rule DDC-VAR. Does not apply because the context here is empty.

- Rule DDC-APP. Have: $\varnothing \vdash b\ a^{\ell_0} :^{\ell} B\{a/x\}$ where $\varnothing \vdash b :^{\ell} \Pi x{:}^{\ell_0} A.B$ and $\varnothing \Vdash a :^{\ell \sqcup \ell_0} A$.
  Need to show: $\exists c$ such that $\vdash b\ a^{\ell_0} \leadsto c$.
  By IH, b is a value or $\exists b'$ such that $\vdash b \leadsto b'$.
  If $\vdash b \leadsto b'$, then this case follows by setting $c \coloneqq b'\ a^{\ell_0}$.
  Otherwise, using inversion lemma B.54 on $\varnothing \vdash b :^{\ell} \Pi x{:}^{\ell_0} A.B$, we have, $b = \lambda x{:}^{\ell_0} A'.b'$.
  This case, then, follows by setting $c \coloneqq b'\{a/x\}$.

- Rule DDC-LETPAIR. Have: $\varnothing \vdash \textbf{let}\ (x^{\ell_0}, y) = a\ \textbf{in}\ b :^{\ell} B\{a/z\}$ where $\varnothing \vdash a :^{\ell} \Sigma x{:}^{\ell_0} A_1.A_2$ and $x{:}^{\ell \sqcup \ell_0} A_1, y{:}^{\ell} A_2 \vdash b :^{\ell} B\{(x^{\ell_0}, y)/z\}$ and $z{:}^{\mathcal{C}} \Sigma x{:}^{\ell_0} A_1.A_2 \vdash B :^{\mathcal{C}} s$.
  Need to show: $\exists c$ such that $\vdash \textbf{let}\ (x^{\ell_0}, y) = a\ \textbf{in}\ b \leadsto c$.
  By IH, a is a value or $\exists a'$ such that $\vdash a \leadsto a'$.

If $\vdash a \leadsto a'$, then this case follows by setting $c \coloneqq \textbf{let}\ (x^{\ell_0}, y) \ = \ a'\ \textbf{in}\ b$.

Otherwise, using inversion lemma B.54 on $\varnothing \vdash a :^{\ell} \Sigma x{:}^{\ell_0} A_1.A_2$, we have, $a = ({a_1}^{\ell_0}, a_2)$.

This case, then, follows by setting $c \coloneqq b\{a_1/x\}\{a_2/y\}$.

$\square$

Lemma B.57 (Translation preserves reduction) If $\vdash a \leadsto a'$, then $\vdash \widetilde{a} \leadsto \widetilde{a'}$.

Proof. By induction on $\vdash a \leadsto a'$. We present the β-case for application below.

Have: $\vdash (\lambda x :^{\ell_0} A.b)\ a^{\ell_0} \leadsto b\{a/x\}$.

Need to show: $\vdash \widetilde{(\lambda x :^{\ell_0} A.b)\ a^{\ell_0}} \leadsto \widetilde{b\{a/x\}}$.

If $\ell_0 \sqsubseteq \mathcal{C}$, then $\widetilde{(\lambda x :^{\ell_0} A.b)\ a^{\ell_0}} = (\lambda(x : \widetilde{A}).\widetilde{b})\ \widetilde{a}$. By rule ICCStep-EBeta, $\vdash (\lambda(x : \widetilde{A}).\widetilde{b})\ \widetilde{a} \leadsto \widetilde{b}\{\widetilde{a}/x\}$. This case, then, follows by the fact that $\widetilde{b\{a/x\}} = \widetilde{b}\{\widetilde{a}/x\}$.

Otherwise, $\ell_0 = \top$ and $\widetilde{(\lambda x :^{\ell_0} A.b)\ a^{\ell_0}} = (\lambda[x : \widetilde{A}].\widetilde{b})\ [\widetilde{a}]$. By rule ICCStep-IBeta, $\vdash (\lambda[x : \widetilde{A}].\widetilde{b})\ [\widetilde{a}] \leadsto \widetilde{b}\{\widetilde{a}/x\}$. This case, then, follows by the fact that $\widetilde{b\{a/x\}} = \widetilde{b}\{\widetilde{a}/x\}$.

$\square$

Lemma B.58 (Translation on indistinguishability relation) If $\Phi \vdash a_1 \sim_{\mathcal{C}} a_2$, then $\widetilde{a_1}^* = \widetilde{a_2}^*$.

Proof. By induction on $\Phi \vdash a_1 \sim_{\mathcal{C}} a_2$. We present some of the interesting cases below.

- Rule IndD-Var. Have: $\Phi_1, x : \ell_0, \Phi_2 \vdash x \sim_{\mathcal{C}} x$ where $\ell_0 \sqsubseteq \mathcal{C}$.
  Need to show: $\widetilde{x}^* = \widetilde{x}^*$.
  Follows by definition.
- Rule IndD-Lam. Have: $\Phi \vdash \lambda x :^{\ell_0} A_1.b_1 \sim_{\mathcal{C}} \lambda x :^{\ell_0} A_2.b_2$ where $\Phi, x : \mathcal{C} \sqcup \ell_0 \vdash b_1 \sim_{\mathcal{C}} b_2$.
  Need to show: $\widetilde{(\lambda x :^{\ell_0} A_1.b_1)}^* = \widetilde{(\lambda x :^{\ell_0} A_2.b_2)}^*$.
  There are two possibilities:
  - $\ell_0 \sqsubseteq \mathcal{C}$. Then, $\widetilde{(\lambda x :^{\ell_0} A_1.b_1)}^* = \lambda x.\widetilde{b_1}^*$ and $\widetilde{(\lambda x :^{\ell_0} A_2.b_2)}^* = \lambda x.\widetilde{b_2}^*$.
    This case, then, follows by IH.
  - $\ell_0 = \top$. Then, $\widetilde{(\lambda x :^{\ell_0} A_1.b_1)}^* = \widetilde{b_1}^*$ and $\widetilde{(\lambda x :^{\ell_0} A_2.b_2)}^* = \widetilde{b_2}^*$.
    This case, then, follows by IH.
- Rule IndD-App. Have: $\Phi \vdash b_1\ {a_1}^{\ell_0} \sim_{\mathcal{C}} b_2\ {a_2}^{\ell_0}$ where $\Phi \vdash b_1 \sim_{\mathcal{C}} b_2$ and $\Phi \vdash^{\ell_0}_{\mathcal{C}} a_1 \sim a_2$.
  Need to show: $\widetilde{(b_1\ {a_1}^{\ell_0})}^* = \widetilde{(b_2\ {a_2}^{\ell_0})}^*$.
  There are two possibilities.
  - $\ell_0 \sqsubseteq \mathcal{C}$. Then, we have, $\Phi \vdash a_1 \sim_{\mathcal{C}} a_2$.
    Now, $\widetilde{(b_1\ {a_1}^{\ell_0})}^* = \widetilde{b_1}^*\ \widetilde{a_1}^*$ and $\widetilde{(b_2\ {a_2}^{\ell_0})}^* = \widetilde{b_2}^*\ \widetilde{a_2}^*$.
    This case, then, follows by IH.

– $\ell_0 = \top$. Then, $(\widetilde{b_1\ a_1{}^{\ell_0}})^* = \widetilde{b_1}^*$ and $(\widetilde{b_2\ a_2{}^{\ell_0}})^* = \widetilde{b_2}^*$.
  This case, then, follows by IH.

□

Lemma B.59 (Translation on parallel reduction relation) If $\Phi \vdash A_1 \Rightarrow_{\mathcal{C}} A_2$, then $\widetilde{A_1}^* \cong_{\beta\eta} \widetilde{A_2}^*$.

Proof. By induction on $\Phi \vdash A_1 \Rightarrow_{\mathcal{C}} A_2$. We present some of the interesting cases below.

- Rule Par-Refl. Have: $\Phi \vdash a \Rightarrow_{\mathcal{C}} a$ where $\Phi \vdash a : \mathcal{C}$.
  Need to show: $\widetilde{a}^* \cong_{\beta\eta} \widetilde{a}^*$.
  Follows by reflexivity.

- Rule Par-Lam. Have: $\Phi \vdash \lambda x{:}^{\ell_0} A_1.b_1 \Rightarrow_{\mathcal{C}} \lambda x{:}^{\ell_0} A_2.b_2$ where $\Phi, x{:}\,\mathcal{C} \sqcup \ell_0 \vdash b_1 \Rightarrow_{\mathcal{C}} b_2$.
  Need to show: $(\widetilde{\lambda x{:}^{\ell_0} A_1.b_1})^* \cong_{\beta\eta} (\widetilde{\lambda x{:}^{\ell_0} A_2.b_2})^*$.
  There are two possibilities:

  – $\ell_0 \sqsubseteq \mathcal{C}$. Then, $\widetilde{\lambda x{:}^{\ell_0} A_1.b_1} = \lambda(x{:}\widetilde{A_1}).\widetilde{b_1}$ and $\widetilde{\lambda x{:}^{\ell_0} A_2.b_2} = \lambda(x{:}\widetilde{A_2}).\widetilde{b_2}$.
    Now, $(\lambda(x{:}\widetilde{A_1}).\widetilde{b_1})^* = \lambda x.\widetilde{b_1}^*$ and $(\lambda(x{:}\widetilde{A_2}).\widetilde{b_2})^* = \lambda x.\widetilde{b_2}^*$.
    By IH, $\widetilde{b_1}^* \cong_{\beta\eta} \widetilde{b_2}^*$.
    This case, then, follows by congruence.

  – $\ell_0 = \top$. Then, $\widetilde{\lambda x{:}^{\ell_0} A_1.b_1} = \lambda[x{:}\widetilde{A_1}].\widetilde{b_1}$ and $\widetilde{\lambda x{:}^{\ell_0} A_2.b_2} = \lambda[x{:}\widetilde{A_2}].\widetilde{b_2}$.
    Now, $(\lambda[x{:}\widetilde{A_1}].\widetilde{b_1})^* = \widetilde{b_1}^*$ and $(\lambda[x{:}\widetilde{A_2}].\widetilde{b_2})^* = \widetilde{b_2}^*$.
    This case, then, follows by IH.

- Rule Par-App. Have: $\Phi \vdash b_1\ a_1{}^{\ell_0} \Rightarrow_{\mathcal{C}} b_2\ a_2{}^{\ell_0}$ where $\Phi \vdash b_1 \Rightarrow_{\mathcal{C}} b_2$ and $\Phi \vdash^{\ell_0}_{\mathcal{C}} a_1 \Rightarrow a_2$.
  Need to show: $(\widetilde{b_1\ a_1{}^{\ell_0}})^* \cong_{\beta\eta} (\widetilde{b_2\ a_2{}^{\ell_0}})^*$.
  There are two possibilities:

  – $\ell_0 \sqsubseteq \mathcal{C}$. Then, we have, $\Phi \vdash a_1 \Rightarrow_{\mathcal{C}} a_2$.
    By IH, $\widetilde{b_1}^* \cong_{\beta\eta} \widetilde{b_2}^*$ and $\widetilde{a_1}^* \cong_{\beta\eta} \widetilde{a_2}^*$.
    Next, $(\widetilde{b_1\ a_1{}^{\ell_0}})^* = \widetilde{b_1}^*\ \widetilde{a_1}^*$ and $(\widetilde{b_2\ a_2{}^{\ell_0}})^* = \widetilde{b_2}^*\ \widetilde{a_2}^*$.
    This case, then, follows by congruence.

  – $\ell_0 = \top$. Then, $(\widetilde{b_1\ a_1{}^{\ell_0}})^* = \widetilde{b_1}^*$ and $(\widetilde{b_2\ a_2{}^{\ell_0}})^* = \widetilde{b_2}^*$.
    This case, then, follows by IH.

- Rule Par-AppBeta. Have: $\Phi \vdash b\ a^{\ell_0} \Rightarrow_{\mathcal{C}} b'\{a'/x\}$ where $\Phi \vdash b \Rightarrow_{\mathcal{C}} \lambda x{:}^{\ell_0} A'.b'$ and $\Phi \vdash^{\ell_0}_{\mathcal{C}} a \Rightarrow a'$.
  Need to show: $(\widetilde{b\ a^{\ell_0}})^* \cong_{\beta\eta} (\widetilde{b'\{a'/x\}})^*$.
  There are two possibilities:

- $\ell_0 \sqsubseteq \mathcal{C}$. Then, we have, $\Phi \vdash a \Rightarrow_{\mathcal{C}} a'$.
  By IH, $\widetilde{b}^* \cong_{\beta\eta} \widetilde{(\lambda x :^{\ell_0} A'.b')}^*$ and $\widetilde{a}^* \cong_{\beta\eta} \widetilde{a'}^*$.
  Now, $\widetilde{(\lambda x :^{\ell_0} A'.b')}^* = (\lambda(x : \widetilde{A'}).\widetilde{b'})^* = \lambda x.\widetilde{b'}^*$.
  Then, $\widetilde{(b\ a^{\ell_0})}^* = (\widetilde{b}\ \widetilde{a})^* = \widetilde{b}^*\ \widetilde{a}^* \cong_{\beta\eta} (\lambda x.\widetilde{b'}^*)\ \widetilde{a'}^* \cong_{\beta\eta} \widetilde{b'}^* \{\widetilde{a'}^*/x\} = (\widetilde{b'\{a'/x\}})^*$.
- $\ell_0 = \top$. Then, $\widetilde{(b\ a^{\ell_0})}^* = (\widetilde{b}\ [\widetilde{a}])^* = \widetilde{b}^*$.
  Now, since $\Phi \vdash b \Rightarrow_{\mathcal{C}} \lambda x :^{\top} A'.b'$, by Lemma B.34, $\Phi \vdash \lambda x :^{\top} A'.b' : \mathcal{C}$.
  Then, by inversion, $\Phi, x : \top \vdash b' : \mathcal{C}$.
  Next, by rule CIndD-Nleq, $\Phi \vdash^{\top}_{\mathcal{C}} x \sim a'$.
  Therefore, by Lemma B.31, $\Phi \vdash b'\{x/x\} \sim_{\mathcal{C}} b'\{a'/x\}$.
  Then, by Lemma B.58, $\widetilde{b'\{x/x\}}^* = \widetilde{b'\{a'/x\}}^*$.
  So, $\widetilde{(b\ a^{\ell_0})}^* = \widetilde{b}^* = \widetilde{b'\{x/x\}}^* = \widetilde{b'\{a'/x\}}^*$.
  This case, then, follows by reflexivity.

□

Lemma B.60 (Translation on definitional equality relation) If $\Phi \vdash A_1 \equiv_{\mathcal{C}} A_2$, then $\widetilde{A_1}^* \cong_{\beta\eta} \widetilde{A_2}^*$.

Proof. Let $\Phi \vdash A_1 \equiv_{\mathcal{C}} A_2$.
By Theorem B.41, $\Phi \vdash A_1 \Leftrightarrow_{\mathcal{C}} A_2$.
In other words, $\exists B$ such that $\Phi \vdash A_1 \Rightarrow^*_{\mathcal{C}} B$ and $\Phi \vdash A_2 \Rightarrow^*_{\mathcal{C}} B$.
Then, by the auxiliary lemma shown below, $\widetilde{A_1}^* \cong_{\beta\eta} \widetilde{B}^*$ and $\widetilde{A_2}^* \cong_{\beta\eta} \widetilde{B}^*$.
The main lemma, then, follows by symmetry and transitivity of the $\cong_{\beta\eta}$-relation.

Auxiliary Lemma: If $\Phi \vdash A_1 \Rightarrow^*_{\mathcal{C}} A_2$, then $\widetilde{A_1}^* \cong_{\beta\eta} \widetilde{A_2}^*$.
The proof of this auxiliary lemma is by induction on $\Phi \vdash A_1 \Rightarrow^*_{\mathcal{C}} A_2$.

- Rule MultiPar-One. Have: $\Phi \vdash A_1 \Rightarrow^*_{\mathcal{C}} A_2$ where $\Phi \vdash A_1 \Rightarrow_{\mathcal{C}} A_2$.
  Need to show: $\widetilde{A_1}^* \cong_{\beta\eta} \widetilde{A_2}^*$.
  Follows by Lemma B.59.
- Rule MultiPar-Many. $\Phi \vdash A_1 \Rightarrow^*_{\mathcal{C}} A_2$ where $\Phi \vdash A_1 \Rightarrow^*_{\mathcal{C}} A_0$ and $\Phi \vdash A_0 \Rightarrow^*_{\mathcal{C}} A_2$.
  Need to show: $\widetilde{A_1}^* \cong_{\beta\eta} \widetilde{A_2}^*$.
  Follows by IH and transitivity of the $\cong_{\beta\eta}$-relation.

□

Lemma B.61 (Translation on typing relation) If $\Omega \vdash a :^{\ell} A$, then $\widetilde{\Omega} \vdash \widetilde{a} : \widetilde{A}$.

Proof. By induction on $\Omega \vdash a :^{\ell} A$. We present some of the interesting cases below.

- Rule Ddc-Pi. Have: $\widetilde{\Omega \vdash \Pi x}{:}^{\ell_0} A.B :^{\ell} s_3$ where $\Omega \vdash A :^{\ell} s_1$ and $\Omega, x{:}^{\ell} A \vdash B :^{\ell} s_2$ and $(s_1, s_2, s_3) \in \mathcal{R}$.
  Need to show: $\widetilde{\Omega} \vdash \Pi x{:}^{\ell_0} A.B : s_3$.
  By IH, $\widetilde{\Omega} \vdash \widetilde{A} : s_1$ and $\widetilde{\Omega}, x : \widetilde{A} \vdash \widetilde{B} : s_2$.
  This case, then, follows by rule ICCS-EPi if $\ell_0 \sqsubseteq \mathcal{C}$ and by rule ICCS-IPi otherwise.

- Rule Ddc-Lam. Have: $\widetilde{\Omega \vdash} \lambda x{:}^{\ell_0} \widetilde{A.b :^{\ell}} \Pi x{:}^{\ell_0} A.B$ where $\Omega, x{:}^{\ell \sqcup \ell_0} A \vdash b :^{\ell} B$ and $\mathcal{C} \sqcap \Omega \vdash \Pi x{:}^{\ell_0} A.B :^{\mathcal{C}} s$.
  Need to show: $\widetilde{\Omega} \vdash \lambda x{:}^{\ell_0} A.b : \Pi x{:}^{\ell_0} A.B$.
  There are two possibilities:

  - $\ell_0 \sqsubseteq \mathcal{C}$. By IH, $\widetilde{\Omega}, x : \widetilde{A} \vdash \widetilde{b} : \widetilde{B}$ and $\widetilde{\Omega} \vdash \Pi(x{:}\widetilde{A}).\widetilde{B} : s$.
    This case, then, follows by rule ICCS-ELam.
  - $\ell_0 = \top$. First, by IH, $\widetilde{\Omega}, x : \widetilde{A} \vdash \widetilde{b} : \widetilde{B}$ and $\widetilde{\Omega} \vdash \Pi[x{:}\widetilde{A}].\widetilde{B} : s$.
    Next, we show $x \notin \mathsf{fv}\,\widetilde{b}^*$.
    We have, $\Omega, x{:}^{\top} A \vdash b :^{\ell} B$.
    By subsumption lemma B.46, $\Omega, x{:}^{\top} A \vdash b :^{\mathcal{C}} B$.
    Then, by Lemma B.28, $\lceil \Omega \rceil, x{:}\,\top \vdash b : \mathcal{C}$.
    Next, by rule CIndD-Nleq, $\lceil \Omega \rceil \vdash^{\top}_{\mathcal{C}} x \sim \mathbf{unit}$.
    So, by Lemma B.31, $\lceil \Omega \rceil \vdash b\{x/x\} \sim_{\mathcal{C}} b\{\mathbf{unit}/x\}$.
    Then, by Lemma B.58, $(\widetilde{b\{x/x\}})^* = (\widetilde{b\{\mathbf{unit}/x\}})^*$.
    Since $x \notin \mathsf{fv}\, b\{\mathbf{unit}/x\}$, so $x \notin \mathsf{fv}\,\widetilde{b}^*$.
    This case, then, follows by rule ICCS-ILam.

- Rule Ddc-App. Have: $\Omega \vdash \widetilde{b\, a^{\ell_0}} :^{\ell} B\{a/x\}$ where $\Omega \vdash b :^{\ell} \Pi x{:}^{\ell_0} A.B$ and $\Omega \Vdash a :^{\ell \sqcup \ell_0} A$.
  Need to show: $\widetilde{\Omega} \vdash \widetilde{b\, a^{\ell_0}} : \widetilde{B}\{\widetilde{a}/x\}$.
  There are two possibilities:

  - $\ell_0 \sqsubseteq \mathcal{C}$. Then, we have, $\Omega \vdash a :^{\ell \sqcup \ell_0} A$.
    By IH, $\widetilde{\Omega} \vdash \widetilde{b} : \Pi(x{:}\widetilde{A}).\widetilde{B}$ and $\widetilde{\Omega} \vdash \widetilde{a} : \widetilde{A}$.
    This case, then, follows by rule ICCS-EApp.
  - $\ell_0 = \top$. Then, we have, $\mathcal{C} \sqcap \Omega \vdash a :^{\mathcal{C}} A$.
    By IH, $\widetilde{\Omega} \vdash \widetilde{b} : \Pi[x{:}\widetilde{A}].\widetilde{B}$ and $\widetilde{\Omega} \vdash \widetilde{a} : \widetilde{A}$.
    This case, then, follows by rule ICCS-IApp.

- Rule Ddc-Conv. Have: $\Omega \vdash a :^{\ell} B$ where $\Omega \vdash a :^{\ell} A$ and $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash A \equiv_{\mathcal{C}} B$ and $\mathcal{C} \sqcap \Omega \vdash B :^{\mathcal{C}} s$.
  Need to show: $\widetilde{\Omega} \vdash \widetilde{a} : \widetilde{B}$.
  By IH, $\widetilde{\Omega} \vdash \widetilde{a} : \widetilde{A}$ and $\widetilde{\Omega} \vdash \widetilde{B} : s$.
  Now, since $\lceil \mathcal{C} \sqcap \Omega \rceil \vdash A \equiv_{\mathcal{C}} B$, by Lemma B.60, $\widetilde{A}^* \cong_{\beta\eta} \widetilde{B}^*$.
  This case, then, follows by rule ICCS-Conv.

□

Lemma B.62 (Lemma 3.39) The translation function, $\widetilde{\ }$, preserves meaning and typing:

- If $\vdash a \rightsquigarrow a'$, then $\vdash \widetilde{a} \rightsquigarrow \widetilde{a'}$.
- If $\Phi \vdash A \equiv_{\mathcal{C}} B$, then $\widetilde{A}^* \cong_{\beta\eta} \widetilde{B}^*$.
- If $\Omega \vdash a :^{\ell} A$, then $\widetilde{\Omega} \vdash \widetilde{a} : \widetilde{A}$.

Proof. Follows by lemmas B.57, B.60 and B.61. □

Appendix C

Linearity Analysis in Pure Type Systems

Note: Some of the proofs presented in this appendix have been mechanized in Coq: see Choudhury et al. [2020b].

C.1 Proof of a Proposition Stated in Section 4.2

Proposition C.1. Given a preordered semiring $\mathcal{Q}$ and a standard context Δ, contexts Γ, graded over $\mathcal{Q}$ and satisfying $\lfloor \Gamma \rfloor = \Delta$, form a preordered left $\mathcal{Q}$-semimodule.

Proof. The proof of this proposition is straightforward once we know the definitions of the algebraic structures involved. So first, we look at the definition of a left semimodule.

Definition (Left $\mathcal{Q}'$-semimodule). Given a semiring $\mathcal{Q}' = (Q, +, \cdot, 0, 1)$, a left $\mathcal{Q}'$-semimodule is a 4-tuple $(M, \oplus, \mathbb{0}, \odot)$ where $(M, \oplus, \mathbb{0})$ is a commutative monoid and $\odot : Q \times M \to M$ is a left multiplication function such that:

- for $q_1, q_2 \in Q$ and $m \in M$, we have, $(q_1 + q_2) \odot m = q_1 \odot m \oplus q_2 \odot m$,
- for $q \in Q$ and $m_1, m_2 \in M$, we have, $q \odot (m_1 \oplus m_2) = q \odot m_1 \oplus q \odot m_2$,
- for $q_1, q_2 \in Q$ and $m \in M$, we have, $(q_1 \cdot q_2) \odot m = q_1 \odot (q_2 \odot m)$,
- for $m \in M$, we have, $1 \odot m = m$, and
- for $q \in Q$ and $m \in M$, we have, $0 \odot m = q \odot \mathbb{0} = \mathbb{0}$.

Now, given any standard context Δ, let $\mathbf{\Gamma}$ denote the set of graded contexts, Γ, such that $\lfloor \Gamma \rfloor = \Delta$. Next, let $\Gamma_0 \in \mathbf{\Gamma}$ be the graded context for which $\lfloor \Gamma_0 \rfloor = \mathbf{0}$. Then, $(\mathbf{\Gamma}, +, \Gamma_0, \cdot)$, with $+$ and $\cdot$ as defined in the chapter, forms a left $\mathcal{Q}'$-semimodule.

Next, we define preordered left semimodules.

Definition (Preordered left $\mathcal{Q}$-semimodule). Given a preordered semiring $\mathcal{Q} = (\mathcal{Q}', <:)$, a left $\mathcal{Q}'$-semimodule, $(M, \oplus, \mathbb{0}, \odot)$, is said to be preordered iff there exists a preorder $\leq_M$ on M such that:

- for $m_1, m_2, m \in M$, if $m_1 \leq_M m_2$, then $m \oplus m_1 \leq_M m \oplus m_2$
- for $q \in Q$ and $m_1, m_2 \in M$, if $m_1 \leq_M m_2$, then $q \odot m_1 \leq_M q \odot m_2$
- for $q_1, q_2 \in Q$ and $m \in M$, if $q_1 <: q_2$, then $q_1 \odot m \leq_M q_2 \odot m$.

The ordering $<:$ on $\mathbf{\Gamma}$, as defined in the chapter, is a preorder that satisfies the above properties. Therefore, $(\mathbf{\Gamma}, +, \Gamma_0, \cdot)$, with this ordering, forms a preordered left $\mathcal{Q}$-semimodule. ☐

C.2 Proofs of Lemmas/Theorems Stated in Section 4.2

Lemma C.2 (Lemma 4.1) If $\Gamma_1, \Gamma_2 \vdash a : A$, then $\Gamma_1, z :^0 C, \Gamma_2 \vdash a : A$.

Proof. By induction on $\Gamma_1, \Gamma_2 \vdash a : A$. ☐

Lemma C.3 (Lemma 4.2) If $\Gamma_1, z :^r C, \Gamma_2 \vdash a : A$ and $\Gamma \vdash c : C$ where $\lfloor \Gamma_1 \rfloor = \lfloor \Gamma \rfloor$, then $\Gamma_1 + r \cdot \Gamma, \Gamma_2 \vdash a\{c/z\} : A$.

Proof. By induction on $\Gamma_1, z :^q C, \Gamma_2 \vdash a : A$.

- Rule GLC-Var. There are two cases to consider.
  - Have: $0 \cdot \Gamma_1, z :^0 C, 0 \cdot \Gamma_2, x :^1 A \vdash x : A$ and $\Gamma \vdash c : C$, where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
    Need to show: $0 \cdot \Gamma_1 + 0 \cdot \Gamma, 0 \cdot \Gamma_2, x :^1 A \vdash x : A$.
    Follows by rule GLC-Var.
  - Have: $0 \cdot \Gamma_1, x :^1 A \vdash x : A$ and $\Gamma \vdash a : A$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
    Need to show: $0 \cdot \Gamma_1 + \Gamma \vdash a : A$.
    Follows from hypothesis.
- Rule GLC-Weak. Again, there are two cases to consider.
  - Have: $\Gamma_1, z :^r C, \Gamma_2, y :^0 B \vdash a : A$ where $\Gamma_1, z :^r C, \Gamma_2 \vdash a : A$. Also, $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
    Need to show: $\Gamma_1 + r \cdot \Gamma, \Gamma_2, y :^0 B \vdash a\{c/z\} : A$.
    Follows by IH and rule GLC-Weak.

– Have: $\Gamma_1, y{:}^0 B \vdash a : A$ where $\Gamma_1 \vdash a : A$. Also, $\Gamma \vdash b : B$ where $\lfloor\Gamma\rfloor = \lfloor\Gamma_1\rfloor$.
  Need to show: $\Gamma_1 + 0 \cdot \Gamma \vdash a\{b/y\} : A$.
  Follows from hypothesis.

- Rule GLC-LetUnit. Have: $\Gamma_{11} + \Gamma_{12}, z :^{r_1+r_2} C, \Gamma_{21} + \Gamma_{22} \vdash \textbf{let unit} = a \textbf{ in } b : B$ where $\Gamma_{11}, z :^{r_1} C, \Gamma_{21} \vdash a : \textbf{Unit}$ and $\Gamma_{12}, z{:}^{r_2} C, \Gamma_{22} \vdash b : B$. Also, $\Gamma \vdash c : C$ where $\lfloor\Gamma\rfloor = \lfloor\Gamma_{11}\rfloor$.
  Need to show: $\Gamma_{11} + \Gamma_{12} + (r_1 + r_2) \cdot \Gamma, \Gamma_{21}, \Gamma_{22} \vdash \textbf{let unit} = a\{c/z\} \textbf{ in } b\{c/z\} : B$.
  By IH, $\Gamma_{11} + r_1 \cdot \Gamma, \Gamma_{21} \vdash a\{c/z\} : \textbf{Unit}$ and $\Gamma_{12} + r_2 \cdot \Gamma, \Gamma_{22} \vdash b\{c/z\} : B$.
  This case, then, follows by rule GLC-LetUnit.

- Rule GLC-Lam. Have: $\Gamma_1, z :^r C, \Gamma_2 \vdash \lambda^q x : A.b :^q A \to B$ where $\Gamma_1, z :^r C, \Gamma_2, x :^q A \vdash b : B$. Also, $\Gamma \vdash c : C$ where $\lfloor\Gamma\rfloor = \lfloor\Gamma_1\rfloor$.
  Need to show: $\Gamma_1 + r \cdot \Gamma, \Gamma_2 \vdash \lambda^q x{:}A.b\{c/z\} :^q A \to B$.
  Follows by IH and rule GLC-Lam.

- Rule GLC-App. Have: $\Gamma_{11} + q \cdot \Gamma_{12}, z{:}^{r_1+q\cdot r_2} C, \Gamma_{21} + q \cdot \Gamma_{22} \vdash b\, a^q : B$ where $\Gamma_{11}, z{:}^{r_1} C, \Gamma_{21} \vdash b :^q A \to B$ and $\Gamma_{12}, z{:}^{r_2} C, \Gamma_{22} \vdash a : A$. Also, $\Gamma \vdash c : C$ where $\lfloor\Gamma\rfloor = \lfloor\Gamma_{11}\rfloor$.
  Need to show: $\Gamma_{11} + q \cdot \Gamma_{12} + (r_1 + q \cdot r_2) \cdot \Gamma, \Gamma_{21} + q \cdot \Gamma_{22} \vdash b\{c/z\}\, a^q\{c/z\} : B$.
  By IH, $\Gamma_{11} + r_1 \cdot \Gamma, \Gamma_{21} \vdash b\{c/z\} :^q A \to B$ and $\Gamma_{12} + r_2 \cdot \Gamma, \Gamma_{22} \vdash a\{c/z\} : A$.
  This case, then, follows by rule GLC-App.

- Rule GLC-Box. Have: $q \cdot \Gamma_1, z{:}^{q\cdot r} C, q \cdot \Gamma_2 \vdash \textbf{box}_q\, a : \Box^q A$ where $\Gamma_1, z{:}^r C, \Gamma_2 \vdash a : A$. Also, $\Gamma \vdash c : C$ where $\lfloor\Gamma\rfloor = \lfloor\Gamma_1\rfloor$.
  Need to show: $q \cdot \Gamma_1 + q \cdot r \cdot \Gamma, q \cdot \Gamma_2 \vdash \textbf{box}_q\, a\{c/z\} : \Box^q A$.
  Follows by IH and rule GLC-Box.

- Rule GLC-LetBox. Have: $\Gamma_{11} + \Gamma_{12}, z :^{r_1+r_2} C, \Gamma_{21} + \Gamma_{22} \vdash \textbf{let box}_q\, x = a \textbf{ in } b : B$ where $\Gamma_{11}, z :^{r_1} C, \Gamma_{21} \vdash a : \Box^q A$ and $\Gamma_{12}, z{:}^{r_2} C, \Gamma_{22}, x{:}^q A \vdash b : B$. Also, $\Gamma \vdash c : C$ where $\lfloor\Gamma\rfloor = \lfloor\Gamma_{11}\rfloor$.
  Need to show: $\Gamma_{11} + \Gamma_{12} + (r_1 + r_2) \cdot \Gamma, \Gamma_{21} + \Gamma_{22} \vdash \textbf{let box}_q\, x = a\{c/z\} \textbf{ in } b\{c/z\} : B$.
  By IH, $\Gamma_{11} + r_1 \cdot \Gamma, \Gamma_{21} \vdash a\{c/z\} : \Box^q A$ and $\Gamma_{12} + r_2 \cdot \Gamma, \Gamma_2, x{:}^q A \vdash b\{c/z\} : B$.
  This case, then, follows by rule GLC-LetBox.

- Rule GLC-Pair. Have: $q \cdot \Gamma_{11} + \Gamma_{12}, z :^{q\cdot r_1+r_2} C, q \cdot \Gamma_{21} + \Gamma_{22} \vdash (a_1{}^q, a_2) : {}^q A_1 \times A_2$ where $\Gamma_{11}, z :^{r_1} C, \Gamma_{21} \vdash a_1 : A_1$ and $\Gamma_{12}, z{:}^{r_2} C, \Gamma_{22} \vdash a_2 : A_2$. Also, $\Gamma \vdash c : C$ where $\lfloor\Gamma\rfloor = \lfloor\Gamma_{11}\rfloor$.
  Need to show: $q \cdot \Gamma_{11} + \Gamma_{12} + (q \cdot r_1 + r_2) \cdot \Gamma, q \cdot \Gamma_{21} + \Gamma_{22} \vdash (a_1\{c/z\}^q, a_2\{c/z\}) : {}^q A_1 \times A_2$.
  By IH, $\Gamma_{11} + r_1 \cdot \Gamma, \Gamma_{21} \vdash a_1\{c/z\} : A_1$ and $\Gamma_{12} + r_2 \cdot \Gamma, \Gamma_{22} \vdash a_2\{c/z\} : A_2$.
  This case, then, follows by rule GLC-WPair.

- Rule GLC-LetPair. Have: $\Gamma_{11} + \Gamma_{12}, z :^{r_1+r_2} C, \Gamma_{21} + \Gamma_{22} \vdash \textbf{let}\, (x_1{}^q, x_2) = a \textbf{ in } b : B$ where $\Gamma_{11}, z :^{r_1} C, \Gamma_{21} \vdash a : {}^q A_1 \times A_2$ and $\Gamma_{12}, z{:}^{r_2} C, \Gamma_{22}, x_1{:}^q A_1, x_2{:}^1 A_2 \vdash b : B$. Also, $\Gamma \vdash c : C$ where $\lfloor\Gamma\rfloor = \lfloor\Gamma_{11}\rfloor$.
  Need to show: $\Gamma_{11} + \Gamma_{12} + (r_1 + r_2) \cdot \Gamma, \Gamma_{21} + \Gamma_{22} \vdash \textbf{let}\, (x_1{}^q, x_2) = a\{c/z\} \textbf{ in } b\{c/z\} : B$.
  By IH, $\Gamma_{11} + r_1 \cdot \Gamma, \Gamma_{21} \vdash a\{c/z\} : {}^q A_1 \times A_2$ and $\Gamma_{12} + r_2 \cdot \Gamma, \Gamma_{22}, x_1{:}^q A_1, x_2{:}^1 A_2 \vdash b\{c/z\} : B$.
  This case, then, follows by rule GLC-LetPair.

- Rule GLC-Inj1. Have: $\Gamma_1, z :^r C, \Gamma_2 \vdash \mathbf{inj}_1\, a_1 : A_1 + A_2$ where $\Gamma_1, z :^r C, \Gamma_2 \vdash a_1 : A_1$. Also, $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
  Need to show: $\Gamma_1 + r \cdot \Gamma, \Gamma_2 \vdash \mathbf{inj}_1\, a_1\{c/z\} : A_1 + A_2$.
  Follows by IH and rule GLC-Inj1.

- Rule GLC-Inj2. Similar to rule GLC-Inj1.

- Rule GLC-Case. Have: $q \cdot \Gamma_{11} + \Gamma_{12}, z :^{q \cdot r_1 + r_2} C, q \cdot \Gamma_{21} + \Gamma_{22} \vdash \mathbf{case}_q\, a\, \mathbf{of}\, x_1.b_1; x_2.b_2 : B$ where $\Gamma_{11}, z :^{r_1} C, \Gamma_{21} \vdash a : A_1 + A_2$ and $\Gamma_{12}, z :^{r_2} C, \Gamma_{22}, x_1 :^q A_1 \vdash b_1 : B$ and $\Gamma_{12}, z :^{r_2} C, \Gamma_{22}, x_2 :^q A_2 \vdash b_2 : B$ and $q <: 1$. Also, $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_{11} \rfloor$.
  Need to show: $\Gamma_{11} + \Gamma_{12} + (r_1 + r_2) \cdot \Gamma, \Gamma_{21} + \Gamma_{22} \vdash \mathbf{case}_q\, a\{c/z\}\, \mathbf{of}\, x_1.b_1\{c/z\}; x_2.b_2\{c/z\} : B$.
  By IH, $\Gamma_{11} + r_1 \cdot \Gamma, \Gamma_{21} \vdash a\{c/z\} : A_1 + A_2$ and $\Gamma_{12} + r_2 \cdot \Gamma, \Gamma_{22}, x_1 :^q A_1 \vdash b_1\{c/z\} : B$ and $\Gamma_{12} + r_2 \cdot \Gamma, \Gamma_{22}, x_2 :^q A_2 \vdash b_2\{c/z\} : B$.
  This case, then, follows by rule GLC-Case.

- Rule GLC-Sub. Have: $\Gamma_1, z :^r C, \Gamma_2 \vdash a : A$ where $\Gamma_1', z :^{r'} C, \Gamma_2' \vdash a : A$ where $\Gamma_1 <: \Gamma_1'$ and $r <: r'$ and $\Gamma_2 <: \Gamma_2'$. Also, $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
  Need to show: $\Gamma_1 + r \cdot \Gamma, \Gamma_2 \vdash a\{c/z\} : A$.
  By IH, $\Gamma_1' + r' \cdot \Gamma, \Gamma_2' \vdash a\{c/z\} : A$.
  Since $r <: r'$, so $r \cdot \Gamma <: r' \cdot \Gamma$.
  This case, then, follows by rule GLC-Sub.

□

Theorem C.4 (Theorem 4.3) If $\Gamma \vdash a : A$ and $\vdash a \rightsquigarrow a'$, then $\Gamma \vdash a' : A$.

Proof. By induction on $\Gamma \vdash a : A$ and inversion on $\vdash a \rightsquigarrow a'$.

- Rule GLC-App. Have: $\Gamma_1 + q \cdot \Gamma_2 \vdash b\ a^q : B$ where $\Gamma_1 \vdash b :^q A \to B$ and $\Gamma_2 \vdash a : A$.
  Let $\vdash b\ a^q \rightsquigarrow c$. By inversion:

  - $\vdash b\ a^q \rightsquigarrow b'\ a^q$ when $\vdash b \rightsquigarrow b'$.
    Need to show: $\Gamma_1 + q \cdot \Gamma_2 \vdash b'\ a^q : B$.
    Follows by IH and rule GLC-App.
  - $b = \lambda^q x{:}A'.b'$ and $\vdash b\ a^q \rightsquigarrow b'\{a/x\}$.
    Need to show: $\Gamma_1 + q \cdot \Gamma_2 \vdash b'\{a/x\} : B$.
    By inversion on $\Gamma_1 \vdash \lambda^q x{:}A'.b' :^q A \to B$, we get $A' = A$ and $\Gamma_1, x :^q A \vdash b' : B$.
    This case, then, follows by the substitution lemma.

- Rule GLC-LetPair. Have: $\Gamma_1 + \Gamma_2 \vdash \mathbf{let}\, (x_1{}^q, x_2) = a\ \mathbf{in}\ b : B$ where $\Gamma_1 \vdash a : {}^qA_1 \times A_2$ and $\Gamma_2, x_1 :^q A_1, x_2 :^1 A_2 \vdash b : B$.
  Let $\vdash \mathbf{let}\, (x_1{}^q, x_2) = a\ \mathbf{in}\ b \rightsquigarrow c$. By inversion:

- $\vdash \textbf{let}\,({x_1}^q, x_2) = a \textbf{ in } b \leadsto \textbf{let}\,({x_1}^q, x_2) = a' \textbf{ in } b$ when $\vdash a \leadsto a'$.
  Need to show: $\Gamma_1 + \Gamma_2 \vdash \textbf{let}\,({x_1}^q, x_2) = a' \textbf{ in } b : B$.
  Follows by IH and rule GLC-LetPair.
- $\vdash \textbf{let}\,({x_1}^q, x_2) = ({a_1}^q, a_2) \textbf{ in } b \leadsto b\{a_1/x_1\}\{a_2/x_2\}$.
  Need to show: $\Gamma_1 + \Gamma_2 \vdash b\{a_1/x_1\}\{a_2/x_2\} : B$.
  By inversion on $\Gamma_1 \vdash ({a_1}^q, a_2) : {}^q A_1 \times A_2$, we have:
  $\exists \Gamma_{11}, \Gamma_{12}$ such that $\Gamma_{11} \vdash a_1 : A_1$ and $\Gamma_{12} \vdash a_2 : A_2$ and $\Gamma_1 <: q \cdot \Gamma_{11} + \Gamma_{12}$.
  Applying the substitution lemma, $\Gamma_2 + q \cdot \Gamma_{11}, y :^1 A_2 \vdash b\{a_1/x_1\} : B$.
  Reapplying the substitution lemma, $\Gamma_2 + q \cdot \Gamma_{11} + \Gamma_{12} \vdash b\{a_1/x_1\}\{a_2/x_2\} : B$.
  This case, then, follows by rule GLC-Sub.

- Rule GLC-LetUnit. Have: $\Gamma_1 + \Gamma_2 \vdash \textbf{let unit} = a \textbf{ in } b : B$ where $\Gamma_1 \vdash a : \textbf{Unit}$ and $\Gamma_2 \vdash b : B$.
  Let $\vdash \textbf{let unit} = a \textbf{ in } b \leadsto c$. By inversion:
  - $\vdash \textbf{let unit} = a \textbf{ in } b \leadsto \textbf{let unit} = a' \textbf{ in } b$ when $\vdash a \leadsto a'$.
    Need to show: $\Gamma_1 + \Gamma_2 \vdash \textbf{let unit} = a' \textbf{ in } b : B$.
    Follows by IH and rule GLC-LetUnit.
  - $\vdash \textbf{let unit} = \textbf{unit in } b \leadsto b$.
    Need to show: $\Gamma_1 + \Gamma_2 \vdash b : B$.
    Since $\Gamma_1 \vdash \textbf{unit} : \textbf{Unit}$, we have, $\Gamma_1 <: 0 \cdot \Gamma_1$. Hence, $\Gamma_1 + \Gamma_2 <: \Gamma_2$. This case, then, follows by rule GLC-Sub.

- Rule GLC-Case. Have: $q \cdot \Gamma_1 + \Gamma_2 \vdash \textbf{case}_q\, a \,\textbf{of}\, x_1.b_1; x_2.b_2 : B$ where $\Gamma_1 \vdash a : A_1 + A_2$ and $\Gamma_2, x_1 :^q A_1 \vdash b_1 : B$ and $\Gamma_2, x_2 :^q A_2 \vdash b_2 : B$.
  Let $\vdash \textbf{case}_q\, a \,\textbf{of}\, x_1.b_1; x_2.b_2 \leadsto c$. By inversion:
  - $\vdash \textbf{case}_q\, a \,\textbf{of}\, x_1.b_1; x_2.b_2 \leadsto \textbf{case}_q\, a' \,\textbf{of}\, x_1.b_1; x_2.b_2$ when $\vdash a \leadsto a'$.
    Need to show: $q \cdot \Gamma_1 + \Gamma_2 \vdash \textbf{case}_q\, a' \,\textbf{of}\, x_1.b_1; x_2.b_2 : B$.
    Follows by IH and rule GLC-Case.
  - $\vdash \textbf{case}_q\, (\textbf{inj}_1\, a_1) \,\textbf{of}\, x_1.b_1; x_2.b_2 \leadsto b_1\{a_1/x_1\}$.
    Need to show $q \cdot \Gamma_1 + \Gamma_2 \vdash b_1\{a_1/x_1\} : B$.
    By inversion on $\Gamma_1 \vdash \textbf{inj}_1\, a_1 : A_1 + A_2$, we have $\Gamma_1 \vdash a_1 : A_1$.
    This case, then, follows by the substitution lemma.
  - $\vdash \textbf{case}_q\, (\textbf{inj}_2\, a_2) \,\textbf{of}\, x_1.b_1; x_2.b_2 \leadsto b_2\{a_2/x_2\}$.
    Similar to the previous case.

- Rules GLC-Weak and GLC-Sub. Follows by IH.

□

Theorem C.5 (Theorem 4.4) If $\varnothing \vdash a : A$, then either a is a value or there exists a' such that $\vdash a \leadsto a'$.

Proof. By induction on $\varnothing \vdash a : A$.

- Rules GLC-Var and GLC-Weak. Do not apply since the context must be empty.
- Rule GLC-App. Have: $\Gamma_1 + q \cdot \Gamma_2 \vdash b\ a^q : B$ where $\Gamma_1 \vdash b : A \to B$ and $\Gamma_2 \vdash a : A$.
  Need to show: $\exists c, \vdash b\ a^q \leadsto c$.
  By IH, b is either a value or $\vdash b \leadsto b'$.
  If b is a value, then $b = \lambda^q x{:}A.b'$ for some b'. Therefore, $\vdash b\ a^q \leadsto b'\{a/x\}$.
  Otherwise, $\vdash b\ a^q \leadsto b'\ a^q$.
- Rule GLC-LetPair. Have: $\Gamma_1 + \Gamma_2 \vdash \textbf{let}\,({x_1}^q, x_2) = a\ \textbf{in}\ b : B$ where $\Gamma_1 \vdash a : {}^q A_1 \times A_2$ and $\Gamma_2, x_1 :^q A_1, x_2 :^1 A_2 \vdash b : B$.
  Need to show: $\exists c, \vdash \textbf{let}\,({x_1}^q, x_2) = a\ \textbf{in}\ b \leadsto c$.
  By IH, a is either a value or $\vdash a \leadsto a'$.
  If a is a value, then $a = ({a_1}^q, a_2)$. Therefore, $\vdash \textbf{let}\,({x_1}^q, x_2) = a\ \textbf{in}\ b \leadsto b\{a_1/x_1\}\{a_2/x_2\}$.
  Otherwise, $\vdash \textbf{let}\,({x_1}^q, x_2) = a\ \textbf{in}\ b \leadsto \textbf{let}\,({x_1}^q, x_2) = a'\ \textbf{in}\ b$.
- Rule GLC-LetUnit. Have: $\Gamma_1 + \Gamma_2 \vdash \textbf{let}\,\textbf{unit} = a\ \textbf{in}\ b : B$ where $\Gamma_1 \vdash a : \textbf{Unit}$ and $\Gamma_2 \vdash b : B$.
  Need to show: $\exists c, \vdash \textbf{let}\,\textbf{unit} = a\ \textbf{in}\ b \leadsto c$.
  By IH, a is either a value or $\vdash a \leadsto a'$.
  If a is a value, then $a = \textbf{unit}$. Therefore, $\vdash \textbf{let}\,\textbf{unit} = a\ \textbf{in}\ b \leadsto b$.
  Otherwise, $\vdash \textbf{let}\,\textbf{unit} = a\ \textbf{in}\ b \leadsto \textbf{let}\,\textbf{unit} = a'\ \textbf{in}\ b$.
- Rule GLC-Case. Have: $q \cdot \Gamma_1 + \Gamma_2 \vdash \textbf{case}_q\ a\ \textbf{of}\ x_1.b_1; x_2.b_2 : B$ where $\Gamma_1 \vdash a : A_1 + A_2$ and $\Gamma_2, x_1 :^q A_1 \vdash b_1 : B$ and $\Gamma_2, x_2 :^q A_2 \vdash b_2 : B$.
  Need to show: $\exists c, \vdash \textbf{case}_q\ a\ \textbf{of}\ x_1.b_1; x_2.b_2 \leadsto c$.
  By IH, a is either a value or $\vdash a \leadsto a'$.
  If a is a value, then $a = \textbf{inj}_1\ a_1$ or $a = \textbf{inj}_2\ a_2$.
  Then, $\vdash \textbf{case}_q\ a\ \textbf{of}\ x_1.b_1; x_2.b_2 \leadsto b_1\{a_1/x_1\}$ or $\vdash \textbf{case}_q\ a\ \textbf{of}\ x_1.b_1; x_2.b_2 \leadsto b_2\{a_2/x_2\}$.
  Otherwise, $\vdash \textbf{case}_q\ a\ \textbf{of}\ x_1.b_1; x_2.b_2 \leadsto \textbf{case}_q\ a'\ \textbf{of}\ x_1.b_1; x_2.b_2$.
- Rule GLC-Sub. Follows by IH.
- Rules GLC-Lam, GLC-Pair, GLC-Unit, GLC-Inj1, and GLC-Inj2. The terms typed by these rules are values.

□

C.3 Proofs of Lemmas/Theorems Stated in Section 4.3

Lemma C.6 (Lemma 4.5) If $[H_1]\, a_1 \Rightarrow^{r_1}_{S_1} [H_1'; \mathbf{u}_1'; \Gamma_1']\, a_1'$ and $[H_2]\, a_2 \Rightarrow^{r_2}_{S_2} [H_2'; \mathbf{u}_2'; \Gamma_2']\, a_2'$ such that $(H_1, a_1) \sim_\alpha (H_2, a_2)$, then $(H_1', a_1') \sim_\alpha (H_2', a_2')$.

Proof. By induction on $[H_1]\, a_1 \Rightarrow^{r_1}_{S_1} [H'_1\,;\, \mathbf{u}'_1\,;\, \Gamma'_1]\, a'_1$ and subsequent inversion on $[H_2]\, a_2 \Rightarrow^{r_2}_{S_2} [H'_2\,;\, \mathbf{u}'_2\,;\, \Gamma'_2]\, a'_2$. □

Lemma C.7 (Lemma 4.6) Let $[H]\, a \Rightarrow^r_S [H'\,;\, \mathbf{u}'\,;\, \Gamma']\, a'$. If $|H| = |H'|$, then $a\{H\} = a'\{H'\}$; otherwise $\vdash a\{H\} \rightsquigarrow a'\{H'\}$.

Proof. By induction on $[H]\, a \Rightarrow^r_S [H'\,;\, \mathbf{u}'\,;\, \Gamma']\, a'$. We present some of the interesting cases below. Note that we simplify judgments for the ease of presentation.

- Rule Heap-Var. Have: $[H_1, x \overset{(q+r)}{\mapsto} a, H_2]\, x \Rightarrow^r_S [H_1, x \overset{q}{\mapsto} a, H_2]\, a$.
  Need to show: $x\{H_1, x \overset{(q+r)}{\mapsto} a, H_2\} = a\{H_1, x \overset{q}{\mapsto} a, H_2\}$.
  By our assumption, heaps do not define any variable more than once.
  So, $x\{H_1, x \overset{(q+r)}{\mapsto} a, H_2\} = a\{H_1, x \overset{(q+r)}{\mapsto} a, H_2\}$.
  This case follows.

- Rule Heap-AppL. Have: $[H]\, b\ a^q \Rightarrow^r_S [H']\, b'\ a^q$ where $[H]\, b \Rightarrow^r_{S \cup \mathrm{fv}\, a} [H']\, b'$.
  Need to show: If $|H| = |H'|$, then $b\{H\}\ (a\{H\})^q = b'\{H'\}\ (a\{H'\})^q$. Otherwise, $\vdash b\{H\}\ (a\{H\})^q \rightsquigarrow b'\{H'\}\ (a\{H'\})^q$.
  Say, $|H| = |H'|$. Then, by IH, $b\{H\} = b'\{H'\}$. Further, we have, $\lfloor H \rfloor = \lfloor H' \rfloor$. So, $a\{H\} = a\{H'\}$. Therefore, $b\{H\}\ (a\{H\})^q = b'\{H'\}\ (a\{H'\})^q$.
  Otherwise, by IH, $\vdash b\{H\} \rightsquigarrow b'\{H'\}$. Also, since any fresh variable in H' is chosen avoiding the free variables of a, so $a\{H\} = a\{H'\}$. Therefore, $\vdash b\{H\}\ (a\{H\})^q \rightsquigarrow b'\{H'\}\ (a\{H'\})^q$.

- Rule Heap-AppBeta. Have: $[H]\, (\lambda^q y{:}A_0.b)\ a^q \Rightarrow^r_S [H, y \overset{r\cdot q}{\mapsto} a]\, b$, assuming y is a fresh variable.
  Need to show: $\vdash (\lambda^q y{:}A_0.b\{H\})\ (a\{H\})^q \rightsquigarrow b\{H, y \overset{r\cdot q}{\mapsto} a\}$.
  We have, $b\{H, y \overset{r\cdot q}{\mapsto} a\} = (b\{a/y\})\{H\} = b\{H\}\{a\{H\}/y\}$.
  By β-rule, $\vdash (\lambda^q y{:}A_0.b\{H\})\ (a\{H\})^q \rightsquigarrow b\{H\}\{a\{H\}/y\}$.
  This case follows.

□

Lemma C.8 (Lemma 4.7) If $[H]\, a \Rrightarrow^r_S [H'\,;\, \mathbf{u}'\,;\, \Gamma']\, a'$, then $\overline{H} \diamond \overline{\Gamma'} <: \overline{H'} + \mathbf{u}'$.

Proof. First, we show that if $[H]\, a \Rightarrow^r_S [H'\,;\, \mathbf{u}'\,;\, \Gamma']\, a'$, then $\overline{H} \diamond \overline{\Gamma'} <: \overline{H'} + \mathbf{u}'$.
The proof is by induction on $[H]\, a \Rightarrow^r_S [H'\,;\, \mathbf{u}'\,;\, \Gamma']\, a'$. We present some of the interesting cases below.

- Rule Heap-Var. Have: $[H_1, x \overset{(q+r)}{\mapsto} a, H_2]\, x \Rightarrow^r_S [H_1, x \overset{q}{\mapsto} a, H_2\,;\, \mathbf{0}^{|H_1|} \diamond r \diamond \mathbf{0}^{|H_2|}\,;\, \varnothing]\, a$.
  Need to show: $\overline{H_1} \diamond (q+r) \diamond \overline{H_2} <: \overline{H_1} \diamond (q+r) \diamond \overline{H_2}$.
  Follows by reflexivity.

- Rule Heap-AppL. Have: $[H]\, b\ a^q \Rightarrow^r_S [H'\,;\, \mathbf{u}'\,;\, \Gamma']\, b'\ a^q$ where $[H]\, b \Rightarrow^r_{S \cup \mathrm{fv}\, a} [H'\,;\, \mathbf{u}'\,;\, \Gamma']\, b'$.
  Need to show: $\overline{H} \diamond \overline{\Gamma'} <: \overline{H'} + \mathbf{u}'$.
  Follows by IH.

- Rule Heap-AppBeta. Have: $[H]\,(\lambda^q y\!:\!A_0.b)\; a^q \Rightarrow^r_S [H, y \overset{r\cdot q}{\mapsto} (\Gamma|a|A)\,;\,\mathbf{0}\,;\, y\!:^{r\cdot q} A]\, b$, assuming y is a fresh variable.
  Need to show: $\overline{H} \diamond (r\cdot q) <: \overline{H} \diamond (r \cdot q)$.
  Follows by reflexivity.

- Rule Heap-SubL. Have: $[H_2]\, a \Rightarrow^r_S [H'\,;\,\mathbf{u}'\,;\,\Gamma']\, a'$ where $[H_1]\, a \Rightarrow^r_S [H'\,;\,\mathbf{u}'\,;\,\Gamma']\, a'$ and $H_2 <: H_1$.
  Need to show: $\overline{H_2} \diamond \overline{\Gamma'} <: \overline{H'} + \mathbf{u}'$.
  By IH, $\overline{H_1} \diamond \overline{\Gamma'} <: \overline{H'} + \mathbf{u}'$.
  Since $H_2 <: H_1$, so $\overline{H_2} <: \overline{H_1}$.
  Then, $\overline{H_2} \diamond \overline{\Gamma'} <: \overline{H_1} \diamond \overline{\Gamma'} <: \overline{H'} + \mathbf{u}'$.

- Rule Heap-SubR. Have: $[H]\, a \Rightarrow^{r_2}_S [H'\,;\,\mathbf{u}'\,;\,\Gamma']\, a'$ where $[H]\, a \Rightarrow^{r_1}_S [H'\,;\,\mathbf{u}'\,;\,\Gamma']\, a'$ and $r_1 <: r_2$.
  Need to show: $\overline{H} \diamond \overline{\Gamma'} <: \overline{H'} + \mathbf{u}'$.
  Follows by IH.

Next we show that if $[H]\, a \Rrightarrow^r_S [H'\,;\,\mathbf{u}'\,;\,\Gamma']\, a'$, then $\overline{H} \diamond \overline{\Gamma'} <: \overline{H'} + \mathbf{u}'$.
The proof is by induction on $[H]\, a \Rrightarrow^r_S [H'\,;\,\mathbf{u}'\,;\,\Gamma']\, a'$:

- Rule Multi-One. Have: $[H]\, a \Rrightarrow^r_S [H'\,;\,\mathbf{u}'\,;\,\Gamma']\, a'$ where $[H]\, a \Rightarrow^r_S [H'\,;\,\mathbf{u}'\,;\,\Gamma']\, a'$.
  Need to show: $\overline{H} \diamond \overline{\Gamma'} <: \overline{H'} + \mathbf{u}'$.
  Follows by the proof above.

- Rule Multi-Many. Have: $[H]\, a \Rrightarrow^r_S [H''\,;\,(\mathbf{u}' \diamond \mathbf{0}) + \mathbf{u}''\,;\,\Gamma', \Gamma'']\, b$ where
  $[H]\, a \Rrightarrow^r_S [H'\,;\,\mathbf{u}'\,;\,\Gamma']\, b_1$ and $[H']\, b_1 \Rrightarrow^r_S [H''\,;\,\mathbf{u}''\,;\,\Gamma'']\, b$.
  Need to show: $\overline{H} \diamond \overline{\Gamma'} \diamond \overline{\Gamma''} <: \overline{H''} + (\mathbf{u}' \diamond \mathbf{0}) + \mathbf{u}''$.
  By IH, $\overline{H} \diamond \overline{\Gamma'} <: \overline{H'} + \mathbf{u}'$ and $\overline{H'} \diamond \overline{\Gamma''} <: \overline{H''} + \mathbf{u}''$.
  Then, $\overline{H} \diamond \overline{\Gamma'} \diamond \overline{\Gamma''} <: (\overline{H'} + \mathbf{u}') \diamond \overline{\Gamma''} = (\mathbf{u}' \diamond \mathbf{0}) + (\overline{H'} \diamond \overline{\Gamma''}) <: (\mathbf{u}' \diamond \mathbf{0}) + (\overline{H''} + \mathbf{u}'') = \overline{H''} + (\mathbf{u}' \diamond \mathbf{0}) + \mathbf{u}''$.

□

Lemma C.9 (Lemma 4.8) If $H \Vdash \Gamma$ and $\Gamma \vdash a : A$, then $\varnothing \vdash a\{H\} : A$.

Proof. By induction on $H \Vdash \Gamma$.

- Rule Compat-Empty. Have: $\varnothing \Vdash \varnothing$. Further, $\varnothing \vdash a : A$.
  Need to show: $\varnothing \vdash a : A$.
  Follows from premises.

- Rule Compat-Cons. Have: $H, y \overset{q}{\mapsto} (\Gamma_2|b|B) \Vdash \Gamma_1, y\!:^q B$ where $H \Vdash \Gamma_1 + q \cdot \Gamma_2$ and $\Gamma_2 \vdash b : B$. Further, $\Gamma_1, y\!:^q B \vdash a : A$.
  Need to show: $\varnothing \vdash (a\{b/y\})\{H\} : A$.
  Since $\Gamma_1, y\!:^q B \vdash a : A$ and $\Gamma_2 \vdash b : B$, by Lemma C.3, $\Gamma_1 + q \cdot \Gamma_2 \vdash a\{b/y\} : A$.
  This case, then, follows by IH.

□

Lemma C.10 (Lemma 4.9) If $H \Vdash \Gamma$, then $\overline{H} = \overline{H} \times \langle H \rangle + \overline{\Gamma}$.

Proof. We show this by induction on $H \Vdash \Gamma$. The base case is trivial.

For the cons-case, let $H', x \overset{q}{\mapsto} (\Gamma_2|a|A) \Vdash \Gamma_1, x :^q A$ where $H' \Vdash \Gamma_1 + (q \cdot \Gamma_2)$.
By IH, $\overline{H'} = \overline{H'} \times \langle H' \rangle + \overline{\Gamma_1 + (q \cdot \Gamma_2)}$.
Therefore, $\overline{H'} \diamond q = (\overline{H'} \diamond q) \times \begin{pmatrix} \langle H' \rangle & \mathbf{0}^\top \\ \overline{\Gamma_2} & 0 \end{pmatrix} + \overline{\Gamma_1} \diamond q$. (Here, $\mathbf{u}^\top$ denotes the transpose of vector $\mathbf{u}$.) □

Lemma C.11 (Compatibility and inserting definitions) If $H \Vdash (\Gamma_1 + q \cdot \Gamma_0), \Gamma_2$ and $\Gamma_0 \vdash b : B$, then $H_1, y \overset{q}{\mapsto} (\Gamma_0|b|B), H_2' \Vdash \Gamma_1, y :^q B, \Gamma_2$ where y is fresh and $H = H_1, H_2$ such that $|H_1| = |\Gamma_1|$ and H_2' is H_2 with each embedded context weakened by inserting $y :^0 B$ at the $|H_1|$-th position.

Proof. By induction on the length of Γ_2.
The base case where $|\Gamma_2| = 0$ follows by rule Compat-Cons.
For the inductive case, let $|\Gamma_2| = n + 1$. Then, $\Gamma_2 = \Gamma_{21}, z :^r C$.
As such, $H \Vdash (\Gamma_1 + q \cdot \Gamma_0), \Gamma_{21}, z :^r C$.
Then, by inversion, $H = H_0, z \overset{r}{\mapsto} (\Gamma'|c|C)$ where $H_0 \Vdash ((\Gamma_1 + q \cdot \Gamma_0), \Gamma_{21}) + r \cdot \Gamma'$ and $\Gamma' \vdash c : C$.
In other words, $H_0 \Vdash (\Gamma_1 + q \cdot \Gamma_0 + r \cdot \Gamma_1'), (\Gamma_{21} + r \cdot \Gamma_2')$ where $\Gamma' = \Gamma_1', \Gamma_2'$ such that $|\Gamma_1'| = |\Gamma_1|$.
Now by IH, $H_{01}, y \overset{q}{\mapsto} (\Gamma_0|b|B), H_{02}' \Vdash (\Gamma_1 + r \cdot \Gamma_1'), y :^q B, (\Gamma_{21} + r \cdot \Gamma_2')$, where $H_0 = H_{01}, H_{02}$ such that $|H_{01}| = |\Gamma_1|$ and H_{02}' is H_{02} with each embedded context weakened by inserting $y :^0 B$ at the $|H_{01}|$-th position.
Next, by weakening, $\Gamma_1', y :^0 B, \Gamma_2' \vdash c : C$.
By rewriting, we have, $H_{01}, y \overset{q}{\mapsto} (\Gamma_0|b|B), H_{02}' \Vdash (\Gamma_1, y :^q B, \Gamma_{21}) + r \cdot (\Gamma_1', y :^0 B, \Gamma_2')$.
By rule Compat-Cons, $H_{01}, y \overset{q}{\mapsto} (\Gamma_0|b|B), H_{02}', z \overset{r}{\mapsto} (\Gamma_1', y :^0 B, \Gamma_2'|c|C) \Vdash \Gamma_1, y :^q B, \Gamma_{21}, z :^r C$.
The case follows. □

Lemma C.12 (Compatibility and splitting) If $H \Vdash \Gamma_1 + \Gamma_2$, then there exists H_1 and H_2 such that $H_1 \Vdash \Gamma_1$ and $H_2 \Vdash \Gamma_2$ and $H = H_1 + H_2$. Note that given $\lfloor H_1 \rfloor = \lfloor H_2 \rfloor$, the heap $H_1 + H_2$ is defined as: $\lfloor H_1 + H_2 \rfloor = \lfloor H_1 \rfloor$ and $\overline{H_1 + H_2} = \overline{H_1} + \overline{H_2}$.

Proof. By induction on $H \Vdash \Gamma_1 + \Gamma_2$. The base case is trivial.

For the cons case, we have, $H', x \overset{q}{\mapsto} (\Gamma|a|A) \Vdash \Gamma_0, x :^q A$ where $H' \Vdash \Gamma_0 + q \cdot \Gamma$ and $\Gamma \vdash a : A$. Further, $\Gamma_0, x :^q A = \Gamma_1 + \Gamma_2$.
Need to show: $\exists H_1$ and H_2 such that $H_1 \Vdash \Gamma_1$ and $H_2 \Vdash \Gamma_2$ and $H', x \overset{q}{\mapsto} (\Gamma|a|A) = H_1 + H_2$.
Since $\Gamma_1 + \Gamma_2 = \Gamma_0, x :^q A$, so $\Gamma_1 = \Gamma_{01}, x :^{q_1} A$ and $\Gamma_2 = \Gamma_{02}, x :^{q_2} A$ where $\Gamma_0 = \Gamma_{01} + \Gamma_{02}$ and $q = q_1 + q_2$.
Then, $H' \Vdash (\Gamma_{01} + \Gamma_{02}) + (q_1 + q_2) \cdot \Gamma$. As such, $H' \Vdash (\Gamma_{01} + q_1 \cdot \Gamma) + (\Gamma_{02} + q_2 \cdot \Gamma)$.
By IH, there exists H_1' and H_2' such that $H_1' \Vdash \Gamma_{01} + q_1 \cdot \Gamma$ and $H_2' \Vdash \Gamma_{02} + q_2 \cdot \Gamma$ and $H' = H_1' + H_2'$.
Then, by rule Compat-Cons, $H_1', x \overset{q_1}{\mapsto} (\Gamma|a|A) \Vdash \Gamma_{01}, x :^{q_1} A$ and $H_2', x \overset{q_2}{\mapsto} (\Gamma|a|A) \Vdash \Gamma_{02}, x :^{q_2} A$.
The case follows by setting $H_1 := H_1', x \overset{q_1}{\mapsto} (\Gamma|a|A)$ and $H_2 := H_2', x \overset{q_2}{\mapsto} (\Gamma|a|A)$. □

Lemma C.13 (Compatibility and order) If $H_1 \Vdash \Gamma_1$ and $\Gamma_1 <: \Gamma_2$, then there exists H_2 such that $H_1 <: H_2$ and $H_2 \Vdash \Gamma_2$.

Proof. By induction on $H_1 \Vdash \Gamma_1$. The base case is trivial.

For the cons case, we have, $H, x \overset{q}{\mapsto} (\Gamma|a|A) \Vdash \Gamma_0, x :^q A$ where $H \Vdash \Gamma_0 + q \cdot \Gamma$ and $\Gamma \vdash a : A$.
Further, $\Gamma_0, x :^q A <: \Gamma_2$.
Need to show: exists H_2 such that $H, x \overset{q}{\mapsto} (\Gamma|a|A) <: H_2$ and $H_2 \Vdash \Gamma_2$.
Since $\Gamma_0, x :^q A <: \Gamma_2$, we have, $\Gamma_2 = \Gamma'_0, x :^{q'} A$ where $\Gamma_0 <: \Gamma'_0$ and $q <: q'$.
Then, $\Gamma_0 + q \cdot \Gamma <: \Gamma'_0 + q' \cdot \Gamma$. Next, by IH, exists H' such that $H <: H'$ and $H' \Vdash \Gamma'_0 + q' \cdot \Gamma$.
But by rule Compat-Cons, $H', x \overset{q'}{\mapsto} (\Gamma|a|A) \Vdash \Gamma'_0, x :^{q'} A$.
This case, then, follows by setting $H_2 := H', x \overset{q'}{\mapsto} (\Gamma|a|A)$. □

Theorem C.14 (Invariance) If $H \Vdash \Gamma_0 + r \cdot \Gamma$ and $\Gamma \vdash a : A$ and $r <: 1$, then either a is a value or there exists H', $\mathbf{u'}$, Γ'_0 and Γ' such that:

- $[H]\, a \Rightarrow^r_S [H' ; \mathbf{u'} ; \Gamma'_0]\, a'$
- $\Gamma' \vdash a' : A$
- $H' \Vdash (\Gamma_0, 0 \cdot \Gamma'_0) + r \cdot \Gamma'$
- $r \cdot (\overline{\Gamma} \diamond \mathbf{0}) \;+\; \mathbf{u'} \times \langle H' \rangle \;+\; \mathbf{0} \diamond \overline{\Gamma'_0} \;\;<:\;\; r \cdot \overline{\Gamma'} \;+\; \mathbf{u'} \;+\; (\mathbf{0} \diamond \overline{\Gamma'_0}) \times \langle H' \rangle$

Proof. By induction on $\Gamma \vdash a : A$. The cases where a is a value follow immediately. We present the other cases below.

- Rule GLC-Var. Have: $0 \cdot \Gamma_1, x :^1 A \vdash x : A$.
  Further, $H \Vdash \Gamma_0 + r \cdot (0 \cdot \Gamma_1, x :^1 A)$ where $r <: 1$.
  Then, $\Gamma_0 = \Gamma_{01}, x :^q A$. As such, $H \Vdash \Gamma_{01}, x :^{q+r} A$.
  By inversion, $H = H_1, x \overset{q+r}{\mapsto} (\Gamma_2|a|A)$ where $H_1 \Vdash \Gamma_{01} + (q + r) \cdot \Gamma_2$ and $\Gamma_2 \vdash a : A$.
  Then, we can show the clauses of the theorem one by one:
  - $[H_1, x \overset{q+r}{\mapsto} (\Gamma_2|a|A)]\, x \Rightarrow^r_S [H_1, x \overset{q}{\mapsto} (\Gamma_2|a|A) ; \mathbf{0}^{|H_1|} \diamond r ; \varnothing]\, a$, by rule Heap-Var.
  - Next, we have, $\Gamma_2 \vdash a : A$. By rule GLC-Weak, $\Gamma_2, x :^0 A \vdash a : A$.
  - Now, by rule Compat-Cons, $H_1, x \overset{q}{\mapsto} (\Gamma_2|a|A) \Vdash (\Gamma_{01} + r \cdot \Gamma_2), x :^q A$.
    Then, $H_1, x \overset{q}{\mapsto} (\Gamma_2|a|A) \Vdash (\Gamma_{01}, x :^q A) + r \cdot (\Gamma_2, x :^0 A)$.
  - For the fourth clause, need to show: $r \cdot (\mathbf{0} \diamond 1) + (\mathbf{0} \diamond r) \times \begin{pmatrix} \langle H_1 \rangle & \mathbf{0}^\top \\ \overline{\Gamma_2} & 0 \end{pmatrix} <: r \cdot (\overline{\Gamma_2} \diamond 0) \;+\; (\mathbf{0} \diamond r)$.
    The clause follows by reflexivity.

- Rule GLC-Weak. Have: $\Gamma_1, y :^0 B \vdash a : A$ where $\Gamma_1 \vdash a : A$.
  Further, $H \Vdash \Gamma_0 + r \cdot (\Gamma_1, y :^0 B)$ where $r <: 1$.
  Then, $\Gamma_0 = \Gamma_{01}, y :^q B$. As such, $H \Vdash (\Gamma_{01} + r \cdot \Gamma_1), y :^q B$.
  By inversion, $H = H_1, y \overset{q}{\mapsto} (\Gamma_2 | b | B)$ where $H_1 \Vdash \Gamma_{01} + r \cdot \Gamma_1 + q \cdot \Gamma_2$ and $\Gamma_2 \vdash b : B$.
  Now, if a is a value, this case follows immediately.
  Otherwise, since $H_1 \Vdash \Gamma_{01} + q \cdot \Gamma_2 + r \cdot \Gamma_1$ and $\Gamma_1 \vdash a : A$, by IH, there exists H_1', $\mathbf{u'}$, Γ_0' and Γ_1' such that:

  - $[H_1]\, a \Rightarrow^r_{S \cup \{y\}} [H_1' ; \mathbf{u'} ; \Gamma_0']\, a'$
  - $\Gamma_1' \vdash a' : A$
  - $H_1' \Vdash (\Gamma_{01} + q \cdot \Gamma_2, 0 \cdot \Gamma_0') + r \cdot \Gamma_1'$
  - $r \cdot (\overline{\Gamma_1} \diamond \mathbf{0}) + \mathbf{u'} \times \langle H_1' \rangle + \mathbf{0} \diamond \overline{\Gamma_0'} <: r \cdot \overline{\Gamma_1'} + \mathbf{u'} + (\mathbf{0} \diamond \overline{\Gamma_0'}) \times \langle H_1' \rangle$

  Using these clauses, we show the corresponding clauses of the theorem:

  - By weakening, $[H_1, y \overset{q}{\mapsto} (\Gamma_2 | b | B)]\, a \Rightarrow^r_S [H_{11}', y \overset{q}{\mapsto} (\Gamma_2 | b | B), H_{12}' ; \mathbf{u}_1' \diamond 0 \diamond \mathbf{u}_2' ; \Gamma_0']\, a'$, where $H_1' = H_{11}', H_{12}'$ and $\mathbf{u'} = \mathbf{u}_1' \diamond \mathbf{u}_2'$ and $|H_{11}'| = |\mathbf{u}_1'| = |H_1|$.
    Next, in the above judgment, we replace H_{12}' with H_{12}'', which is basically H_{12}' with each embedded context weakened by inserting $y :^0 B$ at the $|H_1|$-th position. This replacement is permitted by Lemma C.6 and is needed to ensure compatibility.
  - Again, by weakening, $\Gamma_{11}', y :^0 B, \Gamma_{12}' \vdash a' : A$, where $\Gamma_1' = \Gamma_{11}', \Gamma_{12}'$ and $|\Gamma_{11}'| = |H_1|$.
  - $H_{11}', y \overset{q}{\mapsto} (\Gamma_2 | b | B), H_{12}'' \Vdash (\Gamma_{01}, y :^q B, 0 \cdot \Gamma_0') + r \cdot (\Gamma_{11}', y :^0 B, \Gamma_{12}')$, by Lemma C.11.
  - The fourth clause follows upon inserting 0 at the $|H_1|$-th position on both sides of the fourth clause above.

- Rule GLC-LetUnit. Have: $\Gamma_1 + \Gamma_2 \vdash \mathbf{let\, unit} = a \ \mathbf{in}\ b : B$ where $\Gamma_1 \vdash a : \mathbf{Unit}$ and $\Gamma_2 \vdash b : B$.
  Further, $H \Vdash \Gamma_0 + r \cdot (\Gamma_1 + \Gamma_2)$ where $r <: 1$.
  There are two cases to consider.

  - a is not a value.
    Since $H \Vdash \Gamma_0 + r \cdot \Gamma_2 + r \cdot \Gamma_1$ and $\Gamma_1 \vdash a : \mathbf{Unit}$, by IH, there exists H', $\mathbf{u'}$, Γ_0' and Γ_1' such that:
    * $[H]\, a \Rightarrow^r_{S \cup \mathrm{fv}\, b} [H' ; \mathbf{u'} ; \Gamma_0']\, a'$
    * $\Gamma_1' \vdash a' : \mathbf{Unit}$
    * $H' \Vdash (\Gamma_0 + r \cdot \Gamma_2, 0 \cdot \Gamma_0') + r \cdot \Gamma_1'$
    * $r \cdot (\overline{\Gamma_1} \diamond \mathbf{0}) + \mathbf{u'} \times \langle H' \rangle + \mathbf{0} \diamond \overline{\Gamma_0'} <: r \cdot \overline{\Gamma_1'} + \mathbf{u'} + (\mathbf{0} \diamond \overline{\Gamma_0'}) \times \langle H' \rangle$

    Using these clauses, we show the corresponding clauses of the theorem:
    * $[H]\, \mathbf{let\, unit} = a \ \mathbf{in}\ b \Rightarrow^r_S [H' ; \mathbf{u'} ; \Gamma_0']\, \mathbf{let\, unit} = a' \ \mathbf{in}\ b$, by rule Heap-LetUnitL.
    * By weakening and rule GLC-LetUnit, $\Gamma_1' + (\Gamma_2, 0 \cdot \Gamma_0') \vdash \mathbf{let\, unit} = a' \ \mathbf{in}\ b : B$.
    * Next, $H' \Vdash (\Gamma_0, 0 \cdot \Gamma_0') + r \cdot (\Gamma_1' + (\Gamma_2, 0 \cdot \Gamma_0'))$, by rearrangement of the third clause.
    * The fourth clause follows upon adding $r \cdot (\overline{\Gamma_2} \diamond \mathbf{0})$ to both sides of the fourth clause above.

– a is a value.
Since $\Gamma_1 \vdash a : \mathbf{Unit}$, so $a = \mathbf{unit}$ and $\Gamma_1 <: 0 \cdot \Gamma_1$.
Then, we can show the clauses one by one:

  * $[H]\, \mathbf{let\, unit} = \mathbf{unit\ in}\ b \Rightarrow^r_S [H]\, b$.
  * We have, $\Gamma_2 \vdash b : B$. Since $\Gamma_1 <: 0 \cdot \Gamma_1$, so $\Gamma_1 + \Gamma_2 <: \Gamma_2$.
    Then, by rule GLC-Sub, $\Gamma_1 + \Gamma_2 \vdash b : B$.
  * The third clause follows from premises.
  * The fourth clause, $r \cdot (\overline{\Gamma_1} + \overline{\Gamma_2}) <: r \cdot (\overline{\Gamma_1} + \overline{\Gamma_2})$, follows by reflexivity.

- Rule GLC-App. Have: $\Gamma_1 + q \cdot \Gamma_2 \vdash b\ a^q : B$ where $\Gamma_1 \vdash b : {}^qA \to B$ and $\Gamma_2 \vdash a : A$.
Further, $H \Vdash \Gamma_0 + r \cdot (\Gamma_1 + q \cdot \Gamma_2)$ where $r <: 1$.
There are two cases to consider.

  – b is not a value.
  Since $H \Vdash \Gamma_0 + r \cdot q \cdot \Gamma_2 + r \cdot \Gamma_1$ and $\Gamma_1 \vdash b : {}^qA \to B$, there exists H', $\mathbf{u'}$, Γ'_0 and Γ'_1 such that:

    * $[H]\, b \Rightarrow^r_{S \cup \mathrm{fv}\, a} [H' ; \mathbf{u'} ; \Gamma'_0]\, b'$
    * $\Gamma'_1 \vdash b' : {}^qA \to B$
    * $H' \Vdash (\Gamma_0 + r \cdot q \cdot \Gamma_2, 0 \cdot \Gamma'_0) + r \cdot \Gamma'_1$
    * $r \cdot (\overline{\Gamma_1} \diamond \mathbf{0}) + \mathbf{u'} \times \langle H' \rangle + \mathbf{0} \diamond \overline{\Gamma'_0} <: r \cdot \overline{\Gamma'_1} + \mathbf{u'} + (\mathbf{0} \diamond \overline{\Gamma'_0}) \times \langle H' \rangle$

  Using these clauses, we show the corresponding clauses of the theorem:

    * $[H]\, b\ a^q \Rightarrow^r_S [H' ; \mathbf{u'} ; \Gamma'_0]\, b'\ a^q$, by rule Heap-AppL.
    * By weakening and rule GLC-App, $\Gamma'_1 + q \cdot (\Gamma_2, 0 \cdot \Gamma'_0) \vdash b'\ a^q : B$.
    * Next, $H' \Vdash (\Gamma_0, 0 \cdot \Gamma'_0) + r \cdot (\Gamma'_1 + q \cdot (\Gamma_2, 0 \cdot \Gamma'_0))$, by rearrangement of the third clause.
    * The fourth clause follows upon adding $r \cdot q \cdot (\overline{\Gamma_2} \diamond \mathbf{0})$ to both sides of the fourth clause above.

  – b is a value.
  Since $\Gamma_1 \vdash b : {}^qA \to B$, so $b = \lambda^q y{:}A.b'$ and $\Gamma_1, y :^q A \vdash b' : B$.
  Then, we can show the clauses one by one:

    * $[H]\, (\lambda^q y{:}A.b')\ a \Rightarrow^r_S [H, y \overset{r \cdot q}{\mapsto} (\Gamma_2 | a | A) ; \mathbf{0} ; y :^{r \cdot q} A]\, b'$, assuming y *fresh*.
    * $\Gamma_1, y :^q A \vdash b' : B$.
    * Next, since $H \Vdash \Gamma_0 + r \cdot \Gamma_1 + r \cdot q \cdot \Gamma_2$ and $\Gamma_2 \vdash a : A$, by the cons-rule, $H, y \overset{r \cdot q}{\mapsto} (\Gamma_2 | a | A) \Vdash (\Gamma_0 + r \cdot \Gamma_1), y :^{r \cdot q} A$.
      In other words, $H, y \overset{r \cdot q}{\mapsto} (\Gamma_2 | a | A) \Vdash (\Gamma_0, y :^0 A) + r \cdot (\Gamma_1, y :^q A)$.
    * Next, need to show: $r \cdot ((\overline{\Gamma_1} + q \cdot \overline{\Gamma_2}) \diamond 0) + (\mathbf{0} \diamond r \cdot q)\ <:\ r \cdot (\overline{\Gamma_1} \diamond q) + (\mathbf{0} \diamond r \cdot q) \times \begin{pmatrix} \langle H \rangle & \mathbf{0}^{\top} \\ \overline{\Gamma_2} & 0 \end{pmatrix}$.
      This clause follows by reflexivity.

- Rule GLC-LetBox. Have: $\Gamma_1 + \Gamma_2 \vdash \mathbf{let\, box}_q\, x = a\ \mathbf{in}\ b : B$ where $\Gamma_1 \vdash a : \Box^q A$ and $\Gamma_2, x :^q A \vdash b : B$.
Further, $H \Vdash \Gamma_0 + r \cdot (\Gamma_1 + \Gamma_2)$ where $r <: 1$.
There are two cases to consider.

- a is not a value.
  Since $H \Vdash \Gamma_0 + r \cdot \Gamma_2 + r \cdot \Gamma_1$ and $\Gamma_1 \vdash a : \Box^q A$, by IH, there exists H', $\mathbf{u}'$, Γ'_0 and Γ'_1 such that:
  * $[H]\, a \Rightarrow^r_{S \cup (\text{fv } b - \{x\})} [H' ; \mathbf{u}' ; \Gamma'_0]\, a'$
  * $\Gamma'_1 \vdash a' : \Box^q A$
  * $H' \Vdash (\Gamma_0 + r \cdot \Gamma_2, 0 \cdot \Gamma'_0) + r \cdot \Gamma'_1$
  * $r \cdot (\overline{\Gamma_1} \diamond \mathbf{0}) + \mathbf{u}' \times \langle H' \rangle + \mathbf{0} \diamond \overline{\Gamma'_0} <: r \cdot \overline{\Gamma'_1} + \mathbf{u}' + (\mathbf{0} \diamond \overline{\Gamma'_0}) \times \langle H' \rangle$

  Using these clauses, we show the corresponding clauses of the theorem:
  * $[H]\, \mathbf{let\, box}_q\, x = a\ \mathbf{in}\ b \Rightarrow^r_S [H' ; \mathbf{u}' ; \Gamma'_0]\, \mathbf{let\, box}_q\, x = a'\ \mathbf{in}\ b$, by left rule.
  * By weakening and rule GLC-LetBox, $\Gamma'_1 + (\Gamma_2, 0 \cdot \Gamma'_0) \vdash \mathbf{let\, box}_q\, x = a'\ \mathbf{in}\ b : B$.
  * Next, $H' \Vdash (\Gamma_0, 0 \cdot \Gamma'_0) + r \cdot (\Gamma'_1 + (\Gamma_2, 0 \cdot \Gamma'_0))$, by rearrangement of the third clause.
  * The fourth clause follows upon adding $r \cdot (\overline{\Gamma_2} \diamond \mathbf{0})$ to both sides of the fourth clause above.

- a is a value.
  Since $\Gamma_1 \vdash a : \Box^q A$, so $a = \mathbf{box}_q\, a'$. Further, there exists Γ'_1 such that $\Gamma'_1 \vdash a' : A$ and $\Gamma_1 <: q \cdot \Gamma'_1$.
  Next, we have, $H \Vdash \Gamma_0 + r \cdot (\Gamma_1 + \Gamma_2)$. But $\Gamma_0 + r \cdot (\Gamma_1 + \Gamma_2) <: \Gamma_0 + r \cdot (q \cdot \Gamma'_1 + \Gamma_2)$.
  So, by Lemma C.13, $\exists H'$ such that $H <: H'$ and $H' \Vdash \Gamma_0 + r \cdot (q \cdot \Gamma'_1 + \Gamma_2)$.
  Now, we show the clauses one by one:
  * $[H']\, \mathbf{let\, box}_q\, x = \mathbf{box}_q\, a'\ \mathbf{in}\ b \Rightarrow^r_S [H', x \overset{r \cdot q}{\mapsto} (\Gamma'_1 | a' | A) ; \mathbf{0} ; x :^{r \cdot q} A]\, b$, by rule Heap-LetBoxBeta (assuming x *fresh*).
    But since $H <: H'$, by rule Heap-SubL, $[H]\, \mathbf{let\, box}_q\, x = \mathbf{box}_q\, a'\ \mathbf{in}\ b \Rightarrow^r_S [H', x \overset{r \cdot q}{\mapsto} (\Gamma'_1 | a' | A) ; \mathbf{0} ; x :^{r \cdot q} A]\, b$.
  * $\Gamma_2, x :^q A \vdash b : B$, from premises.
  * Next, since $H' \Vdash \Gamma_0 + r \cdot (q \cdot \Gamma'_1 + \Gamma_2)$ and $\Gamma'_1 \vdash a' : A$, by the cons-rule, $H', x \overset{r \cdot q}{\mapsto} (\Gamma'_1 | a' | A) \Vdash (\Gamma_0 + r \cdot \Gamma_2), x :^{r \cdot q} A$.
    In other words, $H', x \overset{r \cdot q}{\mapsto} (\Gamma'_1 | a' | A) \Vdash (\Gamma_0, x :^0 A) + r \cdot (\Gamma_2, x :^q A)$.
  * Next, need to show: $r \cdot ((\overline{\Gamma_1} + \overline{\Gamma_2}) \diamond 0) + (\mathbf{0} \diamond r \cdot q)\ <:\ r \cdot (\overline{\Gamma_2} \diamond q) + (\mathbf{0} \diamond r \cdot q) \times \begin{pmatrix} \langle H' \rangle & \mathbf{0}^{\top} \\ \overline{\Gamma'_1} & 0 \end{pmatrix}$.
    This clause follows from $\Gamma_1 <: q \cdot \Gamma'_1$.

- Rule GLC-LetPair. Have: $\Gamma_1 + \Gamma_2 \vdash \mathbf{let}\, ({x_1}^q, x_2) = a\ \mathbf{in}\ b : B$ where $\Gamma_1 \vdash a : {}^q A_1 \times A_2$ and $\Gamma_2, x_1 :^q A_1, x_2 :^1 A_2 \vdash b : B$.
  Further, $H \Vdash \Gamma_0 + r \cdot (\Gamma_1 + \Gamma_2)$ where $r <: 1$.
  There are two cases to consider.

  - a is not a value.
    Since $H \Vdash \Gamma_0 + r \cdot \Gamma_2 + r \cdot \Gamma_1$ and $\Gamma_1 \vdash a : {}^q A_1 \times A_2$, by IH, there exists H', $\mathbf{u}'$, Γ'_0 and Γ'_1 such that:
    * $[H]\, a \Rightarrow^r_{S \cup (\text{fv } b - \{x_1, x_2\})} [H' ; \mathbf{u}' ; \Gamma'_0]\, a'$
    * $\Gamma'_1 \vdash a' : {}^q A_1 \times A_2$
    * $H' \Vdash (\Gamma_0 + r \cdot \Gamma_2, 0 \cdot \Gamma'_0) + r \cdot \Gamma'_1$
    * $r \cdot (\overline{\Gamma_1} \diamond \mathbf{0}) + \mathbf{u}' \times \langle H' \rangle + \mathbf{0} \diamond \overline{\Gamma'_0} <: r \cdot \overline{\Gamma'_1} + \mathbf{u}' + (\mathbf{0} \diamond \overline{\Gamma'_0}) \times \langle H' \rangle$

Using the above clauses, we show the corresponding clauses of the theorem:

- $[H]\,\textbf{let}\,(x_1{}^q, x_2) = a \textbf{ in } b \Rightarrow^r_S [H' \,;\, \mathbf{u}' \,;\, \Gamma'_0]\,\textbf{let}\,(x_1{}^q, x_2) = a' \textbf{ in } b$, by left rule.
- By weakening and rule GLC-LetPair, $\Gamma'_1 + (\Gamma_2, 0 \cdot \Gamma'_0) \vdash \textbf{let}\,(x_1{}^q, x_2) = a' \textbf{ in } b : B$.
- Next, $H' \Vdash (\Gamma_0, 0 \cdot \Gamma'_0) + r \cdot (\Gamma'_1 + (\Gamma_2, 0 \cdot \Gamma'_0))$, by rearrangement of the third clause.
- The fourth clause follows upon adding $r \cdot (\overline{\Gamma_2} \diamond \mathbf{0})$ to both sides of the fourth clause above.

– a is a value.

Since $\Gamma_1 \vdash a : {}^q A_1 \times A_2$, so $a = (a_1{}^q, a_2)$. Further, there exists Γ_{11} and Γ_{12} such that $\Gamma_{11} \vdash a_1 : A_1$ and $\Gamma_{12} \vdash a_2 : A_2$ and $\Gamma_1 <: q \cdot \Gamma_{11} + \Gamma_{12}$.

Next, we have, $H \Vdash \Gamma_0 + r \cdot (\Gamma_1 + \Gamma_2)$. But $\Gamma_0 + r \cdot (\Gamma_1 + \Gamma_2) <: \Gamma_0 + r \cdot (q \cdot \Gamma_{11} + \Gamma_{12} + \Gamma_2)$.

So, by Lemma C.13, $\exists H'$ such that $H <: H'$ and $H' \Vdash \Gamma_0 + r \cdot (q \cdot \Gamma_{11} + \Gamma_{12} + \Gamma_2)$.

Now, we show the clauses one by one:

- By rule Heap-LetPairBeta, $[H']\,\textbf{let}\,(x_1{}^q, x_2) = (a_1{}^q, a_2) \textbf{ in } b \Rightarrow^r_S [H', x_1 \overset{r \cdot q}{\mapsto} (\Gamma_{11}|a_1|A_1), x_2 \overset{r}{\mapsto} (\Gamma_{12}, x_1 :^0 A_1|a_2|A_2) \,;\, \mathbf{0} \,;\, x_1 :^{r \cdot q} A_1, x_2 :^r A_2]\, b$ (assuming x_1, x_2 *fresh*).
  But since $H <: H'$, by rule Heap-SubL, $[H]\,\textbf{let}\,(x_1{}^q, x_2) = (a_1{}^q, a_2) \textbf{ in } b \Rightarrow^r_S [H', x_1 \overset{r \cdot q}{\mapsto} (\Gamma_{11}|a_1|A_1), x_2 \overset{r}{\mapsto} (\Gamma_{12}, x_1 :^0 A_1|a_2|A_2) \,;\, \mathbf{0} \,;\, x_1 :^{r \cdot q} A_1, x_2 :^r A_2]\, b$.
- $\Gamma_2, x_1 :^q A_1, x_2 :^1 A_2 \vdash b : B$, from premises.
- Next, since $H' \Vdash \Gamma_0 + r \cdot (q \cdot \Gamma_{11} + \Gamma_{12} + \Gamma_2)$ and $\Gamma_{11} \vdash a_1 : A_1$, by the cons-rule, $H', x_1 \overset{r \cdot q}{\mapsto} (\Gamma_{11}|a_1|A_1) \Vdash (\Gamma_0 + r \cdot \Gamma_2 + r \cdot \Gamma_{12}), x_1 :^{r \cdot q} A_1$.
  Now, by weakening $\Gamma_{12}, x_1 :^0 A_1 \vdash a_2 : A_2$.
  Then, using the cons-rule again, $H', x_1 \overset{r \cdot q}{\mapsto} (\Gamma_{11}|a_1|A_1), x_2 \overset{r}{\mapsto} (\Gamma_{12}, x_1 :^0 A_1|a_2|A_2) \Vdash (\Gamma_0 + r \cdot \Gamma_2), x_1 :^{r \cdot q} A_1, x_2 :^r A_2$.
  In other words, $H', x_1 \overset{r \cdot q}{\mapsto} (\Gamma_{11}|a_1|A_1), x_2 \overset{r}{\mapsto} (\Gamma_{12}, x_1 :^0 A_1|a_2|A_2) \Vdash (\Gamma_0, x_1 :^0 A_1, x_2 :^0 A_2) + r \cdot (\Gamma_2, x_1 :^q A_1, x_2 :^1 A_2)$.
- Next, need to show: $r \cdot ((\overline{\Gamma_1} + \overline{\Gamma_2}) \diamond 0 \diamond 0) + (\mathbf{0} \diamond (r \cdot q) \diamond r) \;<:\; r \cdot (\overline{\Gamma_2} \diamond q \diamond 1) + (\mathbf{0} \diamond (r \cdot q) \diamond r) \times \begin{pmatrix} \langle H' \rangle & \mathbf{0}^\top & \mathbf{0}^\top \\ \overline{\Gamma_{11}} & 0 & 0 \\ \overline{\Gamma_{12}} & 0 & 0 \end{pmatrix}$.
  This clause follows from $\Gamma_1 <: q \cdot \Gamma_{11} + \Gamma_{12}$.

- Rule GLC-Case. Have: $q \cdot \Gamma_1 + \Gamma_2 \vdash \textbf{case}_q\, a \textbf{ of } x_1.b_1; x_2.b_2 : B$ where $\Gamma_1 \vdash a : A_1 + A_2$ and $\Gamma_2, x_1 :^q A_1 \vdash b_1 : B$ and $\Gamma_2, x_2 :^q A_2 \vdash b_2 : B$ and $q <: 1$.
  Further, $H \Vdash \Gamma_0 + r \cdot (q \cdot \Gamma_1 + \Gamma_2)$ where $r <: 1$.
  There are two cases to consider.

  – a is not a value.

  Since $r <: 1$ and $q <: 1$, so $r \cdot q <: 1$.

  Now, since $H \Vdash \Gamma_0 + r \cdot \Gamma_2 + r \cdot q \cdot \Gamma_1$ and $\Gamma_1 \vdash a : A_1 + A_2$, by IH, there exists H', $\mathbf{u}'$, Γ'_0 and Γ'_1 such that:

  - $[H]\, a \Rightarrow^{r \cdot q}_{S \cup (\text{fv}\, b_1 - \{x_1\}) \cup (\text{fv}\, b_2 - \{x_2\})} [H' \,;\, \mathbf{u}' \,;\, \Gamma'_0]\, a'$
  - $\Gamma'_1 \vdash a' : A_1 + A_2$
  - $H' \Vdash (\Gamma_0 + r \cdot \Gamma_2, 0 \cdot \Gamma'_0) + r \cdot q \cdot \Gamma'_1$

* $r \cdot q \cdot (\overline{\Gamma_1} \diamond \mathbf{0}) + \mathbf{u'} \times \langle H' \rangle + \mathbf{0} \diamond \overline{\Gamma'_0} <: r \cdot q \cdot \overline{\Gamma'_1} + \mathbf{u'} + (\mathbf{0} \diamond \overline{\Gamma'_0}) \times \langle H' \rangle$

Using the above clauses, we show the corresponding clauses of the theorem:

* $[H]\, \mathbf{case}_q\ a\ \mathbf{of}\ x_1.b_1; x_2.b_2 \Rightarrow_S^{r \cdot q} [H' ; \mathbf{u'} ; \Gamma'_0]\, \mathbf{case}_q\ a'\ \mathbf{of}\ x_1.b_1; x_2.b_2$, by left rule.
  Then, $[H]\, \mathbf{case}_q\ a\ \mathbf{of}\ x_1.b_1; x_2.b_2 \Rightarrow_S^{r} [H' ; \mathbf{u'} ; \Gamma'_0]\, \mathbf{case}_q\ a'\ \mathbf{of}\ x_1.b_1; x_2.b_2$, by rule Heap-SubR, since $r \cdot q <: r$.
* By weakening and rule GLC-Case, $q \cdot \Gamma'_1 + (\Gamma_2, 0 \cdot \Gamma'_0) \vdash \mathbf{case}_q\ a'\ \mathbf{of}\ x_1.b_1; x_2.b_2 : B$.
* Next, $H' \Vdash (\Gamma_0, 0 \cdot \Gamma'_0) + r \cdot (q \cdot \Gamma'_1 + (\Gamma_2, 0 \cdot \Gamma'_0))$, by rearrangement of the third clause.
* The fourth clause follows upon adding $r \cdot (\overline{\Gamma_2} \diamond \mathbf{0})$ to both sides of the fourth clause above.

- a is a value.
  Since $\Gamma_1 \vdash a : A_1 + A_2$, so $a = \mathbf{inj}_1\, a_1$ or $a = \mathbf{inj}_2\, a_2$.
  Say, $a = \mathbf{inj}_1\, a_1$. The reasoning is similar when $a = \mathbf{inj}_2\, a_2$.
  Since $a = \mathbf{inj}_1\, a_1$, by inversion, $\Gamma_1 \vdash a_1 : A_1$.
  Now, we show the clauses one by one:
  * $[H]\, \mathbf{case}_q\, (\mathbf{inj}_1\, a_1)\, \mathbf{of}\ x_1.b_1; x_2.b_2 \Rightarrow_S^{r} [H, x_1 \overset{r \cdot q}{\mapsto} (\Gamma_1 | a_1 | A_1) ; \mathbf{0} ; x_1 :^{r \cdot q} A_1]\, b_1$, by rule Heap-CaseBeta1 (assuming x_1 *fresh*).
  * $\Gamma_2, x_1 :^q A_1 \vdash b_1 : B$, from premises.
  * Next, since $H \Vdash \Gamma_0 + r \cdot (q \cdot \Gamma_1 + \Gamma_2)$ and $\Gamma_1 \vdash a_1 : A_1$, by the cons-rule, $H, x_1 \overset{r \cdot q}{\mapsto} (\Gamma_1 | a_1 | A_1) \Vdash (\Gamma_0 + r \cdot \Gamma_2), x_1 :^{r \cdot q} A_1$.
    In other words, $H, x_1 \overset{r \cdot q}{\mapsto} (\Gamma_1 | a_1 | A_1) \Vdash (\Gamma_0, x_1 :^0 A_1) + r \cdot (\Gamma_2, x_1 :^q A_1)$.
  * Next, need to show: $r \cdot ((q \cdot \overline{\Gamma_1} + \overline{\Gamma_2}) \diamond 0) + \mathbf{0} \diamond r \cdot q <: r \cdot (\overline{\Gamma_2} \diamond q) + (\mathbf{0} \diamond r \cdot q) \times \begin{pmatrix} \langle H \rangle & \mathbf{0}^\top \\ \overline{\Gamma_1} & 0 \end{pmatrix}$.
    This clause follows by reflexivity.

* Rule GLC-Sub. Have: $\Gamma_2 \vdash a : A$ where $\Gamma_1 \vdash a : A$ and $\Gamma_2 <: \Gamma_1$.
  Further, $H \Vdash \Gamma_0 + r \cdot \Gamma_2$ where $r <: 1$.
  Since $\Gamma_2 <: \Gamma_1$, so $\Gamma_0 + r \cdot \Gamma_2 <: \Gamma_0 + r \cdot \Gamma_1$.
  Then, by Lemma C.13, there exists H_1 such that $H <: H_1$ and $H_1 \Vdash \Gamma_0 + r \cdot \Gamma_1$.
  Now, if a is a value, this case follows immediately.
  Otherwise, since $H_1 \Vdash \Gamma_0 + r \cdot \Gamma_1$ and $\Gamma_1 \vdash a : A$, by IH, $\exists H'$, $\mathbf{u'}$, Γ'_0 and Γ'_1 such that:
  - $[H_1]\, a \Rightarrow_S^r [H' ; \mathbf{u'} ; \Gamma'_0]\, a'$.
  - $\Gamma'_1 \vdash a' : A$
  - $H' \Vdash (\Gamma_0, 0 \cdot \Gamma'_0) + r \cdot \Gamma'_1$
  - $r \cdot (\overline{\Gamma_1} \diamond \mathbf{0}) + \mathbf{u'} \times \langle H' \rangle + \mathbf{0} \diamond \overline{\Gamma'_0} <: r \cdot \overline{\Gamma'_1} + \mathbf{u'} + (\mathbf{0} \diamond \overline{\Gamma'_0}) \times \langle H' \rangle$

  Using the above clauses, we show the corresponding clauses of the theorem:
  - From the first clause, by rule Heap-SubL, $[H]\, a \Rightarrow_S^r [H' ; \mathbf{u'} ; \Gamma'_0]\, a'$.
  - $\Gamma'_1 \vdash a' : A$, same as the second clause.
  - $H' \Vdash (\Gamma_0, 0 \cdot \Gamma'_0) + r \cdot \Gamma'_1$, same as the third clause.

– Need to show: $r \cdot (\overline{\Gamma_2} \diamond \mathbf{0}) + \mathbf{u'} \times \langle H' \rangle + \mathbf{0} \diamond \overline{\Gamma'_0} <: r \cdot \overline{\Gamma'_1} + \mathbf{u'} + (\mathbf{0} \diamond \overline{\Gamma'_0}) \times \langle H' \rangle$
Follows from the fourth clause and the fact that $\Gamma_2 <: \Gamma_1$.

□

Theorem C.15 (Theorem 4.10) If $H \Vdash \Gamma$ and $\Gamma \vdash a : A$, then either a is a value or there exists H', $\mathbf{u'}$, Γ'_0 and Γ' such that:

- $[H]\, a \Rightarrow^{1}_{S} [H' ; \mathbf{u'} ; \Gamma'_0]\, a'$
- $\Gamma' \vdash a' : A$
- $H' \Vdash \Gamma'$
- $\overline{\Gamma} \diamond \mathbf{0} \; + \; \mathbf{u'} \times \langle H' \rangle \; + \; \mathbf{0} \diamond \overline{\Gamma'_0} \;\; <: \;\; \overline{\Gamma'} \; + \; \mathbf{u'} \; + \; (\mathbf{0} \diamond \overline{\Gamma'_0}) \times \langle H' \rangle$

Proof. Follows from Theorem C.14 by setting $r \coloneqq 1$ and $\Gamma_0 \coloneqq 0 \cdot \Gamma$. □

C.4 Proofs of Lemmas/Theorems Stated in Section 4.4

Lemma C.16 (Unused) If $[H]\, a \Rightarrow^{r}_{S} [H' ; \mathbf{u'} ; \Gamma'_0]\, a'$ and $x_i \overset{q_0}{\mapsto} (\Gamma_i | a_i | A_i) \in H$ such that $\neg(\exists q, q_0 <: q + r)$, then $\mathbf{u'}(x_i) = 0$ and $x_i \overset{q'_0}{\mapsto} (\Gamma_i | a_i | A_i) \in H'$, where $q_0 <: q'_0$.

Proof. By induction on $[H]\, a \Rightarrow^{r}_{S} [H' ; \mathbf{u'} ; \Gamma'_0]\, a'$. We present some of the interesting cases below.

- Rule Heap-Var.
  Have: $[H_1, x \overset{(q+r)}{\mapsto} (\Gamma | a | A), H_2]\, x \Rightarrow^{r}_{S} [H_1, x \overset{q}{\mapsto} (\Gamma | a | A), H_2 ; \mathbf{0}^{|H_1|} \diamond r \diamond \mathbf{0}^{|H_2|} ; \varnothing]\, a$ where $r <: 1$. Further, $x_i \overset{q_0}{\mapsto} (\Gamma_i | a_i | A_i) \in H_1, x \overset{(q+r)}{\mapsto} (\Gamma | a | A), H_2$.
  Now, if $x_i = x$, then $q_0 = q + r$, a contradiction. So, $x_i \neq x$.
  The case follows.

- Rule Heap-AppL. Have: $[H]\, b\ a^{q} \Rightarrow^{r}_{S} [H' ; \mathbf{u'} ; \Gamma'_0]\, b'\ a^{q}$ where $[H]\, b \Rightarrow^{r}_{S \cup \mathrm{fv}\, a} [H' ; \mathbf{u'} ; \Gamma'_0]\, b'$.
  Further, $x_i \overset{q_0}{\mapsto} (\Gamma_i | a_i | A_i) \in H$.
  Need to show: $\mathbf{u'}(x_i) = 0$ and $x_i \overset{q'_0}{\mapsto} (\Gamma_i | a_i | A_i) \in H'$, where $q_0 <: q'_0$.
  Follows by IH.

- Rule Heap-AppBeta. Have: $[H]\, (\lambda^{q} y \colon A_0 . b)\ a^{q} \Rightarrow^{r}_{S} [H, y \overset{r \cdot q}{\mapsto} (\Gamma | a | A) ; \mathbf{0} ; y \colon^{r \cdot q} A]\, b$ (assuming y *fresh*).
  Further, $x_i \overset{q_0}{\mapsto} (\Gamma_i | a_i | A_i) \in H$.
  Therefore, $x_i \overset{q_0}{\mapsto} (\Gamma_i | a_i | A_i) \in H, y \overset{r \cdot q}{\mapsto} (\Gamma | a | A)$.
  This case follows.

- Rule Heap-SubL. Have: $[H_2]\, a \Rightarrow^r_S [H' \,;\, \mathbf{u'} \,;\, \Gamma'_0]\, a'$ where $[H_1]\, a \Rightarrow^r_S [H' \,;\, \mathbf{u'} \,;\, \Gamma'_0]\, a'$ and $H_2 <: H_1$.
  Further, $x_i \overset{q_0}{\mapsto} (\Gamma_i|a_i|A_i) \in H_2$ and $\neg(\exists q, q_0 <: q + r)$.
  Need to show: $\mathbf{u'}(x_i) = 0$ and $x_i \overset{q'_0}{\mapsto} (\Gamma_i|a_i|A_i) \in H'$, where $q_0 <: q'_0$.
  Since $x_i \overset{q_0}{\mapsto} (\Gamma_i|a_i|A_i) \in H_2$, so $x_i \overset{q_1}{\mapsto} (\Gamma_i|a_i|A_i) \in H_1$, for some q_1, where $q_0 <: q_1$.
  Now, say $\exists q$ such that $q_1 <: q + r$. Then, $q_0 <: q + r$, a contradiction.
  Therefore, $\neg(\exists q, q_1 <: q + r)$.
  Then, by IH, $\mathbf{u'}(x_i) = 0$ and $x_i \overset{q'_0}{\mapsto} (\Gamma_i|a_i|A_i) \in H'$ where $q_1 <: q'_0$.
  By transitivity, $q_0 <: q'_0$.

- Rule Heap-SubR. Have: $[H]\, a \Rightarrow^{r_2}_S [H' \,;\, \mathbf{u'} \,;\, \Gamma'_0]\, a'$ where $[H]\, a \Rightarrow^{r_1}_S [H' \,;\, \mathbf{u'} \,;\, \Gamma'_0]\, a'$ and $r_1 <: r_2$.
  Further, $x_i \overset{q_0}{\mapsto} (\Gamma_i|a_i|A_i) \in H$ and $\neg(\exists q, q_0 <: q + r_2)$.
  Need to show: $\mathbf{u'}(x_i) = 0$ and $x_i \overset{q'_0}{\mapsto} (\Gamma_i|a_i|A_i) \in H'$, where $q_0 <: q'_0$.
  Say, $\exists q$ such that $q_0 <: q + r_1$. But then, $q + r_1 <: q + r_2$. So, $q_0 <: q + r_2$. A contradiction.
  Therefore, $\neg(\exists q, q_0 <: q + r_1)$.
  This case, then, follows by IH.

□

Lemma C.17 (Lemma 4.11) Let $\mathcal{Q}$ be a zero-unusable semiring. In GLC($\mathcal{Q}$), if $[H]\, a \Rightarrow^1_S [H' \,;\, \mathbf{u'} \,;\, \Gamma'_0]\, a'$ and $x_i \overset{0}{\mapsto} (\Gamma_i|a_i|A_i) \in H$, then $\mathbf{u'}(x_i) = 0$ and $x_i \overset{0}{\mapsto} (\Gamma_i|a_i|A_i) \in H'$.

Proof. Since $\mathcal{Q}$ is zero-unusable, the inequation $0 <: q + 1$ has no solution. Then, setting $r := 1$ and $q_0 := 0$ in Lemma C.16, we have, $\mathbf{u'}(x_i) = 0$ and $x_i \overset{q'_0}{\mapsto} (\Gamma_i|a_i|A_i) \in H'$, where $0 <: q'_0$. But since 0 is maximal with respect to the order relation, $q'_0 = 0$. The lemma follows. □

Lemma C.18 (Irrelevant) Let $H_i = x \overset{q_i}{\mapsto} (\Gamma_i|a_i|A_i)$ and $H_j = x \overset{q_j}{\mapsto} (\Gamma_j|a_j|A_j)$. Suppose, $\neg(\exists q, q_i <: q + r)$ and $\neg(\exists q, q_j <: q + r)$. If $[H_1, H_i, H_2]\, b \Rightarrow^r_{S \cup \mathrm{fv}\, a_j} [H'_1, H'_i, H'_2 \,;\, \mathbf{u'} \,;\, \Gamma'_0]\, b'$ where $\lfloor H'_1 \rfloor = \lfloor H_1 \rfloor$ and $\lfloor H'_i \rfloor = \lfloor H_i \rfloor$, then $[H_1, H_j, H_2]\, b \Rightarrow^r_{S \cup \mathrm{fv}\, a_i} [H'_1, H'_j, H'_2 \,;\, \mathbf{u'} \,;\, \Gamma'_0]\, b'$, where $\lfloor H'_j \rfloor = \lfloor H_j \rfloor$.

Proof. By induction on $[H_1, H_i, H_2]\, b \Rightarrow^r_{S \cup \mathrm{fv}\, a_j} [H'_1, H'_i, H'_2 \,;\, \mathbf{u'} \,;\, \Gamma'_0]\, b'$. We present some of the interesting cases below.

- Rule Heap-Var. There are three cases to consider.
  - Variable being looked-up is in H_1.
    Have: $[H_{11}, y \overset{(q+r)}{\mapsto} (\Gamma|b|B), H_{12}, H_i, H_2]\, y \Rightarrow^r_{S \cup \mathrm{fv}\, a_j} [H_{11}, y \overset{q}{\mapsto} (\Gamma|b|B), H_{12}, H_i, H_2]\, b$.
    Need to show:
    $[H_{11}, y \overset{(q+r)}{\mapsto} (\Gamma|b|B), H_{12}, H_j, H_2]\, y \Rightarrow^r_{S \cup \mathrm{fv}\, a_i} [H_{11}, y \overset{q}{\mapsto} (\Gamma|b|B), H_{12}, H'_j, H_2]\, b$
    Follows by rule Heap-Var.
  - The case when the variable being looked-up is in H_2 is similar.

– The third case, when the variable being looked-up is the one in H_i, leads to contradiction because $\neg(\exists q, q_i <: q + r)$.

- Rule Heap-AppL. Have: $[H_1, H_i, H_2]\, b\ a^q \Rightarrow^r_{S \cup \mathrm{fv}\, a_j} [H'_1, H'_i, H'_2\,;\, \mathbf{u'}\,;\, \Gamma'_0]\, b'\ a^q$ where
  $[H_1, H_i, H_2]\, b \Rightarrow^r_{S \cup \mathrm{fv}\, a_j \cup \mathrm{fv}\, a} [H'_1, H'_i, H'_2\,;\, \mathbf{u'}\,;\, \Gamma'_0]\, b'$.
  Need to show: $[H_1, H_j, H_2]\, b\ a^q \Rightarrow^r_{S \cup \mathrm{fv}\, a_i} [H'_1, H'_j, H'_2\,;\, \mathbf{u'}\,;\, \Gamma'_0]\, b'\ a^q$.
  Follows by IH and rule Heap-AppL.

- Rule Heap-AppBeta.
  Have: $[H_1, H_i, H_2]\, (\lambda^q y\!:\!A_0.b)\ a^q \Rightarrow^r_{S \cup \mathrm{fv}\, a_j} [H_1, H_i, H_2, y \overset{r\cdot q}{\mapsto} (\Gamma|a|A)\,;\, \mathbf{0}\,;\, y\!:^{r\cdot q} A]\, b$ (assuming y *fresh*).
  Need to show: $[H_1, H_j, H_2]\, (\lambda^q y\!:\!A_0.b)\ a^q \Rightarrow^r_{S \cup \mathrm{fv}\, a_i} [H_1, H'_j, H_2, y \overset{r\cdot q}{\mapsto} (\Gamma|a|A)\,;\, \mathbf{0}\,;\, y\!:^{r\cdot q} A]\, b$.
  Follows by rule Heap-AppBeta.

- Rule Heap-SubL. Have: $[H_{21}, H_i, H_{22}]\, b \Rightarrow^r_{S \cup \mathrm{fv}\, a_j} [H'_1, H'_i, H'_2\,;\, \mathbf{u'}\,;\, \Gamma'_0]\, b'$ where
  $[H_{11}, H_0, H_{12}]\, b \Rightarrow^r_{S \cup \mathrm{fv}\, a_j} [H'_1, H'_i, H'_2\,;\, \mathbf{u'}\,;\, \Gamma'_0]\, b'$ and $H_{21} <: H_{11}$ and $H_{22} <: H_{12}$ and $H_0 = x \overset{q_0}{\mapsto} (\Gamma_i|a_i|A_i)$ where $q_i <: q_0$.
  Further, $\neg(\exists q, q_i <: q + r)$ and $\neg(\exists q, q_j <: q + r)$.
  Need to show: $[H_{21}, H_j, H_{22}]\, b \Rightarrow^r_{S \cup \mathrm{fv}\, a_i} [H'_1, H'_j, H'_2\,;\, \mathbf{u'}\,;\, \Gamma'_0]\, b'$.
  Since $q_i <: q_0$, so, $\neg(\exists q, q_0 <: q + r)$.
  By IH, $[H_{11}, H_j, H_{12}]\, b \Rightarrow^r_{S \cup \mathrm{fv}\, a_i} [H'_1, H'_j, H'_2\,;\, \mathbf{u'}\,;\, \Gamma'_0]\, b'$.
  This case, then, follows by rule Heap-SubL.

- Rule Heap-SubR. Have: $[H_1, H_i, H_2]\, b \Rightarrow^{r_2}_{S \cup \mathrm{fv}\, a_j} [H'_1, H'_i, H'_2\,;\, \mathbf{u'}\,;\, \Gamma'_0]\, b'$ where
  $[H_1, H_i, H_2]\, b \Rightarrow^{r_1}_{S \cup \mathrm{fv}\, a_j} [H'_1, H'_i, H'_2\,;\, \mathbf{u'}\,;\, \Gamma'_0]\, b'$ and $r_1 <: r_2$.
  Further, $\neg(\exists q, q_i <: q + r_2)$ and $\neg(\exists q, q_j <: q + r_2)$.
  Need to show: $[H_1, H_j, H_2]\, b \Rightarrow^{r_2}_{S \cup \mathrm{fv}\, a_i} [H'_1, H'_j, H'_2\,;\, \mathbf{u'}\,;\, \Gamma'_0]\, b'$
  Since $r_1 <: r_2$, so $\neg(\exists q, q_i <: q + r_1)$ and $\neg(\exists q, q_j <: q + r_1)$.
  By IH, $[H_1, H_j, H_2]\, b \Rightarrow^{r_1}_{S \cup \mathrm{fv}\, a_i} [H'_1, H'_j, H'_2\,;\, \mathbf{u'}\,;\, \Gamma'_0]\, b'$.
  This case, then, follows by rule Heap-SubR.

□

Lemma C.19 (Lemma 4.12) Let $\mathcal{Q}$ be a zero-unusable semiring. Next, let $H_i = x \overset{0}{\mapsto} (\Gamma_i|a_i|A_i)$ and $H_j = x \overset{0}{\mapsto} (\Gamma_j|a_j|A_j)$. Then, in GLC($\mathcal{Q}$), if $[H_1, H_i, H_2]\, b \Rightarrow^1_{S \cup \mathrm{fv}\, a_j} [H'_1, H_i, H'_2\,;\, \mathbf{u'}\,;\, \Gamma'_0]\, b'$, then $[H_1, H_j, H_2]\, b \Rightarrow^1_{S \cup \mathrm{fv}\, a_i} [H'_1, H_j, H'_2\,;\, \mathbf{u'}\,;\, \Gamma'_0]\, b'$.

Proof. Since $\mathcal{Q}$ is zero-unusable, the inequation $0 <: q + 1$ has no solution. Then, setting $r := 1$ and $q_i := 0$ and $q_j := 0$ in Lemma C.18, we have, $[H_1, H_j, H_2]\, b \Rightarrow^1_{S \cup \mathrm{fv}\, a_i} [H'_1, H'_j, H'_2\,;\, \mathbf{u'}\,;\, \Gamma'_0]\, b'$. But by Lemma C.17, $H'_j = H_j$. The lemma follows. □

Lemma C.20 (Lemma 4.13) Let $\mathcal{Q}$ be a zero-unusable semiring. Let $\varnothing \vdash f : {}^0A \to B$ and $\varnothing \vdash a_1 : A$ and $\varnothing \vdash a_2 : A$ in GLC($\mathcal{Q}$). Then, $f\ {a_1}^0$ and $f\ {a_2}^0$ have the same operational behavior.

Proof. To see why, we consider the reduction of $f\ {a_1}^0$ and $f\ {a_2}^0$.

Let $[H]\,a \Rrightarrow^q_k [H']\,a'$ denote a reduction of k steps where H and a are the initial heap and term, H' and a' are the final heap and term, and q is the grade at which the reduction takes place.

For some k_0, H and b, we have, $[\varnothing] f\ {a_1}^0 \Rrightarrow^1_{k_0} [H]\,(\lambda^0 x.b)\ {a_1}^0$ and $[\varnothing] f\ {a_2}^0 \Rrightarrow^1_{k_0} [H]\,(\lambda^0 x.b)\ {a_2}^0$.

Then, $[H]\,(\lambda^0 x.b)\ {a_1}^0 \Rightarrow [H, x \overset{0}{\mapsto} a_1]\,b$ and $[H]\,(\lambda^0 x.b)\ {a_2}^0 \Rightarrow [H, x \overset{0}{\mapsto} a_2]\,b$ (assuming x *fresh*).

But, if $[H, x \overset{0}{\mapsto} a_1]\,b \Rrightarrow^1_k [H', x \overset{0}{\mapsto} a_1, H'']\,b'$, then $[H, x \overset{0}{\mapsto} a_2]\,b \Rrightarrow^1_k [H', x \overset{0}{\mapsto} a_2, H'']\,b'$, for any k (By Lemma C.19).

Therefore, $f\ {a_1}^0$ and $f\ {a_2}^0$ have the same operational behavior. $\square$

Lemma C.21 (Lemma 4.14) Let $\mathcal{Q}$ be a one-linear semiring. In GLC($\mathcal{Q}$), if $H \Vdash \Gamma$ and $x_i \overset{1}{\mapsto} (\Gamma_i | a_i | A_i) \in H$, then in the graph $G_{H,\Gamma}$, there is exactly one path from the ground node to v_i and further, all the weights on that path are 1.

Proof. Without loss of generality, we can assume that x_i appears at the ith position in H.
Now, since $H \Vdash \Gamma$, so $\overline{H} = \overline{H} \times \langle H \rangle + \overline{\Gamma}$, by Lemma C.10.
Let $\langle H \rangle(:, i)$ denote the ith column of the matrix $\langle H \rangle$.
Then, $\overline{H} \times \langle H \rangle(:, i) + \overline{\Gamma}(i) = 1$.
Now, since $\mathcal{Q}$ is one-linear, so

- either $\overline{\Gamma}(i) = 1$, in which case $\langle H \rangle(i, j) = 0$, for all j;
- or $\overline{\Gamma}(i) = 0$, in which case there exists an index $j_0 > i$ such that $\overline{H}(j_0) = 1$ and $\langle H \rangle(j_0, i) = 1$ and $\langle H \rangle(j, i) = 0$, for all $j \neq j_0$.

In the former case, there is exactly one path from the ground node to v_i; that path is just a single edge of weight 1.
In the latter case, there is only one incoming edge to v_i; that edge has weight 1 and is from v_{j_0}. Further, observe that the allowed usage of the assignment corresponding to v_{j_0} is also 1. So, we can repeat the same argument with index j_0, and then over and over again until we reach an index j_n for which $\overline{\Gamma}(j_n) = 1$. We are guaranteed to reach such an index because in every iteration, the newly discovered index is numerically greater than the current index. This chain of indices, then, give us the required path from the ground node to v_i. $\square$

C.5 Proofs of Lemmas/Theorems Stated in Section 4.5

Lemma C.22 (Lemma 4.15) If $\Gamma \vdash a : A$ in GLC($\mathbb{B}_{\geq}$), then $\vec{\Gamma} \vdash \overline{a} :^1 \overline{A}$ in SDC($\mathbb{B}_{\geq}$). Further, if $\vdash a \rightsquigarrow a'$ in GLC($\mathbb{B}_{\geq}$), then $\vdash \overline{a} \rightsquigarrow^* \overline{a'}$ in SDC($\mathbb{B}_{\geq}$).

Proof. First, let $\Gamma \vdash a : A$ in GLC($\mathbb{B}_{\geq}$). By induction on $\Gamma \vdash a : A$, we show that $\vec{\Gamma} \vdash \overline{a} :^1 \overline{A}$ in SDC($\mathbb{B}_{\geq}$).

- Rule GLC-Var. Have: $0 \cdot \Gamma, x :^1 A \vdash x : A$.
  Need to show: $0 \sqcup \vec{\Gamma}, x :^1 \overline{A} \vdash x :^1 \overline{A}$. (Note that $q \sqcup \vec{\Gamma}$ denotes the point-wise join of q with the grades on the assumptions in $\vec{\Gamma}$.)
  Follows by rule SDC-Var.

- Rule GLC-Weak. Have: $\Gamma, y :^0 B \vdash a : A$ where $\Gamma \vdash a : A$.
  Need to show: $\vec{\Gamma}, y :^0 \overline{B} \vdash \overline{a} :^1 \overline{A}$.
  By IH, $\vec{\Gamma} \vdash \overline{a} :^1 \overline{A}$.
  This case, then, follows by weakening lemma 3.5.

- Rule GLC-Unit. Have: $\varnothing \vdash \mathbf{unit} : \mathbf{Unit}$.
  Need to show: $\varnothing \vdash \mathbf{unit} :^1 \mathbf{Unit}$.
  Follows by rule SDC-Unit.

- Rule GLC-LetUnit. Have: $\Gamma_1 + \Gamma_2 \vdash \mathbf{let\,unit} = a \ \mathbf{in} \ b : B$ where $\Gamma_1 \vdash a : \mathbf{Unit}$ and $\Gamma_2 \vdash b : B$.
  Need to show: $\vec{\Gamma}_1 \sqcap \vec{\Gamma}_2 \vdash \mathbf{let\,unit} = \overline{a} \ \mathbf{in} \ \overline{b} :^1 \overline{B}$. (Note that $\vec{\Gamma}_1 \sqcap \vec{\Gamma}_2$ denotes the point-wise meet of the grades on the assumptions in $\vec{\Gamma}_1$ and $\vec{\Gamma}_2$.)
  By IH, $\vec{\Gamma}_1 \vdash \overline{a} :^1 \mathbf{Unit}$ and $\vec{\Gamma}_2 \vdash \overline{b} :^1 \overline{B}$.
  By narrowing lemma 3.1, $\vec{\Gamma}_1 \sqcap \vec{\Gamma}_2 \vdash \overline{a} :^1 \mathbf{Unit}$ and $\vec{\Gamma}_1 \sqcap \vec{\Gamma}_2 \vdash \overline{b} :^1 \overline{B}$.
  This case, then, follows by rule SDC-LetUnit.

- Rule GLC-Lam. Have: $\Gamma \vdash \lambda^q x{:}A.b : {}^q A \to B$ where $\Gamma, x :^q A \vdash b : B$.
  Need to show: $\vec{\Gamma} \vdash \lambda y{:}S_q\,\overline{A}.\mathbf{unlock}^q\ x = y\,\mathbf{in}\,\overline{b} :^1 S_q\,\overline{A} \to \overline{B}$, where y is fresh.
  Now, by rule SDC-Var, $\vec{\Gamma}, y :^1 S_q\,\overline{A} \vdash y :^1 S_q\,\overline{A}$.
  By IH, $\vec{\Gamma}, x :^q \overline{A} \vdash \overline{b} :^1 \overline{B}$.
  By weakening lemma 3.5, $\vec{\Gamma}, y :^1 S_q\,\overline{A}, x :^q \overline{A} \vdash \overline{b} :^1 \overline{B}$.
  So, by rule SDC-Unlock, $\vec{\Gamma}, y :^1 S_q\,\overline{A} \vdash \mathbf{unlock}^q\ x = y\,\mathbf{in}\,\overline{b} :^1 \overline{B}$.
  This case, then, follows by rule SDC-Lam.

- Rule GLC-App. Have: $\Gamma_1 + q \cdot \Gamma_2 \vdash b\ a^q : B$ where $\Gamma_1 \vdash b : {}^q A \to B$ and $\Gamma_2 \vdash a : A$.
  Need to show: $\vec{\Gamma}_1 \sqcap (q \sqcup \vec{\Gamma}_2) \vdash \overline{b}\,(\mathbf{lock}^q\ \overline{a}) :^1 \overline{B}$.
  By IH, $\vec{\Gamma}_1 \vdash \overline{b} :^1 S_q\,\overline{A} \to \overline{B}$ and $\vec{\Gamma}_2 \vdash \overline{a} :^1 \overline{A}$.
  Then, by multiplication lemma 3.3, $q \sqcup \vec{\Gamma}_2 \vdash \overline{a} :^q \overline{A}$.
  By rule SDC-Lock, $q \sqcup \vec{\Gamma}_2 \vdash \mathbf{lock}^q\ \overline{a} :^1 S_q\,\overline{A}$.
  By narrowing lemma 3.1, $\vec{\Gamma}_1 \sqcap (q \sqcup \vec{\Gamma}_2) \vdash \mathbf{lock}^q\ \overline{a} :^1 S_q\,\overline{A}$.
  Again by the narrowing lemma, $\vec{\Gamma}_1 \sqcap (q \sqcup \vec{\Gamma}_2) \vdash \overline{b} :^1 S_q\,\overline{A} \to \overline{B}$.
  This case, then, follows by rule SDC-App.

- Rule GLC-Box. Have: $q \cdot \Gamma \vdash \mathbf{box}_q\ a : \Box^q A$ where $\Gamma \vdash a : A$.
  Need to show: $q \sqcup \vec{\Gamma} \vdash \mathbf{lock}^q\ \overline{a} :^1 S_q\,\overline{A}$.

By IH, $\vec{\Gamma} \vdash \overline{a} :^1 \overline{A}$.

By multiplication lemma 3.3, $q \sqcup \vec{\Gamma} \vdash \overline{a} :^q \overline{A}$.

This case, then, follows by rule SDC-Lock.

- Rule GLC-LetBox. Have: $\Gamma_1 + \Gamma_2 \vdash \textbf{let box}_q\, x = a \textbf{ in } b : B$ where $\Gamma_1 \vdash a : \Box^q A$ and $\Gamma_2, x :^q A \vdash b : B$.

  Need to show: $\vec{\Gamma_1} \sqcap \vec{\Gamma_2} \vdash \textbf{unlock}^q\, x = \overline{a}\, \textbf{in}\, \overline{b} :^1 \overline{B}$.

  By IH, $\vec{\Gamma_1} \vdash \overline{a} :^1 S_q\, \overline{A}$ and $\vec{\Gamma_2}, x :^q \overline{A} \vdash \overline{b} :^1 \overline{B}$.

  By narrowing lemma 3.1, $\vec{\Gamma_1} \sqcap \vec{\Gamma_2} \vdash \overline{a} :^1 S_q\, \overline{A}$ and $\vec{\Gamma_1} \sqcap \vec{\Gamma_2}, x :^q \overline{A} \vdash \overline{b} :^1 \overline{B}$.

  This case, then, follows by rule SDC-Unlock.

- Rule GLC-Pair. Have: $q \cdot \Gamma_1 + \Gamma_2 \vdash (a_1{}^q, a_2) : {}^q A_1 \times A_2$ where $\Gamma_1 \vdash a_1 : A_1$ and $\Gamma_2 \vdash a_2 : A_2$.

  Need to show: $(q \sqcup \vec{\Gamma_1}) \sqcap \vec{\Gamma_2} \vdash (\textbf{lock}^q\, \overline{a_1}, \overline{a_2}) :^1 S_q\, \overline{A_1} \times \overline{A_2}$.

  By IH, $\vec{\Gamma_1} \vdash \overline{a_1} :^1 \overline{A_1}$ and $\vec{\Gamma_2} \vdash \overline{a_2} :^1 \overline{A_2}$.

  By multiplication lemma 3.3, $q \sqcup \vec{\Gamma_1} \vdash \overline{a_1} :^q \overline{A_1}$.

  By rule SDC-Lock, $q \sqcup \vec{\Gamma_1} \vdash \textbf{lock}^q\, \overline{a_1} :^1 S_q\, \overline{A_1}$.

  By narrowing lemma 3.1, $(q \sqcup \vec{\Gamma_1}) \sqcap \vec{\Gamma_2} \vdash \textbf{lock}^q\, \overline{a_1} :^1 S_q\, \overline{A_1}$.

  Again by the narrowing lemma, $(q \sqcup \vec{\Gamma_1}) \sqcap \vec{\Gamma_2} \vdash \overline{a_2} :^1 \overline{A_2}$.

  This case, then, follows by rule SDC-Pair.

- Rule GLC-LetPair. Have: $\Gamma_1 + \Gamma_2 \vdash \textbf{let}\, (x_1{}^q, x_2) = a \textbf{ in } b : B$ where $\Gamma_1 \vdash a : {}^q A_1 \times A_2$ and $\Gamma_2, x_1 :^q A_1, x_2 :^1 A_2 \vdash b : B$.

  Need to show: $\vec{\Gamma_1} \sqcap \vec{\Gamma_2} \vdash \textbf{let}\, (x_1', x_2) = \overline{a} \textbf{ in } \textbf{unlock}^q\, x_1 = x_1'\, \textbf{in}\, \overline{b} :^1 \overline{B}$, where x_1' is fresh.

  By IH, $\vec{\Gamma_1} \vdash \overline{a} :^1 S_q\, \overline{A_1} \times \overline{A_2}$ and $\vec{\Gamma_2}, x_1 :^q \overline{A_1}, x_2 :^1 \overline{A_2} \vdash \overline{b} :^1 \overline{B}$.

  By exchange, $\vec{\Gamma_2}, x_2 :^1 \overline{A_2}, x_1 :^q \overline{A_1} \vdash \overline{b} :^1 \overline{B}$.

  By weakening, $\vec{\Gamma_2}, x_2 :^1 \overline{A_2}, x_1' :^1 S_q\, \overline{A_1}, x_1 :^q \overline{A_1} \vdash \overline{b} :^1 \overline{B}$.

  By rule SDC-Var, $\vec{\Gamma_2}, x_2 :^1 \overline{A_2}, x_1' :^1 S_q\, \overline{A_1} \vdash x_1' :^1 S_q\, \overline{A_1}$.

  Then, by rule SDC-Unlock, $\vec{\Gamma_2}, x_2 :^1 \overline{A_2}, x_1' :^1 S_q\, \overline{A_1} \vdash \textbf{unlock}^q\, x_1 = x_1'\, \textbf{in}\, \overline{b} :^1 \overline{B}$.

  By exchange, $\vec{\Gamma_2}, x_1' :^1 S_q\, \overline{A_1}, x_2 :^1 \overline{A_2} \vdash \textbf{unlock}^q\, x_1 = x_1'\, \textbf{in}\, \overline{b} :^1 \overline{B}$.

  This case, then, follows by rule SDC-LetPair.

- Rule GLC-Inj1. Have: $\Gamma \vdash \textbf{inj}_1\, a_1 : A_1 + A_2$ where $\Gamma \vdash a_1 : A_1$.

  Need to show: $\vec{\Gamma} \vdash \textbf{inj}_1\, \overline{a_1} :^1 \overline{A_1} + \overline{A_2}$.

  By IH, $\vec{\Gamma} \vdash \overline{a_1} :^1 \overline{A_1}$.

  This case follows by rule SDC-Inj.

- Rule GLC-Case. Have: $q \cdot \Gamma_1 + \Gamma_2 \vdash \textbf{case}_q\, a\, \textbf{of}\, x_1.b_1; x_2.b_2 : B$ where $\Gamma_1 \vdash a : A_1 + A_2$ and $\Gamma_2, x_1 :^q A_1 \vdash b_1 : B$ and $\Gamma_2, x_2 :^q A_2 \vdash b_2 : B$ and $q <: 1$.

  Now, $q <: 1$ in $\mathbb{B}_{\geq}$ implies $q = 1$.

  As such, $\Gamma_1 + \Gamma_2 \vdash \textbf{case}_1\, a\, \textbf{of}\, x_1.b_1; x_2.b_2 : B$.

  Then, we need to show: $\vec{\Gamma_1} \sqcap \vec{\Gamma_2} \vdash \textbf{case}\, \overline{a}\, \textbf{of}\, x_1.\overline{b_1}; x_2.\overline{b_2} :^1 \overline{B}$.

  By IH, $\vec{\Gamma_1} \vdash \overline{a} :^1 \overline{A_1} + \overline{A_2}$ and $\vec{\Gamma_2}, x_1 :^1 \overline{A_1} \vdash \overline{b_1} :^1 \overline{B}$ and $\vec{\Gamma_2}, x_2 :^1 \overline{A_2} \vdash \overline{b_2} :^1 \overline{B}$.

  By the narrowing lemma, $\vec{\Gamma_1} \sqcap \vec{\Gamma_2} \vdash \overline{a} :^1 \overline{A_1} + \overline{A_2}$.

Again, by the same lemma, $\vec{\overline{\Gamma_1}} \sqcap \vec{\overline{\Gamma_2}}, x_1 :^1 \overline{A_1} \vdash \overline{b_1} :^1 \overline{B}$ and $\vec{\overline{\Gamma_1}} \sqcap \vec{\overline{\Gamma_2}}, x_2 :^1 \overline{A_2} \vdash \overline{b_2} :^1 \overline{B}$.
This case, then, follows by rule SDC-Case.

- Rule GLC-Sub. Have: $\Gamma_2 \vdash a : A$ where $\Gamma_1 \vdash a : A$ and $\Gamma_2 <: \Gamma_1$.
  Need to show: $\vec{\overline{\Gamma_2}} \vdash \overline{a} :^1 \overline{A}$.
  By IH, $\vec{\overline{\Gamma_1}} \vdash \overline{a} :^1 \overline{A}$.
  Now, since $\Gamma_2 <: \Gamma_1$, so $\vec{\overline{\Gamma_2}} \sqsubseteq \vec{\overline{\Gamma_1}}$.
  This case, then, follows by the narrowing lemma.

Next, let $\vdash a \rightsquigarrow a'$ in GLC($\mathbb{B}_{\geq}$). By induction on $\vdash a \rightsquigarrow a'$, we show that $\vdash \overline{a} \rightsquigarrow^* \overline{a'}$ in SDC($\mathbb{B}_{\geq}$). Below, we present some of the interesting cases of the induction.

- Rule S-AppBeta. Have: $\vdash (\lambda^q x : A.b)\ a^q \rightsquigarrow b\{a/x\}$.
  Need to show: $\vdash (\lambda y : S_q\, \overline{A}.\mathbf{unlock}^q\ x = y\ \mathbf{in}\ \overline{b})\ (\mathbf{lock}^q\ \overline{a}) \rightsquigarrow^* \overline{b}\{\overline{a}/x\}$, where y is fresh.
  By β-rule, $\vdash (\lambda y : S_q\, \overline{A}.\mathbf{unlock}^q\ x = y\ \mathbf{in}\ \overline{b})\ (\mathbf{lock}^q\ \overline{a}) \rightsquigarrow \mathbf{unlock}^q\ x = \mathbf{lock}^q\ \overline{a}\ \mathbf{in}\ \overline{b}$.
  Again, by β-rule, $\vdash \mathbf{unlock}^q\ x = \mathbf{lock}^q\ \overline{a}\ \mathbf{in}\ \overline{b} \rightsquigarrow \overline{b}\{\overline{a}/x\}$.
  The case follows.

- Rule S-LetBoxBeta. Have: $\vdash \mathbf{let}\ \mathbf{box}_q\ x = \mathbf{box}_q\ a\ \mathbf{in}\ b \rightsquigarrow b\{a/x\}$.
  Need to show: $\vdash \mathbf{unlock}^q\ x = \mathbf{lock}^q\ \overline{a}\ \mathbf{in}\ \overline{b} \rightsquigarrow^* \overline{b}\{\overline{a}/x\}$.
  Follows by β-rule for unlock.

- Rule S-LetPairBeta. Have: $\vdash \mathbf{let}\ ({x_1}^q, x_2) = ({a_1}^q, a_2)\ \mathbf{in}\ b \rightsquigarrow b\{a_1/x_1\}\{a_2/x_2\}$.
  Need to show: $\vdash \mathbf{let}\ (x_1', x_2) = (\mathbf{lock}^q\ \overline{a_1}, \overline{a_2})\ \mathbf{in}\ \mathbf{unlock}^q\ x_1 = x_1'\ \mathbf{in}\ \overline{b} \rightsquigarrow^* \overline{b}\{\overline{a_1}/x_1\}\{\overline{a_2}/x_2\}$, where x_1' is fresh.
  Now, by β-rule,
  $\vdash \mathbf{let}\ (x_1', x_2) = (\mathbf{lock}^q\ \overline{a_1}, \overline{a_2})\ \mathbf{in}\ \mathbf{unlock}^q\ x_1 = x_1'\ \mathbf{in}\ \overline{b} \rightsquigarrow \mathbf{unlock}^q\ x_1 = \mathbf{lock}^q\ \overline{a_1}\ \mathbf{in}\ \overline{b}\{\overline{a_2}/x_2\}$.
  Again, by β-rule, $\vdash \mathbf{unlock}^q\ x_1 = \mathbf{lock}^q\ \overline{a_1}\ \mathbf{in}\ \overline{b}\{\overline{a_2}/x_2\} \rightsquigarrow \overline{b}\{\overline{a_2}/x_2\}\{\overline{a_1}/x_1\}$.
  But $\overline{b}\{\overline{a_2}/x_2\}\{\overline{a_1}/x_1\} = \overline{b}\{\overline{a_1}/x_1\}\{\overline{a_2}/x_2\}$.
  The case follows.

□

C.6 Proofs of Lemmas/Theorems Stated in Section 4.6

Lemma C.23 (Lemma 4.16) If $\Gamma_1, \Gamma_2 \vdash a : A$ and $\Gamma \vdash C : s$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$, then $\Gamma_1, z :^0 C, \Gamma_2 \vdash a : A$.

Proof. By induction on $\Gamma_1, \Gamma_2 \vdash a : A$. □

Lemma C.24 (Lema 4.17) If $\Gamma_1, z :^r C, \Gamma_2 \vdash a : A$ and $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$, then $\Gamma_1 + r \cdot \Gamma, \Gamma_2\{c/z\} \vdash a\{c/z\} : A\{c/z\}$.

Proof. By induction on $\Gamma_1, z :^r C, \Gamma_2 \vdash a : A$. We present some of the interesting cases below.

- Rule Grad-Var. There are two cases to consider.
  - Have: $0 \cdot \Gamma_{11}, z :^0 C, 0 \cdot \Gamma_{12}, x :^1 A \vdash x : A$ where $\Gamma_{11}, z :^{r_0} C, \Gamma_{12} \vdash A : s$. Further, $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_{11} \rfloor$.
    Need to show: $0 \cdot \Gamma_{11}, 0 \cdot \Gamma_{12}\{c/z\}, x :^1 A\{c/z\} \vdash x : A\{c/z\}$.
    By IH, $\Gamma_{11} + r_0 \cdot \Gamma, \Gamma_{12}\{c/z\} \vdash A\{c/z\} : s$.
    This case, then, follows by rule Grad-Var.
  - Have: $0 \cdot \Gamma_1, z :^1 C \vdash z : C$ where $\Gamma_1 \vdash C : s$. Further, $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
    Need to show: $\Gamma \vdash c : C\{c/z\}$.
    Since $z \notin \text{fv}\, C$, so $C\{c/z\} = C$.
    This case, then, follows from premises.
- Rule Grad-Weak. There are two cases to consider.
  - Have: $\Gamma_1, z :^r C, \Gamma_2, y :^0 B \vdash a : A$ where $\Gamma_1, z :^r C, \Gamma_2 \vdash a : A$ and $\Gamma_{01}, z :^{r_0} C, \Gamma_{02} \vdash B : s$, where $\lfloor \Gamma_{01} \rfloor = \lfloor \Gamma_1 \rfloor$ and $\lfloor \Gamma_{02} \rfloor = \lfloor \Gamma_2 \rfloor$. Further, $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
    Need to show: $\Gamma_1 + r \cdot \Gamma, \Gamma_2\{c/z\}, y :^0 B\{c/z\} \vdash a\{c/z\} : A\{c/z\}$.
    By IH, $\Gamma_1 + r \cdot \Gamma, \Gamma_2\{c/z\} \vdash a\{c/z\} : A\{c/z\}$.
    Again by IH, $\Gamma_{01} + r_0 \cdot \Gamma, \Gamma_{02}\{c/z\} \vdash B\{c/z\} : s$.
    This case, then, follows by rule Grad-Weak.
  - Have: $\Gamma_1, y :^0 B \vdash a : A$ where $\Gamma_1 \vdash a : A$. Further, $\Gamma \vdash b : B$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
    Need to show: $\Gamma_1 \vdash a\{b/y\} : A\{b/y\}$.
    Now, since $y \notin \text{fv}\, a$ and $y \notin \text{fv}\, A$, so $a\{b/y\} = a$ and $A\{b/y\} = A$.
    This case, then, follows from premises.
- Rule Grad-Pi. Have: $\Gamma_{11} + \Gamma_{21}, z :^{r_1 + r_2} C, \Gamma_{12} + \Gamma_{22} \vdash \Pi x :^q A.B : s_3$ where $\Gamma_{11} + z :^{r_1} C, \Gamma_{12} \vdash A : s_1$ and $\Gamma_{21}, z :^{r_2} C, \Gamma_{22}, x :^{q_0} A \vdash B : s_2$ and $\mathcal{R}(s_1, s_2, s_3)$. Further, $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_{11} \rfloor$.
  Need to show: $\Gamma_{11} + \Gamma_{21} + (r_1 + r_2) \cdot \Gamma, \Gamma_{12}\{c/z\} + \Gamma_{22}\{c/z\} \vdash \Pi x :^q A\{c/z\}.B\{c/z\} : s_3$.
  By IH, $\Gamma_{11} + r_1 \cdot \Gamma, \Gamma_{12}\{c/z\} \vdash A\{c/z\} : s_1$.
  Again by IH, $\Gamma_{21} + r_2 \cdot \Gamma, \Gamma_{22}\{c/z\}, x :^{q_0} A\{c/z\} \vdash B\{c/z\} : s_2$.
  This case, then, follows by rule Grad-Pi.
- Rule Grad-Lam. Have: $\Gamma_1, z :^r C, \Gamma_2 \vdash \lambda^q x : A.b : \Pi x :^q A.B$ where $\Gamma_1, z :^r C, \Gamma_2, x :^q A \vdash b : B$ and $\Gamma_{01}, z :^{r_0} C, \Gamma_{02} \vdash \Pi x :^q A.B : s$, where $\lfloor \Gamma_{01} \rfloor = \lfloor \Gamma_1 \rfloor$ and $\lfloor \Gamma_{02} \rfloor = \lfloor \Gamma_2 \rfloor$. Further, $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
  Need to show: $\Gamma_1 + r \cdot \Gamma, \Gamma_2\{c/z\} \vdash \lambda^q x : A\{c/z\}.b\{c/z\} : \Pi x :^q A\{c/z\}.B\{c/z\}$.
  By IH, $\Gamma_1 + r \cdot \Gamma, \Gamma_2\{c/z\}, x :^q A\{c/z\} \vdash b\{c/z\} : B\{c/z\}$.
  Again by IH, $\Gamma_{01} + r_0 \cdot \Gamma, \Gamma_{02}\{c/z\} \vdash \Pi x :^q A\{c/z\}.B\{c/z\} : s$.
  This case, then, follows by rule Grad-Lam.

- Rule Grad-App. Have: $\Gamma_{11} + q \cdot \Gamma_{21}, z :^{r_1 + q \cdot r_2} C, \Gamma_{12} + q \cdot \Gamma_{22} \vdash b\ a^q : B\{a/x\}$ where $\Gamma_{11}, z :^{r_1} C, \Gamma_{12} \vdash b : \Pi x{:}^q A.B$ and $\Gamma_{21}, z :^{r_2} C, \Gamma_{22} \vdash a : A$. Further, $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_{11} \rfloor$.
  Need to show: $\Gamma_{11} + q \cdot \Gamma_{21} + (r_1 + q \cdot r_2) \cdot \Gamma, \Gamma_{12}\{c/z\} + q \cdot \Gamma_{22}\{c/z\} \vdash b\{c/z\}\ a\{c/z\}^q : B\{c/z\}\{a\{c/z\}/x\}$.
  By IH, $\Gamma_{11} + r_1 \cdot \Gamma, \Gamma_{12}\{c/z\} \vdash b\{c/z\} : \Pi x{:}^q A\{c/z\}.B\{c/z\}$.
  Again by IH, $\Gamma_{21} + r_2 \cdot \Gamma, \Gamma_{22}\{c/z\} \vdash a\{c/z\} : A\{c/z\}$.
  This case, then, follows by rule Grad-App.

- Rule Grad-Pair. Have: $q \cdot \Gamma_{11} + \Gamma_{21}, z :^{q \cdot r_1 + r_2} C, q \cdot \Gamma_{12} + \Gamma_{22} \vdash ({a_1}^q, a_2) : \Sigma x{:}^q A_1.A_2$ where $\Gamma_{11}, z :^{r_1} C, \Gamma_{12} \vdash a_1 : A_1$ and $\Gamma_{21}, z :^{r_2} C, \Gamma_{22} \vdash a_2 : A_2\{a_1/x\}$ and $\Gamma_{01}, z :^{r_0} C, \Gamma_{02} \vdash \Sigma x{:}^q A_1.A_2 : s$, where $\lfloor \Gamma_{01} \rfloor = \lfloor \Gamma_{11} \rfloor$ and $\lfloor \Gamma_{02} \rfloor = \lfloor \Gamma_{12} \rfloor$. Further, $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_{11} \rfloor$.
  Need to show: $q \cdot \Gamma_{11} + \Gamma_{21} + (q \cdot r_1 + r_2) \cdot \Gamma, q \cdot \Gamma_{12}\{c/z\} + \Gamma_{22}\{c/z\} \vdash (a_1\{c/z\}^q, a_2\{c/z\}) : \Sigma x{:}^q A_1\{c/z\}.A_2\{c/z\}$.
  By IH, $\Gamma_{11} + r_1 \cdot \Gamma, \Gamma_{12}\{c/z\} \vdash a_1\{c/z\} : A_1\{c/z\}$.
  Again by IH, $\Gamma_{21} + r_2 \cdot \Gamma, \Gamma_{22}\{c/z\} \vdash a_2\{c/z\} : A_2\{c/z\}\{a_1\{c/z\}/x\}$.
  This case, then, follows by rule Grad-Pair.

- Rule Grad-LetPair. Have: $\Gamma_{11} + \Gamma_{21}, z :^{r_1 + r_2} C, \Gamma_{12} + \Gamma_{22} \vdash \textbf{let}\,({x_1}^q, x_2) = a\ \textbf{in}\ b : B\{a/y\}$ where $\Gamma_{11}, z :^{r_1} C, \Gamma_{12} \vdash a : \Sigma x_1{:}^q A_1.A_2$ and $\Gamma_{21}, z :^{r_2} C, \Gamma_{22}, x_1 :^q A_1, x_2 :^1 A_2 \vdash b : B\{({x_1}^q, x_2)/y\}$ and $\Gamma_{01}, z :^{r_0} C, \Gamma_{02}, y :^{q_0} \Sigma x_1{:}^q A_1.A_2 \vdash B : s$, where $\lfloor \Gamma_{01} \rfloor = \lfloor \Gamma_{11} \rfloor$ and $\lfloor \Gamma_{02} \rfloor = \lfloor \Gamma_{12} \rfloor$. Further, $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_{11} \rfloor$.
  Need to show: $\Gamma_{11} + \Gamma_{21} + (r_1 + r_2) \cdot \Gamma, \Gamma_{12}\{c/z\} + \Gamma_{22}\{c/z\} \vdash \textbf{let}\,({x_1}^q, x_2) = a\{c/z\}\ \textbf{in}\ b\{c/z\} : B\{c/z\}\{a\{c/z\}/y\}$.
  By IH, $\Gamma_{11} + r_1 \cdot \Gamma, \Gamma_{12}\{c/z\} \vdash a\{c/z\} : \Sigma x_1{:}A_1\{c/z\}.A_2\{c/z\}$.
  Again by IH, $\Gamma_{21} + r_2 \cdot \Gamma, \Gamma_{22}\{c/z\}, x_1 :^q A_1\{c/z\}, x_2 :^1 A_2\{c/z\} \vdash b\{c/z\} : B\{({x_1}^q, x_2)/y\}\{c/z\}$.
  Also by IH, $\Gamma_{01} + r_0 \cdot \Gamma, \Gamma_{02}\{c/z\}, y :^{q_0} \Sigma x_1{:}^q A_1\{c/z\}.A_2\{c/z\} \vdash B\{c/z\} : s$.
  This case, then, follows by rule Grad-LetPair.

- Rule Grad-Conv. Have: $\Gamma_1, z :^r C, \Gamma_2 \vdash a : B$ where $\Gamma_1, z :^r C, \Gamma_2 \vdash a : A$ and $A =_\beta B$ and $\Gamma_{01}, z :^{r_0} C, \Gamma_{02} \vdash B : s$, where $\lfloor \Gamma_{01} \rfloor = \lfloor \Gamma_1 \rfloor$ and $\lfloor \Gamma_{02} \rfloor = \lfloor \Gamma_2 \rfloor$. Further, $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
  Need to show: $\Gamma_1 + r \cdot \Gamma, \Gamma_2\{c/z\} \vdash a\{c/z\} : B\{c/z\}$.
  By IH, $\Gamma_1 + r \cdot \Gamma, \Gamma_2\{c/z\} \vdash a\{c/z\} : A\{c/z\}$.
  Again by IH, $\Gamma_{01} + r_0 \cdot \Gamma, \Gamma_{02}\{c/z\} \vdash B\{c/z\} : s$.
  Next, since $A =_\beta B$, so $A\{c/z\} =_\beta B\{c/z\}$.
  This case, then, follows by rule Grad-Conv.

- Rule Grad-Sub. Have: $\Gamma_{21}, z :^{r_2} C, \Gamma_{22} \vdash a : A$ where $\Gamma_{11}, z :^{r_1} C, \Gamma_{12} \vdash a : A$ and $\Gamma_{21} <: \Gamma_{11}$ and $r_2 <: r_1$ and $\Gamma_{22} <: \Gamma_{12}$. Further, $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_{21} \rfloor$.
  Need to show: $\Gamma_{21} + r_2 \cdot \Gamma, \Gamma_{22}\{c/z\} \vdash a\{c/z\} : A\{c/z\}$.
  By IH, $\Gamma_{11} + r_1 \cdot \Gamma, \Gamma_{12}\{c/z\} \vdash a\{c/z\} : A\{c/z\}$.
  Since $r_2 <: r_1$, so $r_2 \cdot \Gamma <: r_1 \cdot \Gamma$. Next since $\Gamma_{21} <: \Gamma_{11}$, so $\Gamma_{21} + r_2 \cdot \Gamma <: \Gamma_{11} + r_1 \cdot \Gamma$.
  This case, then, follows by rule Grad-Sub.

□

Lemma C.25 (Equality inversion) The definitional equality relation of GRAD satisfies the following inversion lemmas:

- If $\Pi x :^{q_1} A_1.B_1 =_\beta \Pi x :^{q_2} A_2.B_2$, then $q_1 = q_2$ and $A_1 =_\beta A_2$. Further, if $a_1 =_\beta a_2$, then $B_1\{a_1/x\} =_\beta B_2\{a_2/x\}$.
- If $\Sigma x{:}^{q_1} A_1.B_1 =_\beta \Sigma x{:}^{q_2} A_2.B_2$, then $q_1 = q_2$ and $A_1 =_\beta A_2$. Further, if $a_1 =_\beta a_2$, then $B_1\{a_1/x\} =_\beta B_2\{a_2/x\}$.

Proof. First, note the similarity between this equality inversion lemma for GRAD and the corresponding lemma for DDC$^\top$. In fact, the equality inversion lemma for DDC$^\top$ is essentially the same lemma with ℓ_1 and ℓ_2 replacing q_1 and q_2. However, note that in Appendix B, the equality inversion lemma for DDC$^\top$ is not proved directly but is derived as an instance of the equality inversion lemma for DDC, i.e., Lemma B.53. But if we were to prove the lemma directly, we can follow the same argument and furthermore, with two significant simplifications. First, we can make the definitional equality and parallel reduction relations unindexed. Second, we can remove the context information from these two relations. With these two simplifications, a direct proof of the equality inversion lemma for DDC$^\top$ would be much simpler compared to that of Lemma B.53. But the proof would have the same structure that first establishes equivalence of definitional equality and joinability and thereafter shows joinability implies consistency. That direct proof can then be adapted in a straightforward manner to prove the lemma above. We leave the direct proof and its adaptation as exercises for the reader. □

Lemma C.26 (Typing inversion) GRAD satisfies the following inversion lemmas:

- If $\Gamma \vdash v : A$ and $A =_\beta \Pi x{:}^q B_1.B_2$ and v is a value, then exists A_1, A_2 and a_2 such that
  - $v = \lambda^q x{:}A_1.a_2$
  - $\Gamma, x{:}^q A_1 \vdash a_2 : A_2$
  - $\Gamma_0 \vdash \Pi x{:}^q A_1.A_2 : s$ where $\lfloor \Gamma_0 \rfloor = \lfloor \Gamma \rfloor$
  - $A =_\beta \Pi x{:}^q A_1.A_2$
- If $\Gamma \vdash v : A$ and $A =_\beta \Sigma x{:}^q B_1.B_2$ and v is a value, then exists A_1, A_2, a_1 and a_2 such that
  - $v = (a_1{}^q, a_2)$
  - $\Gamma_1 \vdash a_1 : A_1$ and $\Gamma_2 \vdash a_2 : A_2\{a_1/x\}$ where $\Gamma <: q \cdot \Gamma_1 + \Gamma_2$
  - $\Gamma_0 \vdash \Sigma x{:}^q A_1.A_2 : s$ where $\lfloor \Gamma_0 \rfloor = \lfloor \Gamma \rfloor$
  - $A =_\beta \Sigma x{:}^q A_1.A_2$

Proof. The proof of this lemma is very similar to that of Lemma B.54. Still, we present the proof for the Π-case of the lemma below.
The proof is by induction on $\Gamma \vdash v : A$. Some of the interesting cases of the induction follow:

- Rule GRAD-VAR. Have: $0 \cdot \Gamma, x :^1 A \vdash x : A$ where $\Gamma \vdash A : s$.
  The lemma statement does not apply because x is not a value.

- Rule GRAD-WEAK. Have: $\Gamma, z :^0 C \vdash v : A$ where $\Gamma \vdash v : A$ and $\Gamma_1 \vdash C : s$ where $\lfloor \Gamma_1 \rfloor = \lfloor \Gamma \rfloor$. Further, v is a value and $A =_\beta \Pi x :^q B_1.B_2$.
  Using IH, we have:

  - $v = \lambda^q x : A_1.a_2$.
  - $\Gamma, x :^q A_1 \vdash a_2 : A_2$.
    By weakening, $\Gamma, z :^0 C, x :^q A_1 \vdash a_2 : A_2$.
  - $\Gamma_0 \vdash \Pi x :^q A_1.A_2 : s$ where $\lfloor \Gamma_0 \rfloor = \lfloor \Gamma \rfloor$.
    By weakening, $\Gamma_0, z :^0 C \vdash \Pi x :^q A_1.A_2 : s$.
  - $A =_\beta \Pi x :^q A_1.A_2$.

- Rule GRAD-PI. Have: $\Gamma_1 + \Gamma_2 \vdash \Pi x :^{q_0} A_1.A_2 : s_3$ where $\Gamma_1 \vdash A_1 : s_1$ and $\Gamma_2, x :^{r_0} A_1 \vdash A_2 : s_2$ and $\mathcal{R}(s_1, s_2, s_3)$. Further, $s_3 =_\beta \Pi x :^q B_1.B_2$. But this is a contradiction owing to consistency of definitional equality.

- Rule GRAD-LAM. Have: $\Gamma \vdash \lambda^{q_0} x : A_1.a_2 : \Pi x :^{q_0} A_1.A_2$ where $\Gamma, x :^{q_0} A_1 \vdash a_2 : A_2$ and $\Gamma_0 \vdash \Pi x :^{q_0} A_1.A_2 : s$, where $\lfloor \Gamma_0 \rfloor = \lfloor \Gamma \rfloor$. Further, $\Pi x :^{q_0} A_1.A_2 =_\beta \Pi x :^q B_1.B_2$.
  Now, by consistency of definitional equality, $q_0 = q$.
  So, we have:

  - $v = \lambda^q x : A_1.a_2$.
  - $\Gamma, x :^q A_1 \vdash a_2 : A_2$.
  - $\Gamma_0 \vdash \Pi x :^q A_1.A_2 : s$.
  - $\Pi x :^q A_1.A_2 =_\beta \Pi x :^q A_1.A_2$, by reflexivity.

- Rule GRAD-APP. Have: $\Gamma_1 + q_0 \cdot \Gamma_2 \vdash b\ a^{q_0} : B\{a/x\}$ where $\Gamma_1 \vdash b : \Pi x :^{q_0} A.B$ and $\Gamma_2 \vdash a : A$.
  The lemma statement does not apply because $b\ a^{q_0}$ is not a value.

- Rule GRAD-CONV. Have: $\Gamma \vdash v : B$ where $\Gamma \vdash v : A$ and $A =_\beta B$ and $\Gamma_1 \vdash B : s$, where $\lfloor \Gamma_1 \rfloor = \lfloor \Gamma \rfloor$. Further, v is a value and $B =_\beta \Pi x :^q B_1.B_2$.
  Then, by transitivity, $A =_\beta \Pi x :^q B_1.B_2$.
  Using IH, we have:

  - $v = \lambda^q x : A_1.a_2$.
  - $\Gamma, x :^q A_1 \vdash a_2 : A_2$.
  - $\Gamma_0 \vdash \Pi x :^q A_1.A_2 : s$ where $\lfloor \Gamma_0 \rfloor = \lfloor \Gamma \rfloor$.
  - $A =_\beta \Pi x :^q A_1.A_2$.
    By symmetry and transitivity, $B =_\beta \Pi x :^q A_1.A_2$.

- Rule GRAD-SUB. Have: $\Gamma_2 \vdash v : A$ where $\Gamma_1 \vdash a : A$ and $\Gamma_2 <: \Gamma_1$. Further, $A =_\beta \Pi x :^q B_1 . B_2$.
  Using IH, we have:
  - $v = \lambda^q x : A_1 . a_2$.
  - $\Gamma_1, x :^q A_1 \vdash a_2 : A_2$. By rule GRAD-SUB, $\Gamma_2, x :^q A_1 \vdash a_2 : A_2$.
  - $\Gamma_0 \vdash \Pi x :^q A_1 . A_2 : s$ where $\lfloor \Gamma_0 \rfloor = \lfloor \Gamma_1 \rfloor$.
  - $A =_\beta \Pi x :^q A_1 . A_2$.

$\square$

Lemma C.27 (Regularity) If $\Gamma \vdash a : A$, then $\exists s$ such that $A =_\beta s$ or $\Gamma_0 \vdash A : s$ where $\lfloor \Gamma_0 \rfloor = \lfloor \Gamma \rfloor$.

Proof. By induction on $\Gamma \vdash a : A$. We present some of the interesting cases below.

- Rule GRAD-VAR. Have: $0 \cdot \Gamma, x :^1 A \vdash x : A$ where $\Gamma \vdash A : s$.
  By weakening, $\Gamma, x :^0 A \vdash A : s$.
  The case follows.
- Rule GRAD-WEAK. Have: $\Gamma, y :^0 B \vdash a : A$ where $\Gamma \vdash a : A$ and $\Gamma_1 \vdash B : s_1$ where $\lfloor \Gamma_1 \rfloor = \lfloor \Gamma \rfloor$.
  By IH, $\exists s$ such that $A =_\beta s$ or $\Gamma_0 \vdash A : s$ where $\lfloor \Gamma_0 \rfloor = \lfloor \Gamma \rfloor$.
  Therefore, we have, $A =_\beta s$ or by weakening, $\Gamma_0, y :^0 B \vdash A : s$.
  The case follows.
- Rule GRAD-AXIOM. Have: $\varnothing \vdash s_1 : s_2$ where $\mathcal{A}(s_1, s_2)$.
  By reflexivity, $s_2 =_\beta s_2$.
  The case follows.
- Rule GRAD-PI. Have: $\Gamma_1 + \Gamma_2 \vdash \Pi x :^q A . B : s_3$ where $\Gamma_1 \vdash A : s_1$ and $\Gamma_2, x :^r A \vdash B : s_2$ and $\mathcal{R}(s_1, s_2, s_3)$.
  By reflexivity, $s_3 =_\beta s_3$.
  The case follows.
- Rule GRAD-LAM. Have: $\Gamma \vdash \lambda^q x : A . b : \Pi x :^q A . B$ where $\Gamma, x :^q A \vdash b : B$ and $\Gamma_0 \vdash \Pi x :^q A . B : s$ where $\lfloor \Gamma_0 \rfloor = \lfloor \Gamma \rfloor$.
  The case follows from premises.
- Rule GRAD-APP. Have: $\Gamma_1 + q \cdot \Gamma_2 \vdash b\ a^q : B\{a/x\}$ where $\Gamma_1 \vdash b : \Pi x :^q A . B$ and $\Gamma_2 \vdash a : A$.
  By IH, there exists s such that $\Pi x :^q A . B =_\beta s$ or $\Gamma_0 \vdash \Pi x :^q A . B : s$ where $\lfloor \Gamma_0 \rfloor = \lfloor \Gamma_1 \rfloor$.
  Since definitional equality is consistent, $\Pi x :^q A . B =_\beta s$ cannot hold.
  As such, $\Gamma_0 \vdash \Pi x :^q A . B : s$.
  Then, by inversion, there exists s_1 and s_2 such that $\Gamma_{01} \vdash A : s_1$ and $\Gamma_{02}, x :^r A \vdash B : s_2$ and $\Gamma_0 <: \Gamma_{01} + \Gamma_{02}$.
  By substitution lemma C.24, $\Gamma_{02} + r \cdot \Gamma_2 \vdash B\{a/x\} : s_2$.
  The case follows.

- Rule GRAD-CONV. Have: $\Gamma \vdash a : B$ where $\Gamma \vdash a : A$ and $A =_\beta B$ and $\Gamma_0 \vdash B : s$, where $\lfloor \Gamma_0 \rfloor = \lfloor \Gamma \rfloor$.
  The case follows from premises.

- Rule GRAD-SUB. Have: $\Gamma_2 \vdash a : A$ where $\Gamma_1 \vdash a : A$ and $\Gamma_2 <: \Gamma_1$.
  The case follows by IH.

$\square$

Theorem C.28 (Theorem 4.18) If $\Gamma \vdash a : A$ and $\vdash a \rightsquigarrow a'$, then $\Gamma \vdash a' : A$.

Proof. By induction on $\Gamma \vdash a : A$ and subsequent inversion on $\vdash a \rightsquigarrow a'$. We present some of the interesting cases below.

- Rule GRAD-APP. Have: $\Gamma_1 + q \cdot \Gamma_2 \vdash b\ a^q : B\{a/x\}$ where $\Gamma_1 \vdash b : \Pi x :^q A.B$ and $\Gamma_2 \vdash a : A$. By inversion on $\vdash b\ a^q \rightsquigarrow c$:
  - $\vdash b\ a^q \rightsquigarrow b'\ a^q$ when $\vdash b \rightsquigarrow b'$.
    Need to show: $\Gamma_1 + q \cdot \Gamma_2 \vdash b'\ a^q : B\{a/x\}$.
    By IH, $\Gamma_1 \vdash b' : \Pi x :^q A.B$.
    This case, then, follows by rule GRAD-APP.
  - $b = \lambda^q x : A_0 . b'$ and $\vdash (\lambda^q x : A_0 . b')\ a^q \rightsquigarrow b'\{a/x\}$.
    Need to show: $\Gamma_1 + q \cdot \Gamma_2 \vdash b'\{a/x\} : B\{a/x\}$.
    Using typing inversion lemma C.26 on $\Gamma_1 \vdash \lambda^q x : A_0 . b' : \Pi x :^q A.B$, we have:
    * $\Gamma_1, x :^q A' \vdash b' : B'$
    * $\Gamma_0 \vdash \Pi x :^q A'.B' : s$, where $\lfloor \Gamma_0 \rfloor = \lfloor \Gamma_1 \rfloor$. Therefore, there exists s_1 and s_2 such that $\Gamma_{01} \vdash A' : s_1$ and $\Gamma_{02}, x :^r A' \vdash B' : s_2$ and $\Gamma_0 <: \Gamma_{01} + \Gamma_{02}$.
    * $\Pi x :^q A.B =_\beta \Pi x :^q A'.B'$. Therefore, by equality inversion lemma C.25, $A =_\beta A'$ and $B\{a/x\} =_\beta B'\{a/x\}$.

    Then, by rule GRAD-CONV, $\Gamma_2 \vdash a : A'$.
    Next, by substitution lemma C.24, $\Gamma_1 + q \cdot \Gamma_2 \vdash b'\{a/x\} : B'\{a/x\}$.
    This case, then, follows by using rule GRAD-CONV.

- Rule GRAD-LETPAIR. Have: $\Gamma_1 + \Gamma_2 \vdash \textbf{let}\,(x^q, y) = a \textbf{ in } b : B\{a/z\}$ where $\Gamma_1 \vdash a : \Sigma x{:}^q A_1 . A_2$ and $\Gamma_2, x :^q A_1, y :^1 A_2 \vdash b : B\{(x^q, y)/z\}$ and $\Gamma_0, z :^{r_0} \Sigma x{:}^q A_1 . A_2 \vdash B : s$ where $\lfloor \Gamma_0 \rfloor = \lfloor \Gamma_1 \rfloor$. By inversion on $\vdash \textbf{let}\,(x^q, y) = a \textbf{ in } b \rightsquigarrow c$:
  - $\vdash \textbf{let}\,(x^q, y) = a \textbf{ in } b \rightsquigarrow \textbf{let}\,(x^q, y) = a' \textbf{ in } b$ when $\vdash a \rightsquigarrow a'$.
    Need to show: $\Gamma_1 + \Gamma_2 \vdash \textbf{let}\,(x^q, y) = a' \textbf{ in } b : B\{a/z\}$.
    By IH, $\Gamma_1 \vdash a' : \Sigma x{:}^q A_1 . A_2$.
    Then, by rule GRAD-LETPAIR, $\Gamma_1 + \Gamma_2 \vdash \textbf{let}\,(x^q, y) = a' \textbf{ in } b : B\{a'/z\}$.
    Now, since $\vdash a \rightsquigarrow a'$, so $a =_\beta a'$. Then, $B\{a/z\} =_\beta B\{a'/z\}$.
    This case, then, follows by rule GRAD-CONV.

– $a = (a_1{}^q, a_2)$ and $\vdash \textbf{let}\,(x^q, y) = (a_1{}^q, a_2)$ **in** $b \leadsto b\{a_1/x\}\{a_2/y\}$.
Need to show: $\Gamma_1 + \Gamma_2 \vdash b\{a_1/x\}\{a_2/y\} : B\{(a_1{}^q, a_2)/z\}$.
First, note that since $\Gamma_1 \vdash (a_1{}^q, a_2) : \Sigma x{:}^q A_1.A_2$, by regularity lemma C.27, there exists s such that $\Gamma'_1 \vdash \Sigma x{:}^q A_1.A_2 : s$ where $\lfloor \Gamma'_1 \rfloor = \lfloor \Gamma_1 \rfloor$. Then, by inversion, there exists s_1 and s_2 such that $\Gamma'_{11} \vdash A_1 : s_1$ and $\Gamma'_{12}, x{:}^r A_1 \vdash A_2 : s_2$ and $\Gamma'_1 <: \Gamma'_{11} + \Gamma'_{12}$.
Next, using typing inversion lemma C.26 on $\Gamma_1 \vdash (a_1{}^q, a_2) : \Sigma x{:}^q A_1.A_2$, we have:

  * $\Gamma_{11} \vdash a_1 : A'_1$ and $\Gamma_{12} \vdash a_2 : A'_2\{a_1/x\}$ where $\Gamma_1 <: q \cdot \Gamma_{11} + \Gamma_{12}$
  * $\Sigma x{:}^q A_1.A_2 =_\beta \Sigma x{:}^q A'_1.A'_2$. Therefore, by equality inversion lemma C.25, $A_1 =_\beta A'_1$ and $A_2\{a_1/x\} =_\beta A'_2\{a_1/x\}$.

Then, by rule GRAD-CONV, $\Gamma_{11} \vdash a_1 : A_1$ and $\Gamma_{12} \vdash a_2 : A_2\{a_1/x\}$.
Next, by applying substitution lemma C.24 twice, we have, $\Gamma_2 + q \cdot \Gamma_{11} + \Gamma_{12} \vdash b\{a_1/x\}\{a_2/y\} : B\{(a_1{}^q, a_2)/z\}$.
This case, then, follows by rule GRAD-SUB.

□

Theorem C.29 (Theorem 4.19) If $\varnothing \vdash a : A$, then either a is a value or there exists some a' such that $\vdash a \leadsto a'$.

Proof. By induction on $\varnothing \vdash a : A$. We present some of the interesting cases below.

- Rule GRAD-VAR. Does not apply because the context here is empty.
- Rule GRAD-APP. Have: $\varnothing \vdash b\ a^q : B\{a/x\}$ where $\varnothing \vdash b : \Pi x{:}^q A.B$ and $\varnothing \vdash a : A$.
  Need to show: $\exists c$ such that $\vdash b\ a^q \leadsto c$.
  By IH, b is a value or $\exists b'$ such that $\vdash b \leadsto b'$.
  If $\vdash b \leadsto b'$, then this case follows by setting $c := b'\ a^q$.
  Otherwise, using inversion lemma C.26 on $\varnothing \vdash b : \Pi x{:}^q A.B$, we have, $b = \lambda^q x{:}A'.b'$.
  This case, then, follows by setting $c := b'\{a/x\}$.
- Rule GRAD-LETPAIR. Have: $\varnothing \vdash \textbf{let}\,(x^q, y) = a$ **in** $b : B\{a/z\}$ where $\varnothing \vdash a : \Sigma x{:}^q A_1.A_2$ and $x{:}^q A_1, y{:}^1 A_2 \vdash b : B\{(x^q, y)/z\}$ and $z{:}^{r_0} \Sigma x{:}^q A_1.A_2 \vdash B : s$.
  Need to show: $\exists c$ such that $\vdash \textbf{let}\,(x^q, y) = a$ **in** $b \leadsto c$.
  By IH, a is a value or $\exists a'$ such that $\vdash a \leadsto a'$.
  If $\vdash a \leadsto a'$, then this case follows by setting $c := \textbf{let}\,(x^q, y) = a'$ **in** b.
  Otherwise, using inversion lemma C.26 on $\varnothing \vdash a : \Sigma x{:}^q A_1.A_2$, we have, $a = (a_1{}^q, a_2)$.
  This case, then, follows by setting $c := b\{a_1/x\}\{a_2/y\}$.

□

C.7 Proofs of Lemmas/Theorems/Corollaries Stated in Section 4.7

Lemma C.30 (Lemma 4.20) If $\Gamma_1, z :^r C, \Gamma_2 \vdash a : A$ and $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$, then $\Gamma_1, z = c :^r C, \Gamma_2 \vdash a : A$.

Proof. By induction on $\Gamma_1, z :^r C, \Gamma_2 \vdash a : A$. We present some of the interesting cases below.

- Rule GRAD-VAR. There are two cases to consider.
  - Have: $0 \cdot \Gamma_1, z :^0 C, 0 \cdot \Gamma_2, x :^1 A \vdash x : A$ where $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash A : s$. Further, $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
    Need to show: $0 \cdot \Gamma_1, z = c :^0 C, 0 \cdot \Gamma_2, x :^1 A \vdash x : A$.
    By IH, $\Gamma_1, z = c :^{r_0} C, \Gamma_2 \vdash A : s$.
    This case, then, follows by rule GRAD-VAR.
  - Have: $0 \cdot \Gamma_1, x :^1 A \vdash x : A$ where $\Gamma_1 \vdash A : s$. Further, $\Gamma \vdash a : A$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
    Need to show: $0 \cdot \Gamma_1, x = a :^1 A \vdash x : A$.
    Follows by rule GRAD-VAR-DEF.
- Rule GRAD-WEAK. There are two cases to consider.
  - Have: $\Gamma_1, z :^r C, \Gamma_2, y :^0 B \vdash a : A$ where $\Gamma_1, z :^r C, \Gamma_2 \vdash a : A$ and $\Gamma_{01}, z :^{r_0} C, \Gamma_{02} \vdash B : s$, where $\lfloor \Gamma_{01} \rfloor = \lfloor \Gamma_1 \rfloor$ and $\lfloor \Gamma_{02} \rfloor = \lfloor \Gamma_2 \rfloor$. Further, $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
    Need to show: $\Gamma_1, z = c :^r C, \Gamma_2, y :^0 B \vdash a : A$.
    By IH, $\Gamma_1, z = c :^r C, \Gamma_2 \vdash a : A$ and $\Gamma_{01}, z = c :^{r_0} C, \Gamma_{02} \vdash B : s$.
    This case, then, follows by rule GRAD-WEAK.
  - Have: $\Gamma_1, y :^0 B \vdash a : A$ where $\Gamma_1 \vdash a : A$ and $\Gamma_{01} \vdash B : s$, where $\lfloor \Gamma_{01} \rfloor = \lfloor \Gamma_1 \rfloor$. Further, $\Gamma \vdash b : B$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
    Need to show: $\Gamma_1, y = b :^0 B \vdash a : A$.
    Follows by rule GRAD-WEAK-DEF.
- Rule GRAD-VAR-DEF. Have: $0 \cdot \Gamma_1, z :^0 C, 0 \cdot \Gamma_2, x = a :^1 A \vdash x : A$ where $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash a : A$. Further, $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
  Need to show: $0 \cdot \Gamma_1, z = c :^0 C, 0 \cdot \Gamma_2, x = a :^1 A \vdash x : A$.
  By IH, $\Gamma_1, z = c :^{r_0} C, \Gamma_2 \vdash a : A$.
  This case, then, follows by rule GRAD-VAR-DEF.
- Rule GRAD-WEAK-DEF. Have: $\Gamma_1, z :^r C, \Gamma_2, y = b :^0 B \vdash a : A$ where $\Gamma_1, z :^r C, \Gamma_2 \vdash a : A$ and $\Gamma_{01}, z :^{r_0} C, \Gamma_{02} \vdash b : B$, where $\lfloor \Gamma_{01} \rfloor = \lfloor \Gamma_1 \rfloor$ and $\lfloor \Gamma_{02} \rfloor = \lfloor \Gamma_2 \rfloor$. Further, $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
  Need to show: $\Gamma_1, z = c :^r C, \Gamma_2, y = b :^0 B \vdash a : A$.
  By IH, $\Gamma_1, z = c :^r C, \Gamma_2 \vdash a : A$ and $\Gamma_{01}, z = c :^{r_0} C, \Gamma_{02} \vdash b : B$.
  This case, then, follows by rule GRAD-WEAK-DEF.

- Rule GRAD-CONV-DEF. Have: $\Gamma_1, z :^r C, \Gamma_2 \vdash a : B$ where $\Gamma_1, z :^r C, \Gamma_2 \vdash a : A$ and $A\{\Gamma_1, z :^r C, \Gamma_2\} =_\beta B\{\Gamma_1, z :^r C, \Gamma_2\}$ and $\Gamma_{01}, z :^{r_0} C, \Gamma_{02} \vdash B : s$ where $\lfloor \Gamma_{01} \rfloor = \lfloor \Gamma_1 \rfloor$ and $\lfloor \Gamma_{02} \rfloor = \lfloor \Gamma_2 \rfloor$. Further, $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
  Need to show: $\Gamma_1, z = c :^r C, \Gamma_2 \vdash a : B$.
  By IH, $\Gamma_1, z = c :^r C, \Gamma_2 \vdash a : A$ and $\Gamma_{01}, z = c :^{r_0} C, \Gamma_{02} \vdash B : s$.
  Next, since $A\{\Gamma_2\}\{\Gamma_1\} =_\beta B\{\Gamma_2\}\{\Gamma_1\}$, so $A\{\Gamma_2\}\{\Gamma_1\}\{c\{\Gamma_1\}/z\} =_\beta B\{\Gamma_2\}\{\Gamma_1\}\{c\{\Gamma_1\}/z\}$.
  Then, $A\{\Gamma_2\}\{c/z\}\{\Gamma_1\} =_\beta B\{\Gamma_2\}\{c/z\}\{\Gamma_1\}$.
  As such, $A\{\Gamma_1, z = c :^r C, \Gamma_2\} =_\beta B\{\Gamma_1, z = c :^r C, \Gamma_2\}$.
  This case, then, follows by rule GRAD-CONV-DEF.

□

Lemma C.31 (Lemma 4.21) If $\Gamma_1, \Gamma_2 \vdash a : A$ and $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$, then $\Gamma_1, z = c :^0 C, \Gamma_2 \vdash a : A$.

Proof. By induction on $\Gamma_1, \Gamma_2 \vdash a : A$. □

Lemma C.32 (Substitution) In GRAD extended with definitions, if $\Gamma_1, \Gamma_0, \Gamma_2 \vdash a : A$ where $\Gamma_0 \coloneqq z :^r C$ or $\Gamma_0 \coloneqq z = c :^r C$ and $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$, then $\Gamma_1 + r \cdot \Gamma, \Gamma_2\{c/z\} \vdash a\{c/z\} : A\{c/z\}$.

Proof. By induction on $\Gamma_1, \Gamma_0, \Gamma_2 \vdash a : A$.
The proof is similar to that of Lemma C.24. Below, we present the cases that are different.

- Rule GRAD-VAR-DEF. There are three cases to consider.
  - Have: $0 \cdot \Gamma_{11}, z :^0 C, 0 \cdot \Gamma_{12}, x = a :^1 A \vdash x : A$ where $\Gamma_{11}, z :^{r_0} C, \Gamma_{12} \vdash a : A$. Further, $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_{11} \rfloor$.
    Need to show: $0 \cdot \Gamma_{11}, 0 \cdot \Gamma_{12}\{c/z\}, x = a\{c/z\} :^1 A\{c/z\} \vdash x : A\{c/z\}$.
    By IH, $\Gamma_{11} + r_0 \cdot \Gamma, \Gamma_{12}\{c/z\} \vdash a\{c/z\} : A\{c/z\}$.
    This case, then, follows by rule GRAD-VAR-DEF.
  - Have: $0 \cdot \Gamma_{11}, z = c :^0 C, 0 \cdot \Gamma_{12}, x = a :^1 A \vdash x : A$ where $\Gamma_{11}, z = c :^{r_0} C, \Gamma_{12} \vdash a : A$. Further, $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_{11} \rfloor$.
    Need to show: $0 \cdot \Gamma_{11}, 0 \cdot \Gamma_{12}\{c/z\}, x = a\{c/z\} :^1 A\{c/z\} \vdash x : A\{c/z\}$.
    By IH, $\Gamma_{11} + r_0 \cdot \Gamma, \Gamma_{12}\{c/z\} \vdash a\{c/z\} : A\{c/z\}$.
    This case, then, follows by rule GRAD-VAR-DEF.
  - Have: $0 \cdot \Gamma_1, z = c :^1 C \vdash z : C$ where $\Gamma_1 \vdash c : C$. Further, $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
    Need to show: $\Gamma \vdash c : C\{c/z\}$.
    Since $z \notin \text{fv}\, C$, so $C\{c/z\} = C$.
    This case, then, follows from premises.
- Rule GRAD-WEAK-DEF. There are three cases to consider.

– Have: $\Gamma_1, z :^r C, \Gamma_2, y = b :^0 B \vdash a : A$ where $\Gamma_1, z :^r C, \Gamma_2 \vdash a : A$ and $\Gamma_{01}, z :^{r_0} C, \Gamma_{02} \vdash b : B$, where $\lfloor \Gamma_{01} \rfloor = \lfloor \Gamma_1 \rfloor$ and $\lfloor \Gamma_{02} \rfloor = \lfloor \Gamma_2 \rfloor$. Further, $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
Need to show: $\Gamma_1 + r \cdot \Gamma, \Gamma_2\{c/z\}, y = b\{c/z\} :^0 B\{c/z\} \vdash a\{c/z\} : A\{c/z\}$.
By IH, $\Gamma_1 + r \cdot \Gamma, \Gamma_2\{c/z\} \vdash a\{c/z\} : A\{c/z\}$.
Again by IH, $\Gamma_{01} + r_0 \cdot \Gamma, \Gamma_{02}\{c/z\} \vdash b\{c/z\} : B\{c/z\}$.
This case, then, follows by rule GradD-Weak-Def.

– Have: $\Gamma_1, z = c :^r C, \Gamma_2, y = b :^0 B \vdash a : A$ where $\Gamma_1, z = c :^r C, \Gamma_2 \vdash a : A$ and $\Gamma_{01}, z = c :^{r_0} C, \Gamma_{02} \vdash b : B$, where $\lfloor \Gamma_{01} \rfloor = \lfloor \Gamma_1 \rfloor$ and $\lfloor \Gamma_{02} \rfloor = \lfloor \Gamma_2 \rfloor$. Further, $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
Need to show: $\Gamma_1 + r \cdot \Gamma, \Gamma_2\{c/z\}, y = b\{c/z\} :^0 B\{c/z\} \vdash a\{c/z\} : A\{c/z\}$.
By IH, $\Gamma_1 + r \cdot \Gamma, \Gamma_2\{c/z\} \vdash a\{c/z\} : A\{c/z\}$.
Again by IH, $\Gamma_{01} + r_0 \cdot \Gamma, \Gamma_{02}\{c/z\} \vdash b\{c/z\} : B\{c/z\}$.
This case, then, follows by rule GradD-Weak-Def.

– Have: $\Gamma_1, y = b :^0 B \vdash a : A$ where $\Gamma_1 \vdash a : A$ and $\Gamma_{01} \vdash b : B$, where $\lfloor \Gamma_{01} \rfloor = \lfloor \Gamma_1 \rfloor$. Further, $\Gamma \vdash b : B$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
Need to show: $\Gamma_1 \vdash a\{b/y\} : A\{b/y\}$.
Now, since $y \notin \text{fv}\, a$ and $y \notin \text{fv}\, A$, so $a\{b/y\} = a$ and $A\{b/y\} = A$.
This case, then, follows from premises.

- Rule GradD-Conv-Def. There are two cases to consider.

  – Have: $\Gamma_1, z :^r C, \Gamma_2 \vdash a : B$ where $\Gamma_1, z :^r C, \Gamma_2 \vdash a : A$ and $A\{\Gamma_1, z :^r C, \Gamma_2\} =_\beta B\{\Gamma_1, z :^r C, \Gamma_2\}$ and $\Gamma_{01}, z :^{r_0} C, \Gamma_{02} \vdash B : s$, where $\lfloor \Gamma_{01} \rfloor = \lfloor \Gamma_1 \rfloor$ and $\lfloor \Gamma_{02} \rfloor = \lfloor \Gamma_2 \rfloor$. Further, $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
  Need to show: $\Gamma_1 + r \cdot \Gamma, \Gamma_2\{c/z\} \vdash a\{c/z\} : B\{c/z\}$.
  By IH, $\Gamma_1 + r \cdot \Gamma, \Gamma_2\{c/z\} \vdash a\{c/z\} : A\{c/z\}$.
  Again by IH, $\Gamma_{01} + r_0 \cdot \Gamma, \Gamma_{02}\{c/z\} \vdash B\{c/z\} : s$.
  Next, we have, $A\{\Gamma_2\}\{\Gamma_1\} =_\beta B\{\Gamma_2\}\{\Gamma_1\}$.
  So $A\{\Gamma_2\}\{\Gamma_1\}\{c\{\Gamma_1\}/z\} =_\beta B\{\Gamma_2\}\{\Gamma_1\}\{c\{\Gamma_1\}/z\}$.
  Then, $A\{\Gamma_2\}\{c/z\}\{\Gamma_1\} =_\beta B\{\Gamma_2\}\{c/z\}\{\Gamma_1\}$.
  As such, $(A\{c/z\})\{\Gamma_2\{c/z\}\}\{\Gamma_1\} =_\beta (B\{c/z\})\{\Gamma_2\{c/z\}\}\{\Gamma_1\}$.
  Therefore, $(A\{c/z\})\{\Gamma_1 + r \cdot \Gamma, \Gamma_2\{c/z\}\} =_\beta (B\{c/z\})\{\Gamma_1 + r \cdot \Gamma, \Gamma_2\{c/z\}\}$.
  This case, then, follows by rule GradD-Conv-Def.

  – Have: $\Gamma_1, z = c :^r C, \Gamma_2 \vdash a : B$ where $\Gamma_1, z = c :^r C, \Gamma_2 \vdash a : A$ and $A\{\Gamma_1, z = c :^r C, \Gamma_2\} =_\beta B\{\Gamma_1, z = c :^r C, \Gamma_2\}$ and $\Gamma_{01}, z = c :^{r_0} C, \Gamma_{02} \vdash B : s$, where $\lfloor \Gamma_{01} \rfloor = \lfloor \Gamma_1 \rfloor$ and $\lfloor \Gamma_{02} \rfloor = \lfloor \Gamma_2 \rfloor$. Further, $\Gamma \vdash c : C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
  Need to show: $\Gamma_1 + r \cdot \Gamma, \Gamma_2\{c/z\} \vdash a\{c/z\} : B\{c/z\}$.
  By IH, $\Gamma_1 + r \cdot \Gamma, \Gamma_2\{c/z\} \vdash a\{c/z\} : A\{c/z\}$.
  Again by IH, $\Gamma_{01} + r_0 \cdot \Gamma, \Gamma_{02}\{c/z\} \vdash B\{c/z\} : s$.
  Next, we have, $A\{\Gamma_2\}\{c/z\}\{\Gamma_1\} =_\beta B\{\Gamma_2\}\{c/z\}\{\Gamma_1\}$.
  So $(A\{c/z\})\{\Gamma_2\{c/z\}\}\{\Gamma_1\} =_\beta (B\{c/z\})\{\Gamma_2\{c/z\}\}\{\Gamma_1\}$.

As such, $A\{c/z\}\{\Gamma_1 + r \cdot \Gamma, \Gamma_2\{c/z\}\} =_\beta B\{c/z\}\{\Gamma_1 + r \cdot \Gamma, \Gamma_2\{c/z\}\}$.
This case, then, follows by rule GRAD-CONV-DEF.

□

Lemma C.33 (Lemma 4.22) if $\Gamma_1, z = c :^r C, \Gamma_2 \vdash a : A$ and $\Gamma \vdash c : C$ where $\lfloor\Gamma\rfloor = \lfloor\Gamma_1\rfloor$, then $\Gamma_1 + r \cdot \Gamma, \Gamma_2\{c/z\} \vdash a\{c/z\} : A\{c/z\}$.

Proof. Follows from Lemma C.32. □

Lemma C.34 (Elaboration) If $H \Vdash_o \Gamma'$ and $\Gamma, \Gamma_0 \vdash_o a : A$ where $\lfloor\Gamma\rfloor = \lfloor\Gamma'\rfloor$, then $H_H \Vdash_e \Gamma'_H$ and $\Gamma_H, \Gamma_0 \vdash_e a : A$.

Proof. By induction on $H \Vdash_o \Gamma$.
For the base case, given $\Gamma_0 \vdash_o a : A$, we need to show $\Gamma_0 \vdash_e a : A$. Follows from the fact that any valid derivation in the original system remains so in the extended system.
For the cons-case, given $H, x \overset{q}{\mapsto} (\Gamma''|b|B) \Vdash_o \Gamma', x :^q B$ where $H \Vdash_o \Gamma' + q \cdot \Gamma''$ and $\Gamma'' \vdash_o b : B$. Further, $\Gamma, x :^r B, \Gamma_0 \vdash_o a : A$ where $\lfloor\Gamma\rfloor = \lfloor\Gamma'\rfloor$.
Need to show: $H_H, x \overset{q}{\mapsto} (\Gamma''_H|b|B) \Vdash_e \Gamma'_H, x = b :^q B$ and $\Gamma_H, x = b :^r B, \Gamma_0 \vdash_e a : A$.
By IH, $H_H \Vdash_e \Gamma'_H + q \cdot \Gamma''_H$ and $\Gamma''_H \vdash_e b : B$.
Then, by rule COMPAT-CONS-DEF, $H_H, x \overset{q}{\mapsto} (\Gamma''_H|b|B) \Vdash_e \Gamma'_H, x = b :^q B$.
Also by IH, $\Gamma_H, x :^r B, \Gamma_0 \vdash_e a : A$.
Then, by Lemma C.30, $\Gamma_H, x = b :^r B, \Gamma_0 \vdash_e a : A$. □

Lemma C.35 (Lemma 4.23) If $H \Vdash_o \Gamma$ and $\Gamma \vdash_o a : A$, then $H_H \Vdash_e \Gamma_H$ and $\Gamma_H \vdash_e a : A$.

Proof. Follows from Lemma C.34 by setting $\Gamma' := \Gamma$ and $\Gamma_0 := \varnothing$. □

Lemma C.36 (Typing correspondence) If $\Gamma \vdash_e a : A$ such that Γ does not include any definitions, then $\Gamma \vdash_o a : A$.

Proof. By induction on $\Gamma \vdash_e a : A$. We present some of the interesting cases below.

- Rule GRAD-VAR-DEF. Have: $0 \cdot \Gamma, x = a :^1 A \vdash_e x : A$ where $\Gamma \vdash_e a : A$.
  The lemma statement does not apply since the context includes a definition.

- Rule GRAD-WEAK-DEF. Have: $\Gamma, y = b :^0 B \vdash_e a : A$ where $\Gamma \vdash_e a : A$ and $\Gamma_0 \vdash_e b : B$, where $\lfloor\Gamma_0\rfloor = \lfloor\Gamma\rfloor$.
  The lemma statement does not apply since the context includes a definition.

- Rule GRAD-CONV-DEF. Have: $\Gamma \vdash_e a : B$ where $\Gamma \vdash_e a : A$ and $A\{\Gamma\} =_\beta B\{\Gamma\}$ and $\Gamma_0 \vdash_e B : s$, where $\lfloor\Gamma_0\rfloor = \lfloor\Gamma\rfloor$. Further, Γ does not include any definitions.
  Need to show: $\Gamma \vdash_o a : B$.
  By IH, $\Gamma \vdash_o a : A$ and $\Gamma_0 \vdash_o B : s$.

Next, since Γ does not include any definitions, so $A =_\beta B$.

This case, then, follows by rule GRAD-CONV.

$\square$

Lemma C.37 (Lemma 4.24) If $H \Vdash_e \Gamma$ and $\Gamma \vdash_e a : A$, then $\varnothing \vdash_o a\{H\} : A\{H\}$.

Proof. By induction on $H \Vdash_e \Gamma$.
For the base case, given $\varnothing \vdash_e a : A$, we need to show $\varnothing \vdash_o a : A$. Follows from Lemma C.36.
For the cons-case, we have, $H, y \overset{q}{\mapsto} (\Gamma_2|b|B) \Vdash_e \Gamma_1, y = b :^q B$ where $H \Vdash_e \Gamma_1 + q \cdot \Gamma_2$ and $\Gamma_2 \vdash_e b : B$. Further, $\Gamma_1, y = b :^q B \vdash_e a : A$.
Need to show: $\varnothing \vdash_o a\{b/y\}\{H\} : A\{b/y\}\{H\}$.
Since $\Gamma_1, y = b :^q B \vdash_e a : A$ and $\Gamma_2 \vdash_e b : B$, by Lemma C.33, $\Gamma_1 + q \cdot \Gamma_2 \vdash_e a\{b/y\} : A\{b/y\}$.
This case, then, follows by IH. $\square$

Lemma C.38 (Typing inversion) GRAD, extended with definitions, satisfies the following inversion lemmas:

- If $\Gamma \vdash v : A$ and $A\{\Gamma\} =_\beta \Pi x :^q B_1.B_2$ and v is a value, then exists A_1, A_2 and a_2 such that
  - $v = \lambda^q x : A_1.a_2$
  - $\Gamma, x :^q A_1 \vdash a_2 : A_2$
  - $\Gamma_0 \vdash \Pi x :^q A_1.A_2 : s$ where $\lfloor \Gamma_0 \rfloor = \lfloor \Gamma \rfloor$
  - $A\{\Gamma\} =_\beta \Pi x :^q A_1\{\Gamma\}.A_2\{\Gamma\}$
- If $\Gamma \vdash v : A$ and $A\{\Gamma\} =_\beta \Sigma x{:}^q B_1.B_2$ and v is a value, then exists A_1, A_2, a_1 and a_2 such that
  - $v = ({a_1}^q, a_2)$
  - $\Gamma_1 \vdash a_1 : A_1$ and $\Gamma_2 \vdash a_2 : A_2\{a_1/x\}$ where $\Gamma <: q \cdot \Gamma_1 + \Gamma_2$
  - $\Gamma_0 \vdash \Sigma x{:}^q A_1.A_2 : s$ where $\lfloor \Gamma_0 \rfloor = \lfloor \Gamma \rfloor$
  - $A\{\Gamma\} =_\beta \Sigma x{:}^q A_1\{\Gamma\}.A_2\{\Gamma\}$

Proof. By induction on $\Gamma \vdash v : A$. The proof is similar to that of Lemma C.26.
So, we just present a few cases below. All these cases pertain to inversion on Π-type.

- Rule GRAD-WEAK-DEF. Have: $\Gamma, z = c :^0 C \vdash v : A$ where $\Gamma \vdash v : A$ and $\Gamma_1 \vdash c : C$, where $\lfloor \Gamma_1 \rfloor = \lfloor \Gamma \rfloor$.
  Further, v is a value and $A\{\Gamma, z = c :^0 C\} =_\beta \Pi x :^q B_1.B_2$.
  Observe that since $\Gamma \vdash v : A$ and $z \notin \operatorname{dom}\Gamma$, so $z \notin \operatorname{fv} A$.
  Then, $A\{\Gamma, z = c :^0 C\} = A\{\Gamma\}$. As such, $A\{\Gamma\} =_\beta \Pi x :^q B_1.B_2$.
  Now, using IH, we have:
  - $v = \lambda^q x : A_1.a_2$.

- $\Gamma, x :^q A_1 \vdash a_2 : A_2$.
  By Lemma C.31, $\Gamma, z = c :^0 C, x :^q A_1 \vdash a_2 : A_2$.
- $\Gamma_0 \vdash \Pi x :^q A_1.A_2 : s$ where $\lfloor \Gamma_0 \rfloor = \lfloor \Gamma \rfloor$.
  By Lemma C.31, $\Gamma_0, z = c :^0 C \vdash \Pi x :^q A_1.A_2 : s$.
- $A\{\Gamma\} =_\beta \Pi x :^q A_1\{\Gamma\}.A_2\{\Gamma\}$.
  Then, $A\{\Gamma, z = c :^0 C\} =_\beta \Pi x :^q A_1\{\Gamma, z = c :^0 C\}.A_2\{\Gamma, z = c :^0 C\}$, since z is fresh.

- Rule GRAD-LAM. Have: $\Gamma \vdash \lambda^{q_0} x : A_1.a_2 : \Pi x :^{q_0} A_1.A_2$ where $\Gamma, x :^{q_0} A_1 \vdash a_2 : A_2$ and $\Gamma_0 \vdash \Pi x :^{q_0} A_1.A_2 : s$, where $\lfloor \Gamma_0 \rfloor = \lfloor \Gamma \rfloor$. Further, $\Pi x :^{q_0} A_1\{\Gamma\}.A_2\{\Gamma\} =_\beta \Pi x :^q B_1.B_2$.
  Now, by Lemma C.25, $q_0 = q$.
  So, we have:
  - $v = \lambda^q x : A_1.a_2$.
  - $\Gamma, x :^q A_1 \vdash a_2 : A_2$.
  - $\Gamma_0 \vdash \Pi x :^q A_1.A_2 : s$.
  - $\Pi x :^q A_1\{\Gamma\}.A_2\{\Gamma\} =_\beta \Pi x :^q A_1\{\Gamma\}.A_2\{\Gamma\}$, by reflexivity.

- Rule GRAD-CONV-DEF. Have: $\Gamma \vdash v : B$ where $\Gamma \vdash v : A$ and $A\{\Gamma\} =_\beta B\{\Gamma\}$ and $\Gamma_1 \vdash B : s$, where $\lfloor \Gamma_1 \rfloor = \lfloor \Gamma \rfloor$. Further, v is a value and $B\{\Gamma\} =_\beta \Pi x :^q B_1.B_2$.
  Then, by transitivity, $A\{\Gamma\} =_\beta \Pi x :^q B_1.B_2$.
  Using IH, we have:
  - $v = \lambda^q x : A_1.a_2$.
  - $\Gamma, x :^q A_1 \vdash a_2 : A_2$.
  - $\Gamma_0 \vdash \Pi x :^q A_1.A_2 : s$ where $\lfloor \Gamma_0 \rfloor = \lfloor \Gamma \rfloor$.
  - $A\{\Gamma\} =_\beta \Pi x :^q A_1\{\Gamma\}.A_2\{\Gamma\}$.
    By symmetry and transitivity, $B\{\Gamma\} =_\beta \Pi x :^q A_1\{\Gamma\}.A_2\{\Gamma\}$.

□

Lemma C.39 (Regularity) In GRAD extended with definitions, if $\Gamma \vdash a : A$, then $\exists s$ such that $A =_\beta s$ or $\Gamma_0 \vdash A : s$ where $\lfloor \Gamma_0 \rfloor = \lfloor \Gamma \rfloor$.

Proof. By induction on $\Gamma \vdash a : A$.
The proof is similar to that of Lemma C.27. Below, we present the cases that are different.

- Rule GRAD-VAR-DEF. Have: $0 \cdot \Gamma, x = a :^1 A \vdash x : A$ where $\Gamma \vdash a : A$.
  By IH, $\exists s$ such that $A =_\beta s$ or $\Gamma_0 \vdash A : s$ where $\lfloor \Gamma_0 \rfloor = \lfloor \Gamma \rfloor$.
  Therefore, we have, $A =_\beta s$ or by rule GRAD-WEAK-DEF, $\Gamma_0, x = a :^0 A \vdash A : s$.
  The case follows.

- Rule GRAD-WEAK-DEF. Have: $\Gamma, y = b :^0 B \vdash a : A$ where $\Gamma \vdash a : A$ and $\Gamma_1 \vdash b : B$, where $\lfloor \Gamma_1 \rfloor = \lfloor \Gamma \rfloor$.
  By IH, $\exists s$ such that $A =_\beta s$ or $\Gamma_0 \vdash A : s$ where $\lfloor \Gamma_0 \rfloor = \lfloor \Gamma \rfloor$.
  Therefore, we have, $A =_\beta s$ or by rule GRAD-WEAK-DEF, $\Gamma_0, y = b :^0 B \vdash A : s$.
  The case follows.

- Rule GRAD-CONV-DEF. Have: $\Gamma \vdash a : B$ where $\Gamma \vdash a : A$ and $A\{\Gamma\} =_\beta B\{\Gamma\}$ and $\Gamma_0 \vdash B : s$, where $\lfloor \Gamma_0 \rfloor = \lfloor \Gamma \rfloor$.
  The case follows from premises.

□

Lemma C.40 (Compatibility and inserting definitions) If $H \Vdash (\Gamma_1 + q \cdot \Gamma_0), \Gamma_2$ and $\Gamma_0 \vdash b : B$, then $H_1, y \overset{q}{\mapsto} (\Gamma_0 | b | B), H_2' \Vdash \Gamma_1, y = b :^q B, \Gamma_2$ where y is fresh and $H = H_1, H_2$ such that $|H_1| = |\Gamma_1|$ and H_2' is H_2 with each embedded context weakened by inserting $y = b :^0 B$ at the $|H_1|$-th position.

Proof. By induction on the length of Γ_2. Follow the proof of Lemma C.11. □

Lemma C.41 (Compatibility and splitting) If $H \Vdash \Gamma_1 + \Gamma_2$, then there exists H_1 and H_2 such that $H_1 \Vdash \Gamma_1$ and $H_2 \Vdash \Gamma_2$ and $H = H_1 + H_2$.

Proof. By induction on $H \Vdash \Gamma_1 + \Gamma_2$. Follow the proof of Lemma C.12. □

Lemma C.42 (Compatibility and order) If $H_1 \Vdash \Gamma_1$ and $\Gamma_1 <: \Gamma_2$, then there exists H_2 such that $H_1 <: H_2$ and $H_2 \Vdash \Gamma_2$.

Proof. By induction on $H_1 \Vdash \Gamma_1$. Follow the proof of Lemma C.13. □

Theorem C.43 (Invariance) If $H \Vdash \Gamma_0 + r \cdot \Gamma$ and $\Gamma \vdash a : A$ and $r <: 1$, then either a is a value or there exists H', $\mathbf{u'}$, Γ_0' and Γ' such that:

- $[H]\, a \Rightarrow^r_S [H' ; \mathbf{u'} ; \Gamma_0']\, a'$
- $\Gamma' \vdash a' : A$
- $H' \Vdash (\Gamma_0, 0 \cdot \Gamma_0') + r \cdot \Gamma'$
- $r \cdot (\overline{\Gamma} \diamond \mathbf{0}) \; + \; \mathbf{u'} \times \langle H' \rangle \; + \; \mathbf{0} \diamond \overline{\Gamma_0'} \quad <: \quad r \cdot \overline{\Gamma'} \; + \; \mathbf{u'} \; + \; (\mathbf{0} \diamond \overline{\Gamma_0'}) \times \langle H' \rangle$

Proof. By induction on $\Gamma \vdash a : A$. The cases where a is a value follow immediately. We present some of the other cases below.

- Rule GRAD-VAR-DEF. Have: $0 \cdot \Gamma_1, x = a :^1 A \vdash x : A$ where $\Gamma_1 \vdash a : A$.
  Further, $H \Vdash \Gamma_0 + r \cdot (0 \cdot \Gamma_1, x = a :^1 A)$ where $r <: 1$.
  Then, $\Gamma_0 = \Gamma_{01}, x = a :^q A$. As such, $H \Vdash \Gamma_{01}, x = a :^{q+r} A$.
  By inversion, $H = H_1, x \overset{q+r}{\mapsto} (\Gamma_2|a|A)$ where $H_1 \Vdash \Gamma_{01} + (q + r) \cdot \Gamma_2$ and $\Gamma_2 \vdash a : A$.
  Then, we can show the clauses of the theorem one by one:

  - $[H_1, x \overset{q+r}{\mapsto} (\Gamma_2|a|A)]\, x \Rightarrow^r_S [H_1, x \overset{q}{\mapsto} (\Gamma_2|a|A)\,;\, \mathbf{0}^{|H_1|} \diamond r\,;\, \varnothing]\, a$, by rule HEAP-VAR.
  - Next, we have, $\Gamma_2 \vdash a : A$. By rule GRAD-WEAK-DEF, $\Gamma_2, x = a :^0 A \vdash a : A$.
  - Now, by rule COMPAT-CONS-DEF, $H_1, x \overset{q}{\mapsto} (\Gamma_2|a|A) \Vdash (\Gamma_{01} + r \cdot \Gamma_2), x = a :^q A$.
    As such, $H_1, x \overset{q}{\mapsto} (\Gamma_2|a|A) \Vdash (\Gamma_{01}, x = a :^q A) + r \cdot (\Gamma_2, x = a :^0 A)$.
  - For the fourth clause, need to show: $r \cdot (\mathbf{0} \diamond 1) + (\mathbf{0} \diamond r) \times \begin{pmatrix} \langle H_1 \rangle & \mathbf{0}^\top \\ \overline{\Gamma_2} & 0 \end{pmatrix} <: r \cdot (\overline{\Gamma_2} \diamond 0) \ + \ (\mathbf{0} \diamond r)$.
    The clause follows by reflexivity.

- Rule GRAD-WEAK-DEF. Have: $\Gamma_1, y = b :^0 B \vdash a : A$ where $\Gamma_1 \vdash a : A$ and $\Gamma_2' \vdash b : B$ where $\lfloor \Gamma_2' \rfloor = \lfloor \Gamma_1 \rfloor$.
  Further, $H \Vdash \Gamma_0 + r \cdot (\Gamma_1, y = b :^0 B)$ where $r <: 1$.
  Then, $\Gamma_0 = \Gamma_{01}, y = b :^q B$. As such, $H \Vdash (\Gamma_{01} + r \cdot \Gamma_1), y = b :^q B$.
  By inversion, $H = H_1, y \overset{q}{\mapsto} (\Gamma_2|b|B)$ where $H_1 \Vdash \Gamma_{01} + r \cdot \Gamma_1 + q \cdot \Gamma_2$ and $\Gamma_2 \vdash b : B$.
  Now, if a is a value, this case follows immediately.
  Otherwise, since $H_1 \Vdash \Gamma_{01} + q \cdot \Gamma_2 + r \cdot \Gamma_1$ and $\Gamma_1 \vdash a : A$, by IH, there exists H_1', $\mathbf{u}'$, Γ_0' and Γ_1' such that:

  - $[H_1]\, a \Rightarrow^r_{S \cup \{y\}} [H_1'\,;\, \mathbf{u}'\,;\, \Gamma_0']\, a'$
  - $\Gamma_1' \vdash a' : A$
  - $H_1' \Vdash (\Gamma_{01} + q \cdot \Gamma_2, 0 \cdot \Gamma_0') + r \cdot \Gamma_1'$
  - $r \cdot (\overline{\Gamma_1} \diamond \mathbf{0}) + \mathbf{u}' \times \langle H_1' \rangle + \mathbf{0} \diamond \overline{\Gamma_0'} <: r \cdot \overline{\Gamma_1'} + \mathbf{u}' + (\mathbf{0} \diamond \overline{\Gamma_0'}) \times \langle H_1' \rangle$

  Using these clauses, we show the corresponding clauses of the theorem:

  - By weakening, $[H_1, y \overset{q}{\mapsto} (\Gamma_2|b|B)]\, a \Rightarrow^r_S [H_{11}', y \overset{q}{\mapsto} (\Gamma_2|b|B), H_{12}'\,;\, \mathbf{u}_1' \diamond 0 \diamond \mathbf{u}_2'\,;\, \Gamma_0']\, a'$, where $H_1' = H_{11}', H_{12}'$ and $\mathbf{u}' = \mathbf{u}_1' \diamond \mathbf{u}_2'$ and $|H_{11}'| = |\mathbf{u}_1'| = |H_1|$.
    Next, in the above judgment, we replace H_{12}' with H_{12}'', which is basically H_{12}' with each embedded context weakened by inserting $y = b :^0 B$ at the $|H_1|$-th position. This replacement is permitted by Lemma C.6 and is needed to ensure compatibility.
  - By Lemma C.31, $\Gamma_{11}', y = b :^0 B, \Gamma_{12}' \vdash a' : A$, where $\Gamma_1' = \Gamma_{11}', \Gamma_{12}'$ and $|\Gamma_{11}'| = |H_1|$.
  - $H_{11}', y \overset{q}{\mapsto} (\Gamma_2|b|B), H_{12}'' \Vdash (\Gamma_{01}, y = b :^q B, 0 \cdot \Gamma_0') + r \cdot (\Gamma_{11}', y = b :^0 B, \Gamma_{12}')$, by Lemma C.40.
  - The fourth clause follows upon inserting 0 at the $|H_1|$-th position on both sides of the fourth clause above.

- We do not need to consider rule GRAD-VAR and rule GRAD-WEAK because given $H \Vdash \Gamma_0 + r \cdot \Gamma$, we know that every assumption in Γ is a definition of the form $x = a :^q A$ but the judgments derived by the mentioned rules include assumptions that are not of this form.

- Rule GRAD-APP. Have: $\Gamma_1 + q \cdot \Gamma_2 \vdash b\ a^q : B\{a/y\}$ where $\Gamma_1 \vdash b : \Pi y :^q A.B$ and $\Gamma_2 \vdash a : A$.
  Further, $H \Vdash \Gamma_0 + r \cdot (\Gamma_1 + q \cdot \Gamma_2)$ where $r <: 1$.
  There are two cases to consider.

  - b is not a value.
    Since $H \Vdash \Gamma_0 + r \cdot q \cdot \Gamma_2 + r \cdot \Gamma_1$ and $\Gamma_1 \vdash b : \Pi y :^q A.B$, there exists H', $\mathbf{u'}$, Γ'_0 and Γ'_1 such that:
    * $[H]\, b \Rightarrow^r_{S \cup \mathrm{fv}\, a} [H' ; \mathbf{u'} ; \Gamma'_0]\, b'$
    * $\Gamma'_1 \vdash b' : \Pi y :^q A.B$
    * $H' \Vdash (\Gamma_0 + r \cdot q \cdot \Gamma_2, 0 \cdot \Gamma'_0) + r \cdot \Gamma'_1$
    * $r \cdot (\overline{\Gamma_1} \diamond \mathbf{0}) + \mathbf{u'} \times \langle H' \rangle + \mathbf{0} \diamond \overline{\Gamma'_0} <: r \cdot \overline{\Gamma'_1} + \mathbf{u'} + (\mathbf{0} \diamond \overline{\Gamma'_0}) \times \langle H' \rangle$

    Using these clauses, we show the corresponding clauses of the theorem:
    * $[H]\, b\ a^q \Rightarrow^r_S [H' ; \mathbf{u'} ; \Gamma'_0]\, b'\ a^q$, by rule HEAP-APPL.
    * By weakening and rule GRAD-APP, $\Gamma'_1 + q \cdot (\Gamma_2, 0 \cdot \Gamma'_0) \vdash b'\ a^q : B\{a/y\}$.
    * Next, $H' \Vdash (\Gamma_0, 0 \cdot \Gamma'_0) + r \cdot (\Gamma'_1 + q \cdot (\Gamma_2, 0 \cdot \Gamma'_0))$, by rearrangement of the third clause above.
    * The fourth clause follows upon adding $r \cdot q \cdot (\overline{\Gamma_2} \diamond \mathbf{0})$ to both sides of the fourth clause above.
  - b is a value.
    Since $\Gamma_1 \vdash b : \Pi y :^q A.B$, so by Lemma C.38, there exists b', A' and B' such that:
    * $b = \lambda^q y : A'.b'$
    * $\Gamma_1, y :^q A' \vdash b' : B'$
    * $\Gamma' \vdash \Pi y :^q A'.B' : s'$ where $\lfloor \Gamma' \rfloor = \lfloor \Gamma_1 \rfloor$.
    * $\Pi y :^q A\{\Gamma_1\}.B\{\Gamma_1\} =_\beta \Pi y :^q A'\{\Gamma_1\}.B'\{\Gamma_1\}$.
      Then, by Lemma C.25, $A\{\Gamma_1\} =_\beta A'\{\Gamma_1\}$.
      Now since $\Gamma_2 \vdash a : A$, by rule GRAD-CONV-DEF, $\Gamma_2 \vdash a : A'$.
      Again by Lemma C.25, $B\{\Gamma_1\}\{a\{\Gamma_1\}/y\} =_\beta B'\{\Gamma_1\}\{a\{\Gamma_1\}/y\}$.
      Then, $B\{a/y\}\{\Gamma_1\} =_\beta B'\{a/y\}\{\Gamma_1\}$.

    Again since $\Gamma_1 \vdash b : \Pi y :^q A.B$, so by Lemma C.39, exists s such that $\Gamma_{10} \vdash \Pi y :^q A.B : s$ where $\lfloor \Gamma_{10} \rfloor = \lfloor \Gamma_1 \rfloor$. By inversion, there exists s_1 and s_2 such that $\Gamma_{101} \vdash A : s_1$ and $\Gamma_{102}, y :^{q_0} A \vdash B : s_2$ where $\Gamma_{10} <: \Gamma_{101} + \Gamma_{102}$. Then, by Lemma C.32, $\Gamma_{102} + q_0 \cdot \Gamma_2 \vdash B\{a/y\} : s_2$.
    Now using what we derived above, we show the clauses of the theorem one by one:
    * $[H]\, (\lambda^q y : A'.b')\ a \Rightarrow^r_S [H, y \overset{r \cdot q}{\mapsto} (\Gamma_2 | a | A') ; \mathbf{0} ; y = a :^{r \cdot q} A']\, b'$, by rule HEAP-APPBETA-DEF (assuming y *fresh*).
    * Next we have, $\Gamma_1, y :^q A' \vdash b' : B'$.
      Since $\Gamma_2 \vdash a : A'$, so by Lemma C.30, $\Gamma_1, y = a :^q A' \vdash b' : B'$.
      We also have, $B\{a/y\}\{\Gamma_1\} =_\beta B'\{a/y\}\{\Gamma_1\}$.
      So $B'\{\Gamma_1, y = a :^q A'\} =_\beta B\{a/y\}\{\Gamma_1, y = a :^q A'\}$.
      Further we have, $\Gamma_{102} + q_0 \cdot \Gamma_2 \vdash B\{a/y\} : s_2$.
      So $\Gamma_{102} + q_0 \cdot \Gamma_2, y = a :^0 A' \vdash B\{a/y\} : s_2$.
      Then by rule GRAD-CONV-DEF, $\Gamma_1, y = a :^q A' \vdash b' : B\{a/y\}$.

* Next we have, $H \Vdash \Gamma_0 + r \cdot \Gamma_1 + r \cdot q \cdot \Gamma_2$ and $\Gamma_2 \vdash a : A'$.
  By rule Compat-Cons-Def, $H, y \overset{r \cdot q}{\mapsto} (\Gamma_2 | a | A') \Vdash (\Gamma_0 + r \cdot \Gamma_1), y = a :^{r \cdot q} A'$.
  In other words, $H, y \overset{r \cdot q}{\mapsto} (\Gamma_2 | a | A') \Vdash (\Gamma_0, y = a :^0 A') + r \cdot (\Gamma_1, y = a :^q A')$.
* Next, we need to show: $r \cdot ((\overline{\Gamma_1} + q \cdot \overline{\Gamma_2}) \diamond 0) + (\mathbf{0} \diamond r \cdot q) \ <: \ r \cdot (\overline{\Gamma_1} \diamond q) + (\mathbf{0} \diamond r \cdot q) \times \begin{pmatrix} \frac{\langle H \rangle}{\Gamma_2} & \mathbf{0}^{\top} \\ & 0 \end{pmatrix}$.
  This clause follows by reflexivity.

- Rule Grad-LetPair. Have: $\Gamma_1 + \Gamma_2 \vdash \mathbf{let}\,(x_1{}^q, x_2) = a \ \mathbf{in} \ b : B\{a/z\}$ where $\Gamma_1 \vdash a : \Sigma x_1 :^q A_1 . A_2$ and $\Gamma_2, x_1 :^q A_1, x_2 :^1 A_2 \vdash b : B\{(x_1{}^q, x_2)/z\}$ and $\Gamma_{10}, z :^{r_0} \Sigma x_1 :^q A_1 . A_2 \vdash B : s$, where $\lfloor \Gamma_{10} \rfloor = \lfloor \Gamma_1 \rfloor$.
  Further, $H \Vdash \Gamma_0 + r \cdot (\Gamma_1 + \Gamma_2)$ where $r <: 1$.
  There are two cases to consider.

  - a is not a value.
    Since $H \Vdash \Gamma_0 + r \cdot \Gamma_2 + r \cdot \Gamma_1$ and $\Gamma_1 \vdash a : \Sigma x_1 :^q A_1 . A_2$, by IH, there exists H', $\mathbf{u'}$, Γ'_0 and Γ'_1 such that:
    * $[H]\, a \Rightarrow^{r}_{S \cup \{z\} \cup (\text{fv}\, b - \{x_1, x_2\})} [H' \,;\, \mathbf{u'} \,;\, \Gamma'_0]\, a'$
    * $\Gamma'_1 \vdash a' : \Sigma x_1 :^q A_1 . A_2$
    * $H' \Vdash (\Gamma_0 + r \cdot \Gamma_2, 0 \cdot \Gamma'_0) + r \cdot \Gamma'_1$
    * $r \cdot (\overline{\Gamma_1} \diamond \mathbf{0}) + \mathbf{u'} \times \langle H' \rangle + \mathbf{0} \diamond \overline{\Gamma'_0} <: r \cdot \overline{\Gamma'_1} + \mathbf{u'} + (\mathbf{0} \diamond \overline{\Gamma'_0}) \times \langle H' \rangle$

    Using the above clauses, we show the corresponding clauses of the theorem:
    * $[H]\, \mathbf{let}\,(x_1{}^q, x_2) = a \ \mathbf{in} \ b \Rightarrow^{r}_{S} [H' \,;\, \mathbf{u'} \,;\, \Gamma'_0]\, \mathbf{let}\,(x_1{}^q, x_2) = a' \ \mathbf{in} \ b$, by left rule.
    * By weakening and rule Grad-LetPair, $\Gamma'_1 + (\Gamma_2, 0 \cdot \Gamma'_0) \vdash \mathbf{let}\,(x_1{}^q, x_2) = a' \ \mathbf{in} \ b : B\{a'/z\}$.
      Now since $[H]\, a \Rightarrow [H']\, a'$, by Lemma C.7, $a\{H\} =_\beta a'\{H'\}$.
      Next, we show: $B\{a'/z\}\{\Gamma'_1\} =_\beta B\{a/z\}\{\Gamma'_1\}$.
      Note $B\{a'/z\}\{\Gamma'_1\} = B\{a'/z\}\{H'\} = B\{H'\}\{a'\{H'\}/z\} = B\{\Gamma_2, \Gamma'_0\}\{a'\{H'\}/z\}$.
      But $B\{\Gamma_2, \Gamma'_0\}\{a'\{H'\}/z\} = B\{\Gamma_2\}\{a'\{H'\}/z\}$, since $\text{fv}\, B \cap \text{dom}\, \Gamma'_0 = \varnothing$.
      Then, $B\{\Gamma_2\}\{a'\{H'\}/z\} =_\beta B\{\Gamma_2\}\{a\{H\}/z\} = B\{\Gamma_2\}\{a\{\Gamma_2\}/z\} = B\{a/z\}\{\Gamma_2\}$.
      But $B\{a/z\}\{\Gamma_2\} = B\{a/z\}\{\Gamma_2, \Gamma'_0\}$, since $\text{fv}\, B \cap \text{dom}\, \Gamma'_0 = \ \text{fv}\, a \cap \text{dom}\, \Gamma'_0 = \varnothing$.
      Therefore, $B\{a'/z\}\{\Gamma'_1\} =_\beta B\{a/z\}\{\Gamma'_1\}$.
      Next by substitution, $\Gamma_{10} + r_0 \cdot \Gamma_1 \vdash B\{a/z\} : s$.
      By weakening, $\Gamma_{10} + r_0 \cdot \Gamma_1, 0 \cdot \Gamma'_0 \vdash B\{a/z\} : s$.
      Then by rule Grad-Conv-Def, $\Gamma'_1 + (\Gamma_2, 0 \cdot \Gamma'_0) \vdash \mathbf{let}\,(x_1{}^q, x_2) = a' \ \mathbf{in} \ b : B\{a/z\}$.
    * Next, $H' \Vdash (\Gamma_0, 0 \cdot \Gamma'_0) + r \cdot (\Gamma'_1 + (\Gamma_2, 0 \cdot \Gamma'_0))$, by rearrangement of the third clause.
    * The fourth clause follows upon adding $r \cdot (\overline{\Gamma_2} \diamond \mathbf{0})$ to both sides of the fourth clause above.

  - a is a value.
    Since $\Gamma_1 \vdash a : \Sigma x_1 :^q A_1 . A_2$, so by Lemma C.38, there exists a_1, a_2, A'_1 and A'_2 such that:
    * $a = (a_1{}^q, a_2)$
    * $\Gamma_{11} \vdash a_1 : A'_1$ and $\Gamma_{12} \vdash a_2 : A'_2\{a_1/x_1\}$ where $\Gamma_1 <: q \cdot \Gamma_{11} + \Gamma_{12}$
    * $\Gamma_{20} \vdash \Sigma x_1 :^q A'_1 . A'_2 : s$ where $\lfloor \Gamma_{20} \rfloor = \lfloor \Gamma_1 \rfloor$

- $\Sigma x_1{:}^q A_1\{\Gamma_1\}.A_2\{\Gamma_1\} =_\beta \Sigma x_1{:}^q A_1'\{\Gamma_1\}.A_2'\{\Gamma_1\}$.
  Then, by Lemma C.25, $A_1\{\Gamma_1\} =_\beta A_1'\{\Gamma_1\}$.
  Now since $\Gamma_{11} \vdash a : A_1'$, by rule GRAD-CONV-DEF, $\Gamma_{11} \vdash a : A_1$.
  Again by Lemma C.25, $A_2\{\Gamma_1\}\{a_1\{\Gamma_1\}/x_1\} =_\beta A_2'\{\Gamma_1\}\{a_1\{\Gamma_1\}/x_1\}$.
  Then, $A_2\{a_1/x_1\}\{\Gamma_1\} =_\beta A_2'\{a_1/x_1\}\{\Gamma_1\}$.
  So by rule GRAD-CONV-DEF, $\Gamma_{12} \vdash a_2 : A_2\{a_1/x_1\}$.

Next, we have, $H \Vdash \Gamma_0 + r \cdot (\Gamma_1 + \Gamma_2)$. But $\Gamma_0 + r \cdot (\Gamma_1 + \Gamma_2) <: \Gamma_0 + r \cdot (q \cdot \Gamma_{11} + \Gamma_{12} + \Gamma_2)$.
So, by Lemma C.42, $\exists H'$ such that $H <: H'$ and $H' \Vdash \Gamma_0 + r \cdot (q \cdot \Gamma_{11} + \Gamma_{12} + \Gamma_2)$.
Now using what we derived above, we show the clauses of the theorem one by one:

- By rule HEAP-LETPAIRBETA-DEF, $[H']\,\mathbf{let}\,(x_1{}^q, x_2) = (a_1{}^q, a_2)\,\mathbf{in}\,b \Rightarrow^r_S [H', x_1 \overset{r \cdot q}{\mapsto} (\Gamma_{11}|a_1|A_1), x_2 \overset{r}{\mapsto} (\Gamma_{12}, x_1 = a_1 :^0 A_1|a_2|A_2)\,;\,\mathbf{0}\,;\, x_1 = a_1 :^{r \cdot q} A_1, x_2 = a_2 :^r A_2]\,b$ (assuming x_1, x_2 *fresh*).
  Since $H <: H'$, by rule HEAP-SUBL, $[H]\,\mathbf{let}\,(x_1{}^q, x_2) = (a_1{}^q, a_2)\,\mathbf{in}\,b \Rightarrow^r_S [H', x_1 \overset{r \cdot q}{\mapsto} (\Gamma_{11}|a_1|A_1), x_2 \overset{r}{\mapsto} (\Gamma_{12}, x_1 = a_1 :^0 A_1|a_2|A_2)\,;\,\mathbf{0}\,;\, x_1 = a_1 :^{r \cdot q} A_1, x_2 = a_2 :^r A_2]\,b$.
- Next we have, $\Gamma_2, x_1 :^q A_1, x_2 :^1 A_2 \vdash b : B\{(x_1{}^q, x_2)/z\}$.
  By Lemma C.30, $\Gamma_2, x_1 = a_1 :^q A_1, x_2 :^1 A_2 \vdash b : B\{(x_1{}^q, x_2)/z\}$.
  Next since $\Gamma_{12} \vdash a_2 : A_2\{a_1/x_1\}$, so $\Gamma_{12}, x_1 = a_1 :^0 A_1 \vdash a_2 : A_2$.
  Then using Lemma C.30 again, $\Gamma_2, x_1 = a_1 :^q A_1, x_2 = a_2 :^1 A_2 \vdash b : B\{(x_1{}^q, x_2)/z\}$.
  So by rule GRAD-CONV-DEF, $\Gamma_2, x_1 = a_1 :^q A_1, x_2 = a_2 :^1 A_2 \vdash b : B\{(a_1{}^q, a_2)/z\}$.
- Next we have, $H' \Vdash \Gamma_0 + r \cdot (q \cdot \Gamma_{11} + \Gamma_{12} + \Gamma_2)$ and $\Gamma_{11} \vdash a_1 : A_1$.
  By rule COMPAT-CONS-DEF, $H', x_1 \overset{r \cdot q}{\mapsto} (\Gamma_{11}|a_1|A_1) \Vdash (\Gamma_0 + r \cdot \Gamma_2 + r \cdot \Gamma_{12}), x_1 = a_1 :^{r \cdot q} A_1$.
  Again we have, $\Gamma_{12}, x_1 = a_1 :^0 A_1 \vdash a_2 : A_2$.
  By rule COMPAT-CONS-DEF, $H', x_1 \overset{r \cdot q}{\mapsto} (\Gamma_{11}|a_1|A_1), x_2 \overset{r}{\mapsto} (\Gamma_{12}, x_1 = a_1 :^0 A_1|a_2|A_2) \Vdash (\Gamma_0 + r \cdot \Gamma_2), x_1 = a_1 :^{r \cdot q} A_1, x_2 = a_2 :^r A_2$.
  In other words, $H', x_1 \overset{r \cdot q}{\mapsto} (\Gamma_{11}|a_1|A_1), x_2 \overset{r}{\mapsto} (\Gamma_{12}, x_1 = a_1 :^0 A_1|a_2|A_2) \Vdash (\Gamma_0, x_1 = a_1 :^0 A_1, x_2 = a_2 :^0 A_2) + r \cdot (\Gamma_2, x_1 = a_1 :^q A_1, x_2 = a_2 :^1 A_2)$.
- Next, need to show: $r \cdot ((\overline{\Gamma_1} + \overline{\Gamma_2}) \diamond 0 \diamond 0) + (\mathbf{0} \diamond (r \cdot q) \diamond r) \;<:\; r \cdot (\overline{\Gamma_2} \diamond q \diamond 1) + (\mathbf{0} \diamond (r \cdot q) \diamond r) \times \begin{pmatrix} \langle H' \rangle & \mathbf{0}^\top & \mathbf{0}^\top \\ \overline{\Gamma_{11}} & 0 & 0 \\ \overline{\Gamma_{12}} & 0 & 0 \end{pmatrix}$.
  This clause follows from $\Gamma_1 <: q \cdot \Gamma_{11} + \Gamma_{12}$.

- Rule GRAD-CONV-DEF. Have: $\Gamma \vdash a : B$ where $\Gamma \vdash a : A$ and $A\{\Gamma\} =_\beta B\{\Gamma\}$ and $\Gamma_1 \vdash B : s$, where $\lfloor \Gamma_1 \rfloor = \lfloor \Gamma \rfloor$.
  Further, $H \Vdash \Gamma_0 + r \cdot \Gamma$ where $r <: 1$.
  Now, if a is a value, this case follows immediately.
  Otherwise, since $H \Vdash \Gamma_0 + r \cdot \Gamma$ and $\Gamma \vdash a : A$, by IH, there exists H', $\mathbf{u}'$, Γ_0' and Γ' such that:
  - $[H]\,a \Rightarrow^r_S [H'\,;\,\mathbf{u}'\,;\,\Gamma_0']\,a'$
  - $\Gamma' \vdash a' : A$.
    Now, $A\{\Gamma'\} = A\{\Gamma, \Gamma_0'\} = A\{\Gamma\}$, since $\mathrm{fv}\,A \cap \mathrm{dom}\,\Gamma_0' = \varnothing$.
    But $A\{\Gamma\} =_\beta B\{\Gamma\} = B\{\Gamma, \Gamma_0'\}$, since $\mathrm{fv}\,B \cap \mathrm{dom}\,\Gamma_0' = \varnothing$.
    Then, $A\{\Gamma'\} =_\beta B\{\Gamma'\}$.

Next since $\Gamma_1 \vdash B : s$, by weakening, $\Gamma_1, 0 \cdot \Gamma_0' \vdash B : s$.
So by rule GRAD-CONV-DEF, $\Gamma' \vdash a' : B$.

– $H' \Vdash (\Gamma_0, 0 \cdot \Gamma_0') + r \cdot \Gamma'$
– $r \cdot (\overline{\Gamma} \diamond \mathbf{0}) \;+\; \mathbf{u'} \times \langle H' \rangle \;+\; \mathbf{0} \diamond \overline{\Gamma_0'} \;\;<:\;\; r \cdot \overline{\Gamma'} \;+\; \mathbf{u'} \;+\; (\mathbf{0} \diamond \overline{\Gamma_0'}) \times \langle H' \rangle$

This case follows.

□

Theorem C.44 (Theorem 4.25) If $H \Vdash \Gamma$ and $\Gamma \vdash a : A$, then either a is a value or there exists H', $\mathbf{u'}$, Γ_0' and Γ' such that:

- $[H]\, a \Rightarrow_S^1 [H' ; \mathbf{u'} ; \Gamma_0']\, a'$
- $\Gamma' \vdash a' : A$
- $H' \Vdash \Gamma'$
- $\overline{\Gamma} \diamond \mathbf{0} \;+\; \mathbf{u'} \times \langle H' \rangle \;+\; \mathbf{0} \diamond \overline{\Gamma_0'} \;\;<:\;\; \overline{\Gamma'} \;+\; \mathbf{u'} \;+\; (\mathbf{0} \diamond \overline{\Gamma_0'}) \times \langle H' \rangle$

Proof. Follows from Theorem C.43 by setting $r := 1$ and $\Gamma_0 := 0 \cdot \Gamma$. □

Corollary C.45 (Corollary 4.26) In GRAD parametrized over a zero-unusable semiring, if $\varnothing \vdash f : \Pi x {:}^0 s.x$ and $\varnothing \vdash A : s$, then $f\ A^0$ must diverge.

Proof. Let us assume towards contradiction that for some $\varnothing \vdash A : s$, the term $f\ A^0$ normalizes.
Now, let $B := \mathbf{Unit}$ and $C := \mathbf{Bool}$.
Since $f\ A^0$ normalizes, f must reduce to a value.
Therefore, there exists some k_0 and b such that $[\varnothing] f\ A^0 \twoheadrightarrow_{k_0}^1 [H]\, (\lambda^0 x.b)\ A^0$.
But then, $[\varnothing] f\ B^0 \twoheadrightarrow_{k_0}^1 [H]\, (\lambda^0 x.b)\ B^0$ and $[\varnothing] f\ C^0 \twoheadrightarrow_{k_0}^1 [H]\, (\lambda^0 x.b)\ C^0$.
Next, $[H]\, (\lambda^0 x.b)\ A^0 \Rightarrow [H, x \overset{0}{\mapsto} A]\, b$ (say, *x fresh*).
Similarly, $[H]\, (\lambda^0 x.b)\ B^0 \Rightarrow [H, x \overset{0}{\mapsto} B]\, b$ and $[H]\, (\lambda^0 x.b)\ C^0 \Rightarrow [H, x \overset{0}{\mapsto} C]\, b$.
Now, since $f\ A^0$ normalizes, there exists k_1 and v such that $[H, x \overset{0}{\mapsto} A]\, b \twoheadrightarrow_{k_1}^1 [H', x \overset{0}{\mapsto} A, H'']\, v$.
Then, $[H, x \overset{0}{\mapsto} B]\, b \twoheadrightarrow_{k_1}^1 [H', x \overset{0}{\mapsto} B, H'']\, v$ and $[H, x \overset{0}{\mapsto} C]\, b \twoheadrightarrow_{k_1}^1 [H', x \overset{0}{\mapsto} C, H'']\, v$, by Lemma C.19.
Now by soundness theorem C.44, v must have types $B := \mathbf{Unit}$ and $C := \mathbf{Bool}$.
But there is no value that has types both **Unit** and **Bool**. A contradiction.
The corollary follows. □

Corollary C.46 (Corollary 4.27) In GRAD parametrized over a zero-unusable semiring and a strongly normalizing PTS, if $\varnothing \vdash f : \Pi x {:}^0 s. \Pi y {:}^1 x.x$ and $\varnothing \vdash A : s$ and $\varnothing \vdash a : A$, then $f\ A^0\ a^1 =_\beta a$.

Proof. To see why, let us consider how $f\ A^0\ a^1$ reduces.

Suppose $A_1 \coloneqq \textbf{Unit}$ and $A_2 \coloneqq \textbf{Bool}$. Further, suppose $a_1 \coloneqq \textbf{unit}$ and $a_2 \coloneqq \textbf{true} \triangleq \textbf{inj}_1\ \textbf{unit}$.

Observe that $f\ {A_1}^0\ {a_1}^1$ and $f\ {A_2}^0\ {a_2}^1$ are well-typed.

Now since the parametrizing PTS is strongly normalizing, so f reduces to a value.

Therefore, there exists some k_0 and b such that $[\varnothing] f\ A^0\ a^1 \twoheadrightarrow^1_{k_0} [H_1]\,(\lambda^0 x.b)\ A^0\ a^1$.

But then, $[\varnothing] f\ {A_1}^0\ {a_1}^1 \twoheadrightarrow^1_{k_0} [H_1]\,(\lambda^0 x.b)\ {A_1}^0\ {a_1}^1$ and $[\varnothing] f\ {A_2}^0\ {a_2}^1 \twoheadrightarrow^1_{k_0} [H_1]\,(\lambda^0 x.b)\ {A_2}^0\ {a_2}^1$.

Next, $[H_1]\,(\lambda^0 x.b)\ A^0\ a^1 \Rightarrow [H_1, x \overset{0}{\mapsto} A]\ b\ a^1$ (say *x fresh*).

Similarly, $[H_1]\,(\lambda^0 x.b)\ {A_1}^0\ {a_1}^1 \Rightarrow [H_1, x \overset{0}{\mapsto} A_1]\ b\ {a_1}^1$ and $[H_1]\,(\lambda^0 x.b)\ {A_2}^0\ {a_2}^1 \Rightarrow [H_1, x \overset{0}{\mapsto} A_2]\ b\ {a_2}^1$.

Again since the parametrizing PTS is strongly normalizing, so b reduces to a value.

Therefore, there exists some k_1 and c such that $[H_1, x \overset{0}{\mapsto} A]\ b\ a^1 \twoheadrightarrow^1_{k_1} [H_1', x \overset{0}{\mapsto} A, H_2]\,(\lambda^1 y.c)\ a^1$.

Then, $[H_1, x \overset{0}{\mapsto} A_1]\ b\ {a_1}^1 \twoheadrightarrow^1_{k_1} [H_1', x \overset{0}{\mapsto} A_1, H_2]\,(\lambda^1 y.c)\ {a_1}^1$, by Lemma C.19.

Also, $[H_1, x \overset{0}{\mapsto} A_2]\ b\ {a_2}^1 \twoheadrightarrow^1_{k_1} [H_1', x \overset{0}{\mapsto} A_2, H_2]\,(\lambda^1 y.c)\ {a_2}^1$, by Lemma C.19.

Next, $[H_1', x \overset{0}{\mapsto} A, H_2]\,(\lambda^1 y.c)\ a^1 \Rightarrow [H_1', x \overset{0}{\mapsto} A, H_2, y \overset{1}{\mapsto} a]\ c$ (say *y fresh*).

Similarly, $[H_1', x \overset{0}{\mapsto} A_1, H_2]\,(\lambda^1 y.c)\ {a_1}^1 \Rightarrow [H_1', x \overset{0}{\mapsto} A_1, H_2, y \overset{1}{\mapsto} a_1]\ c$.

And, $[H_1', x \overset{0}{\mapsto} A_2, H_2]\,(\lambda^1 y.c)\ {a_2}^1 \Rightarrow [H_1', x \overset{0}{\mapsto} A_2, H_2, y \overset{1}{\mapsto} a_2]\ c$.

Now since $f\ A^0\ a^1$ normalizes, so the value of y gets looked up.

Till that look-up, the above three reductions are indistinguishable from one another.

Say, the value of y gets looked up after k_2 steps.

In k_2 steps: $[H_1', x \overset{0}{\mapsto} A, H_2, y \overset{1}{\mapsto} a]\ c \twoheadrightarrow^1_{k_2} [H_1'', x \overset{0}{\mapsto} A, H_2', y \overset{1}{\mapsto} a, H_3]\ c'$.

Similarly, $[H_1', x \overset{0}{\mapsto} A_1, H_2, y \overset{1}{\mapsto} a_1]\ c \twoheadrightarrow^1_{k_2} [H_1'', x \overset{0}{\mapsto} A_1, H_2', y \overset{1}{\mapsto} a_1, H_3]\ c'$.

And, $[H_1', x \overset{0}{\mapsto} A_2, H_2, y \overset{1}{\mapsto} a_2]\ c \twoheadrightarrow^1_{k_2} [H_1'', x \overset{0}{\mapsto} A_2, H_2', y \overset{1}{\mapsto} a_2, H_3]\ c'$.

Since the value of y gets looked up at this point, so $c' = y$ or c' is a proper path headed by y, where a path is a series of nested elimination forms headed by a variable.

Now by soundness theorem C.44:

- $\exists \Gamma$ such that $H_1'', x \overset{0}{\mapsto} A, H_2', y \overset{1}{\mapsto} a, H_3 \Vdash \Gamma$ and $\Gamma \vdash c' : A$
- $\exists \Gamma_1$ such that $H_1'', x \overset{0}{\mapsto} A_1, H_2', y \overset{1}{\mapsto} a_1, H_3 \Vdash \Gamma_1$ and $\Gamma_1 \vdash c' : A_1$
- $\exists \Gamma_2$ such that $H_1'', x \overset{0}{\mapsto} A_2, H_2', y \overset{1}{\mapsto} a_2, H_3 \Vdash \Gamma_2$ and $\Gamma_2 \vdash c' : A_2$

Next, if c' is a proper path headed by y, then either $\Gamma_1 \vdash c' : A_1$ or $\Gamma_2 \vdash c' : A_2$ would fail to hold because there is no well-typed proper path that can be headed by both a_1 and a_2 of types **Unit** and **Bool** respectively.

Therefore, $c' = y$.

Hence, $[\varnothing] f\ A^0\ a^1 \twoheadrightarrow^1_k [H_1'', x \overset{0}{\mapsto} A, H_2', y \overset{1}{\mapsto} a, H_3]\ y$, for some k.

Then by Lemma C.7, $f\ A^0\ a^1 =_\beta a$. □

Corollary C.47 (Corollary 4.28) In GRAD parametrized over a one-linear semiring and a strongly normalizing PTS, if $\varnothing \vdash f : \Pi x :^0 s.\Pi y :^1 {}^1x \times x.{}^1x \times x$ and $\varnothing \vdash A : s$ and $\varnothing \vdash a : A$ and $\varnothing \vdash a' : A$, then $f\ A^0\ (a^1, a')^1 =_\beta (a^1, a')$ or $f\ A^0\ (a^1, a')^1 =_\beta (a'^1, a)$.

Proof. To see why, let us consider how $f\ A^0\ (a^1, a')^1$ reduces.
Suppose $A_1 := \textbf{Unit}$ and $A_2 := \textbf{Bool}$.
Further, let $a_1 := \textbf{unit}$ and $a_1' := \textbf{unit}$.
Also, let $a_2 := \textbf{true} \triangleq \textbf{inj}_1\, \textbf{unit}$ and $a_2' := \textbf{false} \triangleq \textbf{inj}_2\, \textbf{unit}$.
Observe that $f\ {A_1}^0\ ({a_1}^1, a_1')^1$ and $f\ {A_2}^0\ ({a_2}^1, a_2')^1$ are well-typed.
Now reasoning as in Corollary C.46, there exists some k_0 and b such that:

- $[\varnothing] f\ A^0\ (a^1, a')^1 \Rrightarrow^1_{k_0} [H_1]\,(\lambda^0 x.b)\ A^0\ (a^1, a')^1$
- $[\varnothing] f\ {A_1}^0\ ({a_1}^1, a_1')^1 \Rrightarrow^1_{k_0} [H_1]\,(\lambda^0 x.b)\ {A_1}^0\ ({a_1}^1, a_1')^1$
- $[\varnothing] f\ {A_2}^0\ ({a_2}^1, a_2')^1 \Rrightarrow^1_{k_0} [H_1]\,(\lambda^0 x.b)\ {A_2}^0\ ({a_2}^1, a_2')^1$

Next, we have,

- $[H_1]\,(\lambda^0 x.b)\ A^0\ (a^1, a')^1 \Rightarrow [H_1, x \overset{0}{\mapsto} A]\ b\ (a^1, a')^1$ (say x *fresh*)
- $[H_1]\,(\lambda^0 x.b)\ {A_1}^0\ ({a_1}^1, a_1')^1 \Rightarrow [H_1, x \overset{0}{\mapsto} A_1]\ b\ ({a_1}^1, a_1')^1$
- $[H_1]\,(\lambda^0 x.b)\ {A_2}^0\ ({a_2}^1, a_2')^1 \Rightarrow [H_1, x \overset{0}{\mapsto} A_2]\ b\ ({a_2}^1, a_2')^1$

Continuing the same argument forward, there exists some k_1 and c such that:

- $[H_1, x \overset{0}{\mapsto} A]\ b\ (a^1, a')^1 \Rrightarrow^1_{k_1} [H_1', x \overset{0}{\mapsto} A, H_2]\,(\lambda^1 y.c)\ (a^1, a')^1$
- $[H_1, x \overset{0}{\mapsto} A_1]\ b\ ({a_1}^1, a_1')^1 \Rrightarrow^1_{k_1} [H_1', x \overset{0}{\mapsto} A_1, H_2]\,(\lambda^1 y.c)\ ({a_1}^1, a_1')^1$
- $[H_1, x \overset{0}{\mapsto} A_2]\ b\ ({a_2}^1, a_2')^1 \Rrightarrow^1_{k_1} [H_1', x \overset{0}{\mapsto} A_2, H_2]\,(\lambda^1 y.c)\ ({a_2}^1, a_2')^1$

Then, we have,

- $[H_1', x \overset{0}{\mapsto} A, H_2]\,(\lambda^1 y.c)\ (a^1, a')^1 \Rightarrow [H_1', x \overset{0}{\mapsto} A, H_2, y \overset{1}{\mapsto} (a^1, a')]\ c$ (say y *fresh*)
- $[H_1', x \overset{0}{\mapsto} A_1, H_2]\,(\lambda^1 y.c)\ ({a_1}^1, a_1')^1 \Rightarrow [H_1', x \overset{0}{\mapsto} A_1, H_2, y \overset{1}{\mapsto} ({a_1}^1, a_1')]\ c$
- $[H_1', x \overset{0}{\mapsto} A_2, H_2]\,(\lambda^1 y.c)\ ({a_2}^1, a_2')^1 \Rightarrow [H_1', x \overset{0}{\mapsto} A_2, H_2, y \overset{1}{\mapsto} ({a_2}^1, a_2')]\ c$

Reasoning as in Corollary C.46, suppose the value of y gets looked up after k_2 steps. In k_2 steps:

- $[H_1', x \overset{0}{\mapsto} A, H_2, y \overset{1}{\mapsto} (a^1, a')]\ c \Rrightarrow^1_{k_2} [H_1'', x \overset{0}{\mapsto} A, H_2', y \overset{1}{\mapsto} (a^1, a'), H_3]\ c'$
- $[H_1', x \overset{0}{\mapsto} A_1, H_2, y \overset{1}{\mapsto} ({a_1}^1, a_1')]\ c \Rrightarrow^1_{k_2} [H_1'', x \overset{0}{\mapsto} A_1, H_2', y \overset{1}{\mapsto} ({a_1}^1, a_1'), H_3]\ c'$
- $[H_1', x \overset{0}{\mapsto} A_2, H_2, y \overset{1}{\mapsto} ({a_2}^1, a_2')]\ c \Rrightarrow^1_{k_2} [H_1'', x \overset{0}{\mapsto} A_2, H_2', y \overset{1}{\mapsto} ({a_2}^1, a_2'), H_3]\ c'$

Since the value of y gets looked up at this point, so $c' = y$ or c' is a proper path headed by y.
If $c' = y$, then by Lemma C.7, $f\ A^0\ (a^1, a')^1 =_\beta (a^1, a')$.
Otherwise, c' is a proper path headed by y.
Owing to well-typedness, c' must be a pair elimination form, i.e., $c' = \textbf{let}\,({z_1}^1, z_2) = y\ \textbf{in}\ c''$, for some c''.
Then, in the next two steps of reduction (assuming z_1 and z_2 are fresh):

- $[H_1'', x \overset{0}{\mapsto} A, H_2', y \overset{1}{\mapsto} (a^1, a'), H_3]\, c' \Rrightarrow_2^1 [H_1'', x \overset{0}{\mapsto} A, H_2', y \overset{0}{\mapsto} (a^1, a'), H_3, z_1 \overset{1}{\mapsto} a, z_2 \overset{1}{\mapsto} a']\, c''$
- $[H_1'', x \overset{0}{\mapsto} A_1, H_2', y \overset{1}{\mapsto} ({a_1}^1, a_1'), H_3]\, c' \Rrightarrow_2^1 [H_1'', x \overset{0}{\mapsto} A_1, H_2', y \overset{0}{\mapsto} ({a_1}^1, a_1'), H_3, z_1 \overset{1}{\mapsto} a_1, z_2 \overset{1}{\mapsto} a_1']\, c''$
- $[H_1'', x \overset{0}{\mapsto} A_2, H_2', y \overset{1}{\mapsto} ({a_2}^1, a_2'), H_3]\, c' \Rrightarrow_2^1 [H_1'', x \overset{0}{\mapsto} A_2, H_2', y \overset{0}{\mapsto} ({a_2}^1, a_2'), H_3, z_1 \overset{1}{\mapsto} a_2, z_2 \overset{1}{\mapsto} a_2']\, c''$

Next we consider how these reductions proceed from here.
By Lemma C.19, the reductions would proceed in the same manner until the value of z_1 or z_2 gets looked up.
Now, say the value of z_1 or z_2 gets looked up after k_3 steps, when the reduct is c'''.
Then, c''' must be either z_1 or z_2 or a proper path headed by one of these variables.
But c''' cannot be z_1 because c''' has a product type and the value of z_1 is **unit** in one of the heaps. Likewise, c''' cannot be z_2.
Further, c''' cannot also be a proper path headed by z_1 (or z_2) because there is no well-typed path that can be headed by both **unit** and **true** (or **false**).
Therefore, the values of z_1 and z_2 do not get looked up.
But since the parametrizing PTS is strongly normalizing, the reductions terminate.
Then, they must terminate at the same reduct (syntactically same but not necessarily as closures).
By soundness theorem C.44, that reduct must be a pair, say $({c_1}^1, c_2)$.
Assuming the reductions terminate in k_4 steps, we have, put concretely,

- $[H_1'', x \overset{0}{\mapsto} A, H_2', y \overset{0}{\mapsto} (a^1, a'), H_3, z_1 \overset{1}{\mapsto} a, z_2 \overset{1}{\mapsto} a']\, c'' \Rrightarrow_{k_4}^1 [H_1''', x \overset{0}{\mapsto} A, H_2'', y \overset{0}{\mapsto} (a^1, a'), H_3', z_1 \overset{1}{\mapsto} a, z_2 \overset{1}{\mapsto} a', H_4]\, ({c_1}^1, c_2)$
- $[H_1'', x \overset{0}{\mapsto} A_1, H_2', y \overset{0}{\mapsto} ({a_1}^1, a_1'), H_3, z_1 \overset{1}{\mapsto} a_1, z_2 \overset{1}{\mapsto} a_1']\, c'' \Rrightarrow_{k_4}^1 [H_1''', x \overset{0}{\mapsto} A_1, H_2'', y \overset{0}{\mapsto} ({a_1}^1, a_1'), H_3', z_1 \overset{1}{\mapsto} a_1, z_2 \overset{1}{\mapsto} a_1', H_4]\, ({c_1}^1, c_2)$
- $[H_1'', x \overset{0}{\mapsto} A_2, H_2', y \overset{0}{\mapsto} ({a_2}^1, a_2'), H_3, z_1 \overset{1}{\mapsto} a_2, z_2 \overset{1}{\mapsto} a_2']\, c'' \Rrightarrow_{k_4}^1 [H_1''', x \overset{0}{\mapsto} A_2, H_2'', y \overset{0}{\mapsto} ({a_2}^1, a_2'), H_3', z_1 \overset{1}{\mapsto} a_2, z_2 \overset{1}{\mapsto} a_2', H_4]\, ({c_1}^1, c_2)$

Now let,

$$H_0 \coloneqq H_1''', x \overset{0}{\mapsto} A, H_2'', y \overset{0}{\mapsto} (a^1, a'), H_3', z_1 \overset{1}{\mapsto} a, z_2 \overset{1}{\mapsto} a', H_4$$
$$H_i \coloneqq H_1''', x \overset{0}{\mapsto} A_1, H_2'', y \overset{0}{\mapsto} ({a_1}^1, a_1'), H_3', z_1 \overset{1}{\mapsto} a_1, z_2 \overset{1}{\mapsto} a_1', H_4$$
$$H_j \coloneqq H_1''', x \overset{0}{\mapsto} A_2, H_2'', y \overset{0}{\mapsto} ({a_2}^1, a_2'), H_3', z_1 \overset{1}{\mapsto} a_2, z_2 \overset{1}{\mapsto} a_2', H_4$$

Then by soundness theorem C.44,

- $\exists \Gamma_0$ such that $H_0 \Vdash \Gamma_0$ and $\Gamma_0 \vdash ({c_1}^1, c_2) : {}^1A \times A$
- $\exists \Gamma_i$ such that $H_i \Vdash \Gamma_i$ and $\Gamma_i \vdash ({c_1}^1, c_2) : {}^1A_1 \times A_1$
- $\exists \Gamma_j$ such that $H_j \Vdash \Gamma_j$ and $\Gamma_j \vdash ({c_1}^1, c_2) : {}^1A_2 \times A_2$

Next by inversion lemma C.38,

- $\exists \Gamma_{01}$ and Γ_{02} such that $\Gamma_{01} \vdash c_1 : A$ and $\Gamma_{02} \vdash c_2 : A$ and $\Gamma_0 <: \Gamma_{01} + \Gamma_{02}$
- $\exists \Gamma_{i\,1}$ and $\Gamma_{i\,2}$ such that $\Gamma_{i\,1} \vdash c_1 : A_1$ and $\Gamma_{i\,2} \vdash c_2 : A_1$ and $\Gamma_i <: \Gamma_{i\,1} + \Gamma_{i\,2}$
- $\exists \Gamma_{j\,1}$ and $\Gamma_{j\,2}$ such that $\Gamma_{j\,1} \vdash c_1 : A_2$ and $\Gamma_{j\,2} \vdash c_2 : A_2$ and $\Gamma_j <: \Gamma_{j\,1} + \Gamma_{j\,2}$

Then by lemmas C.42 and C.41,

- $\exists H_{01}$ and H_{02} such that $H_{01} \Vdash \Gamma_{01}$ and $H_{02} \Vdash \Gamma_{02}$ and $H_0 <: H_{01} + H_{02}$
- $\exists H_{i\,1}$ and $H_{i\,2}$ such that $H_{i\,1} \Vdash \Gamma_{i\,1}$ and $H_{i\,2} \Vdash \Gamma_{i\,2}$ and $H_i <: H_{i\,1} + H_{i\,2}$
- $\exists H_{j\,1}$ and $H_{j\,2}$ such that $H_{j\,1} \Vdash \Gamma_{j\,1}$ and $H_{j\,2} \Vdash \Gamma_{j\,2}$ and $H_j <: H_{j\,1} + H_{j\,2}$

Now let $\mathbf{H} = \{H_{01}, H_{02}, H_{i\,1}, H_{i\,2}, H_{j\,1}, H_{j\,2}\}$.
Observe that any two heaps in $\mathbf{H}$ may differ in the definitions of only four variables, viz., x, y, z_1 and z_2. However, given that the parametrizing semiring is one-linear and both x and y appear at grade 0 in heaps H_0, H_i and H_j, the two variables would also appear at grade 0 in every heap in $\mathbf{H}$. Therefore, as far as reductions are concerned, these variables are unusable. So only the differences in the definitions of z_1, and likewise z_2, matter.
Now let us consider two sets of reductions: reduction of c_1 in heaps H_{01}, $H_{i\,1}$ and $H_{j\,1}$; reduction of c_2 in heaps H_{02}, $H_{i\,2}$ and $H_{j\,2}$.
Since the parametrizing PTS is strongly normalizing, all these reductions terminate.
Since the reductions of c_1 terminate, either the value of z_1 or the value of z_2 gets looked up.
Below, we consider the two cases separately.

- The value of z_1 gets looked up during the reduction of c_1.
  Then, z_1 must appear at grade 1 in the heaps H_{01}, $H_{i\,1}$ and $H_{j\,1}$. As such, z_1 would appear at grade 0 in the heaps H_{02}, $H_{i\,2}$ and $H_{j\,2}$. Now, when the value of z_1 gets looked up, the redex must be either z_1 or a proper path headed by z_1. But since there is no well-typed path that can be headed by both a_1 and a_2, the redex must be z_1. Then, for some k_5, we have,

$$[H_{01}]\, c_1 \twoheadrightarrow^1_{k_5} [H'_{01}]\, a$$
$$[H_{i\,1}]\, c_1 \twoheadrightarrow^1_{k_5} [H'_{i\,1}]\, a_1$$
$$[H_{j\,1}]\, c_1 \twoheadrightarrow^1_{k_5} [H'_{j\,1}]\, a_2$$

By Lemma C.7, $c_1\{H_{01}\} =_\beta a\{H'_{01}\} = a$.
Next, since the reductions of c_2 in heaps H_{02}, $H_{i\,2}$ and $H_{j\,2}$ terminate, so either the value of z_1 or z_2 gets looked up. But the value of z_1 cannot be looked up since it appears at grade 0 in the above three heaps. Therefore, the value of z_2 gets looked up. Now, when the value of z_2 gets looked up, the redex must be either z_2 or a proper path headed by z_2. But since there is no well-typed path that can be headed by both a'_1 and a'_2, the redex must be z_2. Then, for some k_6, we have,

$$[H_{02}]\, c_2 \twoheadrightarrow^1_{k_6} [H'_{02}]\, a'$$
$$[H_{i\,2}]\, c_2 \twoheadrightarrow^1_{k_6} [H'_{i\,2}]\, a'_1$$
$$[H_{j\,2}]\, c_2 \twoheadrightarrow^1_{k_6} [H'_{j\,2}]\, a'_2$$

By Lemma C.7, $c_2\{H_{02}\} =_\beta a'\{H'_{02}\} = a'$.
But $[\varnothing] f\ A^0\ (a^1, a')^1 \twoheadrightarrow^1_k [H_0]\, ({c_1}^1, c_2)$, for some k.
So by Lemma C.7, $f\ A^0\ (a^1, a')^1 =_\beta (c_1\{H_0\}^1, c_2\{H_0\})$.
Therefore, $f\ A^0\ (a^1, a')^1 =_\beta (a^1, a')$.

- The value of z_2 gets looked up during the reduction of c_1.
  Then, z_2 must appear at grade 1 in the heaps H_{01}, $H_{i\,1}$ and $H_{j\,1}$. As such, z_2 would appear at grade 0 in the heaps H_{02}, $H_{i\,2}$ and $H_{j\,2}$. Now, when the value of z_2 gets looked up, the redex must be either z_2 or a proper path headed by z_2. But since there is no well-typed path that can be headed by both a'_1 and a'_2, the redex must be z_2. Then, for some k_7, we have,

$$[H_{01}]\, c_1 \twoheadrightarrow^1_{k_7} [H''_{01}]\, a'$$
$$[H_{i\,1}]\, c_1 \twoheadrightarrow^1_{k_7} [H''_{i\,1}]\, a'_1$$
$$[H_{j\,1}]\, c_1 \twoheadrightarrow^1_{k_7} [H''_{j\,1}]\, a'_2$$

By Lemma C.7, $c_1\{H_{01}\} =_\beta a'\{H''_{01}\} = a'$.
Next, since the reductions of c_2 in heaps H_{02}, $H_{i\,2}$ and $H_{j\,2}$ terminate, so either the value of z_1 or z_2 gets looked up. But the value of z_2 cannot be looked up since it appears at grade 0 in the above three heaps. Therefore, the value of z_1 gets looked up. Now, when the value of z_1 gets looked up, the redex must be either z_1 or a proper path headed by z_1. But since there is no well-typed path that can be headed by both a_1 and a_2, the redex must be z_1. Then, for some k_8, we have,

$$[H_{02}]\, c_2 \twoheadrightarrow^1_{k_8} [H''_{02}]\, a$$
$$[H_{i\,2}]\, c_2 \twoheadrightarrow^1_{k_8} [H''_{i\,2}]\, a_1$$
$$[H_{j\,2}]\, c_2 \twoheadrightarrow^1_{k_8} [H''_{j\,2}]\, a_2$$

By Lemma C.7, $c_2\{H_{02}\} =_\beta a\{H''_{02}\} = a$.
But $[\varnothing] f\ A^0\ (a^1, a')^1 \twoheadrightarrow^1_k [H_0]\, ({c_1}^1, c_2)$.

So by Lemma C.7, $f\ A^0\ (a^1, a')^1 =_\beta (c_1\{H_0\}^1, c_2\{H_0\})$.
Therefore, $f\ A^0\ (a^1, a')^1 =_\beta ({a'}^1, a)$.

The corollary follows. $\square$

C.8 Proofs of Lemmas/Theorems Stated in Section 4.8

Lemma C.48 (Lemma 4.29) If $\Gamma \vdash a : A$ in GRAD($\mathbb{B}_\geq$), then $\vec{\Gamma} \vdash \overline{a} :^1 \overline{A}$ in DDC$^\top$($\mathbb{B}_\geq$), parametrized over the same PTS. Further, if $\vdash a \rightsquigarrow a'$ in GRAD($\mathbb{B}_\geq$), then $\vdash \overline{a} \rightsquigarrow \overline{a'}$ in DDC$^\top$($\mathbb{B}_\geq$).

Proof. First, let $\vdash a \rightsquigarrow a'$ in GRAD($\mathbb{B}_\geq$). By a straightforward induction on $\vdash a \rightsquigarrow a'$, we show that $\vdash \overline{a} \rightsquigarrow \overline{a'}$ in DDC$^\top$($\mathbb{B}_\geq$). Below, we present two cases of the induction.

- Rule S-APPBETA. Have: $\vdash (\lambda^q x : A.b)\ a^q \rightsquigarrow b\{a/x\}$.
  Need to show: $\vdash (\lambda^q x : \overline{A}.\overline{b})\ (\overline{a})^q \rightsquigarrow \overline{b}\{\overline{a}/x\}$.
  Follows by β-rule for application.
- Rule S-LETPAIRBETA. Have: $\vdash \mathbf{let}\,({x_1}^q, x_2) = ({a_1}^q, a_2)\ \mathbf{in}\ b \rightsquigarrow b\{a_1/x_1\}\{a_2/x_2\}$.
  Need to show: $\vdash \mathbf{let}\,({x_1}^q, x_2) = (\overline{a_1}^q, \overline{a_2})\ \mathbf{in}\ \overline{b} \rightsquigarrow \overline{b}\{\overline{a_1}/x_1\}\{\overline{a_2}/x_2\}$.
  Follows by β-rule for pair.

Next, let $\Gamma \vdash a : A$ in GRAD($\mathbb{B}_\geq$). By induction on $\Gamma \vdash a : A$, we show that $\vec{\Gamma} \vdash \overline{a} :^1 \overline{A}$ in DDC$^\top$($\mathbb{B}_\geq$), parametrized over the same PTS. Below, we present some of the interesting cases of the induction.

- Rule GRAD-VAR. Have: $0 \cdot \Gamma, x :^1 A \vdash x : A$ where $\Gamma \vdash A : s$.
  Need to show: $0 \sqcup \vec{\Gamma}, x :^1 \overline{A} \vdash x :^1 \overline{A}$. (Note that $q \sqcup \vec{\Gamma}$ denotes the point-wise join of q with the grades on the assumptions in $\vec{\Gamma}$.)
  By IH, $\vec{\Gamma} \vdash \overline{A} :^1 s$.
  By multiplication, $0 \sqcup \vec{\Gamma} \vdash \overline{A} :^0 s$.
  This case, then, follows by rule DCT-VAR.
- Rule GRAD-WEAK. Have: $\Gamma, y :^0 B \vdash a : A$ where $\Gamma \vdash a : A$ and $\Gamma_0 \vdash B : s$, where $\lfloor \Gamma_0 \rfloor = \lfloor \Gamma \rfloor$.
  Need to show: $\vec{\Gamma}, y :^0 \overline{B} \vdash \overline{a} :^1 \overline{A}$.
  By IH, $\vec{\Gamma} \vdash \overline{a} :^1 \overline{A}$.
  Again by IH, $\vec{\Gamma_0} \vdash \overline{B} :^1 s$.
  By multiplication, $0 \sqcup \vec{\Gamma_0} \vdash \overline{B} :^0 s$. By narrowing, $\vec{\Gamma} \vdash \overline{B} :^0 s$.
  This case, then, follows by rule DCT-WEAK.
- Rule GRAD-AXIOM. Have: $\varnothing \vdash s_1 : s_2$ where $\mathcal{A}(s_1, s_2)$.
  Need to show: $\varnothing \vdash s_1 :^1 s_2$.

Note here that both the calculi are parametrized by the same PTS.

As such, this case follows by rule DCT-Axiom.

- Rule Grad-Pi. Have: $\Gamma_1 + \Gamma_2 \vdash \Pi x :^q A.B : s_3$ where $\Gamma_1 \vdash A_1 : s_1$ and $\Gamma_2, x :^r A \vdash B : s_2$ such that $\mathcal{R}(s_1, s_2, s_3)$ and $\lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor$.
  Need to show: $\vec{\Gamma_1} \sqcap \vec{\Gamma_2} \vdash \Pi x :^q \overline{A}.\overline{B} :^1 \overline{s_3}$. (Note that $\vec{\Gamma_1} \sqcap \vec{\Gamma_2}$ denotes the point-wise meet of the grades on the assumptions in $\vec{\Gamma_1}$ and $\vec{\Gamma_2}$.)
  By IH, $\vec{\Gamma_1} \vdash \overline{A_1} :^1 \overline{s_1}$ and $\vec{\Gamma_2}, x :^r \overline{A} \vdash \overline{B} :^1 \overline{s_2}$.
  By narrowing, $\vec{\Gamma_1} \sqcap \vec{\Gamma_2} \vdash \overline{A} :^1 \overline{s_1}$ and $\vec{\Gamma_1} \sqcap \vec{\Gamma_2}, x :^r \overline{A} \vdash \overline{B} :^1 \overline{s_2}$.
  Now since both the calculi are parametrized by the same PTS, so $\mathcal{R}(\overline{s_1}, \overline{s_2}, \overline{s_3})$.
  This case, then, follows by rule DCT-Pi.

- Rule Grad-Lam. Have: $\Gamma \vdash \lambda^q x : A.b : \Pi x :^q A.B$ where $\Gamma, x :^q A \vdash b : B$ and $\Gamma_0 \vdash \Pi x :^q A.B : s$, where $\lfloor \Gamma_0 \rfloor = \lfloor \Gamma \rfloor$.
  Need to show: $\vec{\Gamma} \vdash \lambda^q x : \overline{A}.\overline{b} :^1 \Pi x :^q \overline{A}.\overline{B}$.
  By IH, $\vec{\Gamma}, x :^q \overline{A} \vdash \overline{b} :^1 \overline{B}$.
  But since $q = 1 \sqcup q$, so $\vec{\Gamma}, x :^{1 \sqcup q} \overline{A} \vdash \overline{b} :^1 \overline{B}$.
  Again by IH, $\vec{\Gamma_0} \vdash \Pi x :^q \overline{A}.\overline{B} :^1 s$.
  Then, by multiplication and narrowing, $\vec{\Gamma} \vdash \Pi x :^q \overline{A}.\overline{B} :^0 s$.
  From here, this case follows by rule DCT-Lam.

- Rule Grad-App. Have: $\Gamma_1 + q \cdot \Gamma_2 \vdash b\ a^q : B\{a/x\}$ where $\Gamma_1 \vdash b : \Pi x :^q A.B$ and $\Gamma_2 \vdash a : A$ such that $\lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor$.
  Need to show: $\vec{\Gamma_1} \sqcap (q \sqcup \vec{\Gamma_2}) \vdash \overline{b}\ \overline{a}^q :^1 \overline{B}\{\overline{a}/x\}$.
  By IH, $\vec{\Gamma_1} \vdash \overline{b} :^1 \Pi x :^q \overline{A}.\overline{B}$.
  By narrowing, $\vec{\Gamma_1} \sqcap (q \sqcup \vec{\Gamma_2}) \vdash \overline{b} :^1 \Pi x :^q \overline{A}.\overline{B}$.
  Again by IH, $\vec{\Gamma_2} \vdash \overline{a} :^1 \overline{A}$.
  Then, by multiplication, $q \sqcup \vec{\Gamma_2} \vdash \overline{a} :^q \overline{A}$.
  Next by narrowing, $\vec{\Gamma_1} \sqcap (q \sqcup \vec{\Gamma_2}) \vdash \overline{a} :^q \overline{A}$.
  But since $q = 1 \sqcup q$, so $\vec{\Gamma_1} \sqcap (q \sqcup \vec{\Gamma_2}) \vdash \overline{a} :^{1 \sqcup q} \overline{A}$.
  This case, then, follows by rule DCT-App.

- Rule Grad-Conv. Have: $\Gamma \vdash a : B$ where $\Gamma \vdash a : A$ and $A =_\beta B$ and $\Gamma_0 \vdash B : s$, where $\lfloor \Gamma_0 \rfloor = \lfloor \Gamma \rfloor$.
  Need to show: $\vec{\Gamma} \vdash \overline{a} :^1 \overline{B}$.
  By IH, $\vec{\Gamma} \vdash \overline{a} :^1 \overline{A}$.
  Again by IH, $\vec{\Gamma_0} \vdash \overline{B} :^1 s$.
  Then, by multiplication and narrowing, $\vec{\Gamma} \vdash \overline{B} :^0 s$.
  Now, the equality relation used in the conversion rule of $\text{DDC}^\top$ is β-equivalence.
  We have already shown that if $\vdash c_1 \rightsquigarrow c_2$ in $\text{Grad}(\mathbb{B}_\geq)$, then $\vdash \overline{c_1} \rightsquigarrow \overline{c_2}$ in $\text{DDC}^\top(\mathbb{B}_\geq)$.
  Therefore, if $A =_\beta B$ in $\text{Grad}(\mathbb{B}_\geq)$, then $\overline{A} =_\beta \overline{B}$ in $\text{DDC}^\top(\mathbb{B}_\geq)$.
  From here, this case follows by rule DCT-Conv.

- Rule GRAD-PAIR. Have: $q \cdot \Gamma_1 + \Gamma_2 \vdash (a^q, b) : \Sigma x{:}^q A.B$ where $\Gamma_1 \vdash a : A$ and $\Gamma_2 \vdash b : B\{a/x\}$ and $\Gamma_0 \vdash \Sigma x{:}^q A.B : s$ such that $\lfloor\Gamma_1\rfloor = \lfloor\Gamma_2\rfloor = \lfloor\Gamma_0\rfloor$.
  Need to show: $(q \sqcup \vec{\Gamma_1}) \sqcap \vec{\Gamma_2} \vdash (\overline{a}^q, \overline{b}) :^1 \Sigma x{:}^q \overline{A}.\overline{B}$.
  By IH, $\vec{\Gamma_1} \vdash \overline{a} :^1 \overline{A}$.
  By multiplication, $q \sqcup \vec{\Gamma_1} \vdash \overline{a} :^q \overline{A}$.
  By narrowing, $(q \sqcup \vec{\Gamma_1}) \sqcap \vec{\Gamma_2} \vdash \overline{a} :^q \overline{A}$.
  But since $q = 1 \sqcup q$, so $(q \sqcup \vec{\Gamma_1}) \sqcap \vec{\Gamma_2} \vdash \overline{a} :^{1 \sqcup q} \overline{A}$.
  Again by IH, $\vec{\Gamma_2} \vdash \overline{b} :^1 \overline{B}\{a/x\}$.
  By narrowing, $(q \sqcup \vec{\Gamma_1}) \sqcap \vec{\Gamma_2} \vdash \overline{b} :^1 \overline{B}\{a/x\}$.
  Next, by IH, $\vec{\Gamma_0} \vdash \Sigma x{:}\overline{A}.\overline{B} :^1 s$.
  Then, by multiplication and narrowing, $(q \sqcup \vec{\Gamma_1}) \sqcap \vec{\Gamma_2} \vdash \Sigma x{:}\overline{A}.\overline{B} :^0 s$.
  From here, this case follows by rule DCT-PAIR.

- Rule GRAD-LETPAIR. Have: $\Gamma_1 + \Gamma_2 \vdash \textbf{let}\,(x^q, y) = a \textbf{ in } b : B\{a/z\}$ where $\Gamma_1 \vdash a : \Sigma x{:}^q A_1.A_2$ and $\Gamma_2, x{:}^q A_1, y{:}^1 A_2 \vdash b : B\{(x^q, y)/z\}$ and $\Gamma_0, z{:}^r \Sigma x{:}^q A_1.A_2 \vdash B : s$ such that $\lfloor\Gamma_1\rfloor = \lfloor\Gamma_2\rfloor = \lfloor\Gamma_0\rfloor$.
  Need to show: $\vec{\Gamma_1} \sqcap \vec{\Gamma_2} \vdash \textbf{let}\,(x^q, y) = \overline{a} \textbf{ in } \overline{b} :^1 \overline{B}\{\overline{a}/z\}$.
  By IH, $\vec{\Gamma_1} \vdash \overline{a} :^1 \Sigma x{:}^q \overline{A_1}.\overline{A_2}$.
  By narrowing, $\vec{\Gamma_1} \sqcap \vec{\Gamma_2} \vdash \overline{a} :^1 \Sigma x{:}^q \overline{A_1}.\overline{A_2}$.
  Again by IH, $\vec{\Gamma_2}, x{:}^q \overline{A_1}, x_2{:}^1 \overline{A_2} \vdash \overline{b} :^1 \overline{B}\{(x^q, y)/z\}$.
  By narrowing, $\vec{\Gamma_1} \sqcap \vec{\Gamma_2}, x{:}^q \overline{A_1}, x_2{:}^1 \overline{A_2} \vdash \overline{b} :^1 \overline{B}\{(x^q, y)/z\}$.
  But since $q = 1 \sqcup q$, so $\vec{\Gamma_1} \sqcap \vec{\Gamma_2}, x{:}^{1 \sqcup q} \overline{A_1}, x_2{:}^1 \overline{A_2} \vdash \overline{b} :^1 \overline{B}\{(x^q, y)/z\}$.
  Next, by IH, $\vec{\Gamma_0}, z{:}^r \Sigma x{:}^q \overline{A_1}.\overline{A_2} \vdash \overline{B} :^1 s$.
  Then, by multiplication and narrowing, $\vec{\Gamma_1} \sqcap \vec{\Gamma_2}, z{:}^0 \Sigma x{:}^q \overline{A_1}.\overline{A_2} \vdash \overline{B} :^0 s$.
  From here, this case follows by rule DCT-LETPAIR.

□

Appendix D

Combined Linearity and Dependency Analysis in Pure Type Systems

D.1 Proof of a Proposition Stated in Section 5.1

Proposition D.1 Graded-context type systems [Abel and Bernardy, 2020, Atkey, 2018, Brunel et al., 2014, Choudhury et al., 2021, Ghica and Smith, 2014, Orchard et al., 2019, Petricek et al., 2014] cannot derive a monadic join operator.

Proof. Graded-context type systems mentioned in the proposition vary slightly in their design. However, all of them contain a core graded calculus. Below, we first present this Core Graded Calculus, GCore, and thereafter show that GCore cannot derive a monadic join operator.

GCore is parametrized by an arbitrary preordered semiring $\mathcal{Q} = (Q, +, \cdot, 0, 1 <:)$. The grammar and typing rules of GCore($\mathcal{Q}$) appear in Figures D.1 and D.2 respectively. The operations on contexts are defined as in Section 5.2.1.

$$
\begin{aligned}
\text{grades}, q \in Q &::= 0 \mid 1 \mid q_1 + q_2 \mid q_1 \cdot q_2 \mid \dots \\
\text{types}, A, B &::= \mathbf{Bool} \mid A \multimap B \mid !_q\ A \\
\text{terms}, a, b &::= x \mid \lambda x : A.b \mid b\ a \mid !_q\ a \mid \mathbf{let}\ !_q x = a\ \mathbf{in}\ b \\
&\quad \mid \mathbf{true} \mid \mathbf{false} \mid \mathbf{if}\ b\ \mathbf{then}\ a_1\ \mathbf{else}\ a_2 \\
\text{contexts}, \Gamma &::= \varnothing \mid \Gamma, x :^q A
\end{aligned}
$$

Figure D.1: Grammar of GC($\mathcal{Q}$)

$\boxed{\Gamma \vdash a : A}$ *(Typing rules)*

$$\text{GC-Var}\quad \frac{}{0 \cdot \Gamma_1, x :^1 A, 0 \cdot \Gamma_2 \vdash x : A}$$

$$\text{GC-Lam}\quad \frac{\Gamma, x :^1 A \vdash b : B}{\Gamma \vdash \lambda x : A.b : A \multimap B}$$

$$\text{GC-App}\quad \frac{\Gamma_1 \vdash b : A \multimap B \qquad \Gamma_2 \vdash a : A \qquad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor}{\Gamma_1 + \Gamma_2 \vdash b\ a : B}$$

$$\text{GC-ExpIntro}\quad \frac{\Gamma \vdash a : A}{q \cdot \Gamma \vdash !_q\ a :!_q A}$$

$$\text{GC-ExpElim}\quad \frac{\Gamma_1 \vdash a :!_q A \qquad \Gamma_2, x :^q A \vdash b : B \qquad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor}{\Gamma_1 + \Gamma_2 \vdash \mathbf{let}\ !_q x = a\ \mathbf{in}\ b : B}$$

$$\text{GC-True}\quad \frac{}{0 \cdot \Gamma \vdash \mathbf{true} : \mathbf{Bool}}$$

$$\text{GC-False}\quad \frac{}{0 \cdot \Gamma \vdash \mathbf{false} : \mathbf{Bool}}$$

$$\text{GC-If}\quad \frac{\Gamma_1 \vdash b : \mathbf{Bool} \qquad \Gamma_2 \vdash a_1 : A \qquad \Gamma_2 \vdash a_2 : A \qquad \lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor}{\Gamma_1 + \Gamma_2 \vdash \mathbf{if}\ b\ \mathbf{then}\ a_1\ \mathbf{else}\ a_2 : A}$$

$$\text{GC-Sub}\quad \frac{\Gamma_1 \vdash a : A \qquad \Gamma_2 <: \Gamma_1}{\Gamma_2 \vdash a : A}$$

FIGURE D.2: Typing rules for GC($\mathcal{Q}$)

Now, towards contradiction, assume that GCORE($\mathcal{Q}$) can derive a monadic join operator. Then, for any A, there exists a closed non-constant function of type $!_{q_1}\ !_{q_2}\ A \to !_{q_1 \cdot q_2}\ A$ for all $q_1, q_2 \in \mathcal{Q}$. By model-theoretic arguments, we shall show that for some $\mathcal{Q}$, A, q_1 and q_2, any function having such a type must be constant.

Fix $\mathcal{Q}$ to be the following preordered semiring. The underlying set is $\{0, 1, k_1, k_2\}$ and the semiring operations and preorder are defined as follows:

$$\begin{aligned} q_1 + q_2 &\triangleq k_2 \text{ if } q_1, q_2 \notin \{0\} \\ q_1 \cdot q_2 &\triangleq k_2 \text{ if } q_1, q_2 \notin \{0, 1\} \\ q_1 <: q_2 &\triangleq q_1 = q_2 \end{aligned}$$

Check that the above definitions satisfy the axioms for preordered semirings. Now, note that $k_1 \cdot k_1 = k_2$. Below, we shall show that there exists no non-constant function of type $!_{k_1} !_{k_1} \mathbf{Bool} \to !_{k_2} \mathbf{Bool}$ in GCORE($\mathcal{Q}$).

Towards contradiction, suppose such a function, $\varnothing \vdash f : !_{k_1}\ !_{k_1}\ \mathbf{Bool} \to !_{k_2}\ \mathbf{Bool}$ exists. Then, $f\ (!_{k_1}\ !_{k_1}\ \mathbf{true})$ and $f\ (!_{k_1}\ !_{k_1}\ \mathbf{false})$ would reduce to different values (assume a call-by-value reduction). Without loss of generality, say $f\ (!_{k_1}\ !_{k_1}\ \mathbf{true}) \leadsto^* !_{k_2}\ \mathbf{true}$ and $f\ (!_{k_1}\ !_{k_1}\ \mathbf{false}) \leadsto^* !_{k_2}\ \mathbf{false}$, where $\leadsto^*$ is the multistep call-by-value reduction relation. Then, for any sound interpretation, $[\![_]\!]_{\mathcal{M}}$, of GCORE($\mathcal{Q}$) in a model $\mathcal{M}$, we should have: $[\![f\ (!_{k_1}\ !_{k_1}\ \mathbf{true})]\!]_{\mathcal{M}} = [\![!_{k_2}\ \mathbf{true}]\!]_{\mathcal{M}}$ and $[\![f\ (!_{k_1}\ !_{k_1}\ \mathbf{false})]\!]_{\mathcal{M}} = [\![!_{k_2}\ \mathbf{false}]\!]_{\mathcal{M}}$. We construct a sound model of GCORE($\mathcal{Q}$) where these equalities lead to a contradiction.

Let **Set** be the category of sets and functions. Note that **Set** is a symmetric monoidal category with the monoidal product given by cartesian product. Now, any $\mathcal{Q}$-graded linear exponential comonad on **Set** provides a sound interpretation of GCORE($\mathcal{Q}$) [Katsumata, 2018]. We define !, a $\mathcal{Q}$-graded linear exponential

comonad on **Set**, as follows:

$$!(0) = !(k_1) = *$$
$$!(1) = !(k_2) = \mathbf{Id}$$

Here, **Id** is the identity functor and $*$ is the functor that maps every object to the terminal object. The morphisms associated with ! are as expected.

Now, interpreting using !, we have:

$$[\![f\ (!_{k_1}\ !_{k_1}\ \mathbf{true})]\!]_{(\mathbf{Set},!)} = [\![f\ (!_{k_1}\ !_{k_1}\ \mathbf{false})]\!]_{(\mathbf{Set},!)} = \text{app} \circ \langle [\![f]\!]_{(\mathbf{Set},!)}, \langle\rangle \rangle$$

But $[\![!_{k_2}\ \mathbf{true}]\!]_{(\mathbf{Set},!)} \neq [\![!_{k_2}\ \mathbf{false}]\!]_{(\mathbf{Set},!)}$. A contradiction.
The proposition follows.

□

D.2 Proofs of Lemmas/Theorems Stated in Section 5.2

Lemma D.2 (Multiplication (Lemma 5.1)) If $\Gamma \vdash a :^q A$, then $r_0 \cdot \Gamma \vdash a :^{r_0 \cdot q} A$.

Proof. By induction on $\Gamma \vdash a :^q A$.

- Rule SLDC-Var. Have: $0 \cdot \Gamma_1, x :^q A, 0 \cdot \Gamma_2 \vdash x :^q A$.
  Need to show: $0 \cdot \Gamma_1, x :^{r_0 \cdot q} A, 0 \cdot \Gamma_2 \vdash x :^{r_0 \cdot q} A$.
  This case follows by rule SLDC-Var.

- Rule SLDC-Lam. Have: $\Gamma \vdash \lambda^r x : A.b :^q {}^r A \to B$ where $\Gamma, x :^{q \cdot r} A \vdash b :^q B$.
  Need to show: $r_0 \cdot \Gamma \vdash \lambda^r x : A.b :^{r_0 \cdot q} {}^r A \to B$.
  By IH, $r_0 \cdot \Gamma, x :^{r_0 \cdot (q \cdot r)} A \vdash b :^{r_0 \cdot q} B$.
  By associativity of multiplication, $r_0 \cdot (q \cdot r) = (r_0 \cdot q) \cdot r$.
  This case, then, follows by rule SLDC-Lam.

- Rule SLDC-App. Have: $\Gamma_1 + \Gamma_2 \vdash b\ a^r :^q B$ where $\Gamma_1 \vdash b :^q {}^r A \to B$ and $\Gamma_2 \vdash a :^{q \cdot r} A$.
  Need to show: $r_0 \cdot (\Gamma_1 + \Gamma_2) \vdash b\ a^r :^{r_0 \cdot q} B$.
  By IH, $r_0 \cdot \Gamma_1 \vdash b :^{r_0 \cdot q} {}^r A \to B$ and $r_0 \cdot \Gamma_2 \vdash a :^{r_0 \cdot (q \cdot r)} A$.
  This case, then, follows by rule SLDC-App, using associativity and distributivity properties.

- Rule SLDC-Pair. Have: $\Gamma_1 + \Gamma_2 \vdash ({a_1}^r, a_2) :^q {}^r A_1 \times A_2$ where $\Gamma_1 \vdash a_1 :^{q \cdot r} A_1$ and $\Gamma_2 \vdash a_2 :^q A_2$.
  Need to show: $r_0 \cdot (\Gamma_1 + \Gamma_2) \vdash ({a_1}^r, a_2) :^{r_0 \cdot q} {}^r A_1 \times A_2$.
  By IH, $r_0 \cdot \Gamma_1 \vdash a_1 :^{r_0 \cdot (q \cdot r)} A_1$ and $r_0 \cdot \Gamma_2 \vdash a_2 :^{r_0 \cdot q} A_2$.
  This case, then, follows by rule SLDC-Pair, using associativity and distributivity properties.

- Rule SLDC-LetPair. Have: $\Gamma_1 + \Gamma_2 \vdash \mathbf{let}_{q_0}\ (x^r, y) = a \ \mathbf{in}\ b :^q B$ where $\Gamma_1 \vdash a :^{q \cdot q_0}\ {}^r A_1 \times A_2$ and $\Gamma_2, x :^{q \cdot q_0 \cdot r} A_1, y :^{q \cdot q_0} A_2 \vdash b :^q B$.
  Need to show: $r_0 \cdot (\Gamma_1 + \Gamma_2) \vdash \mathbf{let}_{q_0}\ (x^r, y) = a \ \mathbf{in}\ b :^{r_0 \cdot q} B$.
  By IH, $r_0 \cdot \Gamma_1 \vdash a :^{r_0 \cdot (q \cdot q_0)}\ {}^r A_1 \times A_2$ and $r_0 \cdot \Gamma_2, x :^{r_0 \cdot (q \cdot q_0 \cdot r)} A_1, y :^{r_0 \cdot (q \cdot q_0)} A_2 \vdash b :^{r_0 \cdot q} B$.
  This case, then, follows by rule SLDC-LetPair, using associativity and distributivity properties.

- Rule SLDC-Unit. Have: $0 \cdot \Gamma \vdash \mathbf{unit} :^q \mathbf{Unit}$.
  Need to show: $r_0 \cdot (0 \cdot \Gamma) \vdash \mathbf{unit} :^{r_0 \cdot q} \mathbf{Unit}$.
  This case follows by rule SLDC-Unit, using associativity and identity properties.

- Rule SLDC-LetUnit. Have: $\Gamma_1 + \Gamma_2 \vdash \mathbf{let}_{q_0}\ \mathbf{unit} = a \ \mathbf{in}\ b :^q B$ where $\Gamma_1 \vdash a :^{q \cdot q_0} \mathbf{Unit}$ and $\Gamma_2 \vdash b :^q B$.
  Need to show: $r_0 \cdot (\Gamma_1 + \Gamma_2) \vdash \mathbf{let}_{q_0}\ \mathbf{unit} = a \ \mathbf{in}\ b :^{r_0 \cdot q} B$.
  By IH, $r_0 \cdot \Gamma_1 \vdash a :^{r_0 \cdot (q \cdot q_0)} \mathbf{Unit}$ and $r_0 \cdot \Gamma_2 \vdash b :^{r_0 \cdot q} B$.
  This case, then, follows by rule SLDC-LetUnit, using associativity and distributivity properties.

- Rule SLDC-Inj1. Have $\Gamma \vdash \mathbf{inj_1}\ a_1 :^q A_1 + A_2$ where $\Gamma \vdash a_1 :^q A_1$.
  Need to show: $r_0 \cdot \Gamma \vdash \mathbf{inj_1}\ a_1 :^{r_0 \cdot q} A_1 + A_2$.
  By IH, $r_0 \cdot \Gamma \vdash a_1 :^{r_0 \cdot q} A_1$.
  This case, then, follows by rule SLDC-Inj1.

- Rule SLDC-Inj2. Similar to rule SLDC-Inj1.

- Rule SLDC-Case. Have: $\Gamma_1 + \Gamma_2 \vdash \mathbf{case}_{q_0}\ a \ \mathbf{of}\ x_1.b_1 \ ; \ x_2.b_2 :^q B$ where $\Gamma_1 \vdash a :^{q \cdot q_0} A_1 + A_2$ and $\Gamma_2, x_1 :^{q \cdot q_0} A_1 \vdash b_1 :^q B$ and $\Gamma_2, x_2 :^{q \cdot q_0} A_2 \vdash b_2 :^q B$.
  Need to show: $r_0 \cdot (\Gamma_1 + \Gamma_2) \vdash \mathbf{case}_{q_0}\ a \ \mathbf{of}\ x_1.b_1 \ ; \ x_2.b_2 :^{r_0 \cdot q} B$.
  By IH, $r_0 \cdot \Gamma_1 \vdash a :^{r_0 \cdot (q \cdot q_0)} A_1 + A_2$ and $r_0 \cdot \Gamma_2, x_1 :^{r_0 \cdot (q \cdot q_0)} A_1 \vdash b_1 :^{r_0 \cdot q} B$ and $r_0 \cdot \Gamma_2, x_2 :^{r_0 \cdot (q \cdot q_0)} A_2 \vdash b_2 :^{r_0 \cdot q} B$.
  This case, then, follows by rule SLDC-Case, using associativity and distributivity properties.

- Rule SLDC-SubL. Have: $\Gamma \vdash a :^q A$ where $\Gamma' \vdash a :^q A$ and $\Gamma <: \Gamma'$.
  Need to show: $r_0 \cdot \Gamma \vdash a :^{r_0 \cdot q} A$.
  By IH, $r_0 \cdot \Gamma' \vdash a :^{r_0 \cdot q} A$.
  Next, $r_0 \cdot \Gamma <: r_0 \cdot \Gamma'$, since $\Gamma <: \Gamma'$.
  This case, then, follows by rule SLDC-SubL.

- Rule SLDC-SubR. Have: $\Gamma \vdash a :^{q'} A$ where $\Gamma \vdash a :^q A$ and $q <: q'$.
  Need to show: $r_0 \cdot \Gamma \vdash a :^{r_0 \cdot q'} A$.
  By IH, $r_0 \cdot \Gamma \vdash a :^{r_0 \cdot q} A$.
  Next, $r_0 \cdot q <: r_0 \cdot q'$, since $q <: q'$.
  This case, then, follows by rule SLDC-SubR.

□

Lemma D.3 (Factorization (Lemma 5.2)) If $\Gamma \vdash a :^q A$ and $q \neq 0$, then there exists Γ' such that $\Gamma' \vdash a :^1 A$ and $\Gamma <: q \cdot \Gamma'$.

Proof. By induction on $\Gamma \vdash a :^q A$.

- Rule SLDC-Var. Have: $0 \cdot \Gamma_1, x :^q A, 0 \cdot \Gamma_2 \vdash x :^q A$.
  Need to show: $\exists \Gamma'$ such that $\Gamma' \vdash x :^1 A$ and $0 \cdot \Gamma_1, x :^q A, 0 \cdot \Gamma_2 <: q \cdot \Gamma'$.
  This case follows by setting $\Gamma' := 0 \cdot \Gamma_1, x :^1 A, 0 \cdot \Gamma_2$.

- Rule SLDC-Lam. Have: $\Gamma \vdash \lambda^r x : A.b :^q {}^r A \to B$ where $\Gamma, x :^{q \cdot r} A \vdash b :^q B$.
  Need to show: $\exists \Gamma'$ such that $\Gamma' \vdash \lambda^r x : A.b :^1 {}^r A \to B$ and $\Gamma <: q \cdot \Gamma'$.
  By IH, $\exists \Gamma'_1$ and r' such that $\Gamma'_1, x :^{r'} A \vdash b :^1 B$ and $\Gamma <: q \cdot \Gamma'_1$ and $q \cdot r <: q \cdot r'$.
  Since $q \cdot r <: q \cdot r'$ and $q \neq 0$, therefore $r <: r'$.
  By rule SLDC-SubL, $\Gamma'_1, x :^r A \vdash b :^1 B$.
  This case, then, follows by rule SLDC-Lam by setting $\Gamma' := \Gamma'_1$.

- Rule SLDC-App. Have: $\Gamma_1 + \Gamma_2 \vdash b\ a^r :^q B$ where $\Gamma_1 \vdash b :^q {}^r A \to B$ and $\Gamma_2 \vdash a :^{q \cdot r} A$.
  Need to show: $\exists \Gamma'$ such that $\Gamma' \vdash b\ a^r :^1 B$ and $\Gamma_1 + \Gamma_2 <: q \cdot \Gamma'$.
  There are two cases to consider.

  - $r \neq 0$. Since $q \neq 0$, therefore $q \cdot r \neq 0$.
    By IH, $\exists \Gamma'_1, \Gamma'_2$ such that $\Gamma'_1 \vdash b :^1 {}^r A \to B$ and $\Gamma'_2 \vdash a :^1 A$ and $\Gamma_1 <: q \cdot \Gamma'_1$ and $\Gamma_2 <: (q \cdot r) \cdot \Gamma'_2$.
    By Lemma D.2, $r \cdot \Gamma'_2 \vdash a :^r A$.
    By rule SLDC-App, $\Gamma'_1 + r \cdot \Gamma'_2 \vdash b\ a^r :^1 B$.
    This case, then, follows by setting $\Gamma' := \Gamma'_1 + r \cdot \Gamma'_2$.
  - $r = 0$. Then, $\Gamma_2 \vdash a :^0 A$. By Lemma D.2, $0 \cdot \Gamma_2 \vdash a :^0 A$.
    By IH, $\exists \Gamma'_1$ such that $\Gamma'_1 \vdash b :^1 {}^0 A \to B$ and $\Gamma_1 <: q \cdot \Gamma'_1$.
    By rule SLDC-App, $\Gamma'_1 \vdash b\ a^0 :^1 B$.
    Further, $\Gamma_1 + \Gamma_2 <: \Gamma_1$ ($\because \Gamma_1 + \Gamma_0 <: \Gamma_1$ for any Γ_0 over $\mathbb{N}_\geq$ whereas over $\mathbb{N}_=$, $\overline{\Gamma_2} = \mathbf{0}$).
    By transitivity, $\Gamma_1 + \Gamma_2 <: q \cdot \Gamma'_1$.
    This case, then, follows by setting $\Gamma' := \Gamma'_1$.

- Rule SLDC-Pair. Have: $\Gamma_1 + \Gamma_2 \vdash ({a_1}^r, a_2) :^q {}^r A_1 \times A_2$ where $\Gamma_1 \vdash a_1 :^{q \cdot r} A_1$ and $\Gamma_2 \vdash a_2 :^q A_2$.
  Need to show: $\exists \Gamma'$ such that $\Gamma' \vdash ({a_1}^r, a_2) :^1 {}^r A_1 \times A_2$ and $\Gamma_1 + \Gamma_2 <: q \cdot \Gamma'$.
  There are two cases to consider.

  - $r \neq 0$. Since $q \neq 0$, therefore $q \cdot r \neq 0$.
    By IH, $\exists \Gamma'_1, \Gamma'_2$ such that $\Gamma'_1 \vdash a_1 :^1 A_1$ and $\Gamma'_2 \vdash a_2 :^1 A_2$ and $\Gamma_1 <: (q \cdot r) \cdot \Gamma'_1$ and $\Gamma_2 <: q \cdot \Gamma'_2$.
    By Lemma D.2, $r \cdot \Gamma'_1 \vdash a_1 :^r A_1$.
    By rule SLDC-Pair, $r \cdot \Gamma'_1 + \Gamma'_2 \vdash ({a_1}^r, a_2) :^1 {}^r A_1 \times A_2$.
    This case, then, follows by setting $\Gamma' := r \cdot \Gamma'_1 + \Gamma'_2$.
  - $r = 0$. Then, $\Gamma_1 \vdash a_1 :^0 A_1$. By Lemma D.2, $0 \cdot \Gamma_1 \vdash a_1 :^0 A_1$.
    By IH, $\exists \Gamma'_2$ such that $\Gamma'_2 \vdash a_2 :^1 A_2$ and $\Gamma_2 <: q \cdot \Gamma'_2$.
    By rule SLDC-Pair, $\Gamma'_2 \vdash ({a_1}^0, a_2) :^1 {}^0 A_1 \times A_2$.
    Further, $\Gamma_1 + \Gamma_2 <: \Gamma_2$ ($\because \Gamma_0 + \Gamma_2 <: \Gamma_2$ for any Γ_0 over $\mathbb{N}_\geq$ whereas over $\mathbb{N}_=$, $\overline{\Gamma_1} = \mathbf{0}$).

By transitivity, $\Gamma_1 + \Gamma_2 <: q \cdot \Gamma'_2$.
This case, then, follows by setting $\Gamma' := \Gamma'_2$.

- Rule SLDC-LetPair. Have: $\Gamma_1 + \Gamma_2 \vdash \mathbf{let}_{q_0}\ (x^r, y) = a\ \mathbf{in}\ b :^q B$ where $\Gamma_1 \vdash a :^{q \cdot q_0}\ {}^r A_1 \times A_2$ and $\Gamma_2, x :^{q \cdot q_0 \cdot r} A_1, y :^{q \cdot q_0} A_2 \vdash b :^q B$ and $q_0 <: 1$.
Need to show: $\exists \Gamma'$ such that $\Gamma' \vdash \mathbf{let}_{q_0}\ (x^r, y) = a\ \mathbf{in}\ b :^1 B$ and $\Gamma_1 + \Gamma_2 <: q \cdot \Gamma'$.
Since $q \neq 0$ and $q_0 <: 1$, therefore $q \cdot q_0 \neq 0$.
By IH, $\exists \Gamma'_1, \Gamma'_2$ and r', q' such that $\Gamma'_1 \vdash a :^1\ {}^r A_1 \times A_2$ and $\Gamma'_2, x :^{r'} A_1, y :^{q'} A_2 \vdash b :^1 B$ and $\Gamma_1 <: (q \cdot q_0) \cdot \Gamma'_1$ and $\Gamma_2 <: q \cdot \Gamma'_2$ and $q \cdot q_0 \cdot r <: q \cdot r'$ and $q \cdot q_0 <: q \cdot q'$.
Since $q \neq 0$, therefore $q_0 \cdot r <: r'$ and $q_0 <: q'$.
By rule SLDC-SubL, $\Gamma'_2, x :^{q_0 \cdot r} A_1, y :^{q_0} A_2 \vdash b :^1 B$.
Again, by Lemma D.2, $q_0 \cdot \Gamma'_1 \vdash a :^{q_0}\ {}^r A_1 \times A_2$.
By rule SLDC-LetPair, $q_0 \cdot \Gamma'_1 + \Gamma'_2 \vdash \mathbf{let}_{q_0}\ (x^r, y) = a\ \mathbf{in}\ b :^1 B$.
This case, then, follows by setting $\Gamma' := q_0 \cdot \Gamma'_1 + \Gamma'_2$.

- Rule SLDC-Unit. Have: $0 \cdot \Gamma \vdash \mathbf{unit} :^q \mathbf{Unit}$.
Need to show: $\exists \Gamma'$ such that $\Gamma' \vdash \mathbf{unit} :^1 \mathbf{Unit}$ and $0 \cdot \Gamma <: q \cdot \Gamma'$.
This case follows by setting $\Gamma' := 0 \cdot \Gamma$.

- Rule SLDC-LetUnit. Have: $\Gamma_1 + \Gamma_2 \vdash \mathbf{let}_{q_0}\ \mathbf{unit} = a\ \mathbf{in}\ b :^q B$ where $\Gamma_1 \vdash a :^{q \cdot q_0} \mathbf{Unit}$ and $\Gamma_2 \vdash b :^q B$ and $q_0 <: 1$.
Need to show: $\exists \Gamma'$ such that $\Gamma' \vdash \mathbf{let}_{q_0}\ \mathbf{unit} = a\ \mathbf{in}\ b :^1 B$ and $\Gamma_1 + \Gamma_2 <: q \cdot \Gamma'$.
By IH, $\exists \Gamma'_1, \Gamma'_2$ such that $\Gamma'_1 \vdash a :^1 \mathbf{Unit}$ and $\Gamma'_2 \vdash b :^1 B$ and $\Gamma_1 <: (q \cdot q_0) \cdot \Gamma'_1$ and $\Gamma_2 <: q \cdot \Gamma'_2$.
By Lemma D.2, $q_0 \cdot \Gamma'_1 \vdash a :^{q_0} \mathbf{Unit}$.
By rule SLDC-LetUnit, $q_0 \cdot \Gamma'_1 + \Gamma'_2 \vdash \mathbf{let}_{q_0}\ \mathbf{unit} = a\ \mathbf{in}\ b :^1 B$.
This case, then, follows by setting $\Gamma' := q_0 \cdot \Gamma'_1 + \Gamma'_2$.

- Rule SLDC-Inj1. Have $\Gamma \vdash \mathbf{inj}_1\ a_1 :^q A_1 + A_2$ where $\Gamma \vdash a_1 :^q A_1$.
Need to show: $\exists \Gamma'$ such that $\Gamma' \vdash \mathbf{inj}_1\ a_1 :^1 A_1 + A_2$ and $\Gamma <: q \cdot \Gamma'$.
By IH, $\exists \Gamma'_1$ such that $\Gamma'_1 \vdash a_1 :^1 A_1$ and $\Gamma <: q \cdot \Gamma'_1$.
This case, then, follows by setting $\Gamma' := \Gamma'_1$.

- Rule SLDC-Inj2. Similar to rule SLDC-Inj1.

- Rule SLDC-Case. Have: $\Gamma_1 + \Gamma_2 \vdash \mathbf{case}_{q_0}\ a\ \mathbf{of}\ x_1.b_1 ; x_2.b_2 :^q B$ where $\Gamma_1 \vdash a :^{q \cdot q_0} A_1 + A_2$ and $\Gamma_2, x_1 :^{q \cdot q_0} A_1 \vdash b_1 :^q B$ and $\Gamma_2, x_2 :^{q \cdot q_0} A_2 \vdash b_2 :^q B$ and $q_0 <: 1$.
Need to show: $\exists \Gamma'$ such that $\Gamma' \vdash \mathbf{case}_{q_0}\ a\ \mathbf{of}\ x_1.b_1 ; x_2.b_2 :^1 B$ and $\Gamma_1 + \Gamma_2 <: q \cdot \Gamma'$.
By IH, $\exists \Gamma'_1, \Gamma'_{21}, \Gamma'_{22}$ and q'_1, q'_2 such that $\Gamma'_1 \vdash a :^1 A_1 + A_2$ and $\Gamma'_{21}, x_1 :^{q'_1} A_1 \vdash b_1 :^1 B$ and $\Gamma'_{22}, x_2 :^{q'_2} A_2 \vdash b_2 :^1 B$ and $\Gamma_1 <: (q \cdot q_0) \cdot \Gamma'_1$ and $\Gamma_2 <: q \cdot \Gamma'_{21}$ and $\Gamma_2 <: q \cdot \Gamma'_{22}$ and $q \cdot q_0 <: q \cdot q'_1$ and $q \cdot q_0 <: q \cdot q'_2$.
Since $q \neq 0$, therefore $q_0 <: q'_1$ and $q_0 <: q'_2$.
The remainder of the proof for this case is different for $\mathbb{N}_=$ and $\mathbb{N}_\geq$.
  - In case of $\mathbb{N}_=$, since $<:$ is discrete, $\Gamma'_{21} = \Gamma'_{22} = \Gamma'_2$ (say).
  Then, $\Gamma_2 <: q \cdot \Gamma'_2$.

Further, by rule SLDC-SubL, $\Gamma_2', x_1 :^{q_0} A_1 \vdash b_1 :^1 B$ and $\Gamma_2', x_2 :^{q_0} A_2 \vdash b_2 :^1 B$.
By Lemma D.2, $q_0 \cdot \Gamma_1' \vdash a :^{q_0} A_1 + A_2$.
So by rule SLDC-Case, $q_0 \cdot \Gamma_1' + \Gamma_2' \vdash \mathbf{case}_{q_0}\ a\ \mathbf{of}\ x_1.b_1\ ;\ x_2.b_2 :^1 B$.
This case, then, follows by setting $\Gamma' := q_0 \cdot \Gamma_1' + \Gamma_2'$.

- In case of $\mathbb{N}_{\geq}$, Γ_{21}' may not be equal to Γ_{22}'. So we need a different approach.
  For any two grades q_1 and q_2, we define an operator as follows: $q_1 ; q_2 = q_1$ if $q_1 <: q_2$ and q_2 otherwise. Next, we extend this operation to graded contexts Γ_1 and Γ_2 where $\lfloor \Gamma_1 \rfloor = \lfloor \Gamma_2 \rfloor$: we define $\Gamma_0 := \Gamma_1 ; \Gamma_2$ as $\lfloor \Gamma_0 \rfloor = \lfloor \Gamma_1 \rfloor$ and $\overline{\Gamma_0} = \overline{\Gamma_1} ; \overline{\Gamma_2}$ (operation defined pointwise).
  Now, let $\Gamma_2' := \Gamma_{21}' ; \Gamma_{22}'$. Observe that $\Gamma_2' <: \Gamma_{21}'$ and $\Gamma_2' <: \Gamma_{22}'$.
  Then by transitivity, $\Gamma_2 <: q \cdot \Gamma_2'$.
  Again, by rule SLDC-SubL, $\Gamma_2', x_1 :^{q_0} A_1 \vdash b_1 :^1 B$ and $\Gamma_2', x_2 :^{q_0} A_2 \vdash b_2 :^1 B$.
  By Lemma D.2, $q_0 \cdot \Gamma_1' \vdash a :^{q_0} A_1 + A_2$.
  So by rule SLDC-Case, $q_0 \cdot \Gamma_1' + \Gamma_2' \vdash \mathbf{case}_{q_0}\ a\ \mathbf{of}\ x_1.b_1\ ;\ x_2.b_2 :^1 B$.
  This case, then, follows by setting $\Gamma' := q_0 \cdot \Gamma_1' + \Gamma_2'$.

- Rule SLDC-SubL. Have: $\Gamma \vdash a :^q A$ where $\Gamma_1 \vdash a :^q A$ and $\Gamma <: \Gamma_1$.
  Need to show: $\exists \Gamma'$ such that $\Gamma' \vdash a :^1 A$ and $\Gamma <: q \cdot \Gamma'$.
  By IH, $\exists \Gamma_1'$ such that $\Gamma_1' \vdash a :^1 A$ and $\Gamma_1 <: q \cdot \Gamma_1'$.
  This case, then, follows by setting $\Gamma' := \Gamma_1'$.

- Rule SLDC-SubR. Have: $\Gamma \vdash a :^{q'} A$ where $\Gamma \vdash a :^q A$ and $q <: q'$.
  Need to show: $\exists \Gamma'$ such that $\Gamma' \vdash a :^1 A$ and $\Gamma <: q' \cdot \Gamma'$.
  Since $q' \neq 0$, therefore $q \neq 0$.
  By IH, $\exists \Gamma_1'$ such that $\Gamma_1' \vdash a :^1 A$ and $\Gamma <: q \cdot \Gamma_1'$.
  Now, since $q <: q'$, so $q \cdot \Gamma_1' <: q' \cdot \Gamma_1'$.
  This case, then, follows by setting $\Gamma' := \Gamma_1'$.

□

Lemma D.4 (Splitting (Lemma 5.3)) If $\Gamma \vdash a :^{q_1 + q_2} A$, then there exists Γ_1 and Γ_2 such that $\Gamma_1 \vdash a :^{q_1} A$ and $\Gamma_2 \vdash a :^{q_2} A$ and $\Gamma = \Gamma_1 + \Gamma_2$.

Proof. If $q_1 + q_2 = 0$, then set $\Gamma_1 := 0 \cdot \Gamma$ and $\Gamma_2 := \Gamma$.
Otherwise, by Lemma D.3, $\exists \Gamma'$ such that $\Gamma' \vdash a :^1 A$ and $\Gamma <: (q_1 + q_2) \cdot \Gamma'$.
Then, $\Gamma = (q_1 + q_2) \cdot \Gamma' + \Gamma_0$, for some Γ_0 (in case of $\mathbb{N}_=$, we have, $\overline{\Gamma_0} = \mathbf{0}$).
Now, by Lemma D.2, $q_1 \cdot \Gamma' \vdash a :^{q_1} A$ and $q_2 \cdot \Gamma' \vdash a :^{q_2} A$.
By rule SLDC-SubL, $q_2 \cdot \Gamma' + \Gamma_0 \vdash a :^{q_2} A$.
The lemma, then, follows by setting $\Gamma_1 := q_1 \cdot \Gamma'$ and $\Gamma_2 := q_2 \cdot \Gamma' + \Gamma_0$. □

Lemma D.5 (Weakening (Lemma 5.4)) If $\Gamma_1, \Gamma_2 \vdash a :^q A$, then $\Gamma_1, z :^0 C, \Gamma_2 \vdash a :^q A$.

Proof. By induction on $\Gamma_1, \Gamma_2 \vdash a :^q A$. □

Lemma D.6 (Substitution (Lemma 5.5)) If $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash a :^q A$ and $\Gamma \vdash c :^{r_0} C$ where $\lfloor\Gamma_1\rfloor = \lfloor\Gamma\rfloor$, then $\Gamma_1 + \Gamma, \Gamma_2 \vdash a\{c/z\} :^q A$.

Proof. By induction on $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash a :^q A$.

- Rule SLDC-Var. There are three cases to consider.
  - $0 \cdot \Gamma_{11}, z :^0 C, 0 \cdot \Gamma_{12}, x :^q A, 0 \cdot \Gamma_2 \vdash x :^q A$. Also, $\Gamma \vdash c :^0 C$ where $\lfloor\Gamma_{11}\rfloor = \lfloor\Gamma\rfloor$.
    Need to show: $\Gamma, 0 \cdot \Gamma_{12}, x :^q A, 0 \cdot \Gamma_2 \vdash x :^q A$.
    In case of $\mathbb{N}_=$, we have, $\overline{\Gamma} = \mathbf{0}$. This case, then, follows by rule SLDC-Var.
    In case of $\mathbb{N}_\geq$, this case follows by rule SLDC-Var and rule SLDC-SubL.
  - $0 \cdot \Gamma_1, x :^q A, 0 \cdot \Gamma_2 \vdash x :^q A$. Also, $\Gamma \vdash a :^q A$ where $\lfloor\Gamma_1\rfloor = \lfloor\Gamma\rfloor$.
    Need to show: $\Gamma, 0 \cdot \Gamma_2 \vdash a :^q A$.
    Follows by weakening lemma D.5.
  - $0 \cdot \Gamma_1, x :^q A, 0 \cdot \Gamma_{21}, z :^0 C, 0 \cdot \Gamma_{22} \vdash x :^q A$. Also, $\Gamma_{31}, x :^r A, \Gamma_{32} \vdash c :^0 C$ where $\lfloor\Gamma_{31}\rfloor = \lfloor\Gamma_1\rfloor$ and $\lfloor\Gamma_{32}\rfloor = \lfloor\Gamma_{21}\rfloor$.
    Need to show: $\Gamma_{31}, x :^{(q+r)} A, \Gamma_{32}, 0 \cdot \Gamma_{22} \vdash x :^q A$.
    In case of $\mathbb{N}_=$, we have, $\overline{\Gamma_{31}} = \mathbf{0}$ and $\overline{\Gamma_{32}} = \mathbf{0}$ and $r = 0$. This case, then, follows by rule SLDC-Var.
    In case of $\mathbb{N}_\geq$, this case follows by rule SLDC-Var and rule SLDC-SubL.
- Rule SLDC-Lam. Have: $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash \lambda^r x : A.b :^q {}^r A \to B$ where $\Gamma_1, z :^{r_0} C, \Gamma_2, x :^{q \cdot r} A \vdash b :^q B$.
  Also, $\Gamma \vdash c :^{r_0} C$ where $\lfloor\Gamma\rfloor = \lfloor\Gamma_1\rfloor$.
  Need to show: $\Gamma_1 + \Gamma, \Gamma_2 \vdash \lambda^r x : A.b\{c/z\} :^q {}^r A \to B$.
  Follows by IH and rule SLDC-Lam.
- Rule SLDC-App. Have: $\Gamma_{11} + \Gamma_{12}, z :^{r_{01}+r_{02}} C, \Gamma_{21} + \Gamma_{22} \vdash b\, a^r :^q B$ where $\Gamma_{11}, z :^{r_{01}} C, \Gamma_{21} \vdash b :^q {}^r A \to B$ and $\Gamma_{12}, z :^{r_{02}} C, \Gamma_{22} \vdash a :^{q \cdot r} A$. Also, $\Gamma \vdash c :^{r_{01}+r_{02}} C$ where $\lfloor\Gamma\rfloor = \lfloor\Gamma_{11}\rfloor$.
  Need to show: $\Gamma_{11} + \Gamma_{12} + \Gamma, \Gamma_{21} + \Gamma_{22} \vdash b\{c/z\}\ a\{c/z\}^r :^q B$.
  Since $\Gamma \vdash c :^{r_{01}+r_{02}} C$, so by lemma D.4, $\exists \Gamma_{31}, \Gamma_{32}$ such that $\Gamma_{31} \vdash c :^{r_{01}} C$ and $\Gamma_{32} \vdash c :^{r_{02}} C$ and $\Gamma = \Gamma_{31} + \Gamma_{32}$.
  By IH, $\Gamma_{11} + \Gamma_{31}, \Gamma_{21} \vdash b\{c/z\} :^q {}^r A \to B$ and $\Gamma_{12} + \Gamma_{32}, \Gamma_{22} \vdash a\{c/z\} :^{q \cdot r} A$.
  This case, then, follows by rule SLDC-App.
- Rule SLDC-Pair. Have: $\Gamma_{11} + \Gamma_{12}, z :^{r_{01}+r_{02}} C, \Gamma_{21} + \Gamma_{22} \vdash (a_1{}^r, a_2) :^q {}^r A_1 \times A_2$ where $\Gamma_{11}, z :^{r_{01}} C, \Gamma_{21} \vdash a_1 :^{q \cdot r} A_1$ and $\Gamma_{12}, z :^{r_{02}} C, \Gamma_{22} \vdash a_2 :^q A_2$. Also, $\Gamma \vdash c :^{r_{01}+r_{02}} C$ where $\lfloor\Gamma\rfloor = \lfloor\Gamma_{11}\rfloor$.
  Need to show: $\Gamma_{11} + \Gamma_{12} + \Gamma, \Gamma_{21} + \Gamma_{22} \vdash (a_1\{c/z\}^r, a_2\{c/z\}) :^q {}^r A_1 \times A_2$.
  Since $\Gamma \vdash c :^{r_{01}+r_{02}} C$, so by lemma D.4, $\exists \Gamma_{31}, \Gamma_{32}$ such that $\Gamma_{31} \vdash c :^{r_{01}} C$ and $\Gamma_{32} \vdash c :^{r_{02}} C$ and $\Gamma = \Gamma_{31} + \Gamma_{32}$.
  By IH, $\Gamma_{11} + \Gamma_{31}, \Gamma_{21} \vdash a_1\{c/z\} :^{q \cdot r} A_1$ and $\Gamma_{12} + \Gamma_{32}, \Gamma_{22} \vdash a_2\{c/z\} :^q A_2$.
  This case, then, follows by rule SLDC-Pair.

- Rule SLDC-LetPair. Have: $\Gamma_{11} + \Gamma_{12}, z :^{r_{01}+r_{02}} C, \Gamma_{21} + \Gamma_{22} \vdash \textbf{let}_{q_0}\ (x^r, y) = a \textbf{ in } b :^q B$ where $\Gamma_{11}, z :^{r_{01}} C, \Gamma_{21} \vdash a :^{q \cdot q_0}\ {}^r A_1 \times A_2$ and $\Gamma_{12}, z :^{r_{02}} C, \Gamma_{22}, x :^{q \cdot q_0 \cdot r} A_1, y :^{q \cdot q_0} A_2 \vdash b :^q B$. Also, $\Gamma \vdash c :^{r_{01}+r_{02}} C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_{11} \rfloor$.
  Need to show: $\Gamma_{11} + \Gamma_{12} + \Gamma, \Gamma_{21} + \Gamma_{22} \vdash \textbf{let}_{q_0}\ (x^r, y) = a\{c/z\} \textbf{ in } b\{c/z\} :^q B$.
  Since $\Gamma \vdash c :^{r_{01}+r_{02}} C$, so by lemma D.4, $\exists \Gamma_{31}, \Gamma_{32}$ such that $\Gamma_{31} \vdash c :^{r_{01}} C$ and $\Gamma_{32} \vdash c :^{r_{02}} C$ and $\Gamma = \Gamma_{31} + \Gamma_{32}$.
  By IH, $\Gamma_{11} + \Gamma_{31}, \Gamma_{21} \vdash a\{c/z\} :^{q \cdot q_0}\ {}^r A_1 \times A_2$ and $\Gamma_{12} + \Gamma_{32}, \Gamma_{22}, x :^{q \cdot q_0 \cdot r} A_1, y :^{q \cdot q_0} A_2 \vdash b\{c/z\} :^q B$.
  This case, then, follows by rule SLDC-LetPair.

- Rule SLDC-Unit. Have: $0 \cdot \Gamma_1, z :^0 C, 0 \cdot \Gamma_2 \vdash \textbf{unit} :^q \textbf{Unit}$. Also, $\Gamma \vdash c :^0 C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
  Need to show: $\Gamma, 0 \cdot \Gamma_2 \vdash \textbf{unit} :^q \textbf{Unit}$.
  In case of $\mathbb{N}_=$, we have, $\overline{\Gamma} = \mathbf{0}$. This case, then, follows by rule SLDC-Unit.
  In case of $\mathbb{N}_\geq$, this case follows by rule SLDC-Unit and rule SLDC-SubL.

- Rule SLDC-LetUnit. Have: $\Gamma_{11} + \Gamma_{12}, z :^{r_{01}+r_{02}} C, \Gamma_{21} + \Gamma_{22} \vdash \textbf{let}_{q_0}\ \textbf{unit} = a \textbf{ in } b :^q B$ where $\Gamma_{11}, z :^{r_{01}} C, \Gamma_{21} \vdash a :^{q \cdot q_0} \textbf{Unit}$ and $\Gamma_{12}, z :^{r_{02}} C, \Gamma_{22} \vdash b :^q B$. Also, $\Gamma \vdash c :^{r_{01}+r_{02}} C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_{11} \rfloor$.
  Need to show: $\Gamma_{11} + \Gamma_{12} + \Gamma, \Gamma_{21} + \Gamma_{22} \vdash \textbf{let}_{q_0}\ \textbf{unit} = a\{c/z\} \textbf{ in } b\{c/z\} :^q B$.
  Since $\Gamma \vdash c :^{r_{01}+r_{02}} C$, so by lemma D.4, $\exists \Gamma_{31}, \Gamma_{32}$ such that $\Gamma_{31} \vdash c :^{r_{01}} C$ and $\Gamma_{32} \vdash c :^{r_{02}} C$ and $\Gamma = \Gamma_{31} + \Gamma_{32}$.
  By IH, $\Gamma_{11} + \Gamma_{31}, \Gamma_{21} \vdash a\{c/z\} :^{q \cdot q_0} \textbf{Unit}$ and $\Gamma_{12} + \Gamma_{32}, \Gamma_{22} \vdash b\{c/z\} :^q B$.
  This case, then, follows by rule SLDC-LetUnit.

- Rule SLDC-Inj1. Have: $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash \textbf{inj}_1\ a_1 :^q A_1 + A_2$ where $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash a_1 :^q A_1$. Also, $\Gamma \vdash c :^{r_0} C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
  Need to show: $\Gamma_1 + \Gamma, \Gamma_2 \vdash \textbf{inj}_1\ a_1\{c/z\} :^q A_1 + A_2$.
  By IH, $\Gamma_1 + \Gamma, \Gamma_2 \vdash a_1\{c/z\} :^q A_1$.
  This case, then, follows by rule SLDC-Inj1.

- Rule SLDC-Inj2. Similar to rule SLDC-Inj1.

- Rule SLDC-Case. Have: $\Gamma_{11} + \Gamma_{12}, z :^{r_{01}+r_{02}} C, \Gamma_{21} + \Gamma_{22} \vdash \textbf{case}_{q_0}\ a \textbf{ of } x_1.b_1 ; x_2.b_2 :^q B$ where $\Gamma_{11}, z :^{r_{01}} C, \Gamma_{21} \vdash a :^{q \cdot q_0} A_1 + A_2$ and $\Gamma_{12}, z :^{r_{02}} C, \Gamma_{22}, x_1 :^{q \cdot q_0} A_1 \vdash b_1 :^q B$ and $\Gamma_{12}, z :^{r_{02}} C, \Gamma_{22}, x_2 :^{q \cdot q_0} A_2 \vdash b_2 :^q B$. Also, $\Gamma \vdash c :^{r_{01}+r_{02}} C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_{11} \rfloor$.
  Need to show: $\Gamma_{11} + \Gamma_{12} + \Gamma, \Gamma_{21} + \Gamma_{22} \vdash \textbf{case}_{q_0}\ a\{c/z\} \textbf{ of } x_1.b_1\{c/z\} ; x_2.b_2\{c/z\} :^q B$.
  Since $\Gamma \vdash c :^{r_{01}+r_{02}} C$, so by lemma D.4, $\exists \Gamma_{31}, \Gamma_{32}$ such that $\Gamma_{31} \vdash c :^{r_{01}} C$ and $\Gamma_{32} \vdash c :^{r_{02}} C$ and $\Gamma = \Gamma_{31} + \Gamma_{32}$.
  By IH, $\Gamma_{11} + \Gamma_{31}, \Gamma_{21} \vdash a\{c/z\} :^{q \cdot q_0} A_1 + A_2$ and $\Gamma_{12} + \Gamma_{32}, \Gamma_{22}, x_1 :^{q \cdot q_0} A_1 \vdash b_1\{c/z\} :^q B$ and $\Gamma_{12} + \Gamma_{32}, \Gamma_{22}, x_2 :^{q \cdot q_0} A_2 \vdash b_2\{c/z\} :^q B$.
  This case, then, follows by rule SLDC-Case.

- Rule SLDC-SubL. Have: $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash a :^q A$ where $\Gamma'_1, z :^{r'_0} C, \Gamma'_2 \vdash a :^q A$ such that $\Gamma_1 <: \Gamma'_1$ and $r_0 <: r'_0$ and $\Gamma_2 <: \Gamma'_2$. Also, $\Gamma \vdash c :^{r_0} C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
  Need to show: $\Gamma_1 + \Gamma, \Gamma_2 \vdash a\{c/z\} :^q A$.

Since $r_0 <: r_0'$, by rule SLDC-SubR, $\Gamma \vdash c :^{r_0'} C$.
By IH, $\Gamma_1' + \Gamma, \Gamma_2' \vdash a\{c/z\} :^q A$.
This case, then, follows by rule SLDC-SubL.

- Rule SLDC-SubR. Have: $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash a :^{q'} A$ where $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash a :^q A$ and $q <: q'$. Also, $\Gamma \vdash c :^{r_0} C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
  Need to show: $\Gamma_1 + \Gamma, \Gamma_2 \vdash a\{c/z\} :^{q'} A$.
  By IH, $\Gamma_1 + \Gamma, \Gamma_2 \vdash a\{c/z\} :^q A$.
  This case, then, follows by rule SLDC-SubR.

□

Theorem D.7 (Preservation (Theorem 5.6)) If $\Gamma \vdash a :^q A$ and $\vdash a \rightsquigarrow a'$, then $\Gamma \vdash a' :^q A$.

Proof. By induction on $\Gamma \vdash a :^q A$ and subsequent inversion on $\vdash a \rightsquigarrow a'$.

- Rule SLDC-App. Have: $\Gamma_1 + \Gamma_2 \vdash b\ a^r :^q B$ where $\Gamma_1 \vdash b :^q {}^r A \to B$ and $\Gamma_2 \vdash a :^{q \cdot r} A$.
  Let $\vdash b\ a^r \rightsquigarrow c$. By inversion:
  - $\vdash b\ a^r \rightsquigarrow b'\ a^r$, when $\vdash b \rightsquigarrow b'$.
    Need to show: $\Gamma_1 + \Gamma_2 \vdash b'\ a^r :^q B$.
    Follows by IH and rule SLDC-App.
  - $b = \lambda^r x : A'.b'$ and $\vdash b\ a^r \rightsquigarrow b'\{a/x\}$.
    Need to show: $\Gamma_1 + \Gamma_2 \vdash b'\{a/x\} :^q B$.
    By inversion on $\Gamma_1 \vdash \lambda^r x : A'.b' :^q {}^r A \to B$, we get $A' = A$ and $\Gamma_1, x :^{q_0 \cdot r} A \vdash b' :^{q_0} B$, for some $q_0 <: q$.
    Now, there are two cases to consider.
    * $q_0 = 0$. Since $q_0 <: q$, so $q = 0$. As such, $q_0 = q$.
      So, we have, $\Gamma_1, x :^{q \cdot r} A \vdash b' :^q B$.
      This case, then, follows by the substitution lemma.
    * $q_0 \neq 0$. By factorization lemma D.3, $\exists \Gamma_1'$ and r' such that $\Gamma_1', x :^{r'} A \vdash b' :^1 B$ and $\Gamma_1 <: q_0 \cdot \Gamma_1'$ and $q_0 \cdot r <: q_0 \cdot r'$.
      Since $q_0 \neq 0$, so $r <: r'$. Hence, by rule SLDC-SubL, $\Gamma_1', x :^r A \vdash b' :^1 B$.
      Now, by multiplication lemma D.2, $q \cdot \Gamma_1', x :^{q \cdot r} A \vdash b' :^q B$.
      And by rule SLDC-SubL, $\Gamma_1, x :^{q \cdot r} A \vdash b' :^q B$.
      This case, then, follows by the substitution lemma.

- Rule SLDC-LetPair. Have: $\Gamma_1 + \Gamma_2 \vdash \mathbf{let}_{q_0}\ (x^r, y) = a\ \mathbf{in}\ b :^q B$ where $\Gamma_1 \vdash a :^{q \cdot q_0} {}^r A_1 \times A_2$ and $\Gamma_2, x :^{q \cdot q_0 \cdot r} A_1, y :^{q \cdot q_0} A_2 \vdash b :^q B$.
  Let $\vdash \mathbf{let}_{q_0}\ (x^r, y) = a\ \mathbf{in}\ b \rightsquigarrow c$. By inversion:

- $\vdash \mathbf{let}_{q_0}\ (x^r, y) = a\ \mathbf{in}\ b \leadsto \mathbf{let}_{q_0}\ (x^r, y) = a'\ \mathbf{in}\ b$, when $\vdash a \leadsto a'$.
  Need to show: $\Gamma_1 + \Gamma_2 \vdash \mathbf{let}_{q_0}\ (x^r, y) = a'\ \mathbf{in}\ b :^q B$.
  Follows by IH and rule SLDC-LetPair.
- $\vdash \mathbf{let}_{q_0}\ (x^r, y) = ({a_1}^r, a_2)\ \mathbf{in}\ b \leadsto b\{a_1/x\}\{a_2/y\}$.
  Need to show: $\Gamma_1 + \Gamma_2 \vdash b\{a_1/x\}\{a_2/y\} :^q B$.
  By inversion on $\Gamma_1 \vdash ({a_1}^r, a_2) :^{q \cdot q_0}\ {}^r A_1 \times A_2$, we have:
  $\exists \Gamma_{11}, \Gamma_{12}$ such that $\Gamma_{11} \vdash a_1 :^{q \cdot q_0 \cdot r} A_1$ and $\Gamma_{12} \vdash a_2 :^{q \cdot q_0} A_2$ and $\Gamma_1 = \Gamma_{11} + \Gamma_{12}$.
  This case, then, follows by applying the substitution lemma twice.

- Rule SLDC-LetUnit. Have: $\Gamma_1 + \Gamma_2 \vdash \mathbf{let}_{q_0}\ \mathbf{unit} = a\ \mathbf{in}\ b :^q B$ where $\Gamma_1 \vdash a :^{q \cdot q_0} \mathbf{Unit}$ and $\Gamma_2 \vdash b :^q B$.
  Let $\vdash \mathbf{let}_{q_0}\ \mathbf{unit} = a\ \mathbf{in}\ b \leadsto c$. By inversion:
  - $\vdash \mathbf{let}_{q_0}\ \mathbf{unit} = a\ \mathbf{in}\ b \leadsto \mathbf{let}_{q_0}\ \mathbf{unit} = a'\ \mathbf{in}\ b$, when $\vdash a \leadsto a'$.
    Need to show: $\Gamma_1 + \Gamma_2 \vdash \mathbf{let}_{q_0}\ \mathbf{unit} = a'\ \mathbf{in}\ b :^q B$.
    Follows by IH and rule SLDC-LetUnit.
  - $\vdash \mathbf{let}_{q_0}\ \mathbf{unit} = \mathbf{unit}\ \mathbf{in}\ b \leadsto b$.
    Need to show: $\Gamma_1 + \Gamma_2 \vdash b :^q B$.
    In case of $\mathbb{N}_=$, we have, $\overline{\Gamma_1} = \mathbf{0}$. This case, then, follows directly from the premise, $\Gamma_2 \vdash b :^q B$.
    In case of $\mathbb{N}_\geq$, this case follows by rule SLDC-SubL.

- Rule SLDC-Case. Have: $\Gamma_1 + \Gamma_2 \vdash \mathbf{case}_{q_0}\ a\ \mathbf{of}\ x_1.b_1\ ;\ x_2.b_2 :^q B$ where $\Gamma_1 \vdash a :^{q \cdot q_0} A_1 + A_2$ and $\Gamma_2, x_1 :^{q \cdot q_0} A_1 \vdash b_1 :^q B$ and $\Gamma_2, x_2 :^{q \cdot q_0} A_2 \vdash b_2 :^q B$.
  Let $\vdash \mathbf{case}_{q_0}\ a\ \mathbf{of}\ x_1.b_1\ ;\ x_2.b_2 \leadsto c$. By inversion:
  - $\vdash \mathbf{case}_{q_0}\ a\ \mathbf{of}\ x_1.b_1\ ;\ x_2.b_2 \leadsto \mathbf{case}_{q_0}\ a'\ \mathbf{of}\ x_1.b_1\ ;\ x_2.b_2$, when $\vdash a \leadsto a'$.
    Need to show: $\Gamma_1 + \Gamma_2 \vdash \mathbf{case}_{q_0}\ a'\ \mathbf{of}\ x_1.b_1\ ;\ x_2.b_2 :^q B$.
    Follows by IH and rule SLDC-Case.
  - $\vdash \mathbf{case}_{q_0}\ (\mathbf{inj}_1\ a_1)\ \mathbf{of}\ x_1.b_1\ ;\ x_2.b_2 \leadsto b_1\{a_1/x_1\}$.
    Need to show: $\Gamma_1 + \Gamma_2 \vdash b_1\{a_1/x_1\} :^q B$.
    By inversion on $\Gamma_1 \vdash \mathbf{inj}_1\ a_1 :^{q \cdot q_0} A_1 + A_2$, we have, $\Gamma_1 \vdash a_1 :^{q \cdot q_0} A_1$.
    This case, then, follows by applying the substitution lemma.
  - $\vdash \mathbf{case}_{q_0}\ (\mathbf{inj}_2\ a_2)\ \mathbf{of}\ x_1.b_1\ ;\ x_2.b_2 \leadsto b_2\{a_2/x_2\}$.
    Similar to the previous case.

- Rules SLDC-SubL and SLDC-SubR. Follows by IH.

□

Theorem D.8 (Progress (Theorem 5.7)) If $\varnothing \vdash a :^q A$, then either a is a value or there exists a' such that $\vdash a \leadsto a'$.

Proof. By induction on $\varnothing \vdash a :^q A$.

- Rule SLDC-Var. Does not apply since the context here is empty.

- Rule SLDC-App. Have: $\varnothing \vdash b\ a^r :^q B$ where $\varnothing \vdash b :^q {}^r A \to B$ and $\varnothing \vdash a :^{q \cdot r} A$.
  Need to show: $\exists c, \vdash b\ a^r \leadsto c$.
  By IH, b is either a value or $\vdash b \leadsto b'$, for some b'.
  If b is a value, then $b = \lambda^r x : A.b''$, for some b''. Therefore, $\vdash b\ a^r \leadsto b''\{a/x\}$.
  Otherwise, $\vdash b\ a^r \leadsto b'\ a^r$.

- Rule SLDC-LetPair. Have: $\varnothing \vdash \mathbf{let}_{q_0}\ (x^r, y) = a\ \mathbf{in}\ b :^q B$ where $\varnothing \vdash a :^{q \cdot q_0}\ {}^r A_1 \times A_2$ and $x :^{q \cdot q_0 \cdot r} A_1, y :^{q \cdot q_0} A_2 \vdash b :^q B$.
  Need to show: $\exists c, \vdash \mathbf{let}_{q_0}\ (x^r, y) = a\ \mathbf{in}\ b \leadsto c$.
  By IH, a is either a value or $\vdash a \leadsto a'$, for some a'.
  If a is a value, then $a = ({a_1}^r, a_2)$. Therefore, $\vdash \mathbf{let}_{q_0}\ (x^r, y) = a\ \mathbf{in}\ b \leadsto b\{a_1/x\}\{a_2/y\}$.
  Otherwise, $\vdash \mathbf{let}_{q_0}\ (x^r, y) = a\ \mathbf{in}\ b \leadsto \mathbf{let}_{q_0}\ (x^r, y) = a'\ \mathbf{in}\ b$.

- Rule SLDC-LetUnit. Have: $\varnothing \vdash \mathbf{let}_{q_0}\ \mathbf{unit} = a\ \mathbf{in}\ b :^q B$ where $\varnothing \vdash a :^{q \cdot q_0} \mathbf{Unit}$ and $\varnothing \vdash b :^q B$.
  Need to show: $\exists c, \vdash \mathbf{let}_{q_0}\ \mathbf{unit} = a\ \mathbf{in}\ b \leadsto c$.
  By IH, a is either a value or $\vdash a \leadsto a'$, for some a'.
  If a is a value, then $a = \mathbf{unit}$. Therefore, $\vdash \mathbf{let}_{q_0}\ \mathbf{unit} = a\ \mathbf{in}\ b \leadsto b$.
  Otherwise, $\vdash \mathbf{let}_{q_0}\ \mathbf{unit} = a\ \mathbf{in}\ b \leadsto \mathbf{let}_{q_0}\ \mathbf{unit} = a'\ \mathbf{in}\ b$.

- Rule SLDC-Case. Have: $\varnothing \vdash \mathbf{case}_{q_0}\ a\ \mathbf{of}\ x_1.b_1\ ;\ x_2.b_2 :^q B$ where $\varnothing \vdash a :^{q \cdot q_0} A_1 + A_2$ and $x_1 :^{q \cdot q_0} A_1 \vdash b_1 :^q B$ and $x_2 :^{q \cdot q_0} A_2 \vdash b_2 :^q B$.
  Need to show: $\exists c, \vdash \mathbf{case}_{q_0}\ a\ \mathbf{of}\ x_1.b_1\ ;\ x_2.b_2 \leadsto c$.
  By IH, a is either a value or $\vdash a \leadsto a'$, for some a'.
  If a is a value, then $a = \mathbf{inj}_1\ a_1$ or $a = \mathbf{inj}_2\ a_2$.
  Then, $\vdash \mathbf{case}_{q_0}\ a\ \mathbf{of}\ x_1.b_1\ ;\ x_2.b_2 \leadsto b_1\{a_1/x_1\}$ or $\vdash \mathbf{case}_{q_0}\ a\ \mathbf{of}\ x_1.b_1\ ;\ x_2.b_2 \leadsto b_2\{a_2/x_2\}$.
  Otherwise, $\vdash \mathbf{case}_{q_0}\ a\ \mathbf{of}\ x_1.b_1\ ;\ x_2.b_2 \leadsto \mathbf{case}_{q_0}\ a'\ \mathbf{of}\ x_1.b_1\ ;\ x_2.b_2$.

- Rules SLDC-SubL and SLDC-SubR. Follows by IH.

- Rules SLDC-Lam, SLDC-Pair, and SLDC-Unit. The terms typed by these rules are values.

□

Lemma D.9 (Multiplication (Lemma 5.8)) If $\Gamma \vdash a :^\ell A$, then $m_0 \sqcup \Gamma \vdash a :^{m_0 \sqcup \ell} A$.

Proof. By induction on $\Gamma \vdash a :^\ell A$.

- Rule SLDC-VarD. Have: $\top \sqcup \Gamma_1, x :^\ell A, \top \sqcup \Gamma_2 \vdash x :^\ell A$.
  Need to show: $\top \sqcup \Gamma_1, x :^{m_0 \sqcup \ell} A, \top \sqcup \Gamma_2 \vdash x :^{m_0 \sqcup \ell} A$.
  This case follows by rule SLDC-VarD.

- Rule SLDC-LamD. Have: $\Gamma \vdash \lambda^m x : A.b :^{\ell} {}^{m}A \to B$ where $\Gamma, x :^{\ell \sqcup m} A \vdash b :^{\ell} B$.
  Need to show: $m_0 \sqcup \Gamma \vdash \lambda^m x : A.b :^{m_0 \sqcup \ell} {}^{m}A \to B$.
  By IH, $m_0 \sqcup \Gamma, x :^{m_0 \sqcup (\ell \sqcup m)} A \vdash b :^{m_0 \sqcup \ell} B$.
  This case, then, follows by rule SLDC-LamD, using associativity of $\sqcup$.

- Rule SLDC-AppD. Have: $\Gamma_1 \sqcap \Gamma_2 \vdash b\ a^m :^{\ell} B$ where $\Gamma_1 \vdash b :^{\ell} {}^{m}A \to B$ and $\Gamma_2 \vdash a :^{\ell \sqcup m} A$.
  Need to show: $m_0 \sqcup (\Gamma_1 \sqcap \Gamma_2) \vdash b\ a^m :^{m_0 \sqcup \ell} B$.
  By IH, $m_0 \sqcup \Gamma_1 \vdash b :^{m_0 \sqcup \ell} {}^{m}A \to B$ and $m_0 \sqcup \Gamma_2 \vdash a :^{m_0 \sqcup (\ell \sqcup m)} A$.
  By rule SLDC-AppD, using associativity of $\sqcup$, $(m_0 \sqcup \Gamma_1) \sqcap (m_0 \sqcup \Gamma_2) \vdash b\ a^m :^{m_0 \sqcup \ell} B$.
  Now, for elements ℓ_1, ℓ_2 and ℓ_3 of any lattice, $\ell_1 \sqcup (\ell_2 \sqcap \ell_3) \sqsubseteq (\ell_1 \sqcup \ell_2) \sqcap (\ell_1 \sqcup \ell_3)$.
  This case, then, follows by rule SLDC-SubLD, using the above relation.

- Rule SLDC-PairD. Have: $\Gamma_1 \sqcap \Gamma_2 \vdash ({a_1}^m, a_2) :^{\ell} {}^{m}A_1 \times A_2$ where $\Gamma_1 \vdash a_1 :^{\ell \sqcup m} A_1$ and $\Gamma_2 \vdash a_2 :^{\ell} A_2$.
  Need to show: $m_0 \sqcup (\Gamma_1 \sqcap \Gamma_2) \vdash ({a_1}^m, a_2) :^{m_0 \sqcup \ell} {}^{m}A_1 \times A_2$.
  By IH, $m_0 \sqcup \Gamma_1 \vdash a_1 :^{m_0 \sqcup (\ell \sqcup m)} A_1$ and $m_0 \sqcup \Gamma_2 \vdash a_2 :^{m_0 \sqcup \ell} A_2$.
  By rule SLDC-PairD, using associativity of $\sqcup$, $(m_0 \sqcup \Gamma_1) \sqcap (m_0 \sqcup \Gamma_2) \vdash ({a_1}^m, a_2) :^{m_0 \sqcup \ell} {}^{m}A_1 \times A_2$.
  For ℓ_1, ℓ_2, ℓ_3, we have, $\ell_1 \sqcup (\ell_2 \sqcap \ell_3) \sqsubseteq (\ell_1 \sqcup \ell_2) \sqcap (\ell_1 \sqcup \ell_3)$.
  This case, then, follows by rule SLDC-SubLD, using the above relation.

- Rule SLDC-LetPairD. Have: $\Gamma_1 \sqcap \Gamma_2 \vdash \textbf{let}\ (x^m, y) = a\ \textbf{in}\ b :^{\ell} B$ where $\Gamma_1 \vdash a :^{\ell} {}^{m}A_1 \times A_2$ and $\Gamma_2, x :^{\ell \sqcup m} A_1, y :^{\ell} A_2 \vdash b :^{\ell} B$.
  Need to show: $m_0 \sqcup (\Gamma_1 \sqcap \Gamma_2) \vdash \textbf{let}\ (x^m, y) = a\ \textbf{in}\ b :^{m_0 \sqcup \ell} B$.
  By IH, $m_0 \sqcup \Gamma_1 \vdash a :^{m_0 \sqcup \ell} {}^{m}A_1 \times A_2$ and $m_0 \sqcup \Gamma_2, x :^{m_0 \sqcup (\ell \sqcup m)} A_1, y :^{m_0 \sqcup \ell} A_2 \vdash b :^{m_0 \sqcup \ell} B$.
  This case follows by rules SLDC-LetPairD and SLDC-SubLD, using associativity of $\sqcup$ and the distributive inequality.

- Rule SLDC-UnitD. Have: $\top \sqcup \Gamma \vdash \textbf{unit} :^{\ell} \textbf{Unit}$.
  Need to show: $\top \sqcup \Gamma \vdash \textbf{unit} :^{m_0 \sqcup \ell} \textbf{Unit}$.
  Follows by rule SLDC-UnitD.

- Rule SLDC-LetUnitD. Have: $\Gamma_1 \sqcap \Gamma_2 \vdash \textbf{let unit} = a\ \textbf{in}\ b :^{\ell} B$ where $\Gamma_1 \vdash a :^{\ell} \textbf{Unit}$ and $\Gamma_2 \vdash b :^{\ell} B$.
  Need to show: $m_0 \sqcup (\Gamma_1 \sqcap \Gamma_2) \vdash \textbf{let unit} = a\ \textbf{in}\ b :^{m_0 \sqcup \ell} B$.
  By IH, $m_0 \sqcup \Gamma_1 \vdash a :^{m_0 \sqcup \ell} \textbf{Unit}$ and $m_0 \sqcup \Gamma_2 \vdash b :^{m_0 \sqcup \ell} B$.
  This case follows by rules SLDC-LetUnitD and SLDC-SubLD, using the distributive inequality.

- Rule SLDC-Inj1D. Have: $\Gamma \vdash \textbf{inj}_1\ a_1 :^{\ell} A_1 + A_2$ where $\Gamma \vdash a_1 :^{\ell} A_1$.
  Need to show: $m_0 \sqcup \Gamma \vdash \textbf{inj}_1\ a_1 :^{m_0 \sqcup \ell} A_1 + A_2$.
  By IH, $m_0 \sqcup \Gamma \vdash a_1 :^{m_0 \sqcup \ell} A_1$.
  This case, then, follows by rule SLDC-Inj1D.

- Rule SLDC-Inj2D. Similar to rule SLDC-Inj1D.

- Rule SLDC-CaseD. Have: $\Gamma_1 \sqcap \Gamma_2 \vdash \textbf{case}\ a\ \textbf{of}\ x_1.b_1\ ;\ x_2.b_2 :^{\ell} B$ where $\Gamma_1 \vdash a :^{\ell} A_1 + A_2$ and $\Gamma_2, x_1 :^{\ell} A_1 \vdash b_1 :^{\ell} B$ and $\Gamma_2, x_2 :^{\ell} A_2 \vdash b_2 :^{\ell} B$.

Need to show: $m_0 \sqcup (\Gamma_1 \sqcap \Gamma_2) \vdash \textbf{case } a \textbf{ of } x_1.b_1 \,;\, x_2.b_2 :^{m_0 \sqcup \ell} B$.

By IH, $m_0 \sqcup \Gamma_1 \vdash a_1 :^{m_0 \sqcup \ell} A_1 + A_2$ and $m_0 \sqcup \Gamma_2, x_1 :^{m_0 \sqcup \ell} A_1 \vdash b_1 :^{m_0 \sqcup \ell} B$ and $m_0 \sqcup \Gamma_2, x_2 :^{m_0 \sqcup \ell} A_2 \vdash b_2 :^{m_0 \sqcup \ell} B$.

This case follows by rules SLDC-CaseD and SLDC-SubLD, using the distributive inequality.

- Rule SLDC-SubLD. Have: $\Gamma \vdash a :^{\ell} A$ where $\Gamma' \vdash a :^{\ell} A$ and $\Gamma \sqsubseteq \Gamma'$.
  Need to show: $m_0 \sqcup \Gamma \vdash a :^{m_0 \sqcup \ell} A$.
  By IH, $m_0 \sqcup \Gamma' \vdash a :^{m_0 \sqcup \ell} A$.
  Since $\Gamma \sqsubseteq \Gamma'$, so $m_0 \sqcup \Gamma \sqsubseteq m_0 \sqcup \Gamma'$.
  This case, then, follows by rule SLDC-SubLD.
- Rule SLDC-SubRD. Have: $\Gamma \vdash a :^{\ell'} A$ where $\Gamma \vdash a :^{\ell} A$ and $\ell \sqsubseteq \ell'$.
  Need to show: $m_0 \sqcup \Gamma \vdash a :^{m_0 \sqcup \ell'} A$.
  By IH, $m_0 \sqcup \Gamma \vdash a :^{m_0 \sqcup \ell} A$.
  Since $\ell \sqsubseteq \ell'$, so $m_0 \sqcup \ell \sqsubseteq m_0 \sqcup \ell'$.
  This case, then, follows by rule SLDC-SubRD.

□

Lemma D.10 (Splitting (Lemma 5.9)) If $\Gamma \vdash a :^{\ell_1 \sqcap \ell_2} A$, then there exists Γ_1 and Γ_2 such that $\Gamma_1 \vdash a :^{\ell_1} A$ and $\Gamma_2 \vdash a :^{\ell_2} A$ and $\Gamma = \Gamma_1 \sqcap \Gamma_2$.

Proof. Have: $\Gamma \vdash a :^{\ell_1 \sqcap \ell_2} A$. By rule SLDC-SubRD, $\Gamma \vdash a :^{\ell_1} A$ and $\Gamma \vdash a :^{\ell_2} A$. The lemma follows by setting $\Gamma_1 := \Gamma$ and $\Gamma_2 := \Gamma$. □

Lemma D.11 (Weakening (Lemma 5.10)) If $\Gamma_1, \Gamma_2 \vdash a :^{\ell} A$, then $\Gamma_1, z :^{\top} C, \Gamma_2 \vdash a :^{\ell} A$.

Proof. By induction on $\Gamma_1, \Gamma_2 \vdash a :^{\ell} A$. □

Lemma D.12 (Substitution (Lemma 5.11)) If $\Gamma_1, z :^{m_0} C, \Gamma_2 \vdash a :^{\ell} A$ and $\Gamma \vdash c :^{m_0} C$ where $\lfloor \Gamma_1 \rfloor = \lfloor \Gamma \rfloor$, then $\Gamma_1 \sqcap \Gamma, \Gamma_2 \vdash a\{c/z\} :^{\ell} A$.

Proof. By induction on $\Gamma_1, z :^{m_0} C, \Gamma_2 \vdash a :^{\ell} A$.

- Rule SLDC-VarD. There are three cases to consider.
  - $\top \sqcup \Gamma_{11}, z :^{\top} C, \top \sqcup \Gamma_{12}, x :^{\ell} A, \top \sqcup \Gamma_2 \vdash x :^{\ell} A$. Also, $\Gamma \vdash c :^{\top} C$ where $\lfloor \Gamma_{11} \rfloor = \lfloor \Gamma \rfloor$.
    Need to show: $\Gamma, \top \sqcup \Gamma_{12}, x :^{\ell} A, \top \sqcup \Gamma_2 \vdash x :^{\ell} A$.
    Follows by rule SLDC-VarD and rule SLDC-SubLD.
  - $\top \sqcup \Gamma_1, x :^{\ell} A, \top \sqcup \Gamma_2 \vdash x :^{\ell} A$. Also, $\Gamma \vdash a :^{\ell} A$ where $\lfloor \Gamma_1 \rfloor = \lfloor \Gamma \rfloor$.
    Need to show: $\Gamma, \top \sqcup \Gamma_2 \vdash a :^{\ell} A$.
    Follows by weakening lemma D.11.

– $\top \sqcup \Gamma_1, x :^{\ell} A, \top \sqcup \Gamma_{21}, z :^{\top} C, \top \sqcup \Gamma_{22} \vdash x :^{\ell} A$. Also, $\Gamma_{31}, x :^{m} A, \Gamma_{32} \vdash c :^{\top} C$ where $\lfloor \Gamma_{31} \rfloor = \lfloor \Gamma_1 \rfloor$ and $\lfloor \Gamma_{32} \rfloor = \lfloor \Gamma_{21} \rfloor$.
  Need to show: $\Gamma_{31}, x :^{\ell \sqcap m} A, \Gamma_{32}, \top \sqcup \Gamma_{22} \vdash x :^{\ell} A$.
  Follows by rule SLDC-VarD and rule SLDC-SubLD.

- Rule SLDC-LamD. Have: $\Gamma_1, z :^{m_0} C, \Gamma_2 \vdash \lambda^m x : A.b :^{\ell}\ {}^{m}A \to B$ where $\Gamma_1, z :^{m_0} C, \Gamma_2, x :^{\ell \sqcup m} A \vdash b :^{\ell} B$. Also, $\Gamma \vdash c :^{m_0} C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
  Need to show: $\Gamma_1 \sqcap \Gamma, \Gamma_2 \vdash \lambda^m x : A.b\{c/z\} :^{\ell}\ {}^{m}A \to B$.
  Follows by IH and rule SLDC-LamD.

- Rule SLDC-AppD. Have: $\Gamma_{11} \sqcap \Gamma_{12}, z :^{m_{01} \sqcap m_{02}} C, \Gamma_{21} \sqcap \Gamma_{22} \vdash b\ a^m :^{\ell} B$ where $\Gamma_{11}, z :^{m_{01}} C, \Gamma_{21} \vdash b :^{\ell}\ {}^{m}A \to B$ and $\Gamma_{12}, z :^{m_{02}} C, \Gamma_{22} \vdash a :^{\ell \sqcup m} A$. Also, $\Gamma \vdash c :^{m_{01} \sqcap m_{02}} C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_{11} \rfloor$.
  Need to show: $\Gamma_{11} \sqcap \Gamma_{12} \sqcap \Gamma, \Gamma_{21} \sqcap \Gamma_{22} \vdash b\{c/z\}\ a\{c/z\}^m :^{\ell} B$.
  Since $\Gamma \vdash c :^{m_{01} \sqcap m_{02}} C$, so by lemma D.10, $\exists \Gamma_{31}, \Gamma_{32}$ such that $\Gamma_{31} \vdash c :^{m_{01}} C$ and $\Gamma_{32} \vdash c :^{m_{02}} C$ and $\Gamma = \Gamma_{31} \sqcap \Gamma_{32}$.
  By IH, $\Gamma_{11} \sqcap \Gamma_{31}, \Gamma_{21} \vdash b\{c/z\} :^{\ell}\ {}^{m}A \to B$ and $\Gamma_{12} \sqcap \Gamma_{32}, \Gamma_{22} \vdash a\{c/z\} :^{\ell \sqcup m} A$.
  This case, then, follows by rule SLDC-AppD.

- Rule SLDC-PairD. Have: $\Gamma_{11} \sqcap \Gamma_{12}, z :^{m_{01} \sqcap m_{02}} C, \Gamma_{21} \sqcap \Gamma_{22} \vdash (a_1{}^m, a_2) :^{\ell}\ {}^{m}A_1 \times A_2$ where $\Gamma_{11}, z :^{m_{01}} C, \Gamma_{21} \vdash a_1 :^{\ell \sqcup m} A_1$ and $\Gamma_{12}, z :^{m_{02}} C, \Gamma_{22} \vdash a_2 :^{\ell} A_2$. Also, $\Gamma \vdash c :^{m_{01} \sqcap m_{02}} C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_{11} \rfloor$.
  Need to show: $\Gamma_{11} \sqcap \Gamma_{12} \sqcap \Gamma, \Gamma_{21} \sqcap \Gamma_{22} \vdash (a_1\{c/z\}^r, a_2\{c/z\}) :^{\ell}\ {}^{m}A_1 \times A_2$.
  Since $\Gamma \vdash c :^{m_{01} \sqcap m_{02}} C$, so by lemma D.10, $\exists \Gamma_{31}, \Gamma_{32}$ such that $\Gamma_{31} \vdash c :^{m_{01}} C$ and $\Gamma_{32} \vdash c :^{m_{02}} C$ and $\Gamma = \Gamma_{31} \sqcap \Gamma_{32}$.
  By IH, $\Gamma_{11} \sqcap \Gamma_{31}, \Gamma_{21} \vdash a_1\{c/z\} :^{\ell \sqcup m} A_1$ and $\Gamma_{12} \sqcap \Gamma_{32}, \Gamma_{22} \vdash a_2\{c/z\} :^{\ell} A_2$.
  This case, then, follows by rule SLDC-PairD.

- Rule SLDC-LetPairD. Have: $\Gamma_{11} \sqcap \Gamma_{12}, z :^{m_{01} \sqcap m_{02}} C, \Gamma_{21} \sqcap \Gamma_{22} \vdash \textbf{let}\ (x^m, y) = a\ \textbf{in}\ b :^{\ell} B$ where $\Gamma_{11}, z :^{m_{01}} C, \Gamma_{21} \vdash a :^{\ell}\ {}^{m}A_1 \times A_2$ and $\Gamma_{12}, z :^{m_{02}} C, \Gamma_{22}, x :^{\ell \sqcup m} A_1, y :^{\ell} A_2 \vdash b :^{\ell} B$. Also, $\Gamma \vdash c :^{m_{01} \sqcap m_{02}} C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_{11} \rfloor$.
  Need to show: $\Gamma_{11} \sqcap \Gamma_{12} \sqcap \Gamma, \Gamma_{21} \sqcap \Gamma_{22} \vdash \textbf{let}\ (x^m, y) = a\{c/z\}\ \textbf{in}\ b\{c/z\} :^{\ell} B$.
  Since $\Gamma \vdash c :^{m_{01} \sqcap m_{02}} C$, so by lemma D.10, $\exists \Gamma_{31}, \Gamma_{32}$ such that $\Gamma_{31} \vdash c :^{m_{01}} C$ and $\Gamma_{32} \vdash c :^{m_{02}} C$ and $\Gamma = \Gamma_{31} \sqcap \Gamma_{32}$.
  By IH, $\Gamma_{11} \sqcap \Gamma_{31}, \Gamma_{21} \vdash a\{c/z\} :^{\ell}\ {}^{m}A_1 \times A_2$ and $\Gamma_{12} \sqcap \Gamma_{32}, \Gamma_{22}, x :^{\ell \sqcup m} A_1, y :^{\ell} A_2 \vdash b\{c/z\} :^{\ell} B$.
  This case, then, follows by rule SLDC-LetPairD.

- Rule SLDC-UnitD. Have: $\top \sqcup \Gamma_1, z :^{\top} C, \top \sqcup \Gamma_2 \vdash \textbf{unit} :^{\ell} \textbf{Unit}$. Also $\Gamma \vdash c :^{\top} C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
  Need to show: $\Gamma, \top \sqcup \Gamma_2 \vdash \textbf{unit} :^{\ell} \textbf{Unit}$.
  Follows by rule SLDC-UnitD and rule SLDC-SubLD.

- Rule SLDC-LetUnitD. Have: $\Gamma_{11} \sqcap \Gamma_{12}, z :^{m_{01} \sqcap m_{02}} C, \Gamma_{21} \sqcap \Gamma_{22} \vdash \textbf{let unit} = a\ \textbf{in}\ b :^{\ell} B$ where $\Gamma_{11}, z :^{m_{01}} C, \Gamma_{21} \vdash a :^{\ell} \textbf{Unit}$ and $\Gamma_{12}, z :^{m_{02}} C, \Gamma_{22} \vdash b :^{\ell} B$. Also, $\Gamma \vdash c :^{m_{01} \sqcap m_{02}} C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_{11} \rfloor$.
  Need to show: $\Gamma_{11} \sqcap \Gamma_{12} \sqcap \Gamma, \Gamma_{21} \sqcap \Gamma_{22} \vdash \textbf{let unit} = a\{c/z\}\ \textbf{in}\ b\{c/z\} :^{\ell} B$.
  Since $\Gamma \vdash c :^{m_{01} \sqcap m_{02}} C$, so by lemma D.10, $\exists \Gamma_{31}, \Gamma_{32}$ such that $\Gamma_{31} \vdash c :^{m_{01}} C$ and $\Gamma_{32} \vdash c :^{m_{02}} C$ and

$\Gamma = \Gamma_{31} \sqcap \Gamma_{32}$.
By IH, $\Gamma_{11} \sqcap \Gamma_{31}, \Gamma_{21} \vdash a\{c/z\} :^{\ell} \textbf{Unit}$ and $\Gamma_{12} \sqcap \Gamma_{32}, \Gamma_{22} \vdash b\{c/z\} :^{\ell} B$.
This case, then, follows by rule SLDC-LetUnitD.

- Rule SLDC-Inj1D. Have: $\Gamma_1, z :^{m_0} C, \Gamma_2 \vdash \textbf{inj}_1\ a_1 :^{\ell} A_1 + A_2$ where $\Gamma_1, z :^{m_0} C, \Gamma_2 \vdash a_1 :^{\ell} A_1$. Also, $\Gamma \vdash c :^{m_0} C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
Need to show: $\Gamma_1 \sqcap \Gamma, \Gamma_2 \vdash \textbf{inj}_1\ a_1\{c/z\} :^{\ell} A_1 + A_2$.
By IH, $\Gamma_1 \sqcap \Gamma, \Gamma_2 \vdash a_1\{c/z\} :^{\ell} A_1$.
This case, then, follows by rule SLDC-Inj1D.

- Rule SLDC-Inj2D. Similar to rule SLDC-Inj1D.

- Rule SLDC-CaseD. Have: $\Gamma_{11} \sqcap \Gamma_{12}, z :^{m_{01} \sqcap m_{02}} C, \Gamma_{21} \sqcap \Gamma_{22} \vdash \textbf{case}\ a\ \textbf{of}\ x_1.b_1\ ;\ x_2.b_2 :^{\ell} B$ where $\Gamma_{11}, z :^{m_{01}} C, \Gamma_{21} \vdash a :^{\ell} A_1 + A_2$ and $\Gamma_{12}, z :^{m_{02}} C, \Gamma_{22}, x_1 :^{\ell} A_1 \vdash b_1 :^{\ell} B$ and $\Gamma_{12}, z :^{m_{02}} C, \Gamma_{22}, x_2 :^{\ell} A_2 \vdash b_2 :^{\ell} B$. Also, $\Gamma \vdash c :^{m_{01} \sqcap m_{02}} C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_{11} \rfloor$.
Need to show: $\Gamma_{11} \sqcap \Gamma_{12} \sqcap \Gamma, \Gamma_{21} \sqcap \Gamma_{22} \vdash \textbf{case}\ a\{c/z\}\ \textbf{of}\ x_1.b_1\{c/z\}\ ;\ x_2.b_2\{c/z\} :^{\ell} B$.
Since $\Gamma \vdash c :^{m_{01} \sqcap m_{02}} C$, so by lemma D.10, $\exists \Gamma_{31}, \Gamma_{32}$ such that $\Gamma_{31} \vdash c :^{m_{01}} C$ and $\Gamma_{32} \vdash c :^{m_{02}} C$ and $\Gamma = \Gamma_{31} \sqcap \Gamma_{32}$.
By IH, $\Gamma_{11} \sqcap \Gamma_{31}, \Gamma_{21} \vdash a\{c/z\} :^{\ell} A_1 + A_2$ and $\Gamma_{12} \sqcap \Gamma_{32}, \Gamma_{22}, x_1 :^{\ell} A_1 \vdash b_1\{c/z\} :^{\ell} B$ and $\Gamma_{12} \sqcap \Gamma_{32}, \Gamma_{22}, x_2 :^{\ell} A_2 \vdash b_2\{c/z\} :^{\ell} B$.
This case, then, follows by rule SLDC-CaseD.

- Rule SLDC-SubLD. Have: $\Gamma_1, z :^{m_0} C, \Gamma_2 \vdash a :^{\ell} A$ where $\Gamma_1', z :^{m_0'} C, \Gamma_2' \vdash a :^{\ell} A$ such that $\Gamma_1 \sqsubseteq \Gamma_1'$ and $m_0 \sqsubseteq m_0'$ and $\Gamma_2 \sqsubseteq \Gamma_2'$. Also, $\Gamma \vdash c :^{m_0} C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
Need to show: $\Gamma_1 \sqcap \Gamma, \Gamma_2 \vdash a\{c/z\} :^{\ell} A$.
Since $m_0 \sqsubseteq m_0'$, by rule SLDC-SubRD, $\Gamma \vdash c :^{m_0'} C$.
By IH, $\Gamma_1' \sqcap \Gamma, \Gamma_2' \vdash a\{c/z\} :^{\ell} A$.
This case, then, follows by rule SLDC-SubLD.

- Rule SLDC-SubRD. Have: $\Gamma_1, z :^{m_0} C, \Gamma_2 \vdash a :^{\ell'} A$ where $\Gamma_1, z :^{m_0} C, \Gamma_2 \vdash a :^{\ell} A$ and $\ell \sqsubseteq \ell'$. Also, $\Gamma \vdash c :^{m_0} C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
Need to show: $\Gamma_1 \sqcap \Gamma, \Gamma_2 \vdash a\{c/z\} :^{\ell'} A$.
By IH, $\Gamma_1 \sqcap \Gamma, \Gamma_2 \vdash a\{c/z\} :^{\ell} A$.
This case, then, follows by rule SLDC-SubRD.

□

Lemma D.13 (Restricted Upgrading) If $\Gamma_1, x :^{m} A, \Gamma_2 \vdash b :^{\ell} B$ and $\ell_0 \sqsubseteq \ell$, then $\Gamma_1, x :^{\ell_0 \sqcup m} A, \Gamma_2 \vdash b :^{\ell} B$.

Proof. By Lemma D.9, $\ell_0 \sqcup \Gamma_1, x :^{\ell_0 \sqcup m} A, \ell_0 \sqcup \Gamma_2 \vdash b :^{\ell_0 \sqcup \ell} B$.
Since $\ell_0 \sqsubseteq \ell$, so $\ell_0 \sqcup \Gamma_1, x :^{\ell_0 \sqcup m} A, \ell_0 \sqcup \Gamma_2 \vdash b :^{\ell} B$.
Again, since $\Gamma_1 \sqsubseteq \ell_0 \sqcup \Gamma_1$ and $\Gamma_2 \sqsubseteq \ell_0 \sqcup \Gamma_2$, so by rule SLDC-SubLD, $\Gamma_1, x :^{\ell_0 \sqcup m} A, \Gamma_2 \vdash b :^{\ell} B$. □

Theorem D.14 (Preservation (Theorem 5.12)) If $\Gamma \vdash a :^{\ell} A$ and $\vdash a \rightsquigarrow a'$, then $\Gamma \vdash a' :^{\ell} A$.

Proof. By induction on $\Gamma \vdash a :^{\ell} A$ and subsequent inversion on $\vdash a \leadsto a'$.

- Rule SLDC-AppD. Have: $\Gamma_1 \sqcap \Gamma_2 \vdash b\ a^m :^{\ell} B$ where $\Gamma_1 \vdash b :^{\ell} {}^m A \to B$ and $\Gamma_2 \vdash a :^{\ell \sqcup m} A$.
  Let $\vdash b\ a^m \leadsto c$. By inversion:
  - $\vdash b\ a^m \leadsto b'\ a^m$, when $\vdash b \leadsto b'$.
    Need to show: $\Gamma_1 \sqcap \Gamma_2 \vdash b'\ a^m :^{\ell} B$.
    Follows by IH and rule SLDC-AppD.
  - $b = \lambda^m x : A'.b'$ and $\vdash b\ a^m \leadsto b'\{a/x\}$.
    Need to show: $\Gamma_1 \sqcap \Gamma_2 \vdash b'\{a/x\} :^{\ell} B$.
    By inversion on $\Gamma_1 \vdash \lambda^m x : A'.b' :^{\ell} {}^m A \to B$, we get $A' = A$ and $\Gamma_1, x :^{\ell_0 \sqcup m} A \vdash b' :^{\ell_0} B$, for some $\ell_0 \sqsubseteq \ell$.
    By rule SLDC-SubRD, $\Gamma_1, x :^{\ell_0 \sqcup m} A \vdash b' :^{\ell} B$.
    By lemma D.13, $\Gamma_1, x :^{\ell \sqcup m} A \vdash b' :^{\ell} B$.
    This case, then, follows by the substitution lemma.
- Rule SLDC-LetPairD. Have: $\Gamma_1 \sqcap \Gamma_2 \vdash \textbf{let}\ (x^m, y) = a\ \textbf{in}\ b :^{\ell} B$ where $\Gamma_1 \vdash a :^{\ell} {}^m A_1 \times A_2$ and $\Gamma_2, x :^{\ell \sqcup m} A_1, y :^{\ell} A_2 \vdash b :^{\ell} B$.
  Let $\vdash \textbf{let}\ (x^m, y) = a\ \textbf{in}\ b \leadsto c$. By inversion:
  - $\vdash \textbf{let}\ (x^m, y) = a\ \textbf{in}\ b \leadsto \textbf{let}\ (x^m, y) = a'\ \textbf{in}\ b$, when $\vdash a \leadsto a'$.
    Need to show: $\Gamma_1 \sqcap \Gamma_2 \vdash \textbf{let}\ (x^m, y) = a'\ \textbf{in}\ b :^{\ell} B$.
    Follows by IH and rule SLDC-LetPairD.
  - $\vdash \textbf{let}\ (x^m, y) = ({a_1}^m, a_2)\ \textbf{in}\ b \leadsto b\{a_1/x\}\{a_2/y\}$.
    Need to show: $\Gamma_1 \sqcap \Gamma_2 \vdash b\{a_1/x\}\{a_2/y\} :^{\ell} B$.
    By inversion on $\Gamma_1 \vdash ({a_1}^m, a_2) :^{\ell} {}^m A_1 \times A_2$, we have:
    $\exists \Gamma_{11}, \Gamma_{12}$ such that $\Gamma_{11} \vdash a_1 :^{\ell \sqcup m} A_1$ and $\Gamma_{12} \vdash a_2 :^{\ell} A_2$ and $\Gamma_1 = \Gamma_{11} \sqcap \Gamma_{12}$.
    This case, then, follows by applying the substitution lemma twice.
- Rule SLDC-LetUnitD. Have: $\Gamma_1 \sqcap \Gamma_2 \vdash \textbf{let unit} = a\ \textbf{in}\ b :^{\ell} B$ where $\Gamma_1 \vdash a :^{\ell} \textbf{Unit}$ and $\Gamma_2 \vdash b :^{\ell} B$.
  Let $\vdash \textbf{let unit} = a\ \textbf{in}\ b \leadsto c$. By inversion:
  - $\vdash \textbf{let unit} = a\ \textbf{in}\ b \leadsto \textbf{let unit} = a'\ \textbf{in}\ b$, when $\vdash a \leadsto a'$.
    Need to show: $\Gamma_1 \sqcap \Gamma_2 \vdash \textbf{let unit} = a'\ \textbf{in}\ b :^{\ell} B$.
    Follows by IH and rule SLDC-LetUnitD.
  - $\vdash \textbf{let unit} = \textbf{unit in}\ b \leadsto b$.
    Need to show: $\Gamma_1 \sqcap \Gamma_2 \vdash b :^{\ell} B$.
    Follows by rule SLDC-SubLD.
- Rule SLDC-CaseD. Have: $\Gamma_1 \sqcap \Gamma_2 \vdash \textbf{case}\ a\ \textbf{of}\ x_1.b_1\ ;\ x_2.b_2 :^{\ell} B$ where $\Gamma_1 \vdash a :^{\ell} A_1 + A_2$ and $\Gamma_2, x_1 :^{\ell} A_1 \vdash b_1 :^{\ell} B$ and $\Gamma_2, x_2 :^{\ell} A_2 \vdash b_2 :^{\ell} B$.
  Let $\vdash \textbf{case}\ a\ \textbf{of}\ x_1.b_1\ ;\ x_2.b_2 \leadsto c$. By inversion:

- $\vdash \textbf{case}\ a\ \textbf{of}\ x_1.b_1\ ;\ x_2.b_2 \rightsquigarrow \textbf{case}\ a'\ \textbf{of}\ x_1.b_1\ ;\ x_2.b_2$, when $\vdash a \rightsquigarrow a'$.
  Need to show: $\Gamma_1 \sqcap \Gamma_2 \vdash \textbf{case}\ a'\ \textbf{of}\ x_1.b_1\ ;\ x_2.b_2 :^\ell B$.
  Follows by IH and rule SLDC-CaseD.
- $\vdash \textbf{case}\ (\textbf{inj}_1\ a_1)\ \textbf{of}\ x_1.b_1\ ;\ x_2.b_2 \rightsquigarrow b_1\{a_1/x_1\}$.
  Need to show $\Gamma_1 \sqcap \Gamma_2 \vdash b_1\{a_1/x_1\} :^\ell B$.
  By inversion on $\Gamma_1 \vdash \textbf{inj}_1\ a_1 :^\ell A_1 + A_2$, we have: $\Gamma_1 \vdash a_1 :^\ell A_1$.
  This case, then, follows by applying the substitution lemma.
- $\vdash \textbf{case}\ (\textbf{inj}_2\ a_2)\ \textbf{of}\ x_1.b_1\ ;\ x_2.b_2 \rightsquigarrow b_2\{a_2/x_2\}$.
  Similar to the previous case.

- Rules SLDC-SubLD and SLDC-SubRD. Follows by IH.

□

Theorem D.15 (Progress (Theorem 5.13)) If $\varnothing \vdash a :^\ell A$, then either a is a value or there exists a' such that $\vdash a \rightsquigarrow a'$.

Proof. By induction on $\varnothing \vdash a :^\ell A$.

- Rule SLDC-VarD. Does not apply since the context here is empty.
- Rule SLDC-AppD. Have: $\varnothing \vdash b\ a^m :^\ell B$ where $\varnothing \vdash b :^\ell {}^m A \to B$ and $\varnothing \vdash a :^{\ell \sqcup m} A$.
  Need to show: $\exists c, \vdash b\ a^m \rightsquigarrow c$.
  By IH, b is either a value or $\vdash b \rightsquigarrow b'$, for some b'.
  If b is a value, then $b = \lambda^m x : A.b''$, for some b''. Therefore, $\vdash b\ a^m \rightsquigarrow b''\{a/x\}$.
  Otherwise, $\vdash b\ a^m \rightsquigarrow b'\ a^m$.
- Rule SLDC-LetPairD. Have: $\varnothing \vdash \textbf{let}\ (x^m, y) = a\ \textbf{in}\ b :^\ell B$ where $\varnothing \vdash a :^\ell {}^m A_1 \times A_2$ and $x :^{\ell \sqcup m} A_1, y :^\ell A_2 \vdash b :^\ell B$.
  Need to show: $\exists c, \vdash \textbf{let}\ (x^m, y) = a\ \textbf{in}\ b \rightsquigarrow c$.
  By IH, a is either a value or $\vdash a \rightsquigarrow a'$, for some a'.
  If a is a value, then $a = ({a_1}^m, a_2)$. Therefore, $\vdash \textbf{let}\ (x^m, y) = a\ \textbf{in}\ b \rightsquigarrow b\{a_1/x\}\{a_2/y\}$.
  Otherwise, $\vdash \textbf{let}\ (x^m, y) = a\ \textbf{in}\ b \rightsquigarrow \textbf{let}\ (x^m, y) = a'\ \textbf{in}\ b$.
- Rule SLDC-LetUnitD. Have: $\varnothing \vdash \textbf{let unit} = a\ \textbf{in}\ b :^\ell B$ where $\varnothing \vdash a :^\ell \textbf{Unit}$ and $\varnothing \vdash b :^\ell B$.
  Need to show: $\exists c, \vdash \textbf{let unit} = a\ \textbf{in}\ b \rightsquigarrow c$.
  By IH, a is either a value or $\vdash a \rightsquigarrow a'$, for some a'.
  If a is a value, then $a = \textbf{unit}$. Therefore, $\vdash \textbf{let unit} = a\ \textbf{in}\ b \rightsquigarrow b$.
  Otherwise, $\vdash \textbf{let unit} = a\ \textbf{in}\ b \rightsquigarrow \textbf{let unit} = a'\ \textbf{in}\ b$.
- Rule SLDC-CaseD. Have: $\varnothing \vdash \textbf{case}\ a\ \textbf{of}\ x_1.b_1\ ;\ x_2.b_2 :^\ell B$ where $\varnothing \vdash a :^\ell A_1 + A_2$ and $x_1 :^\ell A_1 \vdash b_1 :^\ell B$ and $x_2 :^\ell A_2 \vdash b_2 :^\ell B$.

Need to show: $\exists c, \vdash \textbf{case } a \textbf{ of } x_1.b_1 \ ; \ x_2.b_2 \rightsquigarrow c$.
By IH, a is either a value or $\vdash a \rightsquigarrow a'$, for some a'.
If a is a value, then $a = \textbf{inj}_1\ a_1$ or $a = \textbf{inj}_2\ a_2$.
Then, $\vdash \textbf{case } a \textbf{ of } x_1.b_1 \ ; \ x_2.b_2 \rightsquigarrow b_1\{a_1/x_1\}$ or $\vdash \textbf{case } a \textbf{ of } x_1.b_1 \ ; \ x_2.b_2 \rightsquigarrow b_2\{a_2/x_2\}$.
Otherwise, $\vdash \textbf{case } a \textbf{ of } x_1.b_1 \ ; \ x_2.b_2 \rightsquigarrow \textbf{case } a' \textbf{ of } x_1.b_1 \ ; \ x_2.b_2$.

- Rules SLDC-SubLD and SLDC-SubRD. Follows by IH.
- Rules SLDC-LamD, SLDC-PairD, and SLDC-UnitD. The terms typed by these rules are values.

□

Lemma D.16 (Equivalence (Lemma 5.14)) With regard to dependency analysis, SLDC is equivalent to SDC.

Proof. For an arbitrary lattice $\mathcal{L}$, we present meaning-preserving translations from SDC($\mathcal{L}$) to SLDC($\mathcal{L}$) and vice-versa.
The translation function from SDC to SLDC, $\overline{}$, is defined on types and terms as follows:

$$\overline{\textbf{Unit}} = \textbf{Unit} \qquad \overline{A \to B} = {}^{\bot}\overline{A} \to \overline{B} \qquad \overline{A \times B} = {}^{\bot}\overline{A} \times \overline{B} \qquad \overline{A + B} = \overline{A} + \overline{B} \qquad \overline{S_\ell\, A} = {}^{\ell}\overline{A} \times \textbf{Unit}$$

$$\begin{aligned}
\overline{x} &= x & \overline{\textbf{unit}} &= \textbf{unit}\\
\overline{\lambda x : A.b} &= \lambda^{\bot} x : \overline{A}.\overline{b} & \overline{b\ a} &= \overline{b}\ \overline{a}^{\bot}\\
\overline{(a_1, a_2)} &= (\overline{a_1}^{\bot}, \overline{a_2}) & \overline{\textbf{let } (x, y) = a \textbf{ in } b} &= \textbf{let } (x^{\bot}, y) = \overline{a} \textbf{ in } \overline{b}\\
\overline{\textbf{inj}_i\ a_i} &= \textbf{inj}_i\ \overline{a_i} & \overline{\textbf{case } a \textbf{ of } x_1.b_1 \ ; \ x_2.b_2} &= \textbf{case } \overline{a} \textbf{ of } x_1.\overline{b_1} \ ; \ x_2.\overline{b_2}\\
\overline{\textbf{lock}^{\ell}\ a} &= (\overline{a}^{\ell}, \textbf{unit}) & \overline{\textbf{unlock}^{\ell}\ x = a \textbf{ in } b} &= \textbf{let } (x^{\ell}, y) = \overline{a} \textbf{ in } \overline{b} \ \ (y \text{ fresh})
\end{aligned}$$

Conversely, the translation function from SLDC to SDC, $\underline{}$, is defined on types and terms as follows:

$$\underline{\textbf{Unit}} = \textbf{Unit} \qquad \underline{{}^{\ell}A \to B} = S_\ell\ \underline{A} \to \underline{B} \qquad \underline{{}^{\ell}A \times B} = S_\ell\ \underline{A} \times \underline{B} \qquad \underline{A + B} = \underline{A} + \underline{B}$$

$$\begin{aligned}
\underline{x} &= x\\
\underline{\textbf{unit}} &= \textbf{unit}\\
\underline{\textbf{let unit} = a \textbf{ in } b} &= \textbf{let unit} = \underline{a} \textbf{ in } \underline{b}\\
\underline{\lambda^{\ell} x : A.b} &= \lambda y : S_\ell\ \underline{A}.\textbf{unlock}^{\ell}\ x = y \textbf{ in } \underline{b} \ \ (y \text{ fresh})\\
\underline{b\ a^{\ell}} &= \underline{b}\ (\textbf{lock}^{\ell}\ \underline{a})\\
\underline{(a_1{}^{\ell}, a_2)} &= (\textbf{lock}^{\ell}\ \underline{a_1}, \underline{a_2})\\
\underline{\textbf{let } (x^{\ell}, y) = a \textbf{ in } b} &= \textbf{let } (x', y) = \underline{a} \textbf{ in unlock}^{\ell}\ x = x' \textbf{ in } \underline{b} \ \ (x' \text{ fresh})\\
\underline{\textbf{inj}_i\ a_i} &= \textbf{inj}_i\ \underline{a_i}\\
\underline{\textbf{case } a \textbf{ of } x_1.b_1 \ ; \ x_2.b_2} &= \textbf{case } \underline{a} \textbf{ of } x_1.\underline{b_1} \ ; \ x_2.\underline{b_2}
\end{aligned}$$

Notice the similarity between the above translation and the one from GLC($\mathbb{B}_{\geq}$) to SDC($\mathbb{B}_{\geq}$) presented in Section 4.5.1. This similarity is to be expected because the source languages of the two translations, i.e., SLDC and GLC, are quite similar.

Now, we show that the above translations preserve typing and meaning.
First we show: if $\Gamma \vdash a :^{\ell} A$ in SDC($\mathcal{L}$), then $\overrightarrow{\Gamma} \vdash \overline{a} :^{\ell} \overline{A}$ in SLDC($\mathcal{L}$). (Here, $\overrightarrow{\Gamma}$ denotes the context Γ with the types translated.)
By induction on $\Gamma \vdash a :^{\ell} A$. We present some of the interesting cases below.

- Rule SDC-Var. Have: $\Gamma_1, x :^{\ell_0} A, \Gamma_2 \vdash x :^{\ell} A$ where $\ell_0 \sqsubseteq \ell$.
  Need to show: $\overrightarrow{\Gamma_1}, x :^{\ell_0} \overline{A}, \overrightarrow{\Gamma_2} \vdash x :^{\ell} \overline{A}$.
  Follows by rule SLDC-VarD and rule SLDC-SubLD.

- Rule SDC-Lam. Have: $\Gamma \vdash \lambda x : A.b :^{\ell} A \to B$ where $\Gamma, x :^{\ell} A \vdash b :^{\ell} B$.
  Need to show: $\overrightarrow{\Gamma} \vdash \lambda^{\perp} x : \overline{A}.\overline{b} :^{\ell} {}^{\perp}\overline{A} \to \overline{B}$.
  By IH, $\overrightarrow{\Gamma}, x :^{\ell} \overline{A} \vdash \overline{b} :^{\ell} \overline{B}$.
  This case, then, follows by rule SLDC-LamD.

- Rule SDC-App. Have: $\Gamma \vdash b\ a :^{\ell} B$ where $\Gamma \vdash b :^{\ell} A \to B$ and $\Gamma \vdash a :^{\ell} A$.
  Need to show: $\overrightarrow{\Gamma} \vdash \overline{b}\ \overline{a}^{\perp} :^{\ell} \overline{B}$.
  By IH, $\overrightarrow{\Gamma} \vdash \overline{b} :^{\ell} {}^{\perp}\overline{A} \to \overline{B}$ and $\overrightarrow{\Gamma} \vdash \overline{a} :^{\ell} \overline{A}$.
  This case, then, follows by rule SLDC-AppD.

- Rule SDC-Lock. Have: $\Gamma \vdash \mathbf{lock}^{\ell_0}\ a :^{\ell} S_{\ell_0} A$ where $\Gamma \vdash a :^{\ell \sqcup \ell_0} A$.
  Need to show: $\overrightarrow{\Gamma} \vdash (\overline{a}^{\ell}, \mathbf{unit}) :^{\ell} {}^{\ell_0}\overline{A} \times \mathbf{Unit}$.
  By IH, $\overrightarrow{\Gamma} \vdash \overline{a} :^{\ell \sqcup \ell_0} \overline{A}$.
  By rules SLDC-UnitD and SLDC-SubLD, $\overrightarrow{\Gamma} \vdash \mathbf{unit} :^{\ell} \mathbf{Unit}$.
  This case, then, follows by rule SLDC-PairD.

- Rule SDC-Unlock. Have: $\Gamma \vdash \mathbf{unlock}^{\ell_0}\ x = a\ \mathbf{in}\ b :^{\ell} B$ where $\Gamma \vdash a :^{\ell} S_{\ell_0} A$ and $\Gamma, x :^{\ell \sqcup \ell_0} A \vdash b :^{\ell} B$.
  Need to show: $\overrightarrow{\Gamma} \vdash \mathbf{let}\ (x^{\ell_0}, y) = \overline{a}\ \mathbf{in}\ \overline{b} :^{\ell} \overline{B}$, where y is fresh.
  By IH, $\overrightarrow{\Gamma} \vdash \overline{a} :^{\ell} {}^{\ell_0}\overline{A} \times \mathbf{Unit}$ and $\overrightarrow{\Gamma}, x :^{\ell \sqcup \ell_0} \overline{A} \vdash \overline{b} :^{\ell} \overline{B}$.
  Now by weakening and rule SLDC-SubLD, $\overrightarrow{\Gamma}, x :^{\ell \sqcup \ell_0} \overline{A}, y :^{\ell} \mathbf{Unit} \vdash \overline{b} :^{\ell} \overline{B}$.
  This case, then, follows by rule SLDC-LetPairD.

Next, we show that the translation from SDC to SLDC preserves meaning: if $\vdash a \leadsto a'$ in SDC($\mathcal{L}$), then $\vdash \overline{a} \leadsto \overline{a'}$ in SLDC($\mathcal{L}$).
By induction on $\vdash a \leadsto a'$. We present some of the interesting cases below.

- Rule SDCStep-AppBeta. Have: $\vdash (\lambda x : A.b)\ a \leadsto b\{a/x\}$.
  Need to show: $\vdash (\lambda^{\perp} x : \overline{A}.\overline{b})\ \overline{a}^{\perp} \leadsto \overline{b}\{\overline{a}/x\}$.
  Follows by β-rule.

- Rule SDCStep-UnlockBeta. Have: $\vdash \mathbf{unlock}^{\ell_0}\, x = \mathbf{lock}^{\ell_0}\, a\, \mathbf{in}\, b \rightsquigarrow b\{a/x\}$.
  Need to show: $\vdash \mathbf{let}\, (x^{\ell_0}, y) = (\overline{a}^{\ell_0}, \mathbf{unit})\, \mathbf{in}\, \overline{b} \rightsquigarrow \overline{b}\{\overline{a}/x\}$.
  By β-rule, $\vdash \mathbf{let}\, (x^{\ell_0}, y) = (\overline{a}^{\ell_0}, \mathbf{unit})\, \mathbf{in}\, \overline{b} \rightsquigarrow \overline{b}\{\overline{a}/x\}\{\mathbf{unit}/y\}$.
  But since y is fresh, $\overline{b}\{\overline{a}/x\}\{\mathbf{unit}/y\} = \overline{b}\{\overline{a}/x\}$.
  The case follows.

Next, we show that the translation from SLDC to SDC preserves typing: if $\Gamma \vdash a :^{\ell} A$ in SLDC($\mathcal{L}$), then $\underline{\Gamma} \vdash \underline{a} :^{\ell} \underline{A}$ in SDC($\mathcal{L}$). (Here, $\underline{\Gamma}$ denotes the context Γ with the types translated.)
By induction on $\Gamma \vdash a :^{\ell} A$. We present some of the interesting cases below.

- Rule SLDC-VarD. Have: $\top \sqcup \Gamma_1, x :^{\ell} A, \top \sqcup \Gamma_2 \vdash x :^{\ell} A$.
  Ned to show: $\top \sqcup \underline{\Gamma_1}, x :^{\ell} \underline{A}, \top \sqcup \underline{\Gamma_2} \vdash x :^{\ell} \underline{A}$.
  Follows by rule SDC-Var.

- Rule SLDC-LamD. Have: $\Gamma \vdash \lambda^{\ell_0} x : A.b :^{\ell} {}^{\ell_0}A \to B$ where $\Gamma, x :^{\ell \sqcup \ell_0} A \vdash b :^{\ell} B$.
  Need to show: $\underline{\Gamma} \vdash \lambda y : S_{\ell_0}\, \underline{A}.\mathbf{unlock}^{\ell_0}\, x = y\, \mathbf{in}\, \underline{b} :^{\ell} S_{\ell_0}\, \underline{A} \to \underline{B}$, where y is fresh.
  By IH, $\underline{\Gamma}, x :^{\ell \sqcup \ell_0} \underline{A} \vdash \underline{b} :^{\ell} \underline{B}$.
  By weakening, $\underline{\Gamma}, y :^{\ell} S_{\ell_0}\, \underline{A}, x :^{\ell \sqcup \ell_0} \underline{A} \vdash \underline{b} :^{\ell} \underline{B}$.
  Next, by rule SDC-Var, $\underline{\Gamma}, y :^{\ell} S_{\ell_0}\, \underline{A} \vdash y :^{\ell} S_{\ell_0}\, \underline{A}$.
  So by rule SDC-Unlock, $\underline{\Gamma}, y :^{\ell} S_{\ell_0}\, \underline{A} \vdash \mathbf{unlock}^{\ell_0}\, x = y\, \mathbf{in}\, \underline{b} :^{\ell} \underline{B}$.
  This case, then, follows by rule SDC-Lam.

- Rule SLDC-AppD. Have: $\Gamma_1 \sqcap \Gamma_2 \vdash b\, a^{\ell_0} :^{\ell} B$ where $\Gamma_1 \vdash b :^{\ell} {}^{\ell_0}A \to B$ and $\Gamma_2 \vdash a :^{\ell \sqcup \ell_0} A$.
  Need to show: $\underline{\Gamma_1} \sqcap \underline{\Gamma_2} \vdash \underline{b}\, (\mathbf{lock}^{\ell_0}\, \underline{a}) :^{\ell} \underline{B}$.
  By IH, $\underline{\Gamma_1} \vdash \underline{b} :^{\ell} S_{\ell_0}\, \underline{A} \to \underline{B}$ and $\underline{\Gamma_2} \vdash \underline{a} :^{\ell \sqcup \ell_0} \underline{A}$.
  Then, by the narrowing lemma, $\underline{\Gamma_1} \sqcap \underline{\Gamma_2} \vdash \underline{b} :^{\ell} S_{\ell_0}\, \underline{A} \to \underline{B}$ and $\underline{\Gamma_1} \sqcap \underline{\Gamma_2} \vdash \underline{a} :^{\ell \sqcup \ell_0} \underline{A}$.
  Next, by rule SDC-Lock, $\underline{\Gamma_1} \sqcap \underline{\Gamma_2} \vdash \mathbf{lock}^{\ell_0}\, \underline{a} :^{\ell} S_{\ell_0}\, \underline{A}$.
  This case, then, follows by rule SDC-App.

- Rule SLDC-PairD. Have: $\Gamma_1 \sqcap \Gamma_2 \vdash ({a_1}^{\ell_0}, a_2) :^{\ell} {}^{\ell_0}A_1 \times A_2$ where $\Gamma_1 \vdash a_1 :^{\ell \sqcup \ell_0} A_1$ and $\Gamma_2 \vdash a_2 :^{\ell} A_2$.
  Need to show: $\underline{\Gamma_1} \sqcap \underline{\Gamma_2} \vdash (\mathbf{lock}^{\ell_0}\, \underline{a_1}, \underline{a_2}) :^{\ell} S_{\ell_0}\, \underline{A_1} \times \underline{A_2}$.
  By IH, $\underline{\Gamma_1} \vdash \underline{a_1} :^{\ell \sqcup \ell_0} \underline{A_1}$ and $\underline{\Gamma_2} \vdash \underline{a_2} :^{\ell} \underline{A_2}$.
  Then, by the narrowing lemma, $\underline{\Gamma_1} \sqcap \underline{\Gamma_2} \vdash \underline{a_1} :^{\ell \sqcup \ell_0} \underline{A_1}$ and $\underline{\Gamma_1} \sqcap \underline{\Gamma_2} \vdash \underline{a_2} :^{\ell} \underline{A_2}$.
  Next, by rule SDC-Lock, $\underline{\Gamma_1} \sqcap \underline{\Gamma_2} \vdash \mathbf{lock}^{\ell_0}\, \underline{a_1} :^{\ell} S_{\ell_0}\, \underline{A_1}$.
  This case, then, follows by rule SDC-Pair.

- Rule SLDC-LetPairD. Have: $\Gamma_1 \sqcap \Gamma_2 \vdash \mathbf{let}\, (x^{\ell_0}, y) = a\, \mathbf{in}\, b :^{\ell} B$ where $\Gamma_1 \vdash a :^{\ell} {}^{\ell_0}A_1 \times A_2$ and $\Gamma_2, x :^{\ell \sqcup \ell_0} A_1, y :^{\ell} A_2 \vdash b :^{\ell} B$.
  Need to show: $\underline{\Gamma_1} \sqcap \underline{\Gamma_2} \vdash \mathbf{let}\, (x', y) = \underline{a}\, \mathbf{in}\, \mathbf{unlock}^{\ell_0}\, x = x'\, \mathbf{in}\, \underline{b} :^{\ell} \underline{B}$, where x' is fresh.
  By IH, $\underline{\Gamma_1} \vdash \underline{a} :^{\ell} S_{\ell_0}\, \underline{A_1} \times \underline{A_2}$.
  Then, by narrowing, $\underline{\Gamma_1} \sqcap \underline{\Gamma_2} \vdash \underline{a} :^{\ell} S_{\ell_0}\, \underline{A_1} \times \underline{A_2}$.
  Again, by IH, $\underline{\Gamma_2}, x :^{\ell \sqcup \ell_0} \underline{A_1}, y :^{\ell} \underline{A_2} \vdash \underline{b} :^{\ell} \underline{B}$.

By narrowing, $\underline{\Gamma_1} \sqcap \underline{\Gamma_2}, x :^{\ell \sqcup \ell_0} \underline{A_1}, y :^{\ell} \underline{A_2} \vdash \underline{b} :^{\ell} \underline{B}$.
Now, by exchange, $\underline{\Gamma_1} \sqcap \underline{\Gamma_2}, y :^{\ell} \underline{A_2}, x :^{\ell \sqcup \ell_0} \underline{A_1} \vdash \underline{b} :^{\ell} \underline{B}$.
And by weakening, $\underline{\Gamma_1} \sqcap \underline{\Gamma_2}, y :^{\ell} \underline{A_2}, x' :^{\ell} S_{\ell_0} \underline{A_1}, x :^{\ell \sqcup \ell_0} \underline{A_1} \vdash \underline{b} :^{\ell} \underline{B}$.
Next, by rule SDC-Var, $\underline{\Gamma_1} \sqcap \underline{\Gamma_2}, y :^{\ell} \underline{A_2}, x' :^{\ell} S_{\ell_0} \underline{A_1} \vdash x' :^{\ell} S_{\ell_0} \underline{A_1}$.
Then, by rule SDC-Unlock, $\underline{\Gamma_1} \sqcap \underline{\Gamma_2}, y :^{\ell} \underline{A_2}, x' :^{\ell} S_{\ell_0} \underline{A_1} \vdash \mathbf{unlock}^{\ell_0}\ x = x'\ \mathbf{in}\ \underline{b} :^{\ell} \underline{B}$.
By exchange, $\underline{\Gamma_1} \sqcap \underline{\Gamma_2}, x' :^{\ell} S_{\ell_0} \underline{A_1}, y :^{\ell} \underline{A_2} \vdash \mathbf{unlock}^{\ell_0}\ x = x'\ \mathbf{in}\ \underline{b} :^{\ell} \underline{B}$.
This case, then, follows by rule SDC-LetPair.

Finally, we show that the translation from SLDC to SDC preserves meaning: if $\vdash a \rightsquigarrow a'$ in SLDC($\mathcal{L}$), then $\underline{a} \rightsquigarrow^* \underline{a'}$ in SDC($\mathcal{L}$). (Note that we need the multi-step relation here because of the way the translation function is set up for the elimination forms for functions and pairs.)
By induction on $\vdash a \rightsquigarrow a'$. Follow the proof of a similar proposition in Lemma C.22. □

D.3 Proofs of Lemmas/Theorems Stated in Section 5.3

Lemma D.17 (Similarity) If $[H]a \Longrightarrow^q_S [H']a'$, then $a\{H\} = a'\{H'\}$ or $\vdash a\{H\} \rightsquigarrow a'\{H'\}$. Here, $a\{H\}$ denotes the term obtained by substituting in a the definitions in H, in reverse order.

Proof. Follow the proof of Lemma C.7. □

Lemma D.18 (Unchanged (Lemma 5.15)) If $[H_1, x \overset{r}{\mapsto} a, H_2]c \Longrightarrow^q_S [H_1', x \overset{r'}{\mapsto} a, H_2']c'$ (where $|H_1| = |H_1'|$) and $\neg(\exists q_0, r <: q_0 + q)$, then $r' = r$.

Proof. By induction on $[H_1, x \overset{r}{\mapsto} a, H_2]c \Longrightarrow^q_S [H_1', x \overset{r'}{\mapsto} a, H_2']c'$. There are two interesting cases:

- Rule HeapLD-Var. Have: $[H_1, x \overset{r}{\mapsto} a, H_2]c \Longrightarrow^q_S [H_1', x \overset{r'}{\mapsto} a, H_2']c'$, where c is a variable that is defined either in H_1 or in H_2. In either case, $r' = r$. Observe that c cannot be x because then this rule would not apply, owing to the condition, $\neg(\exists q_0, r <: q_0 + q)$.
- Rule HeapLD-SubR. Have: $[H_1, x \overset{r}{\mapsto} a, H_2]c \Longrightarrow^{q'}_S [H_1', x \overset{r'}{\mapsto} a, H_2']c'$ where $[H_1, x \overset{r}{\mapsto} a, H_2]c \Longrightarrow^q_S [H_1', x \overset{r'}{\mapsto} a, H_2']c'$ and $q <: q'$. Further, $\neg(\exists q_0, r <: q_0 + q')$.
Towards contradiction, suppose there exists q_1 such that $r <: q_1 + q$.
Since $q <: q'$, so $q_1 + q <: q_1 + q'$.
By transitivity, $r <: q_1 + q'$. A contradiction.
Therefore, $\neg(\exists q_0, r <: q_0 + q)$.
This case, then, follows by IH.

□

Lemma D.19 (Irrelevant (Lemma 5.16)) If $[H_1, x \overset{r}{\mapsto} a, H_2]c \Longrightarrow^{q}_{S \cup \mathrm{fv}\, b} [H_1', x \overset{r'}{\mapsto} a, H_2']c'$ (where $|H_1| = |H_1'|$) and $\neg(\exists q_0, r <: q_0 + q)$, then $[H_1, x \overset{r}{\mapsto} b, H_2]c \Longrightarrow^{q}_{S \cup \mathrm{fv}\, a} [H_1', x \overset{r'}{\mapsto} b, H_2']c'$.

Proof. By induction on $[H_1, x \overset{r}{\mapsto} a, H_2]c \Longrightarrow^{q}_{S \cup \mathrm{fv}\, b} [H_1', x \overset{r'}{\mapsto} a, H_2']c'$. Follow the proof of Lemma C.18. ☐

Lemma D.20 If $H \Vdash \Gamma_1 + \Gamma_2$ and $\Gamma_2 \vdash a :^{q} A$, then either a is a value or there exists H', Γ_2', a' such that:

- $[H]a \Longrightarrow^{q}_{S} [H']a'$
- $H' \Vdash \Gamma_1 + \Gamma_2'$
- $\Gamma_2' \vdash a' :^{q} A$

(Note that + is overloaded here: $\Gamma_1 + \Gamma_2$ denotes addition of contexts Γ_1 and Γ_2 after padding them as necessary.)

Proof. By induction on $\Gamma_2 \vdash a :^{q} A$.

- Rule SLDC-Var. Have $0 \cdot \Gamma_{21}, x :^{q_2} A, 0 \cdot \Gamma_{22} \vdash x :^{q_2} A$.
  Further, $H \Vdash (\Gamma_{11}, x :^{q_1} A, \Gamma_{12}) + (0 \cdot \Gamma_{21}, x :^{q_2} A, 0 \cdot \Gamma_{22})$.
  By inversion, $\exists H_1, q, a, H_2, \Gamma_0$ and q_0 such that $H = H_1, x \overset{q}{\mapsto} a, H_2$ and $\Gamma_0 \vdash a :^{q} A$ and $q = q_1 + q_2 + q_0$.
  By lemma D.4, $\exists \Gamma_{01}, \Gamma_{02}$ such that $\Gamma_{01} \vdash a :^{q_0 + q_1} A$ and $\Gamma_{02} \vdash a :^{q_2} A$ and $\Gamma_0 = \Gamma_{01} + \Gamma_{02}$.
  Then, we have,
  - $[H_1, x \overset{q_0+q_1+q_2}{\mapsto} a, H_2]x \Longrightarrow^{q_2}_{S} [H_1, x \overset{q_0+q_1}{\mapsto} a, H_2]a$.
  - $H_1, x \overset{q_0+q_1}{\mapsto} a, H_2 \Vdash (\Gamma_{11}, x :^{q_1} A, \Gamma_{12}) + (\Gamma_{02}, x :^{0} A, 0 \cdot \Gamma_{22})$.
  - $\Gamma_{02}, x :^{0} A, 0 \cdot \Gamma_{22} \vdash a :^{q_2} A$.
- Rule SLDC-App. Have: $\Gamma_{21} + \Gamma_{22} \vdash b\ a^{r} :^{q} B$ where $\Gamma_{21} \vdash b :^{q} {}^{r}A \to B$ and $\Gamma_{22} \vdash a :^{q \cdot r} A$.
  Further, $H \Vdash \Gamma_1 + (\Gamma_{21} + \Gamma_{22})$.
  By IH, if b steps, then $[H]b \Longrightarrow^{q}_{S \cup \mathrm{fv}\, a} [H']b'$ and $H' \Vdash (\Gamma_1 + \Gamma_{22}) + \Gamma_{21}'$ and $\Gamma_{21}' \vdash b' :^{q} {}^{r}A \to B$.
  Then, $[H]b\ a^{r} \Longrightarrow^{q}_{S} [H']b'\ a^{r}$ and $H' \Vdash \Gamma_1 + (\Gamma_{21}' + \Gamma_{22})$ and $\Gamma_{21}' + \Gamma_{22} \vdash b'\ a^{r} :^{q} B$.

  Otherwise, b is a value. By inversion $b = \lambda^{r} x : A.b'$.
  Also, by inversion, $\Gamma_{21}, x :^{q \cdot r} A \vdash b' :^{q} B$.
  (Note that by inverting $\Gamma_{21} \vdash \lambda^{r} x : A.b' :^{q} {}^{r}A \to B$, we get, $\Gamma_{21}, x :^{q_0 \cdot r} A \vdash b' :^{q_0} B$, for some $q_0 <: q$. But from this judgment, we can derive $\Gamma_{21}, x :^{q \cdot r} A \vdash b' :^{q} B$, as shown in the rule SLDC-App case of Lemma D.7, and similarly in the rule SLDC-AppD case of Lemma D.14.)
  Then, $[H](\lambda^{r} x : A.b')\ a \Longrightarrow^{q}_{S} [H, x \overset{q \cdot r}{\mapsto} a]b'$ (assuming x *fresh*).
  Further, $H, x \overset{q \cdot r}{\mapsto} a \Vdash (\Gamma_1 + \Gamma_{21}), x :^{q \cdot r} A$.

- Rule SLDC-LetUnit. Have: $\Gamma_{21} + \Gamma_{22} \vdash \textbf{let}_{q_0}\ \textbf{unit} = a\ \textbf{in}\ b :^q B$ where $\Gamma_{21} \vdash a :^{q \cdot q_0} \textbf{Unit}$ and $\Gamma_{22} \vdash b :^q B$.
  Further, $H \Vdash \Gamma_1 + (\Gamma_{21} + \Gamma_{22})$.
  By IH, if a steps, then $[H]a \Longrightarrow^{q \cdot q_0}_{S \cup \text{fv}\ b} [H']a'$ and $H' \Vdash \Gamma_1 + (\Gamma'_{21} + \Gamma_{22})$ and $\Gamma'_{21} \vdash a' :^{q \cdot q_0} \textbf{Unit}$.
  Then, $[H]\textbf{let}_{q_0}\ \textbf{unit} = a\ \textbf{in}\ b \Longrightarrow^{q \cdot q_0}_{S} [H']\textbf{let}_{q_0}\ \textbf{unit} = a'\ \textbf{in}\ b$.
  By rule HeapLD-SubR, $[H]\textbf{let}_{q_0}\ \textbf{unit} = a\ \textbf{in}\ b \Longrightarrow^{q}_{S} [H']\textbf{let}_{q_0}\ \textbf{unit} = a'\ \textbf{in}\ b$. $(\because q_0 <: 1)$
  And by rule SLDC-LetUnit, $\Gamma'_{21} + \Gamma_{22} \vdash \textbf{let}_{q_0}\ \textbf{unit} = a'\ \textbf{in}\ b :^q B$.

  Otherwise, a is a value. By inversion, $a = \textbf{unit}$.
  Then, $[H]\textbf{let}_{q_0}\ \textbf{unit} = \textbf{unit}\ \textbf{in}\ b \Longrightarrow^{q}_{S} [H]b$.
  Further, $\Gamma_{21} + \Gamma_{22} \vdash b :^q B$. ($\because$ In case of $\mathbb{N}_=$, $\overline{\Gamma_{21}} = \mathbf{0}$ and in case of $\mathbb{N}_\geq$, $\Gamma_{21} + \Gamma_{22} <: \Gamma_{22}$.)

- Rule SLDC-LetPair. Have: $\Gamma_{21} + \Gamma_{22} \vdash \textbf{let}_{q_0}\ (x^r, y) = a\ \textbf{in}\ b :^q B$ where $\Gamma_{21} \vdash a :^{q \cdot q_0} {}^r A_1 \times A_2$ and $\Gamma_{22}, x :^{q \cdot q_0 \cdot r} A_1, y :^{q \cdot q_0} A_2 \vdash b :^q B$.
  Further, $H \Vdash \Gamma_1 + (\Gamma_{21} + \Gamma_{22})$.
  By IH, if a steps, then $[H]a \Longrightarrow^{q \cdot q_0}_{S \cup \text{fv}\ b} [H']a'$ and $H' \Vdash \Gamma_1 + (\Gamma'_{21} + \Gamma_{22})$ and $\Gamma'_{21} \vdash a' :^{q \cdot q_0} {}^r A_1 \times A_2$.
  Then, $[H]\textbf{let}_{q_0}\ (x^r, y) = a\ \textbf{in}\ b \Longrightarrow^{q \cdot q_0}_{S} [H']\textbf{let}_{q_0}\ (x^r, y) = a'\ \textbf{in}\ b$.
  By rule HeapLD-SubR, $[H]\textbf{let}_{q_0}\ (x^r, y) = a\ \textbf{in}\ b \Longrightarrow^{q}_{S} [H']\textbf{let}_{q_0}\ (x^r, y) = a'\ \textbf{in}\ b$. $(\because q_0 <: 1)$
  And by rule SLDC-LetPair, $\Gamma'_{21} + \Gamma_{22} \vdash \textbf{let}_{q_0}\ (x^r, y) = a'\ \textbf{in}\ b :^q B$.

  Otherwise, a is a value. By inversion, $a = ({a_1}^r, a_2)$.
  Further, $\exists \Gamma_{211}, \Gamma_{212}$ such that $\Gamma_{211} \vdash a_1 :^{q \cdot q_0 \cdot r} A_1$ and $\Gamma_{212} \vdash a_2 :^{q \cdot q_0} A_2$ and $\Gamma_{21} = \Gamma_{211} + \Gamma_{212}$.
  Now, $[H]\textbf{let}_{q_0}\ (x^r, y) = ({a_1}^r, a_2)\ \textbf{in}\ b \Longrightarrow^{q}_{S} [H, x \overset{q \cdot q_0 \cdot r}{\mapsto} a_1, y \overset{q \cdot q_0}{\mapsto} a_2]b$ (assuming x, y *fresh*).
  And, $H, x \overset{q \cdot q_0 \cdot r}{\mapsto} a_1, y \overset{q \cdot q_0}{\mapsto} a_2 \Vdash (\Gamma_1 + \Gamma_{22}), x :^{q \cdot q_0 \cdot r} A_1, y :^{q \cdot q_0} A_2$.

- Rule SLDC-Case. Have: $\Gamma_{21} + \Gamma_{22} \vdash \textbf{case}_{q_0}\ a\ \textbf{of}\ x_1.b_1\ ;\ x_2.b_2 :^q B$ where $\Gamma_{21} \vdash a :^{q \cdot q_0} A_1 + A_2$ and $\Gamma_{22}, x_1 :^{q \cdot q_0} A_1 \vdash b_1 :^q B$ and $\Gamma_{22}, x_2 :^{q \cdot q_0} A_2 \vdash b_2 :^q B$.
  Further, $H \Vdash \Gamma_1 + (\Gamma_{21} + \Gamma_{22})$.
  By IH, if a steps, then $[H]a \Longrightarrow^{q \cdot q_0}_{S \cup \text{fv}\ b_1 \cup \text{fv}\ b_2} [H']a'$ and $H' \Vdash \Gamma_1 + (\Gamma'_{21} + \Gamma_{22})$ and $\Gamma'_{21} \vdash a' :^{q \cdot q_0} A_1 + A_2$.
  Then, $[H]\textbf{case}_{q_0}\ a\ \textbf{of}\ x_1.b_1\ ;\ x_2.b_2 \Longrightarrow^{q \cdot q_0}_{S} [H']\textbf{case}_{q_0}\ a'\ \textbf{of}\ x_1.b_1\ ;\ x_2.b_2$.
  By rule HeapLD-SubR, $[H]\textbf{case}_{q_0}\ a\ \textbf{of}\ x_1.b_1\ ;\ x_2.b_2 \Longrightarrow^{q}_{S} [H']\textbf{case}_{q_0}\ a'\ \textbf{of}\ x_1.b_1\ ;\ x_2.b_2$. $(\because q_0 <: 1)$
  And by rule SLDC-Case, $\Gamma'_{21} + \Gamma_{22} \vdash \textbf{case}_{q_0}\ a'\ \textbf{of}\ x_1.b_1\ ;\ x_2.b_2 :^q B$.

  Otherwise, a is a value. By inversion, $a = \textbf{inj}_1\ a_1$ or $a = \textbf{inj}_2\ a_2$.
  Say $a = \textbf{inj}_1\ a_1$. Then, $\Gamma_{21} \vdash a_1 :^{q \cdot q_0} A_1$.
  Now, $[H]\textbf{case}_{q_0}\ (\textbf{inj}_1\ a_1)\ \textbf{of}\ x_1.b_1\ ;\ x_2.b_2 \Longrightarrow^{q}_{S} [H, x_1 \overset{q \cdot q_0}{\mapsto} a_1]b_1$ (assuming x_1 *fresh*).
  And, $H, x_1 \overset{q \cdot q_0}{\mapsto} a_1 \Vdash (\Gamma_1 + \Gamma_{22}), x_1 :^{q \cdot q_0} A_1$.
  The case when $a = \textbf{inj}_2\ a_2$ is similar.

- Rule SLDC-SubL. Have: $\Gamma_2 \vdash a :^q A$ where $\Gamma'_2 \vdash a :^q A$ and $\Gamma_2 <: \Gamma'_2$.
  Further, $H \Vdash \Gamma_1 + \Gamma_2$.

Since $\Gamma_2 <: \Gamma_2'$, there exists Γ_0 such that $\Gamma_2 = \Gamma_2' + \Gamma_0$. (In case of $\mathbb{N}_=, \overline{\Gamma_0} = \mathbf{0}$.)
By IH, if a is not value, then $[H]a \Longrightarrow_S^q [H']a'$ and $H' \Vdash (\Gamma_1 + \Gamma_0) + \Gamma_2''$ and $\Gamma_2'' \vdash a' :^q A$.
Then, by rule SLDC-SubL, $\Gamma_0 + \Gamma_2'' \vdash a' :^q A$, since $\Gamma_0 + \Gamma_2'' <: \Gamma_2''$.

- Rule SLDC-SubR. Have $\Gamma_2 \vdash a :^{q'} A$ where $\Gamma_2 \vdash a :^q A$ and $q <: q'$.
  Further, $H \Vdash \Gamma_1 + \Gamma_2$.
  By IH, if a is not a value, then $[H]a \Longrightarrow_S^q [H']a'$ and $H' \Vdash \Gamma_1 + \Gamma_2'$ and $\Gamma_2' \vdash a' :^q A$.
  Then, since $q <: q'$, by rule HeapLD-SubR, $[H]a \Longrightarrow_S^{q'} [H']a'$.
  And by rule SLDC-SubR, $\Gamma_2' \vdash a' :^{q'} A$.

□

Theorem D.21 (Soundness (Theorem 5.17)) If $H \Vdash \Gamma$ and $\Gamma \vdash a :^q A$, then either a is a value or there exists H', Γ', a' such that:

- $[H]a \Longrightarrow_S^q [H']a'$
- $H' \Vdash \Gamma'$
- $\Gamma' \vdash a' :^q A$

Proof. Use lemma D.20 with $\Gamma_1 := 0 \cdot \Gamma$ and $\Gamma_2 := \Gamma$. □

Lemma D.22 If $H \Vdash \Gamma_1 \sqcap \Gamma_2$ and $\Gamma_2 \vdash a :^\ell A$, then either a is a value or there exists H', Γ_2', a' such that:

- $[H]a \Longrightarrow_S^\ell [H']a'$
- $H' \Vdash \Gamma_1 \sqcap \Gamma_2'$
- $\Gamma_2' \vdash a' :^\ell A$

Proof. By induction on $\Gamma_2 \vdash a :^\ell A$. Same as the proof of lemma D.20 when $+, \cdot, 0, 1$ and $<:$ are replaced with $\sqcap, \sqcup, \top, \bot$ and $\sqsubseteq$ respectively. □

Theorem D.23 (Soundness (Theorem 5.17)) If $H \Vdash \Gamma$ and $\Gamma \vdash a :^\ell A$, then either a is a value or there exists H', Γ', a' such that:

- $[H]a \Longrightarrow_S^\ell [H']a'$
- $H' \Vdash \Gamma'$
- $\Gamma' \vdash a' :^\ell A$

Proof. Use lemma D.22 with $\Gamma_1 := \top \sqcup \Gamma$ and $\Gamma_2 := \Gamma$. □

Corollary D.24 (No Usage (Corollary 5.18)) In SLDC($\mathcal{Q}_{\mathbb{N}}$): Let $\varnothing \vdash f :^1 {}^0A \to \textbf{Bool}$. Then, for any $\varnothing \vdash a_1 :^0 A$ and $\varnothing \vdash a_2 :^0 A$, the terms $f\ {a_1}^0$ and $f\ {a_2}^0$ have the same operational behavior, i.e., either both the terms diverge or both reduce to the same value.

Proof. To see why, we consider the reductions of $f\ {a_1}^0$ and $f\ {a_2}^0$.

Let $[H]\, a \Rrightarrow^q_j [H']\, a'$ denote a reduction of j steps where H and a are the initial heap and term, H' and a' are the final heap and term, and q is the label at which the reduction takes place.

For some j, H and b, we have, $[\varnothing] f\ {a_1}^0 \Rrightarrow^1_j [H]\, (\lambda^0 x.b)\ {a_1}^0$ and $[\varnothing] f\ {a_2}^0 \Rrightarrow^1_j [H]\, (\lambda^0 x.b)\ {a_2}^0$.

Then, $[H](\lambda^0 x.b)\ {a_1}^0 \Longrightarrow^1_S [H, x \overset{0}{\mapsto} a_1] b$ and $[H](\lambda^0 x.b)\ {a_2}^0 \Longrightarrow^1_S [H, x \overset{0}{\mapsto} a_2] b$ (for x *fresh*).

But, if $[H, x \overset{0}{\mapsto} a_1]\, b \Rrightarrow^1_k [H', x \overset{0}{\mapsto} a_1, H'']\, b'$, then $[H, x \overset{0}{\mapsto} a_2]\, b \Rrightarrow^1_k [H', x \overset{0}{\mapsto} a_2, H'']\, b'$, for any k (By Lemma D.19).

Therefore, $f\ {a_1}^0$ and $f\ {a_2}^0$ have the same operational behavior.

□

Corollary D.25 (Noninterference (Corollary 5.19)) In SLDC($\mathcal{L}$): Let $\varnothing \vdash f :^\ell {}^mA \to \textbf{Bool}$ such that $\neg(m \sqsubseteq \ell)$. Then, for any $\varnothing \vdash a_1 :^m A$ and $\varnothing \vdash a_2 :^m A$, the terms $f\ {a_1}^m$ and $f\ {a_2}^m$ have the same operational behavior.

Proof. Same as the proof of Corollary D.24 with 0 and 1 replaced by m and ℓ respectively. □

Corollary D.26 (Affine Usage (Corollary 5.20)) In SLDC($\mathbb{N}_\geq$): Let $\varnothing \vdash f :^1 {}^1A \to \textbf{Bool}$. Then, for any $\varnothing \vdash a :^1 A$, the term $f\ a^1$ uses a at most once during reduction.

Proof. To see why, consider the reduction of $f\ a^1$.

For some j, H and b, we have, $[\varnothing] f\ a^1 \Rrightarrow^1_j [H]\, (\lambda^1 x.b)\ a^1$.

Then, $[H](\lambda^1 x.b)\ a^1 \Longrightarrow^1_S [H, x \overset{1}{\mapsto} a] b$ (for x *fresh*).

Now, b may reduce to a value without ever looking up x or b may look up the value of x exactly once. A single use of x will change the allowed usage from 1 to 0, making it essentially unusable thereafter. Hence, a cannot be used more than once. □

D.4 Proofs of Lemmas/Theorems/Corollaries Stated in Section 5.4

Lemma D.27 (Multiplication (Lemma 5.21)) If $\Gamma \vdash a :^q A$, then $r_0 \cdot \Gamma \vdash a :^{r_0 \cdot q} A$.

Proof. By induction on $\Gamma \vdash a :^q A$. Follow the proof of lemma D.2. □

Lemma D.28 (Factorization (Lemma 5.22)) If $\Gamma \vdash a :^q A$ and $q \neq 0$, then there exists Γ' such that $\Gamma' \vdash a :^1 A$ and $\Gamma <: q \cdot \Gamma'$.

Proof. By induction on $\Gamma \vdash a :^q A$. Follow the proof of lemma D.3. □

Lemma D.29 (Splitting (Lemma 5.23)) If $\Gamma \vdash a :^{q_1+q_2} A$, then there exists Γ_1 and Γ_2 such that $\Gamma_1 \vdash a :^{q_1} A$ and $\Gamma_2 \vdash a :^{q_2} A$ and $\Gamma = \Gamma_1 + \Gamma_2$.

Proof. If $q_1 + q_2 = 0$, then set $\Gamma_1 \coloneqq 0 \cdot \Gamma$ and $\Gamma_2 \coloneqq \Gamma$.
Otherwise, by Lemma D.28, $\exists \Gamma'$ such that $\Gamma' \vdash a :^1 A$ and $\Gamma <: (q_1 + q_2) \cdot \Gamma'$.
Then, $\Gamma = (q_1 + q_2) \cdot \Gamma' + \Gamma_0$, for some Γ_0 (in case of $\mathbb{N}_=$, $\overline{\Gamma_0} = \mathbf{0}$).
Now, by Lemma D.27, $q_1 \cdot \Gamma' \vdash a :^{q_1} A$ and $q_2 \cdot \Gamma' \vdash a :^{q_2} A$.
By rule LDC-SubL, $q_2 \cdot \Gamma' + \Gamma_0 \vdash a :^{q_2} A$.
The lemma follows by setting $\Gamma_1 \coloneqq q_1 \cdot \Gamma'$ and $\Gamma_2 \coloneqq q_2 \cdot \Gamma' + \Gamma_0$. □

Lemma D.30 (Weakening (Lemma 5.24)) If $\Gamma_1, \Gamma_2 \vdash a :^q A$ and $\lfloor \Gamma_1 \rfloor \vdash_0 C : s$, then $\Gamma_1, z :^0 C, \Gamma_2 \vdash a :^q A$.

Proof. By induction on $\Gamma_1, \Gamma_2 \vdash a :^q A$. □

Lemma D.31 (Substitution (Lemma 5.25)) If $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash a :^q A$ and $\Gamma \vdash c :^{r_0} C$ where $\lfloor \Gamma_1 \rfloor = \lfloor \Gamma \rfloor$, then $\Gamma_1 + \Gamma, \Gamma_2\{c/z\} \vdash a\{c/z\} :^q A\{c/z\}$.

Proof. By induction on $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash a :^q A$.

- Rule LDC-Var. There are two cases to consider.
  - Have: $0 \cdot \Gamma_1, x :^q A \vdash x :^q A$ where $\lfloor \Gamma_1 \rfloor \vdash_0 A : s$. Further, $\Gamma \vdash a :^q A$ where $\lfloor \Gamma_1 \rfloor = \lfloor \Gamma \rfloor$.
    Need to show: $\Gamma \vdash a :^q A\{a/x\}$.
    Follows from what's given since $A\{a/x\} = A$.
  - Have: $0 \cdot \Gamma_{11}, z :^0 C, 0 \cdot \Gamma_{12}, x :^q A \vdash x :^q A$ where $\lfloor \Gamma_{11} \rfloor, z : C, \lfloor \Gamma_{12} \rfloor \vdash_0 A : s$. Further, $\Gamma \vdash c :^0 C$ where $\lfloor \Gamma_{11} \rfloor = \lfloor \Gamma \rfloor$.
    Need to show: $\Gamma, 0 \cdot \Gamma_{12}\{c/z\}, x :^q A\{c/z\} \vdash x :^q A\{c/z\}$.
    By IH, $\lfloor \Gamma_{11} \rfloor, \lfloor \Gamma_{12} \rfloor\{c/z\} \vdash_0 A\{c/z\} : s$.
    This case, then, follows by rules LDC-Var and LDC-SubL (note that in case of $\mathbb{N}_=$, $\overline{\Gamma} = \mathbf{0}$).
- Rule LDC-Weak. There are two cases to consider.
  - Have: $\Gamma_1, y :^0 B \vdash a :^q A$ where $\Gamma_1 \vdash a :^q A$ and $\lfloor \Gamma_1 \rfloor \vdash_0 B : s$. Further, $\Gamma \vdash b :^0 B$ where $\lfloor \Gamma_1 \rfloor = \lfloor \Gamma \rfloor$.
    Need to show: $\Gamma_1 + \Gamma \vdash a\{b/y\} :^q A\{b/y\}$.
    Since $y \notin \text{fv } a$ and $y \notin \text{fv } A$, we need to show: $\Gamma_1 + \Gamma \vdash a :^q A$.
    This case follows by rule LDC-SubL (note that in case of $\mathbb{N}_=$, $\overline{\Gamma} = \mathbf{0}$).

– Have: $\Gamma_{11}, z :^{r_0} C, \Gamma_{12}, y :^0 B \vdash a :^q A$ where $\Gamma_{11}, z :^{r_0} C, \Gamma_{12} \vdash a :^q A$ and $\lfloor \Gamma_{11} \rfloor, z : C, \lfloor \Gamma_{12} \rfloor \vdash_0 B : s$. Further, $\Gamma \vdash c :^{r_0} C$ where $\lfloor \Gamma_{11} \rfloor = \lfloor \Gamma \rfloor$.
Need to show: $\Gamma_{11} + \Gamma, \Gamma_{12}\{c/z\}, y :^0 B\{c/z\} \vdash a\{c/z\} :^q A\{c/z\}$.
By IH, $\Gamma_{11} + \Gamma, \Gamma_{12}\{c/z\} \vdash a\{c/z\} :^q A\{c/z\}$.
Next, by multiplication, $0 \cdot \Gamma \vdash c :^0 C$.
Again by IH, $\lfloor \Gamma_{11} \rfloor, \lfloor \Gamma_{12} \rfloor\{c/z\} \vdash_0 B\{c/z\} : s$.
This case, then, follows by rule LDC-Weak.

- Rule LDC-Pi. Have: $\Gamma_{11} + \Gamma_{12}, z :^{r_{01}+r_{02}} C, \Gamma_{21} + \Gamma_{22} \vdash \Pi x :^r A.B :^q s_3$ where $\Gamma_{11}, z :^{r_{01}} C, \Gamma_{21} \vdash A :^q s_1$ and $\Gamma_{12}, z :^{r_{02}} C, \Gamma_{22}, x :^{q_0} A \vdash B :^q s_2$ and $\mathcal{R}(s_1, s_2, s_3)$.
Further, $\Gamma \vdash c :^{r_{01}+r_{02}} C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_{11} \rfloor$.
Need to show: $\Gamma_{11} + \Gamma_{12} + \Gamma, \Gamma_{21}\{c/z\} + \Gamma_{22}\{c/z\} \vdash \Pi x :^r A\{c/z\}.B\{c/z\} :^q s_3$.
By lemma D.29, $\exists \Gamma_{31}, \Gamma_{32}$ such that $\Gamma_{31} \vdash c :^{r_{01}} C$ and $\Gamma_{32} \vdash c :^{r_{02}} C$ and $\Gamma_{31} + \Gamma_{32} = \Gamma$.
By IH, $\Gamma_{11} + \Gamma_{31}, \Gamma_{21}\{c/z\} \vdash A\{c/z\} :^q s_1$ and $\Gamma_{12} + \Gamma_{32}, \Gamma_{22}\{c/z\}, x :^{q_0} A\{c/z\} \vdash B\{c/z\} :^q s_2$.
This case, then, follows by rule LDC-Pi.

- Rule LDC-Lam. Have: $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash \lambda^r x : A.b :^q \Pi x :^r A.B$ where $\Gamma_1, z :^{r_0} C, \Gamma_2, x :^{q \cdot r} A \vdash b :^q B$.
Further, $\Gamma \vdash c :^{r_0} C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
Need to show: $\Gamma_1 + \Gamma, \Gamma_2\{c/z\} \vdash \lambda^r x : A\{c/z\}.b\{c/z\} :^q \Pi x :^r A\{c/z\}.B\{c/z\}$.
By IH, $\Gamma_1 + \Gamma, \Gamma_2\{c/z\}, x :^{q \cdot r} A\{c/z\} \vdash b\{c/z\} :^q B\{c/z\}$.
This case, then, follows by rule LDC-Lam.

- Rule LDC-App. Have: $\Gamma_{11} + \Gamma_{12}, z :^{r_{01}+r_{02}} C, \Gamma_{21} + \Gamma_{22} \vdash b\ a^r :^q B\{a/x\}$ where $\Gamma_{11}, z :^{r_{01}} C, \Gamma_{21} \vdash b :^q \Pi x :^r A.B$ and $\Gamma_{12}, z :^{r_{02}} C, \Gamma_{22} \vdash a :^{q \cdot r} A$.
Further, $\Gamma \vdash c :^{r_{01}+r_{02}} C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_{11} \rfloor$.
Need to show: $\Gamma_{11} + \Gamma_{12} + \Gamma, \Gamma_{21}\{c/z\} + \Gamma_{22}\{c/z\} \vdash b\{c/z\}\ a\{c/z\}^r :^q B\{c/z\}\{a\{c/z\}/x\}$.
By lemma D.29, $\exists \Gamma_{31}, \Gamma_{32}$ such that $\Gamma_{31} \vdash c :^{r_{01}} C$ and $\Gamma_{32} \vdash c :^{r_{02}} C$ and $\Gamma_{31} + \Gamma_{32} = \Gamma$.
By IH, $\Gamma_{11} + \Gamma_{31}, \Gamma_{21}\{c/z\} \vdash b\{c/z\} :^q \Pi x :^r A\{c/z\}.B\{c/z\}$
and $\Gamma_{12} + \Gamma_{32}, \Gamma_{22}\{c/z\} \vdash a\{c/z\} :^{q \cdot r} A\{c/z\}$.
This case, then, follows by rule LDC-App.

- Rule LDC-Conv. Have: $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash a :^q B$ where $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash a :^q A$ and $A =_\beta B$.
Further, $\Gamma \vdash c :^{r_0} C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
Need to show: $\Gamma_1 + \Gamma, \Gamma_2\{c/z\} \vdash a\{c/z\} :^q B\{c/z\}$.
By IH, $\Gamma_1 + \Gamma, \Gamma_2\{c/z\} \vdash a\{c/z\} :^q A\{c/z\}$.
Next, since $A =_\beta B$, so $A\{c/z\} =_\beta B\{c/z\}$.
This case, then, follows by rule LDC-Conv.

- Rule LDC-Pair. Have: $\Gamma_{11} + \Gamma_{12}, z :^{r_{01}+r_{02}} C, \Gamma_{21} + \Gamma_{22} \vdash (a_1{}^r, a_2) :^q \Sigma x :^r A_1.A_2$ where $\Gamma_{11}, z :^{r_{01}} C, \Gamma_{21} \vdash a_1 :^{q \cdot r} A_1$ and $\Gamma_{12}, z :^{r_{02}} C, \Gamma_{22} \vdash a_2 :^q A_2\{a_1/x\}$.
Further, $\Gamma \vdash c :^{r_{01}+r_{02}} C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_{11} \rfloor$.
Need to show: $\Gamma_{11} + \Gamma_{12} + \Gamma, \Gamma_{21}\{c/z\} + \Gamma_{22}\{c/z\} \vdash (a_1\{c/z\}^r, a_2\{c/z\}) :^q \Sigma x :^r A_1\{c/z\}.A_2\{c/z\}$.

By lemma D.29, $\exists \Gamma_{31}, \Gamma_{32}$ such that $\Gamma_{31} \vdash c :^{r_{01}} C$ and $\Gamma_{32} \vdash c :^{r_{02}} C$ and $\Gamma = \Gamma_{31} + \Gamma_{32}$.
By IH, $\Gamma_{11} + \Gamma_{31}, \Gamma_{21}\{c/z\} \vdash a_1\{c/z\} :^{q \cdot r} A_1\{c/z\}$
and $\Gamma_{12} + \Gamma_{32}, \Gamma_{22}\{c/z\} \vdash a_2\{c/z\} :^{q} A_2\{a_1/x\}\{c/z\}$.
Now, $A_2\{a_1/x\}\{c/z\} = A_2\{c/z\}\{a_1\{c/z\}/x\}$.
This case, then, follows by rule LDC-Pair.

- Rule LDC-LetPair. Have: $\Gamma_{11} + \Gamma_{12}, z :^{r_{01}+r_{02}} C, \Gamma_{21} + \Gamma_{22} \vdash \mathbf{let}_{q_0}\ (x^r, y) = a \mathbf{\ in\ } b :^q B\{a/w\}$ where $\Delta, w : \Sigma x :^r A_1.A_2 \vdash_0 B : s$ and $\Gamma_{11}, z :^{r_{01}} C, \Gamma_{21} \vdash a :^{q \cdot q_0} \Sigma x :^r A_1.A_2$ and $\Gamma_{12}, z :^{r_{02}} C, \Gamma_{22}, x :^{q \cdot q_0 \cdot r} A_1, y :^{q \cdot q_0} A_2 \vdash b :^q B\{(x^r, y)/w\}$ such that $\Delta = \lfloor \Gamma_{11} \rfloor, z : C, \lfloor \Gamma_{21} \rfloor$.
  Further, $\Gamma \vdash c :^{r_{01}+r_{02}} C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_{11} \rfloor$.
  Need to show: $\Gamma_{11}+\Gamma_{12}+\Gamma, \Gamma_{21}\{c/z\}+\Gamma_{22}\{c/z\} \vdash \mathbf{let}_{q_0}\,(x^r, y) = a\{c/z\}\mathbf{in}\, b\{c/z\} :^q B\{c/z\}\{a\{c/z\}/w\}$.
  By lemma D.29, $\exists \Gamma_{31}, \Gamma_{32}$ such that $\Gamma_{31} \vdash c :^{r_{01}} C$ and $\Gamma_{32} \vdash c :^{r_{02}} C$ and $\Gamma = \Gamma_{31} + \Gamma_{32}$.
  By IH, $\Gamma_{11} + \Gamma_{31}, \Gamma_{21}\{c/z\} \vdash a\{c/z\} :^{q \cdot q_0} \Sigma x :^r A_1\{c/z\}.A_2\{c/z\}$ and
  $\Gamma_{12} + \Gamma_{32}, \Gamma_{22}\{c/z\}, x :^{q \cdot q_0 \cdot r} A_1\{c/z\}, y :^{q \cdot q_0} A_2\{c/z\} \vdash b\{c/z\} :^q B\{(x^r, y)/w\}\{c/z\}$.
  This case, then, follows by rule LDC-LetPair.

- Rules LDC-Sum, LDC-Inj1, and LDC-Inj2. By IH.

- Rule LDC-Case. Have: $\Gamma_{11} + \Gamma_{12}, z :^{r_{01}+r_{02}} C, \Gamma_{21} + \Gamma_{22} \vdash \mathbf{case}_{q_0}\ a \mathbf{\ of\ } x_1.b_1\ ;\ x_2.b_2 :^q B\{a/w\}$ where $\Delta, w : A_1 + A_2 \vdash_0 B : s$ and $\Gamma_{11}, z :^{r_{01}} C, \Gamma_{21} \vdash a :^{q \cdot q_0} A_1 + A_2$ and $\Gamma_{12}, z :^{r_{02}} C, \Gamma_{22}, x_1 :^{q \cdot q_0} A_1 \vdash b_1 :^q B\{\mathbf{inj}_1\ x_1/w\}$ and $\Gamma_{12}, z :^{r_{02}} C, \Gamma_{22}, x_2 :^{q \cdot q_0} A_2 \vdash b_2 :^q B\{\mathbf{inj}_2\ x_2/w\}$ such that $\Delta = \lfloor \Gamma_{11} \rfloor, z : C, \lfloor \Gamma_{21} \rfloor$.
  Further, $\Gamma \vdash c :^{r_{01}+r_{02}} C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_{11} \rfloor$.
  Need to show:
  $\Gamma_{11} + \Gamma_{12} + \Gamma, \Gamma_{21}\{c/z\} + \Gamma_{22}\{c/z\} \vdash \mathbf{case}_{q_0}\ a\{c/z\} \mathbf{\ of\ } x_1.b_1\{c/z\}\ ;\ x_2.b_2\{c/z\} :^q B\{c/z\}\{a\{c/z\}/w\}$.
  By lemma D.29, $\exists \Gamma_{31}, \Gamma_{32}$ such that $\Gamma_{31} \vdash c :^{r_{01}} C$ and $\Gamma_{32} \vdash c :^{r_{02}} C$ and $\Gamma = \Gamma_{31} + \Gamma_{32}$.
  By IH, $\Gamma_{11} + \Gamma_{31}, \Gamma_{21}\{c/z\} \vdash a\{c/z\} :^{q \cdot q_0} A_1\{c/z\} + A_2\{c/z\}$ and
  $\Gamma_{12} + \Gamma_{32}, \Gamma_{22}\{c/z\}, x_1 :^{q \cdot q_0} A_1\{c/z\} \vdash b_1\{c/z\} :^q B\{\mathbf{inj}_1\ x_1/w\}\{c/z\}$ and
  $\Gamma_{12} + \Gamma_{32}, \Gamma_{22}\{c/z\}, x_2 :^{q \cdot q_0} A_2\{c/z\} \vdash b_2\{c/z\} :^q B\{\mathbf{inj}_2\ x_2/w\}\{c/z\}$.
  This case, then, follows by rule LDC-Case.

- Rules LDC-SubL and LDC-SubR. By IH.

□

Lemma D.32 (Equality inversion) The definitional equality relation of LDC satisfies the following inversion lemmas:

- If $\Pi x :^{r_1} A_1.B_1 =_\beta \Pi x :^{r_2} A_2.B_2$, then $r_1 = r_2$ and $A_1 =_\beta A_2$. Further, if $a_1 =_\beta a_2$, then $B_1\{a_1/x\} =_\beta B_2\{a_2/x\}$.

- If $\Sigma x :^{r_1} A_1.B_1 =_\beta \Sigma x :^{r_2} A_2.B_2$, then $r_1 = r_2$ and $A_1 =_\beta A_2$. Further, if $a_1 =_\beta a_2$, then $B_1\{a_1/x\} =_\beta B_2\{a_2/x\}$.

Proof. Same as Lemma C.25. □

Lemma D.33 (Typing inversion) LDC satisfies the following inversion lemmas:

- If $\Gamma \vdash v :^q A$ and $A =_\beta \Pi x :^r B_1.B_2$ and v is a value, then exists A_1, A_2 and a_2 such that
  - $v = \lambda^r x : A_1.a_2$
  - $\Gamma, x :^{q_0 \cdot r} A_1 \vdash a_2 :^{q_0} A_2$ where $q_0 <: q$
  - $\lfloor \Gamma \rfloor \vdash_0 \Pi x :^r A_1.A_2 : s$
  - $A =_\beta \Pi x :^r A_1.A_2$
- If $\Gamma \vdash v :^q A$ and $A =_\beta \Sigma x :^r B_1.B_2$ and v is a value, then exists A_1, A_2, a_1 and a_2 such that
  - $v = ({a_1}^r, a_2)$
  - $\Gamma_1 \vdash a_1 :^{q \cdot r} A_1$ and $\Gamma_2 \vdash a_2 :^q A_2\{a_1/x\}$ where $\Gamma = \Gamma_1 + \Gamma_2$
  - $\lfloor \Gamma \rfloor \vdash_0 \Sigma x :^r A_1.A_2 : s$
  - $A =_\beta \Sigma x :^r A_1.A_2$

Proof. Follow the proof of Lemma C.26. □

Theorem D.34 (Preservation (Theorem 5.26)) If $\Gamma \vdash a :^q A$ and $\vdash a \leadsto a'$, then $\Gamma \vdash a' :^q A$.

Proof. By induction on $\Gamma \vdash a :^q A$ and subsequent inversion on $\vdash a \leadsto a'$.

- Rule LDC-App. Have: $\Gamma_1 + \Gamma_2 \vdash b\ a^r :^q B\{a/x\}$ where $\Gamma_1 \vdash b :^q \Pi x :^r A.B$ and $\Gamma_2 \vdash a :^{q \cdot r} A$.
  Let $\vdash b\ a^r \leadsto c$. By inversion:
  - $\vdash b\ a^r \leadsto b'\ a^r$, when $\vdash b \leadsto b'$.
    Need to show: $\Gamma_1 + \Gamma_2 \vdash b'\ a^r :^q B\{a/x\}$.
    Follows by IH and rule LDC-App.
  - $b = \lambda^r x : A'.b'$ and $\vdash b\ a^r \leadsto b'\{a/x\}$.
    Need to show: $\Gamma_1 + \Gamma_2 \vdash b'\{a/x\} :^q B\{a/x\}$.
    Since $\Gamma_1 \vdash \lambda^r x : A'.b' :^q \Pi x :^r A.B$, by Lemmas D.33 and D.32, we have, $A' =_\beta A$ and $\Gamma_1, x :^{q_0 \cdot r} A' \vdash b' :^{q_0} B$, for some $q_0 <: q$
    Now, there are two cases to consider.
    * $q_0 = 0$. Since $q_0 <: q$, so $q = 0$. As such, $q_0 = q$.
      So, we have, $\Gamma_1, x :^{q \cdot r} A' \vdash b' :^q B$.
      Next, by rule LDC-Conv, $\Gamma_2 \vdash a :^{q \cdot r} A'$.
      This case, then, follows by the substitution lemma.

* $q_0 \neq 0$. By lemma D.28, $\exists \Gamma_1'$ and r' such that $\Gamma_1', x :^{r'} A' \vdash b' :^1 B$ and $\Gamma_1 <: q_0 \cdot \Gamma_1'$ and $q_0 \cdot r <: q_0 \cdot r'$.
  Since $q_0 \neq 0$, therefore $r <: r'$. Hence, by rule LDC-SubL, $\Gamma_1', x :^r A' \vdash b' :^1 B$.
  Now, by lemma D.27, $q \cdot \Gamma_1', x :^{q \cdot r} A' \vdash b' :^q B$.
  By rule LDC-SubL, $\Gamma_1, x :^{q \cdot r} A' \vdash b' :^q B$.
  Next by rule LDC-Conv, $\Gamma_2 \vdash a :^{q \cdot r} A'$.
  This case, then, follows by the substitution lemma.

- Rule LDC-LetUnit. Have: $\Gamma_1 + \Gamma_2 \vdash \mathbf{let}_{q_0}\ \mathbf{unit} = a\ \mathbf{in}\ b :^q B\{a/z\}$ where $\Delta, z : \mathbf{Unit} \vdash_0 B : s$ and $\Gamma_1 \vdash a :^{q \cdot q_0} \mathbf{Unit}$ and $\Gamma_2 \vdash b :^q B\{\mathbf{unit}/z\}$ such that $\Delta = \lfloor \Gamma_1 \rfloor$.
  Let $\vdash \mathbf{let}_{q_0}\ \mathbf{unit} = a\ \mathbf{in}\ b \rightsquigarrow c$. By inversion:
  - $\vdash \mathbf{let}_{q_0}\ \mathbf{unit} = a\ \mathbf{in}\ b \rightsquigarrow \mathbf{let}_{q_0}\ \mathbf{unit} = a'\ \mathbf{in}\ b$, when $\vdash a \rightsquigarrow a'$.
    Need to show: $\Gamma_1 + \Gamma_2 \vdash \mathbf{let}_{q_0}\ \mathbf{unit} = a'\ \mathbf{in}\ b :^q B\{a/z\}$.
    By IH, $\Gamma_1 \vdash a' :^{q \cdot q_0} \mathbf{Unit}$.
    By rule LDC-LetUnit, $\Gamma_1 + \Gamma_2 \vdash \mathbf{let}_{q_0}\ \mathbf{unit} = a'\ \mathbf{in}\ b :^q B\{a'/z\}$.
    Next, since $\vdash a \rightsquigarrow a'$, so $a =_\beta a'$. As such, $B\{a/z\} =_\beta B\{a'/z\}$.
    This case, then, follows by rule LDC-Conv.
  - $\vdash \mathbf{let}_{q_0}\ \mathbf{unit} = \mathbf{unit}\ \mathbf{in}\ b \rightsquigarrow b$.
    Need to show: $\Gamma_1 + \Gamma_2 \vdash b :^q B\{\mathbf{unit}/z\}$.
    Follows by rule LDC-SubL (note that in case of $\mathbb{N}_=$, $\overline{\Gamma_1} = \mathbf{0}$).

- Rule LDC-LetPair. Have: $\Gamma_1 + \Gamma_2 \vdash \mathbf{let}_{q_0}\ (x^r, y) = a\ \mathbf{in}\ b :^q B\{a/z\}$ where $\Delta, z : \Sigma x :^r A_1.A_2 \vdash_0 B : s$ and $\Gamma_1 \vdash a :^{q \cdot q_0} \Sigma x :^r A_1.A_2$ and $\Gamma_2, x :^{q \cdot q_0 \cdot r} A_1, y :^{q \cdot q_0} A_2 \vdash b :^q B\{(x^r, y)/z\}$ such that $\Delta = \lfloor \Gamma_1 \rfloor$.
  Let $\vdash \mathbf{let}_{q_0}\ (x^r, y) = a\ \mathbf{in}\ b \rightsquigarrow c$. By inversion:
  - $\vdash \mathbf{let}_{q_0}\ (x^r, y) = a\ \mathbf{in}\ b \rightsquigarrow \mathbf{let}_{q_0}\ (x^r, y) = a'\ \mathbf{in}\ b$, when $\vdash a \rightsquigarrow a'$.
    Need to show: $\Gamma_1 + \Gamma_2 \vdash \mathbf{let}_{q_0}\ (x^r, y) = a'\ \mathbf{in}\ b :^q B\{a/z\}$.
    By IH, $\Gamma_1 \vdash a' :^{q \cdot q_0} \Sigma x :^r A_1.A_2$.
    By rule LDC-LetPair, $\Gamma_1 + \Gamma_2 \vdash \mathbf{let}_{q_0}\ (x^r, y) = a'\ \mathbf{in}\ b :^q B\{a'/z\}$.
    Next, since $\vdash a \rightsquigarrow a'$, so $a =_\beta a'$. As such, $B\{a/z\} =_\beta B\{a'/z\}$.
    This case, then, follows by rule LDC-Conv.
  - $\vdash \mathbf{let}_{q_0}\ (x^r, y) = ({a_1}^r, a_2)\ \mathbf{in}\ b \rightsquigarrow b\{a_1/x\}\{a_2/y\}$.
    Need to show: $\Gamma_1 + \Gamma_2 \vdash b\{a_1/x\}\{a_2/y\} :^q B\{({a_1}^r, a_2)/z\}$.
    Since $\Gamma_1 \vdash ({a_1}^r, a_2) :^{q \cdot q_0} \Sigma x :^r A_1.A_2$, by Lemmas D.33 and D.32, we have,
    $\Gamma_{11} \vdash a_1 :^{q \cdot q_0 \cdot r} A_1$ and $\Gamma_{12} \vdash a_2 :^{q \cdot q_0} A_2\{a_1/x\}$ and $\Gamma_1 = \Gamma_{11} + \Gamma_{12}$, for some Γ_{11} and Γ_{12}.
    Now, by the substitution lemma, $\Gamma_2 + \Gamma_{11}, y :^{q \cdot q_0} A_2\{a_1/x\} \vdash b\{a_1/x\} :^q B\{({a_1}^r, y)/z\}$.
    Applying the lemma again, we get, $\Gamma_2 + \Gamma_{11} + \Gamma_{12} \vdash b\{a_1/x\}\{a_2/y\} :^q B\{({a_1}^r, a_2)/z\}$, as required.

- Rule LDC-Case. Have: $\Gamma_1 + \Gamma_2 \vdash \mathbf{case}_{q_0}\ a\ \mathbf{of}\ x_1.b_1\ ;\ x_2.b_2 :^q B\{a/z\}$ where $\Delta, z : A_1 + A_2 \vdash_0 B : s$ and $\Gamma_1 \vdash a :^{q \cdot q_0} A_1 + A_2$ and $\Gamma_2, x_1 :^{q \cdot q_0} A_1 \vdash b_1 :^q B\{\mathbf{inj}_1\ x_1/z\}$ and $\Gamma_2, x_2 :^{q \cdot q_0} A_2 \vdash b_2 :^q B\{\mathbf{inj}_2\ x_2/z\}$.
  Let $\vdash \mathbf{case}_{q_0}\ a\ \mathbf{of}\ x_1.b_1\ ;\ x_2.b_2 \rightsquigarrow c$ such that $\Delta = \lfloor \Gamma_1 \rfloor$. By inversion:

– $\vdash \textbf{case}_{q_0}\ a \textbf{ of } x_1.b_1 \ ;\ x_2.b_2 \leadsto \textbf{case}_{q_0}\ a' \textbf{ of } x_1.b_1 \ ;\ x_2.b_2$, when $\vdash a \leadsto a'$.
Need to show: $\Gamma_1 + \Gamma_2 \vdash \textbf{case}_{q_0}\ a' \textbf{ of } x_1.b_1 \ ;\ x_2.b_2 :^q B\{a/z\}$.
By IH, $\Gamma_1 \vdash a' :^{q \cdot q_0} A_1 + A_2$.
By rule LDC-Case, $\Gamma_1 + \Gamma_2 \vdash \textbf{case}_{q_0}\ a' \textbf{ of } x_1.b_1 \ ;\ x_2.b_2 :^q B\{a'/z\}$.
Next, since $\vdash a \leadsto a'$, so $a =_\beta a'$. As such, $B\{a/z\} =_\beta B\{a'/z\}$.
This case, then, follows by rule LDC-Conv.

– $\vdash \textbf{case}_{q_0}\ (\textbf{inj}_1\ a_1) \textbf{ of } x_1.b_1 \ ;\ x_2.b_2 \leadsto b_1\{a_1/x_1\}$.
Need to show $\Gamma_1 + \Gamma_2 \vdash b_1\{a_1/x_1\} :^q B\{\textbf{inj}_1\ a_1/z\}$.
By inversion on $\Gamma_1 \vdash \textbf{inj}_1\ a_1 :^{q \cdot q_0} A_1 + A_2$, we have $\Gamma_1 \vdash a_1 :^{q \cdot q_0} A_1$.
This case, then, follows by applying the substitution lemma.

– $\vdash \textbf{case}_{q_0}\ (\textbf{inj}_2\ a_2) \textbf{ of } x_1.b_1 \ ;\ x_2.b_2 \leadsto b_2\{a_2/x_2\}$.
Similar to the previous case.

- Rules LDC-Weak, LDC-Conv, LDC-SubL, and LDC-SubR. Follows by IH.

□

Theorem D.35 (Progress (Theorem 5.27)) If $\varnothing \vdash a :^q A$, then either a is a value or there exists a' such that $\vdash a \leadsto a'$.

Proof. By induction on $\varnothing \vdash a :^q A$.

- Rule LDC-App. Have: $\varnothing \vdash b\ a^r :^q B\{a/x\}$ where $\varnothing \vdash b :^q \Pi x :^r A.B$ and $\varnothing \vdash a :^{q \cdot r} A$.
Need to show: $\exists c, \vdash b\ a^r \leadsto c$.
By IH, b is either a value or $\vdash b \leadsto b'$, for some b'.
If b is a value, then by Lemma D.33, $b = \lambda^r x : A'.b''$, for some b''. Therefore, $\vdash b\ a^r \leadsto b''\{a/x\}$.
Otherwise, $\vdash b\ a^r \leadsto b'\ a^r$.

- Rule LDC-LetUnit. Have: $\varnothing \vdash \textbf{let}_{q_0}\ \textbf{unit} = a \textbf{ in } b :^q B\{a/z\}$ where $z : \textbf{Unit} \vdash_0 B : s$ and $\varnothing \vdash a :^{q \cdot q_0}$ $\textbf{Unit}$ and $\varnothing \vdash b :^q B\{\textbf{unit}/z\}$.
Need to show: $\exists c, \vdash \textbf{let}_{q_0}\ \textbf{unit} = a \textbf{ in } b \leadsto c$.
By IH, a is either a value or $\vdash a \leadsto a'$, for some a'.
If a is a value, then by inversion, $a = \textbf{unit}$. Therefore, $\vdash \textbf{let}_{q_0}\ \textbf{unit} = a \textbf{ in } b \leadsto b$.
Otherwise, $\vdash \textbf{let}_{q_0}\ \textbf{unit} = a \textbf{ in } b \leadsto \textbf{let}_{q_0}\ \textbf{unit} = a' \textbf{ in } b$.

- Rule LDC-LetPair. Have: $\varnothing \vdash \textbf{let}_{q_0}\ (x^r, y) = a \textbf{ in } b :^q B\{a/z\}$ where $z : \Sigma x :^r A_1.A_2 \vdash_0 B : s$ and $\varnothing \vdash a :^{q \cdot q_0} \Sigma x :^r A_1.A_2$ and $x :^{q \cdot q_0 \cdot r} A_1, y :^{q \cdot q_0} A_2 \vdash b :^q B\{(x^r, y)/z\}$.
Need to show: $\exists c, \vdash \textbf{let}_{q_0}\ (x^r, y) = a \textbf{ in } b \leadsto c$.
By IH, a is either a value or $\vdash a \leadsto a'$, for some a'.
If a is a value, then by Lemma D.33, $a = ({a_1}^r, a_2)$. Therefore, $\vdash \textbf{let}_{q_0}\ (x^r, y) = a \textbf{ in } b \leadsto b\{a_1/x\}\{a_2/y\}$.
Otherwise, $\vdash \textbf{let}_{q_0}\ (x^r, y) = a \textbf{ in } b \leadsto \textbf{let}_{q_0}\ (x^r, y) = a' \textbf{ in } b$.

- Rule LDC-Case. Have: $\varnothing \vdash \textbf{case}_{q_0}\ a\ \textbf{of}\ x_1.b_1\ ;\ x_2.b_2 :^q B\{a/z\}$ where $z : A_1 + A_2 \vdash_0 B : s$ and $\varnothing \vdash a :^{q \cdot q_0} A_1 + A_2$ and $x_1 :^{q \cdot q_0} A_1 \vdash b_1 :^q B\{\textbf{inj}_1\ x_1/z\}$ and $x_2 :^{q \cdot q_0} A_2 \vdash b_2 :^q B\{\textbf{inj}_2\ x_2/z\}$.
  Need to show: $\exists c, \vdash \textbf{case}_{q_0}\ a\ \textbf{of}\ x_1.b_1\ ;\ x_2.b_2 \leadsto c$.
  By IH, a is either a value or $\vdash a \leadsto a'$, for some a'.
  If a is a value, then $a = \textbf{inj}_1\ a_1$ or $a = \textbf{inj}_2\ a_2$.
  Then, $\vdash \textbf{case}_{q_0}\ a\ \textbf{of}\ x_1.b_1\ ;\ x_2.b_2 \leadsto b_1\{a_1/x_1\}$ or $\vdash \textbf{case}_{q_0}\ a\ \textbf{of}\ x_1.b_1\ ;\ x_2.b_2 \leadsto b_2\{a_2/x_2\}$.
  Otherwise, $\vdash \textbf{case}_{q_0}\ a\ \textbf{of}\ x_1.b_1\ ;\ x_2.b_2 \leadsto \textbf{case}_{q_0}\ a'\ \textbf{of}\ x_1.b_1\ ;\ x_2.b_2$.

- Rules LDC-Weak, LDC-Conv, LDC-SubL, and LDC-SubR. Follows by IH.

- Rule LDC-Var. Does not apply since the context here is empty.

- The terms typed by the remaining rules are values.

□

Lemma D.36 (Multi-Substitution (Lemma 5.28)) If $H \Vdash^{\mathbf{e}} \Gamma$ and $\Gamma \vdash^{\mathbf{e}} a :^q A$, then $\varnothing \vdash^{\mathbf{u}} a\{\Gamma\} :^q A\{\Gamma\}$.

Proof. By induction on $H \Vdash^{\mathbf{e}} \Gamma$. Follow the proof of Lemma C.37. □

Lemma D.37 (Elaboration (Lemma 5.29)) If $H \Vdash^{\mathbf{u}} \Gamma$ and $\Gamma \vdash^{\mathbf{u}} a :^q A$, then $H \Vdash^{\mathbf{e}} \Gamma_H$ and $\Gamma_H \vdash^{\mathbf{e}} a :^q A$.

Proof. Follow proof of Lemma C.35. □

Lemma D.38 (Inserting definition) If $\Gamma_1, z :^r C, \Gamma_2 \vdash a :^q A$ and $\lfloor \Gamma_1 \rfloor \vdash_0 c : C$, then $\Gamma_1, z = c :^r C, \Gamma_2 \vdash a :^q A$.

Proof. Follow the proof of Lemma C.30. □

Lemma D.39 (Typing inversion) LDC, extended with definitions, satisfies the following inversion lemmas:

- If $\Gamma \vdash v :^q A$ and $A\{\Gamma\} =_\beta \Pi x :^r B_1.B_2$ and v is a value, then exists A_1, A_2 and a_2 such that
  - $v = \lambda^r x : A_1.a_2$
  - $\Gamma, x :^{q_0 \cdot r} A_1 \vdash a_2 :^{q_0} A_2$ where $q_0 <: q$
  - $\lfloor \Gamma \rfloor \vdash_0 \Pi x :^r A_1.A_2 : s$
  - $A\{\Gamma\} =_\beta \Pi x :^r A_1\{\Gamma\}.A_2\{\Gamma\}$

- If $\Gamma \vdash v :^q A$ and $A\{\Gamma\} =_\beta \Sigma x :^r B_1.B_2$ and v is a value, then exists A_1, A_2, a_1 and a_2 such that
  - $v = ({a_1}^r, a_2)$
  - $\Gamma_1 \vdash a_1 :^{q \cdot r} A_1$ and $\Gamma_2 \vdash a_2 :^q A_2\{a_1/x\}$ where $\Gamma = \Gamma_1 + \Gamma_2$
  - $\lfloor \Gamma \rfloor \vdash_0 \Sigma x :^r A_1.A_2 : s$

– $A\{\Gamma\} =_\beta \Sigma x :^r A_1\{\Gamma\}.A_2\{\Gamma\}$

Proof. Follow the proof of Lemma C.38. □

Lemma D.40 If $H \Vdash \Gamma_1 + \Gamma_2$ and $\Gamma_2 \vdash a :^q A$, then either a is a value or there exists H', Γ_2', a' such that:

- $[H]a \Longrightarrow^q_S [H']a'$
- $H' \Vdash \Gamma_1 + \Gamma_2'$
- $\Gamma_2' \vdash a' :^q A$

(Note that + is overloaded here: $\Gamma_1 + \Gamma_2$ denotes addition of contexts Γ_1 and Γ_2 after padding them as necessary.)

Proof. By induction on $\Gamma_2 \vdash a :^q A$.

- Rule LDC-Var-Def. Have: $0 \cdot \Gamma_{21}, x = a :^{q_2} A \vdash x :^{q_2} A$ where $\lfloor \Gamma_{21} \rfloor \vdash_0 a : A$.
  Further, $H \Vdash (\Gamma_{11}, x = a :^{q_1} A) + (0 \cdot \Gamma_{21}, x = a :^{q_2} A)$.
  By inversion, $\exists H_1, \Gamma_0$ such that $H = H_1, x \overset{q_1+q_2}{\mapsto} a$ and $\Gamma_0 \vdash a :^{q_1+q_2} A$.
  By lemma D.29, $\exists \Gamma_{01}, \Gamma_{02}$ such that $\Gamma_{01} \vdash a :^{q_1} A$ and $\Gamma_{02} \vdash a :^{q_2} A$ and $\Gamma_0 = \Gamma_{01} + \Gamma_{02}$.
  Then, we have,
  - $[H_1, x \overset{q_1+q_2}{\mapsto} a]x \Longrightarrow^{q_2}_S [H_1, x \overset{q_1}{\mapsto} a]a$.
  - $H_1, x \overset{q_1}{\mapsto} a \Vdash (\Gamma_{11}, x = a :^{q_1} A) + (\Gamma_{02}, x = a :^0 A)$.
  - $\Gamma_{02}, x = a :^0 A \vdash a :^{q_2} A$.
- Rule LDC-Weak-Def. Have: $\Gamma_{21}, y = b :^0 B \vdash a :^q A$ where $\Gamma_{21} \vdash a :^q A$ and $\lfloor \Gamma_{21} \rfloor \vdash_0 b : B$.
  Further, $H \Vdash (\Gamma_{11}, y = b :^{q_1} B) + (\Gamma_{21}, y = b :^0 B)$.
  By inversion, $\exists H_1, \Gamma_0$ such that $H = H_1, y \overset{q_1}{\mapsto} b$ and $\Gamma_0 \vdash b :^{q_1} B$ and $H_1 \Vdash \Gamma_{11} + \Gamma_{21} + \Gamma_0$.
  By IH, $\exists H_1', H_2', \Gamma_{21}', \Gamma_{22}', a'$ such that
  - $[H_1]a \Longrightarrow^q_{S \cup \{y\}} [H_1', H_2']a'$ where $|H_1'| = |H_1|$
  - $H_1', H_2' \Vdash (\Gamma_{11} + \Gamma_0 + \Gamma_{21}'), \Gamma_{22}'$ where $|\Gamma_{21}'| = |H_1'|$
  - $\Gamma_{21}', \Gamma_{22}' \vdash a' :^q A$

  Then, we have,
  - $[H_1, y \overset{q_1}{\mapsto} b]a \Longrightarrow^q_S [H_1', y \overset{q_1}{\mapsto} b, H_2']a'$
  - $H_1', y \overset{q_1}{\mapsto} b, H_2' \Vdash ((\Gamma_{11}, y = b :^{q_1} B) + (\Gamma_{21}', y = b :^0 B)), \Gamma_{22}'$
  - $\Gamma_{21}', y = b :^0 B, \Gamma_{22}' \vdash a' :^q A$

- Rules LDC-Var and LDC-Weak. Do not apply since whenever $H \Vdash \Gamma_1 + \Gamma_2$ holds, every assumption in Γ_2 is a definition.

- Rule LDC-App. Have: $\Gamma_{21} + \Gamma_{22} \vdash b\ a^r :^q B\{a/x\}$ where $\Gamma_{21} \vdash b :^q \Pi x :^r A.B$ and $\Gamma_{22} \vdash a :^{q \cdot r} A$.
Further, $H \Vdash \Gamma_1 + (\Gamma_{21} + \Gamma_{22})$.
By IH, if b steps, then $[H]b \Longrightarrow^q_{S \cup \mathrm{fv}\, a} [H']b'$ and $H' \Vdash (\Gamma_1 + \Gamma_{22}) + \Gamma'_{21}$ and $\Gamma'_{21} \vdash b' :^q \Pi x :^r A.B$.
Then, $[H]b\ a^r \Longrightarrow^q_S [H']b'\ a^r$ and $H' \Vdash \Gamma_1 + (\Gamma'_{21} + \Gamma_{22})$ and $\Gamma'_{21} + \Gamma_{22} \vdash b'\ a^r :^q B\{a/x\}$.
Otherwise, b is a value. Since $\Gamma_{21} \vdash b :^q \Pi x :^r A.B$, so by Lemma D.39, $b = \lambda^r x : A_1.b'$.
Further, $\Gamma_{21}, x :^{q \cdot r} A_1 \vdash b' :^q B_1$ and $\Pi x :^r A_1\{\Gamma_{21}\}.B_1\{\Gamma_{21}\} =_\beta \Pi x :^r A\{\Gamma_{21}\}.B\{\Gamma_{21}\}$. (Note that Lemma D.39 gives us $\Gamma_{21}, x :^{q_0 \cdot r} A_1 \vdash b' :^{q_0} B_1$, for some $q_0 <: q$. But from this judgment, we can derive $\Gamma_{21}, x :^{q \cdot r} A_1 \vdash b' :^q B_1$, as shown in the rule LDC-App case of Lemma D.34.)
Next, by Lemma D.32, $A_1\{\Gamma_{21}\} =_\beta A\{\Gamma_{21}\}$ and $B_1\{\Gamma_{21}\} =_\beta B\{\Gamma_{21}\}$.
Then, by rule LDC-Conv-Def, $\Gamma_{22} \vdash a :^{q \cdot r} A_1$.
So by Lemma D.38, $\Gamma_{21}, x = a :^{q \cdot r} A_1 \vdash b' :^q B_1$.
Now,

  - $[H](\lambda^r x : A_1.b')\ a^r \Longrightarrow^q_S [H, x \overset{q \cdot r}{\mapsto} a]b'$
  - $H, x \overset{q \cdot r}{\mapsto} a \Vdash (\Gamma_1 + \Gamma_{21}), x = a :^{q \cdot r} A_1$
  - $\Gamma_{21}, x = a :^{q \cdot r} A_1 \vdash b' :^q B\{a/x\}$, by rule LDC-Conv-Def.

- Rule LDC-Conv-Def. Have: $\Gamma_2 \vdash a :^q B$ where $\Gamma_2 \vdash a :^q A$ and $A\{\Gamma_2\} =_\beta B\{\Gamma_2\}$.
Further, $H \Vdash \Gamma_1 + \Gamma_2$.
By IH, $[H]a \Longrightarrow^q_S [H']a'$ and $H' \Vdash \Gamma_1 + \Gamma'_2$ and $\Gamma'_2 \vdash a' :^q A$.
Since $\mathrm{fv}\, A, \mathrm{fv}\, B \in \mathrm{dom}\, \Gamma_2$ and $\lfloor \Gamma_2 \rfloor$ is a sublist of $\lfloor \Gamma'_2 \rfloor$, so $A\{\Gamma'_2\} = A\{\Gamma_2\} =_\beta B\{\Gamma_2\} = B\{\Gamma'_2\}$.
This case, then, follows by rule LDC-Conv-Def.

- Rule LDC-LetPair. Have: $\Gamma_{21} + \Gamma_{22} \vdash \mathbf{let}_{q_0}\ (x^r, y) = a\ \mathbf{in}\ b :^q B\{a/z\}$ where $\lfloor \Gamma_{21} \rfloor, z : \Sigma x :^r A_1.A_2 \vdash_0 B : s$ and $\Gamma_{21} \vdash a :^{q \cdot q_0} \Sigma x :^r A_1.A_2$ and $\Gamma_{22}, x :^{q \cdot q_0 \cdot r} A_1, y :^{q \cdot q_0} A_2 \vdash b :^q B\{(x^r, y)/z\}$.
Further, $H \Vdash \Gamma_1 + (\Gamma_{21} + \Gamma_{22})$.
By IH, if a steps, then $[H]a \Longrightarrow^{q \cdot q_0}_{S \cup \{z\} \cup \mathrm{fv}\, b} [H']a'$ and $H' \Vdash (\Gamma_1 + \Gamma_{22}) + \Gamma'_{21}$ and $\Gamma'_{21} \vdash a' :^{q \cdot q_0} \Sigma x :^r A_1.A_2$.
Then,

  - $[H]\mathbf{let}_{q_0}\ (x^r, y) = a\ \mathbf{in}\ b \Longrightarrow^{q \cdot q_0}_S [H']\mathbf{let}_{q_0}\ (x^r, y) = a'\ \mathbf{in}\ b$.
    By rule HeapLD-SubR, $[H]\mathbf{let}_{q_0}\ (x^r, y) = a\ \mathbf{in}\ b \Longrightarrow^q_S [H']\mathbf{let}_{q_0}\ (x^r, y) = a'\ \mathbf{in}\ b$. $(\because q_0 <: 1)$
  - $H' \Vdash \Gamma_1 + (\Gamma'_{21} + \Gamma_{22})$
  - By rule LDC-LetPair, $\Gamma'_{21} + \Gamma_{22} \vdash \mathbf{let}_{q_0}\ (x^r, y) = a'\ \mathbf{in}\ b :\ B\{a'/z\}$.
    Now, by lemma D.17, $a'\{\Gamma'_{21}\} =_\beta a\{\Gamma_{21}\}$.
    Then, $B\{a/z\}\{\Gamma'_{21}\} = B\{\Gamma'_{21}\}\{a\{\Gamma'_{21}\}/z\}$.
    But since $\mathrm{fv}\, a \in \mathrm{dom}\, \Gamma_{21}$ and $\lfloor \Gamma_{21} \rfloor$ is a sublist of $\lfloor \Gamma'_{21} \rfloor$, so $a\{\Gamma'_{21}\} = a\{\Gamma_{21}\}$.
    As such, $B\{\Gamma'_{21}\}\{a\{\Gamma'_{21}\}/z\} = B\{\Gamma'_{21}\}\{a\{\Gamma_{21}\}/z\}$.
    Now, $B\{\Gamma'_{21}\}\{a\{\Gamma_{21}\}/z\} =_\beta B\{\Gamma'_{21}\}\{a'\{\Gamma'_{21}\}/z\} = B\{a'/z\}\{\Gamma'_{21}\}$.

So, $B\{a/z\}\{\Gamma'_{21}\} =_\beta B\{a'/z\}\{\Gamma'_{21}\}$.
Therefore, by rule LDC-Conv-Def, $\Gamma'_{21} + \Gamma_{22} \vdash \mathbf{let}_{q_0}\ (x^r, y) = a'\ \mathbf{in}\ b :\ B\{a/z\}$.

Otherwise, a is a value. Since $\Gamma_{21} \vdash a :^{q \cdot q_0} \Sigma x :^r A_1.A_2$, so by Lemma D.39, $a = ({a_1}^r, a_2)$.
Further, $\Gamma_{211} \vdash a_1 :^{q \cdot q_0 \cdot r} A'_1$ and $\Gamma_{212} \vdash a_2 :^{q \cdot q_0} A'_2\{a_1/x\}$ where $\Gamma_{21} = \Gamma_{211} + \Gamma_{212}$ and $\Sigma x :^r A_1\{\Gamma_{21}\}.A_2\{\Gamma_{21}\} =_\beta \Sigma x :^r A'_1\{\Gamma_{21}\}.A'_2\{\Gamma_{21}\}$.
By Lemma D.32, $A_1\{\Gamma_{21}\} =_\beta A'_1\{\Gamma_{21}\}$ and $A_2\{a_1/x\}\{\Gamma_{21}\} =_\beta A'_2\{a_1/x\}\{\Gamma_{21}\}$.
Then, by rule LDC-Conv-Def, $\Gamma_{211} \vdash a_1 :^{q \cdot q_0 \cdot r} A_1$ and $\Gamma_{212} \vdash a_2 :^{q \cdot q_0} A_2\{a_1/x\}$.
Now,

- $[H]\mathbf{let}_{q_0}\ (x^r, y) = ({a_1}^r, a_2)\ \mathbf{in}\ b \Longrightarrow^q_S [H, x \overset{q \cdot q_0 \cdot r}{\mapsto} a_1, y \overset{q \cdot q_0}{\mapsto} a_2]b$ (assuming x, y *fresh*)
- $H, x \overset{q \cdot q_0 \cdot r}{\mapsto} a_1, y \overset{q \cdot q_0}{\mapsto} a_2 \Vdash (\Gamma_1 + \Gamma_{22}), x = a_1 :^{q \cdot q_0 \cdot r} A_1, y = a_2 :^{q \cdot q_0} A_2$
- $\Gamma_{22}, x = a_1 :^{q \cdot q_0 \cdot r} A_1, y = a_2 :^{q \cdot q_0} A_2 \vdash b :^q B\{(x^r, y)/z\}$, by Lemma D.38.
  By rule LDC-Conv-Def, $\Gamma_{22}, x = a_1 :^{q \cdot q_0 \cdot r} A_1, y = a_2 :^{q \cdot q_0} A_2 \vdash b :^q B\{({a_1}^r, a_2)/z\}$.

- Rule LDC-LetUnit. Similar to rule LDC-LetPair.

- Rule LDC-Case. Have: $\Gamma_{21} + \Gamma_{22} \vdash \mathbf{case}_{q_0}\ a\ \mathbf{of}\ x_1.b_1\ ;\ x_2.b_2 :^q B\{a/z\}$ where $\lfloor \Gamma_{21} \rfloor, z : A_1 + A_2 \vdash_0 B : s$ and $\Gamma_{21} \vdash a :^{q \cdot q_0} A_1 + A_2$ and $\Gamma_{22}, x_1 :^{q \cdot q_0} A_1 \vdash b_1 :^q B\{\mathbf{inj}_1 x_1/z\}$ and $\Gamma_{22}, x_2 :^{q \cdot q_0} A_2 \vdash b_2 :^q B\{\mathbf{inj}_2 x_2/z\}$.
  Further, $H \Vdash \Gamma_1 + (\Gamma_{21} + \Gamma_{22})$.
  By IH, if a steps, then $[H]a \Longrightarrow^{q \cdot q_0}_{S \cup \mathrm{fv}\ b_1 \cup \mathrm{fv}\ b_2} [H']a'$ and $H' \Vdash (\Gamma_1 + \Gamma_{22}) + \Gamma'_{21}$ and $\Gamma'_{21} \vdash a' :^{q \cdot q_0} A_1 + A_2$.
  Then,

  - $[H]\mathbf{case}_{q_0}\ a\ \mathbf{of}\ x_1.b_1\ ;\ x_2.b_2 \Longrightarrow^{q \cdot q_0}_S [H']\mathbf{case}_{q_0}\ a'\ \mathbf{of}\ x_1.b_1\ ;\ x_2.b_2$.
    By rule HeapLD-SubR, $[H]\mathbf{case}_{q_0}\ a\ \mathbf{of}\ x_1.b_1\ ;\ x_2.b_2 \Longrightarrow^q_S [H']\mathbf{case}_{q_0}\ a'\ \mathbf{of}\ x_1.b_1\ ;\ x_2.b_2$. $(\because q_0 <: 1)$
  - $H' \Vdash \Gamma_1 + (\Gamma'_{21} + \Gamma_{22})$
  - $\Gamma'_{21} + \Gamma_{22} \vdash \mathbf{case}_{q_0}\ a'\ \mathbf{of}\ x_1.b_1\ ;\ x_2.b_2 :^q B\{a'/z\}$, by rule LDC-Case.
    But $B\{a'/z\}\{\Gamma'_{21}\} =_\beta B\{a/z\}\{\Gamma'_{21}\}$.
    So, $\Gamma'_{21} + \Gamma_{22} \vdash \mathbf{case}_{q_0}\ a'\ \mathbf{of}\ x_1.b_1\ ;\ x_2.b_2 :^q B\{a/z\}$, by rule LDC-Conv-Def.

  Otherwise, a is a value. By inversion, $a = \mathbf{inj}_1\ a_1$ or $a = \mathbf{inj}_2\ a_2$.
  Say $a = \mathbf{inj}_1\ a_1$. Then, $\Gamma_{21} \vdash a_1 :^{q \cdot q_0} A_1$.
  Now,

  - $[H]\mathbf{case}_{q_0}\ (\mathbf{inj}_1\ a_1)\ \mathbf{of}\ x_1.b_1\ ;\ x_2.b_2 \Longrightarrow^q_S [H, x_1 \overset{q \cdot q_0}{\mapsto} a_1]b_1$ (assuming x_1 *fresh*)
  - $H, x_1 \overset{q \cdot q_0}{\mapsto} a_1 \Vdash (\Gamma_1 + \Gamma_{22}), x_1 = a_1 :^{q \cdot q_0} A_1$
  - $\Gamma_{22}, x_1 = a_1 :^{q \cdot q_0} A_1 \vdash b_1 :^q B\{\mathbf{inj}_1\ x_1/z\}$, by Lemma D.38.
    By rule LDC-Conv-Def, $\Gamma_{22}, x_1 = a_1 :^{q \cdot q_0} A_1 \vdash b_1 :^q B\{\mathbf{inj}_1\ a_1/z\}$.

  The case when $a = \mathbf{inj}_2\ a_2$ is similar.

- Rules LDC-SubL and LDC-SubR. By IH.

□

Theorem D.41 (Soundness (Theorem 5.30)) If $H \Vdash \Gamma$ and $\Gamma \vdash a :^{q} A$, then either a is a value or there exists H', Γ', a' such that:

- $[H]a \Longrightarrow_{S}^{q} [H']a'$
- $H' \Vdash \Gamma'$
- $\Gamma' \vdash a' :^{q} A$

Proof. Use lemma D.40 with $\Gamma_1 := 0 \cdot \Gamma$ and $\Gamma_2 := \Gamma$. □

Lemma D.42 If $H \Vdash \Gamma_1 \sqcap \Gamma_2$ and $\Gamma_2 \vdash a :^{\ell} A$, then either a is a value or there exists H', Γ_2', a' such that:

- $[H]a \Longrightarrow_{S}^{\ell} [H']a'$
- $H' \Vdash \Gamma_1 \sqcap \Gamma_2'$
- $\Gamma_2' \vdash a' :^{\ell} A$

Proof. By induction on $\Gamma_2 \vdash a :^{\ell} A$. Same as the proof of lemma D.40 when $+, \cdot, 0, 1$ and $<:$ are replaced with $\sqcap, \sqcup, \top, \bot$ and $\sqsubseteq$ respectively. □

Theorem D.43 (Soundness (Theorem 5.30)) If $H \Vdash \Gamma$ and $\Gamma \vdash a :^{\ell} A$, then either a is a value or there exists H', Γ', a' such that:

- $[H]a \Longrightarrow_{S}^{\ell} [H']a'$
- $H' \Vdash \Gamma'$
- $\Gamma' \vdash a' :^{\ell} A$

Proof. Use lemma D.42 with $\Gamma_1 := \top \sqcup \Gamma$ and $\Gamma_2 := \Gamma$. □

Corollary D.44 (No Usage (Corollary 5.31)) In LDC($\mathcal{Q}_{\mathbb{N}}$): Let $\varnothing \vdash f :^{1}\ {}^{0}A \to \textbf{Bool}$. Then, for any $\varnothing \vdash a_1 :^{0} A$ and $\varnothing \vdash a_2 :^{0} A$, the terms $f\ {a_1}^{0}$ and $f\ {a_2}^{0}$ have the same operational behavior.

Proof. Same as Corollary D.24. □

Corollary D.45 (Noninterference (Corollary 5.32)) In LDC($\mathcal{L}$): Let $\varnothing \vdash f :^{\ell}\ {}^{m}A \to \textbf{Bool}$ such that $\neg(m \sqsubseteq \ell)$. Then, for any $\varnothing \vdash a_1 :^{m} A$ and $\varnothing \vdash a_2 :^{m} A$, the terms $f\ {a_1}^{m}$ and $f\ {a_2}^{m}$ have the same operational behavior.

Proof. Same as Corollary D.25. □

Corollary D.46 (Null Type (Corollary 5.33)) In LDC($\mathcal{Q}_{\mathbb{N}}$), if $\varnothing \vdash f :^1 \Pi x :^0 s.x$ and $\varnothing \vdash_0 A : s$, then $f\ A^0$ must diverge.

Proof. Follow the proof of Corollary C.45. Note that any $\mathcal{Q} \in \mathcal{Q}_{\mathbb{N}}$ is a zero-unusable semiring. ☐

Corollary D.47 (Polymorphic Identity Type (Corollary 5.34)) In LDC($\mathcal{Q}_{\mathbb{N}}$), given the underlying PTS is strongly normalizing, if $\varnothing \vdash f :^1 \Pi x :^0 s.\Pi y :^1 x.x$ and $\varnothing \vdash_0 A : s$ and $\varnothing \vdash a :^1 A$, then $f\ A^0\ a^1 =_\beta a$.

Proof. Follow the proof of Corollary C.46. Note that any $\mathcal{Q} \in \mathcal{Q}_{\mathbb{N}}$ is a zero-unusable semiring. ☐

D.5 Proofs of Lemmas/Theorems Stated in Section 5.5

Lemma D.48 (Multiplication) In LDC($\mathcal{Q}_{\mathbb{N}}^{\omega}$), if $\Gamma \vdash a :^q A$, then $r_0 \cdot \Gamma \vdash a :^{r_0 \cdot q} A$.

Proof. By induction on $\Gamma \vdash a :^q A$. All the cases other than that of rule LDC-LamOmega are similar to those of lemma D.27.

- Rule LDC-LamOmega. Have: $q_0 \cdot \Gamma \vdash \lambda^r x : A.b :^{q_0 \cdot q} \Pi x :^r A.B$ where $\Gamma, x :^{q \cdot r} A \vdash b :^q B$. Further, $q = \omega \Rightarrow r = \omega$ and $q_0 \neq 0$.
  Need to show: $r_0 \cdot q_0 \cdot \Gamma \vdash \lambda^r x : A.b :^{r_0 \cdot q_0 \cdot q} \Pi x :^r A.B$.
  There are two cases to consider:
  - $r_0 = 0$. By IH, $0 \cdot \Gamma, x :^0 A \vdash b :^0 B$.
    This case, then, follows by rule LDC-LamOmega.
  - $r_0 \neq 0$. Therefore, $r_0 \cdot q_0 \neq 0$.
    This case, then, follows by rule LDC-LamOmega.

☐

Lemma D.49 ($0/\omega$ at $0/\omega$) In LDC($\mathbb{N}_{=}^{\omega}$) and LDC($\mathcal{Q}_{\text{Lin}}$), if $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash a :^q A$ and $q \in \{0, \omega\}$, then $r_0 \in \{0, \omega\}$.

Proof. By induction on $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash a :^q A$. We present some of the interesting cases below.

- Rule LDC-Var. Have: $0 \cdot \Gamma_{11}, x :^q A \vdash x :^q A$. Further, $q \in \{0, \omega\}$.
  This case follows from premises.
- Rule LDC-Pi. Have: $\Gamma_{11} + \Gamma_{12}, z :^{r_{01} + r_{02}} C, \Gamma_{21} + \Gamma_{22} \vdash \Pi x :^r A.B :^q s_3$ where $\Gamma_{11}, z :^{r_{01}} C, \Gamma_{21} \vdash A :^q s_1$ and $\Gamma_{12}, z :^{r_{02}} C, \Gamma_{22}, x :^{r'} A \vdash B :^q s_2$. Further, $q \in \{0, \omega\}$.
  Need to show: $r_{01} + r_{02} \in \{0, \omega\}$.
  By IH, $r_{01} \in \{0, \omega\}$ and $r_{02} \in \{0, \omega\}$. Therefore, $r_{01} + r_{02} \in \{0, \omega\}$.

- Rule LDC-LamOmega. Have: $q_0 \cdot \Gamma_1, z :^{q_0 \cdot r_0} C, q_0 \cdot \Gamma_2 \vdash \lambda^r x : A.b :^{q_0 \cdot q} \Pi x :^r A.B$ where $\Gamma_1, z :^{r_0} C, \Gamma_2, x :^{q \cdot r} A \vdash b :^q B$ and $q_0 \neq 0$. Further, $q_0 \cdot q \in \{0, \omega\}$.
  Need to show: $q_0 \cdot r_0 \in \{0, \omega\}$.
  There are two cases to consider.

  - $q_0 \cdot q = 0$. Since $q_0 \neq 0$, so $q = 0$.
    Then, by IH, $r_0 \in \{0, \omega\}$. Therefore, $q_0 \cdot r_0 \in \{0, \omega\}$.
  - $q_0 \cdot q = \omega$. Then, $q_0 = \omega$ or $q = \omega$.
    If $q = \omega$, then by IH, $r_0 \in \{0, \omega\}$. So, $q_0 \cdot r_0 \in \{0, \omega\}$.
    If $q_0 = \omega$, then $q_0 \cdot r_0 \in \{0, \omega\}$.

- Rule LDC-App. Have: $\Gamma_{11} + \Gamma_{12}, z :^{r_{01} + r_{02}} C, \Gamma_{21} + \Gamma_{22} \vdash b\ a^r :^q B\{a/x\}$ where $\Gamma_{11}, z :^{r_{01}} C, \Gamma_{21} \vdash b :^q \Pi x :^r A.B$ and $\Gamma_{12}, z :^{r_{02}} C, \Gamma_{22} \vdash a :^{q \cdot r} A$. Further, $q \in \{0, \omega\}$.
  Need to show: $r_{01} + r_{02} \in \{0, \omega\}$.
  By IH, $r_{01} \in \{0, \omega\}$.
  Next, since $q \in \{0, \omega\}$, so $q \cdot r \in \{0, \omega\}$. Then, by IH, $r_{02} \in \{0, \omega\}$.
  So, $r_{01} + r_{02} \in \{0, \omega\}$.

- Rule LDC-Pair. Have: $\Gamma_{11} + \Gamma_{12}, z :^{r_{01} + r_{02}} C, \Gamma_{21} + \Gamma_{22} \vdash (a_1{}^r, a_2) :^q \Sigma x :^r A_1.A_2$ where $\Gamma_{11}, z :^{r_{01}} C, \Gamma_{21} \vdash a_1 :^{q \cdot r} A_1$ and $\Gamma_{12}, z :^{r_{02}} C, \Gamma_{22} \vdash a_2 :^q A_2\{a_1/x\}$. Further, $q \in \{0, \omega\}$.
  Need to show: $r_{01} + r_{02} \in \{0, \omega\}$.
  By IH, $r_{02} \in \{0, \omega\}$.
  Next, since $q \in \{0, \omega\}$, so $q \cdot r \in \{0, \omega\}$. Then, by IH, $r_{01} \in \{0, \omega\}$.
  So, $r_{01} + r_{02} \in \{0, \omega\}$.

- Rule LDC-LetPair. Have: $\Gamma_{11} + \Gamma_{12}, z :^{r_{01} + r_{02}} C, \Gamma_{21} + \Gamma_{22} \vdash \mathbf{let}_{q_0}\ (x^r, y) = a\ \mathbf{in}\ b :^q B\{a/w\}$ where $\Delta, w : \Sigma x :^r A_1.A_2 \vdash_0 B : s$ and $\Gamma_{11}, z :^{r_{01}} C, \Gamma_{21} \vdash a :^{q \cdot q_0} \Sigma x :^r A_1.A_2$ and $\Gamma_{12}, z :^{r_{02}} C, \Gamma_{22}, x :^{q \cdot q_0 \cdot r} A_1, y :^{q \cdot q_0} A_2 \vdash b :^q B\{(x^r, y)/w\}$ such that $\Delta = \lfloor \Gamma_{11} \rfloor, z : C, \lfloor \Gamma_{21} \rfloor$. Further, $q \in \{0, \omega\}$.
  Need to show: $r_{01} + r_{02} \in \{0, \omega\}$.
  By IH, $r_{02} \in \{0, \omega\}$.
  Next, since $q \in \{0, \omega\}$, so $q \cdot q_0 \in \{0, \omega\}$. Then, by IH, $r_{01} \in \{0, \omega\}$.
  So, $r_{01} + r_{02} \in \{0, \omega\}$.

- Rule LDC-SubL. Have: $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash a :^q A$ where $\Gamma'_1, z :^{r'_0} C, \Gamma'_2 \vdash a :^q A$ such that $\Gamma_1 <: \Gamma'_1$ and $r_0 <: r'_0$ and $\Gamma_2 <: \Gamma'_2$. Further, $q \in \{0, \omega\}$.
  Need to show: $r_0 \in \{0, \omega\}$.
  By IH, $r'_0 \in \{0, \omega\}$.
  Now, in case of $\mathbb{N}^{\omega}_{=}$ and $\mathcal{Q}_{\text{Lin}}$, $(q_0 <: q'_0) \wedge (q'_0 \in \{0, \omega\}) \Rightarrow q_0 \in \{0, \omega\}$.
  Since $r_0 <: r'_0$ and $r'_0 \in \{0, \omega\}$, so $r_0 \in \{0, \omega\}$.

- Rule LDC-SubR. Have: $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash a :^{q'} A$ where $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash a :^q A$ such that $q <: q'$. Further, $q' \in \{0, \omega\}$.
  Need to show: $r_0 \in \{0, \omega\}$.

In case of $\mathbb{N}^{\omega}_{=}$ and $\mathcal{Q}_{\text{Lin}}$, $(q_0 <: q_0') \wedge (q_0' \in \{0, \omega\}) \Rightarrow q_0 \in \{0, \omega\}$.
Since $q <: q'$ and $q' \in \{0, \omega\}$, so $q \in \{0, \omega\}$.
Then, by IH, $r_0 \in \{0, \omega\}$.

□

Lemma D.50 (No usage at 0) In LDC($\mathcal{Q}^{\omega}_{\mathbb{N}}$), if $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash a :^0 A$, then $\Gamma_1, z :^0 C, \Gamma_2 \vdash a :^0 A$.

Proof. We case-split based on the parametrizing structure.

- Parametrizing structure is $\mathbb{N}^{\omega}_{=}$ or $\mathcal{Q}_{\text{Lin}}$.
  Given: $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash a :^0 A$.
  Need to show: $\Gamma_1, z :^0 C, \Gamma_2 \vdash a :^0 A$.
  By Lemma D.49, $\overline{\Gamma_1}$ and $\overline{\Gamma_2}$ are vectors of 0's and ω's.
  Then, $\Gamma_1 <: 0 \cdot \Gamma_1$ and $\Gamma_2 <: 0 \cdot \Gamma_2$.
  Next, by multiplication lemma D.48, $0 \cdot \Gamma_1, z :^0 C, 0 \cdot \Gamma_2 \vdash a :^0 A$.
  This case, then, follows by rule LDC-SubL.

- Parametrizing structure is $\mathbb{N}^{\omega}_{\geq}$ or $\mathbb{B}_{\geq}$ or $\mathcal{Q}_{\text{Aff}}$.
  Given: $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash a :^0 A$.
  Need to show: $\Gamma_1, z :^0 C, \Gamma_2 \vdash a :^0 A$.
  By multiplication lemma D.48, $0 \cdot \Gamma_1, z :^0 C, 0 \cdot \Gamma_2 \vdash a :^0 A$.
  Next, observe that in all of these preordered semirings, 0 is the top element.
  So, $\Gamma_1 <: 0 \cdot \Gamma_1$ and $\Gamma_2 <: 0 \cdot \Gamma_2$.
  This case, then, follows by rule LDC-SubL.

□

Lemma D.51 (Unusability (Lemma 5.35)) If $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash a :^q A$ and $\neg(r_0 \lll: q)$, then $\Gamma_1, z :^0 C, \Gamma_2 \vdash a :^q A$. Here, $r_0 \lll: q \triangleq \exists q_0, r_0 <: q_0 + q$.

Proof. By induction on $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash a :^q A$. We present some of the interesting cases below.

- Rule LDC-Var. Have $0 \cdot \Gamma, x :^q A \vdash x :^q A$ where $\lfloor \Gamma \rfloor \vdash_0 A : s$.
  This case follows immediately because $q \lll: q$ and the other grades are all 0.

- Rule LDC-Pi. Have: $\Gamma_{11} + \Gamma_{12}, z :^{r_{01}+r_{02}} C, \Gamma_{21} + \Gamma_{22} \vdash \Pi x :^r A.B :^q s_3$ where $\Gamma_{11}, z :^{r_{01}} C, \Gamma_{21} \vdash A :^q s_1$ and $\Gamma_{12}, z :^{r_{02}} C, \Gamma_{22}, x :^{r'} A \vdash B :^q s_2$ and $\mathcal{R}(s_1, s_2, s_3)$ and $\neg(r_{01} + r_{02} \lll: q)$.
  Need to show: $\Gamma_{11} + \Gamma_{12}, z :^0 C, \Gamma_{21} + \Gamma_{22} \vdash \Pi x :^r A.B :^q s_3$.
  Now, if $r_{01} \lll: q$ or $r_{02} \lll: q$, then $r_{01} + r_{02} \lll: q$, a contradiction.
  Therefore, $\neg(r_{01} \lll: q)$ and $\neg(r_{02} \lll: q)$.
  This case, then, follows by IH and rule LDC-Pi.

- Rule LDC-LamOmega. Have: $q_0 \cdot \Gamma_1, z :^{q_0 \cdot r_0} C, q_0 \cdot \Gamma_2 \vdash \lambda^r x : A.b :^{q_0 \cdot q} \Pi x :^r A.B$ where $\Gamma_1, z :^{r_0} C, \Gamma_2, x :^{q \cdot r} A \vdash b :^q B$ and $q = \omega \Rightarrow r = \omega$ and $q_0 \neq 0$ and $\neg(q_0 \cdot r_0 \lll: q_0 \cdot q)$.
  Need to show: $q_0 \cdot \Gamma_1, z :^0 C, q_0 \cdot \Gamma_2 \vdash \lambda^r x : A.b :^{q_0 \cdot q} \Pi x :^r A.B$.
  If $r_0 \lll: q$, then $q_0 \cdot r_0 \lll: q_0 \cdot q$, a contradiction. Therefore, $\neg(r_0 \lll: q)$.
  This case, then, follows by IH and rule LDC-LamOmega.

- Rule LDC-App. Have: $\Gamma_{11} + \Gamma_{12}, z :^{r_{01}+r_{02}} C, \Gamma_{21} + \Gamma_{22} \vdash b\ a^r :^q B\{a/x\}$ where $\Gamma_{11}, z :^{r_{01}} C, \Gamma_{21} \vdash b :^q \Pi x :^r A.B$ and $\Gamma_{12}, z :^{r_{02}} C, \Gamma_{22} \vdash a :^{q \cdot r} A$ and $\neg(r_{01} + r_{02} \lll: q)$.
  Need to show: $\Gamma_{11} + \Gamma_{12}, z :^0 C, \Gamma_{21} + \Gamma_{22} \vdash b\ a^r :^q B\{a/x\}$.
  If $r_{01} \lll: q$, then $r_{01} + r_{02} \lll: q$, a contradiction. Therefore, $\neg(r_{01} \lll: q)$.
  So, by IH, $\Gamma_{11}, z :^0 C, \Gamma_{21} \vdash b :^q \Pi x :^r A.B$.
  Now, there are two cases to consider.

  - $r = 0$. So, by Lemma D.50, $\Gamma_{12}, z :^0 C, \Gamma_{22} \vdash a :^0 A$.
    The case, then, follows by rule LDC-App.
  - $r \neq 0$. Then, $r = 1 + r'$, for some r'.
    Now, if $r_{02} \lll: q \cdot r$, or $r_{02} \lll: q + q \cdot r'$, then $r_{02} \lll: q$, which implies, $r_{01} + r_{02} \lll: q$. A contradiction.
    Therefore, $\neg(r_{02} \lll: q \cdot r)$.
    So, by IH, $\Gamma_{12}, z :^0 C, \Gamma_{22} \vdash a :^{q \cdot r} A$.
    This case, then, follows by rule LDC-App.

- Rule LDC-Pair. Have: $\Gamma_{11} + \Gamma_{12}, z :^{r_{01}+r_{02}} C, \Gamma_{21} + \Gamma_{22} \vdash (a_1{}^r, a_2) :^q \Sigma x :^r A_1.A_2$ where $\Gamma_{11}, z :^{r_{01}} C, \Gamma_{21} \vdash a_1 :^{q \cdot r} A_1$ and $\Gamma_{12}, z :^{r_{02}} C, \Gamma_{22} \vdash a_2 :^q A_2\{a_1/x\}$ and $\neg(r_{01} + r_{02} \lll: q)$.
  Need to show: $\Gamma_{11} + \Gamma_{12}, z :^0 C, \Gamma_{21} + \Gamma_{22} \vdash (a_1{}^r, a_2) :^q \Sigma x :^r A_1.A_2$.
  If $r_{02} \lll: q$, then $r_{01} + r_{02} \lll: q$, a contradiction. Therefore, $\neg(r_{02} \lll: q)$.
  So, by IH, $\Gamma_{12}, z :^0 C, \Gamma_{22} \vdash a_2 :^q A_2\{a_1/x\}$.
  Now, there are two cases to consider.

  - $r = 0$. So, by Lemma D.50, $\Gamma_{11}, z :^0 C, \Gamma_{21} \vdash a_1 :^0 A_1$.
    The case, then, follows by rule LDC-Pair.
  - $r \neq 0$. Then, $r = 1 + r'$, for some r'.
    Now, if $r_{01} \lll: q \cdot r$, or $r_{01} \lll: q + q \cdot r'$, then $r_{01} \lll: q$, which implies, $r_{01} + r_{02} \lll: q$. A contradiction.
    Therefore, $\neg(r_{01} \lll: q \cdot r)$.
    So, by IH, $\Gamma_{11}, z :^0 C, \Gamma_{21} \vdash a_1 :^{q \cdot r} A_1$.
    This case, then, follows by rule LDC-Pair.

- Rule LDC-LetPair. Have: $\Gamma_{11} + \Gamma_{12}, z :^{r_{01}+r_{02}} C, \Gamma_{21} + \Gamma_{22} \vdash \mathbf{let}_{q_0}\ (x^r, y) = a\ \mathbf{in}\ b :^q B\{a/w\}$ where $\lfloor \Gamma_{11} \rfloor, z : C, \lfloor \Gamma_{21} \rfloor, w : \Sigma x :^r A_1.A_2 \vdash_0 B : s$ and $\Gamma_{11}, z :^{r_{01}} C, \Gamma_{21} \vdash a :^{q \cdot q_0} \Sigma x :^r A_1.A_2$ and $\Gamma_{12}, z :^{r_{02}} C, \Gamma_{22}, x :^{q \cdot q_0 \cdot r} A_1, y :^{q \cdot q_0} A_2 \vdash b :^q B\{(x^r, y)/w\}$ and $q_0 <: 1$ and $\neg(r_{01} + r_{02} \lll: q)$.
  Need to show: $\Gamma_{11} + \Gamma_{12}, z :^0 C, \Gamma_{21} + \Gamma_{22} \vdash \mathbf{let}_{q_0}\ (x^r, y) = a\ \mathbf{in}\ b :^q B\{a/z\}$.
  Since $q_0 <: 1$, so $q_0 = q_0' + 1$, for some q_0'.
  Now, if $r_{01} \lll: q \cdot q_0$, or $r_{01} \lll: q \cdot q_0' + q$, then $r_{01} \lll: q$, which implies, $r_{01} + r_{02} \lll: q$. A contradiction.

Therefore, $\neg(r_{01} \lll: q \cdot q_0)$.
So, by IH, $\Gamma_{11}, z :^0 C, \Gamma_{21} \vdash a :^{q \cdot q_0} \Sigma x :^r A_1.A_2$.
Next, if $r_{02} \lll: q$, then $r_{01} + r_{02} \lll: q$, a contradiction. Therefore, $\neg(r_{02} \lll: q)$.
So, by IH, $\Gamma_{12}, z :^0 C, \Gamma_{22}, x :^{q \cdot q_0 \cdot r} A_1, y :^{q \cdot q_0} A_2 \vdash b :^q B\{(x^r, y)/w\}$.
This case, then, follows by rule LDC-LetPair.

- Rule LDC-SubL. Have: $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash a :^q A$ where $\Gamma_1', z :^{r_0'} C, \Gamma_2' \vdash a :^q A$ such that $\Gamma_1 <: \Gamma_1'$ and $r_0 <: r_0'$ and $\Gamma_2 <: \Gamma_2'$. Further, $\neg(r_0 \lll: q)$.
Need to show: $\Gamma_1, z :^0 C, \Gamma_2 \vdash a :^q A$.
If $r_0' \lll: q$, then $r_0 \lll: q$, a contradiction. Therefore, $\neg(r_0' \lll: q)$.
So, by IH, $\Gamma_1', z :^0 C, \Gamma_2' \vdash a :^q A$.
This case, then, follows by rule LDC-SubL.

- Rule LDC-SubR. Have: $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash a :^{q'} A$ where $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash a :^q A$ and $q <: q'$. Further, $\neg(r_0 \lll: q')$.
Need to show: $\Gamma_1, z :^0 C, \Gamma_2 \vdash a :^{q'} A$.
If $r_0 \lll: q$, then $r_0 \lll: q'$, a contradiction. Therefore, $\neg(r_0 \lll: q)$.
So, by IH, $\Gamma_1, z :^0 C, \Gamma_2 \vdash a :^q A$.
This case, then, follows by rule LDC-SubR.

□

Lemma D.52 (Factorization (Lemma 5.36)) If $\Gamma \vdash a :^q A$ and $q \neq 0$, then there exists Γ' such that $\Gamma' \vdash a :^1 A$ and $\Gamma <: q \cdot \Gamma'$.

Proof. By induction on $\Gamma \vdash a :^q A$. We present some of the interesting cases below.

- Rule LDC-Pi. Have: $\Gamma_1 + \Gamma_2 \vdash \Pi x :^r A.B :^q s_3$ where $\Gamma_1 \vdash A :^q s_1$ and $\Gamma_2, x :^{r_0} A \vdash B :^q s_2$.
Need to show: $\exists \Gamma'$ such that $\Gamma' \vdash \Pi x :^r A.B :^1 s_3$ and $\Gamma_1 + \Gamma_2 <: q \cdot \Gamma'$.
By IH, $\exists \Gamma_1', \Gamma_2'$ such that $\Gamma_1' \vdash A :^1 s_1$ and $\Gamma_2', x :^{r'} A \vdash B :^1 s_2$ where $\Gamma_1 <: q \cdot \Gamma_1'$ and $\Gamma_2 <: q \cdot \Gamma_2'$.
This case, then, follows by rule LDC-Pi by setting $\Gamma' := \Gamma_1' + \Gamma_2'$.

- Rule LDC-LamOmega. Have: $q_0 \cdot \Gamma \vdash \lambda^r x : A.b :^{q_0 \cdot q} \Pi x :^r A.B$ where $\Gamma, x :^{q \cdot r} A \vdash b :^q B$ and $q = \omega \Rightarrow r = \omega$ and $q_0 \neq 0$.
Need to show: $\exists \Gamma'$ such that $\Gamma' \vdash \lambda^r x : A.b :^1 \Pi x :^r A.B$ and $q_0 \cdot \Gamma <: (q_0 \cdot q) \cdot \Gamma'$.
There are two cases to consider:
  - $q = \omega$. Then, $q_0 \cdot q = \omega$.
  Here, we need to define an operation on contexts. For a context Γ, define $\Gamma^{\{0,\omega\}}$ as:

$$\varnothing^{\{0,\omega\}} = \varnothing$$

$$(\Gamma, x :^q A)^{\{0,\omega\}} = \begin{cases} \Gamma^{\{0,\omega\}}, x :^q A \text{ if } q \in \{0,\omega\} \\ \Gamma^{\{0,\omega\}}, x :^0 A \text{ otherwise} \end{cases}$$

Now, since $q_0 \cdot \Gamma \vdash \lambda^r x : A.b :^\omega \Pi x :^r A.B$, so by lemma D.51, $(q_0 \cdot \Gamma)^{\{0,\omega\}} \vdash \lambda^r x : A.b :^\omega \Pi x :^r A.B$. (Note that ω being an additive annihilator and the bottom element, $q_1 \lll: \omega \Rightarrow q_1 = \omega$.)
Then, by rule LDC-SubR, $(q_0 \cdot \Gamma)^{\{0,\omega\}} \vdash \lambda^r x : A.b :^1 \Pi x :^r A.B$.

Next, we show: $q_0 \cdot \Gamma <: (q_0 \cdot \Gamma)^{\{0,\omega\}}$.
In case of $\mathbb{N}^\omega_\geq$, $\mathbb{B}_\geq$ and $\mathcal{Q}_{\text{Aff}}$, we have, $q_0 \cdot \Gamma <: (q_0 \cdot \Gamma)^{\{0,\omega\}}$, because in these preordered semirings, 0 is the top element.
In case of $\mathbb{N}^\omega_=$ and $\mathcal{Q}_{\text{Lin}}$, given that $q_0 \cdot \Gamma \vdash \lambda^r x : A.b :^\omega \Pi x :^r A.B$, by Lemma D.49, we find $\overline{q_0 \cdot \Gamma}$ is a vector of 0's and ω's. So, $q_0 \cdot \Gamma = (q_0 \cdot \Gamma)^{\{0,\omega\}}$.
As such, for any $\mathcal{Q}^\omega_\mathbb{N}$, we have, $q_0 \cdot \Gamma <: (q_0 \cdot \Gamma)^{\{0,\omega\}}$.

Now, $q_0 \cdot \Gamma <: (q_0 \cdot \Gamma)^{\{0,\omega\}} = \omega \cdot (q_0 \cdot \Gamma)^{\{0,\omega\}} = q_0 \cdot q \cdot (q_0 \cdot \Gamma)^{\{0,\omega\}}$.
This case, then, follows by setting $\Gamma' := (q_0 \cdot \Gamma)^{\{0,\omega\}}$.

- $q \neq \omega$. By IH, $\Gamma', x :^{r'} A \vdash b :^1 B$ where $\Gamma <: q \cdot \Gamma'$ and $q \cdot r <: q \cdot r'$.
  Since $q \cdot r <: q \cdot r'$ and $q \notin \{0,\omega\}$, so $r <: r'$.
  So, by rule LDC-SubL, $\Gamma', x :^r A \vdash b :^1 B$.
  This case, then, follows by rule LDC-LamOmega by setting $\Gamma' := \Gamma'$.

- Rule LDC-App. Have: $\Gamma_1 + \Gamma_2 \vdash b\ a^r :^q B\{a/x\}$ where $\Gamma_1 \vdash b :^q \Pi x :^r A.B$ and $\Gamma_2 \vdash a :^{q \cdot r} A$.
  Need to show: $\exists \Gamma'$ such that $\Gamma' \vdash b\ a^r :^1 B\{a/x\}$ and $\Gamma_1 + \Gamma_2 <: q \cdot \Gamma'$.
  By IH, $\exists \Gamma'_1$ such that $\Gamma'_1 \vdash b :^1 \Pi x :^r A.B$ and $\Gamma_1 <: q \cdot \Gamma'_1$.
  Now, there are two cases to consider:

  - $r = 0$. Then, $\Gamma_2 \vdash a :^0 A$.
    By the multiplication lemma, $0 \cdot \Gamma_2 \vdash a :^0 A$.
    So, by rule LDC-App, $\Gamma'_1 \vdash b\ a^0 :^1 B\{a/x\}$.
    Next, we show that $\Gamma_1 + \Gamma_2 <: q \cdot \Gamma'_1$.
    In case of $\mathbb{N}^\omega_\geq$, $\mathbb{B}_\geq$ and $\mathcal{Q}_{\text{Aff}}$, we have, $\Gamma_1 + \Gamma_2 <: \Gamma_1 <: q \cdot \Gamma'_1$.
    In case of $\mathbb{N}^\omega_=$ and $\mathcal{Q}_{\text{Lin}}$, given that $\Gamma_2 \vdash a :^0 A$, by Lemma D.49, we find that $\overline{\Gamma_2}$ is a vector of 0's and ω's. Therefore, $\Gamma_1 + \Gamma_2 <: \Gamma_1 <: q \cdot \Gamma'_1$.
    This case, then, follows by setting $\Gamma' := \Gamma'_1$.
  - $r \neq 0$. By IH, $\exists \Gamma'_2$ such that $\Gamma'_2 \vdash a :^1 A$ and $\Gamma_2 <: q \cdot r \cdot \Gamma'_2$.
    Then, by the multiplication lemma, $r \cdot \Gamma'_2 \vdash a :^r A$.
    So, by rule LDC-App, $\Gamma'_1 + r \cdot \Gamma'_2 \vdash b\ a^r :^1 B\{a/x\}$.
    This case, then, follows by setting $\Gamma' := \Gamma'_1 + r \cdot \Gamma'_2$.

- Rule LDC-Pair. Have: $\Gamma_1 + \Gamma_2 \vdash (a_1{}^r, a_2) :^q \Sigma x :^r A_1.A_2$ where $\Gamma_1 \vdash a_1 :^{q \cdot r} A_1$ and $\Gamma_2 \vdash a_2 :^q A_2\{a_1/x\}$.
  Need to show: $\exists \Gamma'$ such that $\Gamma' \vdash (a_1{}^r, a_2) :^1 \Sigma x :^r A_1.A_2$ and $\Gamma_1 + \Gamma_2 <: q \cdot \Gamma'$.

By IH, $\exists \Gamma_2'$ such that $\Gamma_2' \vdash a_2 :^1 A_2\{a_1/x\}$ and $\Gamma_2 <: q \cdot \Gamma_2'$.
Now, there are two cases to consider.

- $r = 0$. Then, $\Gamma_1 \vdash a_1 :^0 A_1$.
  By the multiplication lemma, $0 \cdot \Gamma_1 \vdash a_1 :^0 A_1$.
  So, by rule LDC-Pair, $\Gamma_2' \vdash (a_1{}^0, a_2) :^1 \Sigma x :^0 A_1.A_2$.
  Next, we show that $\Gamma_1 + \Gamma_2 <: q \cdot \Gamma_2'$.
  In case of $\mathbb{N}_{\geq}^{\omega}$, $\mathbb{B}_{\geq}$ and $\mathcal{Q}_{\text{Aff}}$, we have, $\Gamma_1 + \Gamma_2 <: \Gamma_2 <: q \cdot \Gamma_2'$.
  In case of $\mathbb{N}_{=}^{\omega}$ and $\mathcal{Q}_{\text{Lin}}$, given that $\Gamma_1 \vdash a_1 :^0 A_1$, by Lemma D.49, we find that $\overline{\Gamma_1}$ is a vector of 0's and ω's. Therefore, $\Gamma_1 + \Gamma_2 <: \Gamma_2 <: q \cdot \Gamma_2'$.
  This case, then, follows by setting $\Gamma' := \Gamma_2'$.
- $r \neq 0$. By IH, $\exists \Gamma_1'$ such that $\Gamma_1' \vdash a_1 :^1 A_1$ and $\Gamma_1 <: (q \cdot r) \cdot \Gamma_1'$.
  Then, by the multiplication lemma, $r \cdot \Gamma_1' \vdash a_1 :^r A_1$.
  So, by rule LDC-Pair, $r \cdot \Gamma_1' + \Gamma_2' \vdash (a_1{}^r, a_2) :^1 \Sigma x :^r A_1.A_2$.
  This case, then, follows by setting $\Gamma' := r \cdot \Gamma_1' + \Gamma_2'$.

- Rule LDC-LetPair. Have: $\Gamma_1 + \Gamma_2 \vdash \textbf{let}_{q_0}\,(x^r, y) = a\,\textbf{in}\,b :^q B\{a/z\}$ where $\lfloor \Gamma_1 \rfloor, z : \Sigma x :^r A_1.A_2 \vdash_0 B : s$ and $\Gamma_1 \vdash a :^{q \cdot q_0} \Sigma x :^r A_1.A_2$ and $\Gamma_2, x :^{q \cdot q_0 \cdot r} A_1, y :^{q \cdot q_0} A_2 \vdash b :^q B\{(x^r, y)/z\}$ and $q_0 <: 1$.
  Need to show: $\exists \Gamma'$ such that $\Gamma' \vdash \textbf{let}_{q_0}\,(x^r, y) = a\,\textbf{in}\,b :^1 B\{a/z\}$ and $\Gamma_1 + \Gamma_2 <: q \cdot \Gamma'$.
  There are two cases to consider:

  - $q = \omega$. Since $\Gamma_1 + \Gamma_2 \vdash \textbf{let}_{q_0}\,(x^r, y) = a\,\textbf{in}\,b :^\omega B\{a/z\}$, so by lemma D.51, $(\Gamma_1 + \Gamma_2)^{\{0,\omega\}} \vdash \textbf{let}_{q_0}\,(x^r, y) = a\,\textbf{in}\,b :^\omega B\{a/z\}$. (Note that ω being an additive annihilator and the bottom element, $q_1 \lll: \omega \Rightarrow q_1 = \omega$.)
    Then, by rule LDC-SubR, $(\Gamma_1 + \Gamma_2)^{\{0,\omega\}} \vdash \textbf{let}_{q_0}\,(x^r, y) = a\,\textbf{in}\,b :^1 B\{a/z\}$.

    Next, we show: $\Gamma_1 + \Gamma_2 <: (\Gamma_1 + \Gamma_2)^{\{0,\omega\}}$.
    In case of $\mathbb{N}_{\geq}^{\omega}$, $\mathbb{B}_{\geq}$ and $\mathcal{Q}_{\text{Aff}}$, we have, $\Gamma_1 + \Gamma_2 <: (\Gamma_1 + \Gamma_2)^{\{0,\omega\}}$, because in these preordered semirings, 0 is the top element.
    In case of $\mathbb{N}_{=}^{\omega}$ and $\mathcal{Q}_{\text{Lin}}$, given that $\Gamma_1 + \Gamma_2 \vdash \textbf{let}_{q_0}\,(x^r, y) = a\,\textbf{in}\,b :^\omega B\{a/z\}$, by Lemma D.49, we find $\overline{\Gamma_1 + \Gamma_2}$ is a vector of 0's and ω's. So, $\Gamma_1 + \Gamma_2 = (\Gamma_1 + \Gamma_2)^{\{0,\omega\}}$.
    As such, for any $\mathcal{Q}_{\mathbb{N}}^{\omega}$, we have, $\Gamma_1 + \Gamma_2 <: (\Gamma_1 + \Gamma_2)^{\{0,\omega\}}$.

    Now, $\Gamma_1 + \Gamma_2 <: (\Gamma_1 + \Gamma_2)^{\{0,\omega\}} = \omega \cdot (\Gamma_1 + \Gamma_2)^{\{0,\omega\}} = q \cdot (\Gamma_1 + \Gamma_2)^{\{0,\omega\}}$.
    This case, then, follows by setting $\Gamma' := (\Gamma_1 + \Gamma_2)^{\{0,\omega\}}$.
  - $q \neq \omega$. By IH, $\exists \Gamma_1'$ such that $\Gamma_1' \vdash a :^1 \Sigma x :^r A_1.A_2$ and $\Gamma_1 <: q \cdot q_0 \cdot \Gamma_1'$.
    By the multiplication lemma, $q_0 \cdot \Gamma_1' \vdash a :^{q_0} \Sigma x :^r A_1.A_2$.
    Again, by IH, $\exists \Gamma_2'$ such that $\Gamma_2', x :^{r_1'} A_1, y :^{r_2'} A_2 \vdash b :^1 B\{(x^r, y)/z\}$ and $\Gamma_2 <: q \cdot \Gamma_2'$ and $q \cdot q_0 \cdot r <: q \cdot r_1'$ and $q \cdot q_0 <: q \cdot r_2'$.
    Now, since $q \notin \{0, \omega\}$, so $q_0 \cdot r <: r_1'$ and $q_0 <: r_2'$.
    Then, by rule LDC-SubL, $\Gamma_2', x :^{q_0 \cdot r} A_1, y :^{q_0} A_2 \vdash b :^1 B\{(x^r, y)/z\}$.

So, by rule LDC-LetPair, $q_0 \cdot \Gamma_1' + \Gamma_2' \vdash \mathbf{let}_{q_0}\ (x^r, y) = a\ \mathbf{in}\ b :^1 B\{a/z\}$.
This case, then, follows by setting $\Gamma' \coloneqq q_0 \cdot \Gamma_1' + \Gamma_2'$.

□

Lemma D.53 (Splitting) In LDC($\mathcal{Q}_{\mathbb{N}}^{\omega}$), if $\Gamma \vdash a :^{q_1+q_2} A$, then there exists Γ_1 and Γ_2 such that $\Gamma_1 \vdash a :^{q_1} A$ and $\Gamma_2 \vdash a :^{q_2} A$ and $\Gamma = \Gamma_1 + \Gamma_2$.

Proof. If $q_1 + q_2 = 0$, then set $\Gamma_1 \coloneqq 0 \cdot \Gamma$ and $\Gamma_2 \coloneqq \Gamma$.
Otherwise, by lemma D.52, $\exists \Gamma'$ such that $\Gamma' \vdash a :^1 A$ and $\Gamma <: (q_1 + q_2) \cdot \Gamma'$.
Then, $\Gamma = (q_1 + q_2) \cdot \Gamma' + \Gamma_0$, for some Γ_0.

Now, we show: $q_2 \cdot \Gamma' + \Gamma_0 <: q_2 \cdot \Gamma'$.
In case of $\mathbb{N}_{\geq}^{\omega}$, $\mathbb{B}_{\geq}$ and $\mathcal{Q}_{\text{Aff}}$, the above relation holds because in these preordered semirings, $r + r' <: r$ for any r and r'.
In case of $\mathbb{N}_{=}^{\omega}$ and $\mathcal{Q}_{\text{Lin}}$, the above relation holds because $\lfloor \Gamma_0 \rfloor$ is a vector of 0's and ω's.

Next, by multiplication lemma D.48, $q_1 \cdot \Gamma' \vdash a :^{q_1} A$ and $q_2 \cdot \Gamma' \vdash a :^{q_2} A$.
Then, by rule LDC-SubL, $q_2 \cdot \Gamma' + \Gamma_0 \vdash a :^{q_2} A$.
The lemma follows by setting $\Gamma_1 \coloneqq q_1 \cdot \Gamma'$ and $\Gamma_2 \coloneqq q_2 \cdot \Gamma' + \Gamma_0$. □

Lemma D.54 (Weakening) In LDC($\mathcal{Q}_{\mathbb{N}}^{\omega}$), if $\Gamma_1, \Gamma_2 \vdash a :^q A$ and $\lfloor \Gamma_1 \rfloor \vdash_0 C : s$, then $\Gamma_1, z :^0 C, \Gamma_2 \vdash a :^q A$.

Proof. By induction on $\Gamma_1, \Gamma_2 \vdash a :^q A$. □

Lemma D.55 (Substitution) In LDC($\mathcal{Q}_{\mathbb{N}}^{\omega}$), if $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash a :^q A$ and $\Gamma \vdash c :^{r_0} C$ where $\lfloor \Gamma_1 \rfloor = \lfloor \Gamma \rfloor$, then $\Gamma_1 + \Gamma, \Gamma_2\{c/z\} \vdash a\{c/z\} :^q A\{c/z\}$.

Proof. By induction on $\Gamma_1, z :^{r_0} C, \Gamma_2 \vdash a :^q A$. All the cases other than that of rule LDC-LamOmega are similar to those of lemma D.31.

- Rule LDC-LamOmega. Have: $q_0 \cdot \Gamma_1, z :^{q_0 \cdot r_0} C, q_0 \cdot \Gamma_2 \vdash \lambda^r x : A.b :^{q_0 \cdot q} \Pi x :^r A.B$ where $\Gamma_1, z :^{r_0} C, \Gamma_2, x :^{q \cdot r} A \vdash b :^q B$ and $q = \omega \Rightarrow r = \omega$ and $q_0 \neq 0$.
  Further, $\Gamma \vdash c :^{q_0 \cdot r_0} C$ where $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
  Need to show: $q_0 \cdot \Gamma_1 + \Gamma, q_0 \cdot \Gamma_2\{c/z\} \vdash \lambda^r x : A\{c/z\}.b\{c/z\} :^{q_0 \cdot q} \Pi x :^r A\{c/z\}.B\{c/z\}$.
  There are two cases to consider:
  - $r_0 = 0$. Then, $\Gamma \vdash c :^0 C$.
    By multiplication lemma D.48, $0 \cdot \Gamma \vdash c :^0 C$.
    Then, by IH, $\Gamma_1, \Gamma_2\{c/z\}, x :^{q \cdot r} A\{c/z\} \vdash b\{c/z\} :^q B\{c/z\}$.

By rule LDC-LamOmega, $q_0 \cdot \Gamma_1, q_0 \cdot \Gamma_2\{c/z\} \vdash \lambda^r x : A\{c/z\}.b\{c/z\} :^{q_0 \cdot q} \Pi x :^r A\{c/z\}.B\{c/z\}$.

Next, we show: $q_0 \cdot \Gamma_1 + \Gamma <: q_0 \cdot \Gamma_1$.
In case of $\mathbb{N}^{\omega}_{\geq}$, $\mathbb{B}_{\geq}$ and $\mathcal{Q}_{\text{Aff}}$, the above relation holds because in these preordered semirings, $r_1 + r_2 <: r_1$ for any r_1 and r_2.
In case of $\mathbb{N}^{\omega}_{=}$ and $\mathcal{Q}_{\text{Lin}}$, the above relation holds because given that $\Gamma \vdash c :^0 C$, by Lemma D.50, $\lfloor \Gamma \rfloor$ is a vector of 0's and ω's.

This case, then, follows by rule LDC-SubL.

- $r_0 \neq 0$. Then $q_0 \cdot r_0 \neq 0$.
  By the factorization lemma, $\exists \Gamma'$ such that $\Gamma' \vdash c :^1 C$ where $\Gamma <: q_0 \cdot r_0 \cdot \Gamma'$.
  By the multiplication lemma, $r_0 \cdot \Gamma' \vdash c :^{r_0} C$.
  By IH, $\Gamma_1 + r_0 \cdot \Gamma', \Gamma_2\{c/z\}, x :^{q \cdot r} A\{c/z\} \vdash b\{c/z\} :^q B\{c/z\}$.
  By rule LDC-LamOmega, $q_0 \cdot \Gamma_1 + q_0 \cdot r_0 \cdot \Gamma', q_0 \cdot \Gamma_2\{c/z\} \vdash \lambda^r x : A\{c/z\}.b\{c/z\} :^{q_0 \cdot q} \Pi x :^r A\{c/z\}.B\{c/z\}$.
  This case, then, follows by rule LDC-SubL.

□

Lemma D.56 (Lambda Inversion) In LDC($\mathcal{Q}^{\omega}_{\mathbb{N}}$), if $\Gamma \vdash v :^q A$ and $A =_\beta \Pi x :^r B_1.B_2$ and v is a value, then exists A_1, A_2, a_2, Γ_0, q_0 and q_1 such that

- $v = \lambda^r x : A_1.a_2$
- $\Gamma_0, x :^{q_0 \cdot r} A_1 \vdash a_2 :^{q_0} A_2$ where $\Gamma <: q_1 \cdot \Gamma_0$ and $q_1 \cdot q_0 <: q$
- $q_0 = \omega \Rightarrow r = \omega$ and $q_1 \neq 0$
- $\lfloor \Gamma \rfloor \vdash_0 \Pi x :^r A_1.A_2 : s$
- $A =_\beta \Pi x :^r A_1.A_2$

Proof. By induction on $\Gamma \vdash v :^q A$. We present some of the interesting cases below.

- Rule LDC-LamOmega. Have: $q_1 \cdot \Gamma_0 \vdash \lambda^r x : A_1.a_2 :^{q_1 \cdot q_0} \Pi x :^r A_1.A_2$ where $\Gamma_0, x :^{q_0 \cdot r} A_1 \vdash a_2 :^{q_0} A_2$ and $\lfloor \Gamma_0 \rfloor \vdash_0 \Pi x :^r A_1.A_2 : s$ and $q_0 = \omega \Rightarrow r = \omega$ and $q_1 \neq 0$. Further, $\Pi x :^r A_1.A_2 =_\beta \Pi x :^{r'} B_1.B_2$.
  By Lemma D.32, $r = r'$.
  This case, then, follows from premises.
- Rule LDC-Conv. Have: $\Gamma \vdash v :^q B$ where $\Gamma \vdash v :^q A$ and $A =_\beta B$ and $\lfloor \Gamma \rfloor \vdash_0 B : s$. Further, v is a value and $B =_\beta \Pi x :^r B_1.B_2$.
  Then, by transitivity, $A =_\beta \Pi x :^r B_1.B_2$.
  Using IH, we have:

- $v = \lambda^r x : A_1.a_2$.
- $\Gamma_0, x :^{q_0 \cdot r} A_1 \vdash a_2 :^{q_0} A_2$ where $\Gamma <: q_1 \cdot \Gamma_0$ and $q_1 \cdot q_0 <: q$
- $q_0 = \omega \Rightarrow r = \omega$ and $q_1 \neq 0$
- $\lfloor \Gamma \rfloor \vdash_0 \Pi x :^q A_1.A_2 : s$
- $A =_\beta \Pi x :^q A_1.A_2$.
  By symmetry and transitivity, $B =_\beta \Pi x :^q A_1.A_2$.

- Rule LDC-SubL. Have: $\Gamma \vdash v :^q A$ where $\Gamma' \vdash v :^q A$ and $\Gamma <: \Gamma'$. Further, v is a value and $A =_\beta \Pi x :^r B_1.B_2$.
  Using IH, we have:
  - $v = \lambda^r x : A_1.a_2$.
  - $\Gamma_0, x :^{q_0 \cdot r} A_1 \vdash a_2 :^{q_0} A_2$ where $\Gamma' <: q_1 \cdot \Gamma_0$ and $q_1 \cdot q_0 <: q$.
    Then, by transitivity, $\Gamma <: q_1 \cdot \Gamma_0$.
  - $q_0 = \omega \Rightarrow r = \omega$ and $q_1 \neq 0$
  - $\lfloor \Gamma' \rfloor \vdash_0 \Pi x :^q A_1.A_2 : s$
  - $A =_\beta \Pi x :^q A_1.A_2$.

- Rule LDC-SubR. Have: $\Gamma \vdash v :^{q'} A$ where $\Gamma \vdash v :^q A$ and $q <: q'$. Further, v is a value and $A =_\beta \Pi x :^r B_1.B_2$.
  Using IH, we have:
  - $v = \lambda^r x : A_1.a_2$.
  - $\Gamma_0, x :^{q_0 \cdot r} A_1 \vdash a_2 :^{q_0} A_2$ where $\Gamma <: q_1 \cdot \Gamma_0$ and $q_1 \cdot q_0 <: q$.
    Then, by transitivity, $q_1 \cdot q_0 <: q'$.
  - $q_0 = \omega \Rightarrow r = \omega$ and $q_1 \neq 0$
  - $\lfloor \Gamma \rfloor \vdash_0 \Pi x :^q A_1.A_2 : s$
  - $A =_\beta \Pi x :^q A_1.A_2$.

□

Theorem D.57 (Preservation (Theorem 5.37)) If $\Gamma \vdash a :^q A$ and $\vdash a \rightsquigarrow a'$, then $\Gamma \vdash a' :^q A$.

Proof. By induction on $\Gamma \vdash a :^q A$ and subsequent inversion on $\vdash a \rightsquigarrow a'$. All the cases other than that of rule LDC-App are similar to those of lemma D.34.

- Rule LDC-App. Have: $\Gamma_1 + \Gamma_2 \vdash b\ a^r :^q B\{a/x\}$ where $\Gamma_1 \vdash b :^q \Pi x :^r A.B$ and $\Gamma_2 \vdash a :^{q \cdot r} A$.
  Let $\vdash b\ a^r \rightsquigarrow c$. By inversion:

– $\vdash b\ a^r \rightsquigarrow b'\ a^r$, when $\vdash b \rightsquigarrow b'$.

Need to show: $\Gamma_1 + \Gamma_2 \vdash b'\ a^r :^q B\{a/x\}$.

Follows by IH and rule LDC-App.

– $b = \lambda^r x : A'.b'$ and $\vdash b\ a^r \rightsquigarrow b'\{a/x\}$.

Need to show: $\Gamma_1 + \Gamma_2 \vdash b'\{a/x\} :^q B\{a/x\}$.

Applying lemma D.56 on $\Gamma_1 \vdash \lambda^r x : A'.b' :^q \Pi x :^r A.B$, we get:

  * $\Gamma_0, x :^{q_0 \cdot r} A' \vdash b' :^{q_0} B'$
  * $\Gamma_1 <: q_1 \cdot \Gamma_0$ and $q_1 \cdot q_0 <: q$
  * $q_0 = \omega \Rightarrow r = \omega$ and $q_1 \neq 0$
  * $\Pi x :^r A.B =_\beta \Pi x :^r A'.B'$.

    Then, by Lemma D.32, $A =_\beta A'$ and $B\{a/x\} =_\beta B'\{a/x\}$.

Now, there are three cases to consider.

  * $q = 0$. Then, $\Gamma_2 \vdash a :^0 A$.

    By multiplication lemma D.48, $0 \cdot \Gamma_2 \vdash a :^0 A$.

    By rule LDC-Conv, $0 \cdot \Gamma_2 \vdash a :^0 A'$.

    Now, since $\Gamma_0, x :^0 A' \vdash b' :^0 B'$, so by substitution lemma, $\Gamma_0 \vdash b'\{a/x\} :^0 B'\{a/x\}$.

    By rule LDC-Conv, $\Gamma_0 \vdash b'\{a/x\} :^0 B\{a/x\}$.

    By multiplication lemma D.48, $q_1 \cdot \Gamma_0 \vdash b'\{a/x\} :^0 B\{a/x\}$.

    By rule LDC-SubL, $\Gamma_1 \vdash b'\{a/x\} :^0 B\{a/x\}$.

    Now, $\Gamma_1 + \Gamma_2 <: \Gamma_1$ because in case of $\mathbb{N}^\omega_\geq$, $\mathbb{B}_\geq$ and $\mathcal{Q}_{\text{Aff}}$, $r_1 + r_2 <: r_1$ for any r_1 and r_2 whereas in case of $\mathbb{N}^\omega_=$ and $\mathcal{Q}_{\text{Lin}}$, $\lfloor \Gamma_2 \rfloor$ is a vector of 0's and ω's since $\Gamma_2 \vdash a :^0 A$.

    So, by rule LDC-SubL, $\Gamma_1 + \Gamma_2 \vdash b'\{a/x\} :^0 B\{a/x\}$.

  * $q \neq 0$ and $q_0 = \omega$. Then, $r = \omega$.

    Since $q \cdot r = \omega$, so we have, $\Gamma_2 \vdash a :^\omega A$.

    By rule LDC-Conv, $\Gamma_2 \vdash a :^\omega A'$.

    Next, since $q_0 \cdot r = \omega$, so we have, $\Gamma_0, x :^\omega A' \vdash b' :^\omega B'$.

    By multiplication lemma D.48, $q_1 \cdot \Gamma_0, x :^\omega A' \vdash b' :^\omega B'$.

    By rule LDC-SubL, $\Gamma_1, x :^\omega A' \vdash b' :^\omega B'$.

    Then, by the substitution lemma, $\Gamma_1 + \Gamma_2 \vdash b'\{a/x\} :^\omega B'\{a/x\}$.

    By rule LDC-SubR, $\Gamma_1 + \Gamma_2 \vdash b'\{a/x\} :^q B'\{a/x\}$, since $\omega <: q$, for all q.

    Finally, by rule LDC-Conv, $\Gamma_1 + \Gamma_2 \vdash b'\{a/x\} :^q B\{a/x\}$.

  * $q \neq 0$ and $q_0 \neq \omega$. Since $q_1 \cdot q_0 <: q$, so $q_0 \neq 0$. Then, $q_0 \notin \{0, \omega\}$.

    Now, since $\Gamma_0, x :^{q_0 \cdot r} A' \vdash b' :^{q_0} B'$, so by factorization lemma D.52, $\exists \Gamma'_0$ and r' such that $\Gamma'_0, x :^{r'} A' \vdash b' :^1 B'$ and $\Gamma_0 <: q_0 \cdot \Gamma'_0$ and $q_0 \cdot r <: q_0 \cdot r'$.

    But since $q_0 \notin \{0, \omega\}$, so $r <: r'$.

    Then, by rule LDC-SubL, $\Gamma'_0, x :^r A' \vdash b' :^1 B'$.

    By multiplication lemma D.48, $q \cdot \Gamma'_0, x :^{q \cdot r} A' \vdash b' :^q B'$.

    By rule LDC-SubL, $\Gamma_1, x :^{q \cdot r} A' \vdash b' :^q B'$, since $\Gamma_1 <: q_1 \cdot \Gamma_0 <: q_1 \cdot q_0 \cdot \Gamma'_0 <: q \cdot \Gamma'_0$.

    Next, since $\Gamma_2 \vdash a :^{q \cdot r} A$, so by rule LDC-Conv, $\Gamma_2 \vdash a :^{q \cdot r} A'$.

Then, by the substitution lemma, $\Gamma_1 + \Gamma_2 \vdash b'\{a/x\} :^q B'\{a/x\}$.
Finally, by rule LDC-Conv, $\Gamma_1 + \Gamma_2 \vdash b'\{a/x\} :^q B\{a/x\}$.

□

Theorem D.58 (Progress (Theorem 5.38)) If $\varnothing \vdash a :^q A$, then either a is a value or there exists a' such that $\vdash a \rightsquigarrow a'$.

Proof. By induction on $\varnothing \vdash a :^q A$. Follow the proof of theorem D.35. □

Lemma D.59 (Inserting definition) In LDC($\mathcal{Q}_{\mathbb{N}}^{\omega}$), extended with definitions, if $\Gamma_1, z :^r C, \Gamma_2 \vdash a :^q A$ and $\lfloor \Gamma_1 \rfloor \vdash_0 c : C$, then $\Gamma_1, z = c :^r C, \Gamma_2 \vdash a :^q A$.

Proof. Same as Lemma D.38. □

Lemma D.60 (Lambda Inversion) In LDC($\mathcal{Q}_{\mathbb{N}}^{\omega}$), extended with definitions, if $\Gamma \vdash v :^q A$ and $A\{\Gamma\} =_\beta \Pi x :^r B_1.B_2$ and v is a value, then exists A_1, A_2, a_2, Γ_0, q_0 and q_1 such that

- $v = \lambda^r x : A_1.a_2$
- $\Gamma_0, x :^{q_0 \cdot r} A_1 \vdash a_2 :^{q_0} A_2$ where $\Gamma <: q_1 \cdot \Gamma_0$ and $q_1 \cdot q_0 <: q$
- $q_0 = \omega \Rightarrow r = \omega$ and $q_1 \neq 0$
- $\lfloor \Gamma \rfloor \vdash_0 \Pi x :^r A_1.A_2 : s$
- $A\{\Gamma\} =_\beta \Pi x :^r A_1\{\Gamma\}.A_2\{\Gamma\}$

Proof. By induction on $\Gamma \vdash v :^q A$. Follow the proof of Lemma D.56. □

Lemma D.61 In LDC($\mathcal{Q}_{\mathbb{N}}^{\omega}$), extended with definitions, if $H \Vdash \Gamma_1 + \Gamma_2$ and $\Gamma_2 \vdash a :^q A$, then either a is a value or there exists H', Γ_2', a' such that:

- $[H]a \Longrightarrow^q_S [H']a'$
- $H' \Vdash \Gamma_1 + \Gamma_2'$
- $\Gamma_2' \vdash a' :^q A$

Proof. By induction on $\Gamma_2 \vdash a :^q A$. All the cases other than that of rule LDC-App are similar to those of lemma D.40.

- Rule LDC-App. Have: $\Gamma_{21} + \Gamma_{22} \vdash b\ a^r :^q B\{a/x\}$ where $\Gamma_{21} \vdash b :^q \Pi x :^r A.B$ and $\Gamma_{22} \vdash a :^{q \cdot r} A$.
  Further, $H \Vdash \Gamma_1 + (\Gamma_{21} + \Gamma_{22})$.
  If b steps, then this case follows by IH. Otherwise, b is a value.
  Then, using lemma D.60 on $\Gamma_{21} \vdash b :^q \Pi x :^r A.B$, we get:

- $b = \lambda^r x : A'.b'$
- $\Gamma_{20}, x :^{q_0 \cdot r} A' \vdash b' :^{q_0} B'$ where $\Gamma_{21} <: q_1 \cdot \Gamma_{20}$ and $q_1 \cdot q_0 <: q$
- $q_0 = \omega \Rightarrow r = \omega$ and $q_1 \neq 0$
- $\Pi x :^r A\{\Gamma_{21}\}.B\{\Gamma_{21}\} =_\beta \Pi x :^r A'\{\Gamma_{21}\}.B'\{\Gamma_{21}\}$.
  Then, by Lemma D.32, $A\{\Gamma_{21}\} =_\beta A'\{\Gamma_{21}\}$ and $B\{\Gamma_{21}\} =_\beta B'\{\Gamma_{21}\}$.

Now, there are three cases to consider.

- $q = 0$. Then, $\Gamma_{21} \vdash b :^0 \Pi x :^r A.B$ and $\Gamma_{22} \vdash a :^0 A$.
  Next, we have, $\Gamma_{20}, x :^{q_0 \cdot r} A' \vdash b' :^{q_0} B'$.
  By the multiplication lemma, $0 \cdot \Gamma_{20}, x :^0 A' \vdash b' :^0 B'$.
  Now, we show: $\Gamma_{21} <: 0 \cdot \Gamma_{20}$.
  In case of $\mathbb{N}^\omega_\geq$, $\mathbb{B}_\geq$ and $\mathcal{Q}_{\text{Aff}}$, the above relation holds because 0 is the top element.
  In case of $\mathbb{N}^\omega_=$ and $\mathcal{Q}_{\text{Lin}}$, the above relation holds because $\lfloor \Gamma_{21} \rfloor$ is a vector of 0's and ω's. This is so by Lemma D.50 since we have, $\Gamma_{21} \vdash b :^0 \Pi x :^r A.B$.
  Then, by rule LDC-SubL, $\Gamma_{21}, x :^0 A' \vdash b' :^0 B'$.
  Next, since $\Gamma_{22} \vdash a :^0 A$, so by rule LDC-Conv-Def, $\Gamma_{22} \vdash a :^0 A'$.
  By Lemma D.59, $\Gamma_{21}, x = a :^0 A' \vdash b' :^0 B'$.
  Now, since $B'\{a/x\}\{\Gamma_{21}\} =_\beta B\{a/x\}\{\Gamma_{21}\}$, so $\Gamma_{21}, x = a :^0 A' \vdash b' :^0 B\{a/x\}$, by rule LDC-Conv-Def. Then,
  * $[H](\lambda^r x : A'.b')\ a^r \Longrightarrow^0_S [H, x \overset{0}{\mapsto} a]b'$
  * $H, x \overset{0}{\mapsto} a \Vdash (\Gamma_1 + \Gamma_{21}), x = a :^0 A'$
  * $\Gamma_{21}, x = a :^0 A' \vdash b' :^q B\{a/x\}$.
- $q \neq 0$ and $q_0 = \omega$. Since $q_0 = \omega$, so $r = \omega$. Further, since $q \neq 0$, so $q \cdot r = \omega$.
  Now, we have, $\Gamma_{20}, x :^\omega A' \vdash b' :^\omega B'$.
  By the multiplication lemma, $q_1 \cdot \Gamma_{20}, x :^\omega A' \vdash b' :^\omega B'$.
  By rule LDC-SubL, $\Gamma_{21}, x :^\omega A' \vdash b' :^\omega B'$.
  Next, since $\Gamma_{22} \vdash a :^\omega A$, so by rule LDC-Conv-Def, $\Gamma_{22} \vdash a :^\omega A'$.
  Then, by Lemma D.59, $\Gamma_{21}, x = a :^\omega A' \vdash b' :^\omega B'$.
  And, by rule LDC-SubR, $\Gamma_{21}, x = a :^\omega A' \vdash b' :^q B'$.
  Now, since $B'\{a/x\}\{\Gamma_{21}\} =_\beta B\{a/x\}\{\Gamma_{21}\}$, so $\Gamma_{21}, x = a :^\omega A' \vdash b' :^q B\{a/x\}$, by rule LDC-Conv-Def. Then,
  * $[H](\lambda^\omega x : A'.b')\ a^\omega \Longrightarrow^q_S [H, x \overset{\omega}{\mapsto} a]b'$
  * $H, x \overset{\omega}{\mapsto} a \Vdash (\Gamma_1 + \Gamma_{21}), x = a :^\omega A'$
  * $\Gamma_{21}, x = a :^\omega A' \vdash b' :^q B\{a/x\}$.
- $q \neq 0$ and $q_0 \neq \omega$. Since $q_1 \cdot q_0 <: q$, so $q_0 \neq 0$. Then, $q_0 \notin \{0, \omega\}$.
  Now, since $\Gamma_{20}, x :^{q_0 \cdot r} A' \vdash b' :^{q_0} B'$, so by factorization lemma D.52, $\exists \Gamma'_{20}$ and r' such that $\Gamma'_{20}, x :^{r'} A' \vdash b' :^1 B'$ and $\Gamma_{20} <: q_0 \cdot \Gamma'_{20}$ and $q_0 \cdot r <: q_0 \cdot r'$.
  But since $q_0 \notin \{0, \omega\}$, so $r <: r'$.

Then, by rule LDC-SubL, $\Gamma'_{20}, x :^{r} A' \vdash b' :^{1} B'$.
By multiplication lemma D.48, $q \cdot \Gamma'_{20}, x :^{q \cdot r} A' \vdash b' :^{q} B'$.
By rule LDC-SubL, $\Gamma_{21}, x :^{q \cdot r} A' \vdash b' :^{q} B'$, since $\Gamma_{21} <: q_1 \cdot \Gamma_{20} <: q_1 \cdot q_0 \cdot \Gamma'_{20} <: q \cdot \Gamma'_{20}$.
Next, since $\Gamma_{22} \vdash a :^{q \cdot r} A$, so by rule LDC-Conv-Def, $\Gamma_{22} \vdash a :^{q \cdot r} A'$.
Then, by Lemma D.59, $\Gamma_{21}, x = a :^{q \cdot r} A' \vdash b' :^{q} B'$.
Now, since $B'\{a/x\}\{\Gamma_{21}\} =_{\beta} B\{a/x\}\{\Gamma_{21}\}$, so $\Gamma_{21}, x = a :^{q \cdot r} A' \vdash b' :^{q} B\{a/x\}$, by rule LDC-Conv-Def. Then,

* $[H](\lambda^{r} x : A'.b')\ a^{r} \Longrightarrow_{S}^{q} [H, x \overset{q \cdot r}{\mapsto} a] b'$
* $H, x \overset{q \cdot r}{\mapsto} a \Vdash (\Gamma_1 + \Gamma_{21}), x = a :^{q \cdot r} A'$
* $\Gamma_{21}, x = a :^{q \cdot r} A' \vdash b' :^{q} B\{a/x\}$.

□

Theorem D.62 (Soundness) In LDC($\mathcal{Q}_{\mathbb{N}}^{\omega}$), extended with definitions, if $H \Vdash \Gamma$ and $\Gamma \vdash a :^{q} A$, then either a is a value or there exists H', Γ', a' such that $[H]a \Longrightarrow_{S}^{q} [H']a'$ and $H' \Vdash \Gamma'$ and $\Gamma' \vdash a' :^{q} A$.

Proof. Use lemma D.61 with $\Gamma_1 := 0 \cdot \Gamma$ and $\Gamma_2 := \Gamma$. □

Theorem D.63 (Theorem 5.39) LDC($\mathcal{Q}_{\mathbb{N}}^{\omega}$) satisfies heap soundness.

Proof. Follows from theorem D.62 via multi-substitution and elaboration. □

D.6 Proofs of Lemmas/Theorems Stated in Section 5.6

Lemma D.64 (LDC and GraD (Lemma 5.40)) For any $\mathcal{Q} \in \mathcal{Q}_{\mathbb{N}}^{\omega}$, if $\Gamma \vdash a : A$ in GraD($\mathcal{Q}$), then $\overrightarrow{\Gamma} \vdash \overline{a} :^{1} \overline{A}$ in LDC($\mathcal{Q}$). Further, if $\vdash a \rightsquigarrow a'$ in GraD($\mathcal{Q}$), then $\vdash \overline{a} \rightsquigarrow \overline{a'}$ in LDC($\mathcal{Q}$).

Proof. First, we show that if $\Gamma \vdash a : A$ in GraD($\mathcal{Q}$), then $\overrightarrow{\Gamma} \vdash \overline{a} :^{1} \overline{A}$ in LDC($\mathcal{Q}$).
By induction on $\Gamma \vdash a : A$. We present some of the interesting cases below.

- Rule GraD-Var. Have: $0 \cdot \Gamma_1, x :^{1} A \vdash x : A$ where $\Gamma_1 \vdash A : s$.
  Need to show: $0 \cdot \overrightarrow{\Gamma_1}, x :^{1} \overline{A} \vdash x :^{1} \overline{A}$.
  By IH, $\overrightarrow{\Gamma_1} \vdash \overline{A} :^{1} s$.
  By the multiplication lemma, $0 \cdot \overrightarrow{\Gamma_1} \vdash \overline{A} :^{0} s$.
  This case, then, follows by rule LDC-Var.
- Rule GraD-Pi. Have: $\Gamma_1 + \Gamma_2 \vdash \Pi x :^{r} A.B : s_3$ where $\Gamma_1 \vdash A : s_1$ and $\Gamma_2, x :^{r_0} A \vdash B : s_2$ and $\mathcal{R}(s_1, s_2, s_3)$.
  Need to show: $\overrightarrow{\Gamma_1} + \overrightarrow{\Gamma_2} \vdash \Pi x :^{r} \overline{A}.\overline{B} :^{1} s_3$.
  By IH, $\overrightarrow{\Gamma_1} \vdash \overline{A} :^{1} s_1$ and $\overrightarrow{\Gamma_2}, x :^{r_0} \overline{A} \vdash \overline{B} :^{1} s_2$.
  This case, then, follows by rule LDC-Pi.

- Rule GRAD-LAM. Have: $\Gamma \vdash \lambda^r x : A.b : \Pi x :^r A.B$ where $\Gamma, x :^r A \vdash b : B$.
  Need to show: $\vec{\Gamma} \vdash \lambda^r x : \overline{A}.\overline{b} :^1 \Pi x :^r \overline{A}.\overline{B}$.
  By IH, $\vec{\Gamma}, x :^r \overline{A} \vdash \overline{b} :^1 \overline{B}$.
  This case, then, follows by rule LDC-LAMOMEGA.
  (Note that rule LDC-LAMOMEGA has a side condition, $q = \omega \Rightarrow r = \omega$. But this condition is not important in the above application of the rule because $1 \neq \omega$ in case of all the preordered semirings in $\mathcal{Q}_{\mathbb{N}}^{\omega}$ barring $\mathbb{B}_{\geq}$. In case of $\mathbb{B}_{\geq}$, $1 = \omega$, but still the condition is not important since it can be ignored for $\mathbb{B}_{\geq}$, as pointed out in Section 5.5.2.)

- Rule GRAD-APP. Have: $\Gamma_1 + r \cdot \Gamma_2 \vdash b\ a^r : B\{a/x\}$ where $\Gamma_1 \vdash b : \Pi x :^r A.B$ and $\Gamma_2 \vdash a : A$.
  Need to show: $\vec{\Gamma_1} + r \cdot \vec{\Gamma_2} \vdash \overline{b}\ \overline{a}^r :^1 \overline{B}\{\overline{a}/x\}$.
  By IH, $\vec{\Gamma_1} \vdash \overline{b} :^1 \Pi x :^r \overline{A}.\overline{B}$ and $\vec{\Gamma_2} \vdash \overline{a} :^1 \overline{A}$.
  By the multiplication lemma, $r \cdot \vec{\Gamma_2} \vdash \overline{a} :^r \overline{A}$.
  This case, then, follows by rule LDC-APP.

- Rule GRAD-PAIR. Have: $r \cdot \Gamma_1 + \Gamma_2 \vdash (a_1{}^r, a_2) : \Sigma x :^r A_1.A_2$ where $\Gamma_1 \vdash a_1 : A_1$ and $\Gamma_2 \vdash a_2 : A_2\{a_1/x\}$.
  Need to show: $r \cdot \vec{\Gamma_1} + \vec{\Gamma_2} \vdash (\overline{a_1}^r, \overline{a_2}) :^1 \Sigma x :^r \overline{A_1}.\overline{A_2}$.
  By IH, $\vec{\Gamma_1} \vdash \overline{a_1} :^1 \overline{A_1}$ and $\vec{\Gamma_2} \vdash \overline{a_2} :^1 \overline{A_2}\{\overline{a_1}/x\}$.
  By the multiplication lemma, $r \cdot \vec{\Gamma_1} \vdash \overline{a_1} :^r \overline{A_1}$.
  This case, then, follows by rule LDC-PAIR.

- Rule GRAD-LETPAIR. Have: $\Gamma_1 + \Gamma_2 \vdash \mathbf{let}\ (x^r, y) = a\ \mathbf{in}\ b : B\{a/z\}$ where $\Gamma, z :^{r_0} \Sigma x :^r A_1.A_2 \vdash B : s$ and $\Gamma_1 \vdash a : \Sigma x :^r A_1.A_2$ and $\Gamma_2, x :^r A_1, y :^1 A_2 \vdash b : B\{(x^r, y)/z\}$ such that $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
  Need to show: $\vec{\Gamma_1} + \vec{\Gamma_2} \vdash \mathbf{let}_1\ (x^r, y) = \overline{a}\ \mathbf{in}\ \overline{b} :^1 \overline{B}\{\overline{a}/z\}$.
  By IH, $\vec{\Gamma}, z :^{r_0} \Sigma x :^r \overline{A_1}.\overline{A_2} \vdash \overline{B} :^1 s$.
  By the multiplication lemma, $0 \cdot \vec{\Gamma}, z :^0 \Sigma x :^r \overline{A_1}.\overline{A_2} \vdash \overline{B} :^0 s$.
  Again, by IH, $\vec{\Gamma_1} \vdash \overline{a} :^1 \Sigma x :^r \overline{A_1}.\overline{A_2}$ and $\vec{\Gamma_2}, x :^r \overline{A_1}, y :^1 \overline{A_2} \vdash \overline{b} :^1 \overline{B}\{(x^r, y)/z\}$.
  This case, then, follows by rule LDC-LETPAIR.

- Rule GRAD-CASE. Have: $q \cdot \Gamma_1 + \Gamma_2 \vdash \mathbf{case}_q\ a\ \mathbf{of}\ x_1.b_1\ ;\ x_2.b_2 : B\{a/z\}$ where $\Gamma, z :^r A_1 + A_2 \vdash B : s$ and $\Gamma_1 \vdash a : A_1 + A_2$ and $\Gamma_2, x_1 :^q A_1 \vdash b_1 : B\{\mathbf{inj}_1\ x_1/z\}$ and $\Gamma_2, x_2 :^q A_2 \vdash b_2 : B_2\{\mathbf{inj}_2\ x_2/z\}$ such that $q <: 1$ and $\lfloor \Gamma \rfloor = \lfloor \Gamma_1 \rfloor$.
  Need to show: $q \cdot \vec{\Gamma_1} + \vec{\Gamma_2} \vdash \mathbf{case}_q\ \overline{a}\ \mathbf{of}\ x_1.\overline{b_1}\ ;\ x_2.\overline{b_2} :^1 \overline{B}\{\overline{a}/z\}$.
  By IH, $\vec{\Gamma}, z :^r \overline{A_1} + \overline{A_2} \vdash \overline{B} :^1 s$.
  By the multiplication lemma, $0 \cdot \vec{\Gamma}, z :^0 \overline{A_1} + \overline{A_2} \vdash \overline{B} :^0 s$.
  Again, by IH, $\vec{\Gamma_1} \vdash \overline{a} :^1 \overline{A_1} + \overline{A_2}$.
  By the multiplication lemma, $q \cdot \vec{\Gamma_1} \vdash \overline{a} :^q \overline{A_1} + \overline{A_2}$.
  Yet again, by IH, $\vec{\Gamma_2}, x_1 :^q \overline{A_1} \vdash \overline{b_1} :^1 \overline{B}\{\mathbf{inj}_1\ x_1/z\}$ and $\vec{\Gamma_2}, x_2 :^q \overline{A_2} \vdash \overline{b_2} :^1 \overline{B}\{\mathbf{inj}_2\ x_2/z\}$.
  This case, then, follows by rule LDC-CASE.

- Rule GRAD-SUB. Have: $\Gamma \vdash a : A$ where $\Gamma' \vdash a : A$ and $\Gamma <: \Gamma'$.
  Need to show: $\vec{\Gamma} \vdash \overline{a} :^1 \overline{A}$.
  By IH, $\vec{\Gamma'} \vdash \overline{a} :^1 \overline{A}$.

Further, $\vec{\Gamma} <: \vec{\Gamma'}$.

This case, then, follows by rule LDC-SubL.

Next, by straightforward induction, we can show that if $\vdash a \rightsquigarrow a'$ in GRAD($\mathcal{Q}$), then $\vdash \overline{a} \rightsquigarrow \overline{a'}$ in LDC($\mathcal{Q}$). $\square$

Lemma D.65 (Multiplication) In LDC($\mathcal{L}$), if $\Gamma \vdash a :^{\ell} A$, then $m_0 \sqcup \Gamma \vdash a :^{m_0 \sqcup \ell} A$.

Proof. By induction on $\Gamma \vdash a :^{\ell} A$. The proof is similar to that of Lemma D.9. However, the case of rule LDC-LamOmega is somewhat different and as such, presented below.

- Rule LDC-LamOmega. Have: $\ell_0 \sqcup \Gamma \vdash \lambda^m x : A.b :^{\ell_0 \sqcup \ell} \Pi x :^m A.B$ where $\Gamma, x :^{\ell \sqcup m} A \vdash b :^{\ell} B$ and $\lfloor \Gamma \rfloor \vdash_{\top} \Pi x :^m A.B : s$ and $(\ell \sqcap \ell = \ell) \land (\ell \neq \top) \Rightarrow m \sqcap m = m$ and $\ell_0 \neq \top$. (Note that here we are using an alternative formulation of the constraint $q = \omega \Rightarrow r = \omega$, as justified in Section 5.5.1.)
  Need to show: $m_0 \sqcup \ell_0 \sqcup \Gamma \vdash \lambda^m x : A.b :^{m_0 \sqcup \ell_0 \sqcup \ell} \Pi x :^m A.B$.
  By IH, $m_0 \sqcup \ell_0 \sqcup \Gamma, x :^{m_0 \sqcup \ell_0 \sqcup \ell \sqcup m} A \vdash b :^{m_0 \sqcup \ell_0 \sqcup \ell} B$.
  This case, then, follows by rule LDC-LamOmega.

$\square$

Lemma D.66 (LDC and DDC$^{\top}$ (Lemma 5.41)) For any lattice $\mathcal{L}$, $\Gamma \vdash a :^{\ell} A$ in DDC$^{\top}$($\mathcal{L}$) if and only if $\Gamma \vdash a :^{\ell} A$ in LDC($\mathcal{L}$). Further, $\vdash a \rightsquigarrow a'$ in DDC$^{\top}$($\mathcal{L}$) if and only if $\vdash a \rightsquigarrow a'$ in LDC($\mathcal{L}$).

Proof. First, we show that if $\Gamma \vdash a :^{\ell} A$ in DDC$^{\top}$($\mathcal{L}$), then $\Gamma \vdash a :^{\ell} A$ in LDC($\mathcal{L}$).
By induction on $\Gamma \vdash a :^{\ell} A$. We present some of the interesting cases below.

- Rule DCT-Var. Have: $\Gamma, x :^{\ell_0} A \vdash x :^{\ell} A$ where $\Gamma \vdash A :^{\top} s$ and $\ell_0 \sqsubseteq \ell$.
  Need to show: $\Gamma, x :^{\ell_0} A \vdash x :^{\ell} A$ in LDC($\mathcal{L}$).
  By IH, $\Gamma \vdash A :^{\top} s$ in LDC($\mathcal{L}$).
  By multiplication lemma D.65, $\top \sqcup \Gamma \vdash A :^{\top} s$ in LDC($\mathcal{L}$), i.e., $\lfloor \Gamma \rfloor \vdash_{\top} A : s$.
  Then, by rule LDC-Var, $\top \sqcup \Gamma, x :^{\ell} A \vdash x :^{\ell} A$.
  This case, then, follows by rule LDC-SubL.
- Rule DCT-Pi. Have: $\Gamma \vdash \Pi x :^m A.B :^{\ell} s_3$ where $\Gamma \vdash A :^{\ell} s_1$ and $\Gamma, x :^{\ell} A \vdash B :^{\ell} s_2$ and $\mathcal{R}(s_1, s_2, s_3)$.
  Need to show: $\Gamma \vdash \Pi x :^m A.B :^{\ell} s_3$ in LDC($\mathcal{L}$).
  By IH, $\Gamma \vdash A :^{\ell} s_1$ and $\Gamma, x :^{\ell} A \vdash B :^{\ell} s_2$ in LDC($\mathcal{L}$).
  This case, then, follows by rule LDC-Pi.
- Rule DCT-Lam. Have: $\Gamma \vdash \lambda^m x : A.b :^{\ell} \Pi x :^m A.B$ where $\Gamma, x :^{\ell \sqcup m} A \vdash b :^{\ell} B$ and $\Gamma \vdash \Pi x :^m A.B :^{\top} s$.
  Need to show: $\Gamma \vdash \lambda^m x : A.b :^{\ell} \Pi x :^m A.B$ in LDC($\mathcal{L}$).
  By IH, $\Gamma, x :^{\ell \sqcup m} A \vdash b :^{\ell} B$ in LDC($\mathcal{L}$).
  Again, by IH, $\Gamma \vdash \Pi x :^m A.B :^{\top} s$ in LDC($\mathcal{L}$).
  By multiplication lemma D.65, $\top \sqcup \Gamma \vdash \Pi x :^m A.B :^{\top} s$, i.e., $\lfloor \Gamma \rfloor \vdash_{\top} \Pi x :^m A.B : s$.
  This case, then, follows by rule LDC-LamOmega.

- Rule DCT-App. Have: $\Gamma \vdash b\ a^m :^\ell B\{a/x\}$ where $\Gamma \vdash b :^\ell \Pi x :^m A.B$ and $\Gamma \vdash a :^{\ell \sqcup m} A$.
  Need to show: $\Gamma \vdash b\ a^m :^\ell B\{a/x\}$ in LDC($\mathcal{L}$).
  By IH, $\Gamma \vdash b :^\ell \Pi x :^m A.B$ and $\Gamma \vdash a :^{\ell \sqcup m} A$ in LDC($\mathcal{L}$).
  This case, then, follows by rule LDC-App.

Next, we show that if $\Gamma \vdash a :^\ell A$ in LDC($\mathcal{L}$), then $\Gamma \vdash a :^\ell A$ in DDC$^\top$($\mathcal{L}$).
By induction on $\Gamma \vdash a :^\ell A$. We present some of the interesting cases below.

- Rule LDC-Var. Have: $\top \sqcup \Gamma, x :^\ell A \vdash x :^\ell A$ where $\lfloor \Gamma \rfloor \vdash_\top A : s$.
  Need to show: $\top \sqcup \Gamma, x :^\ell A \vdash x :^\ell A$ in DDC$^\top$($\mathcal{L}$).
  By IH, $\top \sqcup \Gamma \vdash A :^\top s$ in DDC$^\top$($\mathcal{L}$).
  This case, then, follows by rule DCT-Var.

- Rule LDC-Pi. Have: $\Gamma_1 \sqcap \Gamma_2 \vdash \Pi x :^m A.B :^\ell s_3$ where $\Gamma_1 \vdash A_1 :^\ell s_1$ and $\Gamma_2, x :^{m_0} A \vdash B :^\ell s_2$ and $\mathcal{R}(s_1, s_2, s_3)$.
  Need to show: $\Gamma_1 \sqcap \Gamma_2 \vdash \Pi x :^m A.B :^\ell s_3$ in DDC$^\top$($\mathcal{L}$).
  By IH, $\Gamma_1 \vdash A :^\ell s_1$ in DDC$^\top$($\mathcal{L}$).
  Then, by narrowing lemma 3.31, $\Gamma_1 \sqcap \Gamma_2 \vdash A :^\ell s_1$ in DDC$^\top$($\mathcal{L}$).
  Again, by IH, $\Gamma_2, x :^{m_0} A \vdash B :^\ell s_2$ in DDC$^\top$($\mathcal{L}$).
  Then, by restricted upgrading lemma 3.32, $\Gamma_2, x :^{\ell \sqcup m_0} A \vdash B :^\ell s_2$ in DDC$^\top$($\mathcal{L}$).
  Next, by narrowing lemma 3.31, $\Gamma_1 \sqcap \Gamma_2, x :^\ell A \vdash B :^\ell s_2$ in DDC$^\top$($\mathcal{L}$).
  This case, then, follows by rule DCT-Pi.

- Rule LDC-LamOmega. Have: $\ell_0 \sqcup \Gamma \vdash \lambda^m x : A.b :^{\ell_0 \sqcup \ell} \Pi x :^m A.B$ where $\Gamma, x :^{\ell \sqcup m} A \vdash b :^\ell B$ and $\lfloor \Gamma \rfloor \vdash_\top \Pi x :^m A.B : s$.
  Need to show: $\ell_0 \sqcup \Gamma \vdash \lambda^m x : A.b :^{\ell_0 \sqcup \ell} \Pi x :^m A.B$ in DDC$^\top$($\mathcal{L}$).
  By IH, $\Gamma, x :^{\ell \sqcup m} A \vdash b :^\ell B$ in DDC$^\top$($\mathcal{L}$).
  Again, by IH, $\top \sqcup \Gamma \vdash \Pi x :^m A.B :^\top s$ in DDC$^\top$($\mathcal{L}$).
  Then, by narrowing lemma 3.31, $\Gamma \vdash \Pi x :^m A.B :^\top s$ in DDC$^\top$($\mathcal{L}$).
  Therefore, by rule DCT-Lam, $\Gamma \vdash \lambda^m x : A.b :^\ell \Pi x :^m A.B$ in DDC$^\top$($\mathcal{L}$).
  This case, then, follows by multiplication lemma 3.33.

- Rule LDC-App. Have: $\Gamma_1 \sqcap \Gamma_2 \vdash b\ a^m :^\ell B\{a/x\}$ where $\Gamma_1 \vdash b :^\ell \Pi x :^m A.B$ and $\Gamma_2 \vdash a :^{\ell \sqcup m} A$.
  Need to show: $\Gamma_1 \sqcap \Gamma_2 \vdash b\ a^m :^\ell B\{a/x\}$ in DDC$^\top$($\mathcal{L}$).
  By IH, $\Gamma_1 \vdash b :^\ell \Pi x :^m A.B$ and $\Gamma_2 \vdash a :^{\ell \sqcup m} A$ in DDC$^\top$($\mathcal{L}$).
  By narrowing lemma 3.31, $\Gamma_1 \sqcap \Gamma_2 \vdash b :^\ell \Pi x :^m A.B$ and $\Gamma_1 \sqcap \Gamma_2 \vdash a :^{\ell \sqcup m} A$ in DDC$^\top$($\mathcal{L}$).
  This case, then, follows by rule DCT-App.

- Rule LDC-SubL. Have: $\Gamma \vdash a :^\ell A$ where $\Gamma' \vdash a :^\ell A$ and $\Gamma \sqsubseteq \Gamma'$.
  Need to show: $\Gamma \vdash a :^\ell A$ in DDC$^\top$($\mathcal{L}$).
  By IH, $\Gamma' \vdash a :^\ell A$ in DDC$^\top$($\mathcal{L}$).
  This case, then, follows by narrowing lemma 3.31.

- Rule LDC-SubR. Have: $\Gamma \vdash a :^{\ell_2} A$ where $\Gamma \vdash a :^{\ell_1} A$ and $\ell_1 \sqsubseteq \ell_2$.
  Need to show: $\Gamma \vdash a :^{\ell_2} A$ in $\text{DDC}^{\top}(\mathcal{L})$.
  By IH, $\Gamma \vdash a :^{\ell_1} A$ in $\text{DDC}^{\top}(\mathcal{L})$.
  This case, then, follows by subsumption lemma 3.34.

Finally, by straightforward induction, we can show that $\vdash a \rightsquigarrow a'$ in $\text{DDC}^{\top}(\mathcal{L})$ if and only if $\vdash a \rightsquigarrow a'$ in $\text{LDC}(\mathcal{L})$. □

D.7 Proofs of Lemmas/Theorems Stated in Section 5.7

Lemma D.67 (Lemma 5.42) The translation from LNL λ-calculus to $\text{SLDC}(\mathcal{Q}_{\text{Lin}})$ is sound:

- If $\Theta; \Gamma \vdash_{\mathcal{L}} e : A$, then $\overline{\Theta}^{\omega}, \overline{\Gamma}^{1} \vdash \overline{e} :^{1} \overline{A}$.
- If $\Theta \vdash_{\mathcal{C}} s : X$, then $\overline{\Theta}^{\omega} \vdash \overline{s} :^{\omega} \overline{X}$.
- If $e =_{\beta} f$, then $\overline{e} =_{\beta} \overline{f}$. If $s =_{\beta} t$ then $\overline{s} =_{\beta} \overline{t}$.

Proof. To show the first two clauses, we do mutual induction on $\Theta; \Gamma \vdash_{\mathcal{L}} e : A$ and $\Theta \vdash_{\mathcal{C}} s : X$. But before moving on to the cases of the induction, we would like to put out a technical note: the contexts of LNL λ-calculus are multisets of assumptions whereas those of SLDC are lists, but this difference is not significant with regard to our proof below because SLDC enjoys the exchange property.

- Rule L-Var. Have: $\Theta; a : A \vdash_{\mathcal{L}} a : A$.
  Need to show: $\overline{\Theta}^{\omega}, a :^{1} \overline{A} \vdash a :^{1} \overline{A}$.
  By rule SLDC-Var, $\overline{\Theta}^{0}, a :^{1} \overline{A} \vdash a :^{1} \overline{A}$.
  Now, since $\omega <: 0$, so $\overline{\Theta}^{\omega} <: \overline{\Theta}^{0}$.
  This case, then, follows by rule SLDC-SubL.

- Rule NL-Var. Have: $\Theta, x : X \vdash_{\mathcal{C}} x : X$.
  Need to show: $\overline{\Theta}^{\omega}, x :^{\omega} \overline{X} \vdash x :^{\omega} \overline{X}$.
  By IH, $\overline{\Theta}^{0}, x :^{\omega} \overline{X} \vdash x :^{\omega} \overline{X}$.
  Now, since $\omega <: 0$, so $\overline{\Theta}^{\omega} <: \overline{\Theta}^{0}$.
  This case, then, follows by rule SLDC-SubL.

- Rule L-Unit. Have: $\Theta; \varnothing \vdash_{\mathcal{L}} * : I$.
  Need to show: $\overline{\Theta}^{\omega} \vdash \textbf{unit} :^{1} \textbf{Unit}$.
  By rule SLDC-Unit, $\overline{\Theta}^{0} \vdash \textbf{unit} :^{1} \textbf{Unit}$.
  This case, then, follows by rule SLDC-SubL.

- Rule NL-Unit. Have: $\Theta \vdash_{\mathcal{C}} () : 1$.
  Need to show: $\overline{\Theta}^{\omega} \vdash \mathbf{unit} :^{\omega} \mathbf{Unit}$.
  By rule SLDC-Unit, $\overline{\Theta}^{0} \vdash \mathbf{unit} :^{\omega} \mathbf{Unit}$.
  This case, then, follows by rule SLDC-SubL.

- Rule L-LetUnit. Have: $\Theta; \Gamma_1, \Gamma_2 \vdash_{\mathcal{L}} \mathbf{let} * = e \,\mathbf{in}\, f : B$ where $\Theta; \Gamma_1 \vdash_{\mathcal{L}} e : I$ and $\Theta; \Gamma_2 \vdash_{\mathcal{L}} f : B$.
  Need to show: $\overline{\Theta}^{\omega}, \overline{\Gamma_1}^{1}, \overline{\Gamma_2}^{1} \vdash \mathbf{let}_1\ \mathbf{unit} = \overline{e} \,\mathbf{in}\, \overline{f} :^{1} \overline{B}$.
  By IH, $\overline{\Theta}^{\omega}, \overline{\Gamma_1}^{1} \vdash \overline{e} :^{1} \mathbf{Unit}$ and $\overline{\Theta}^{\omega}, \overline{\Gamma_2}^{1} \vdash \overline{f} :^{1} \overline{B}$.
  By weakening, $\overline{\Theta}^{\omega}, \overline{\Gamma_1}^{1}, \overline{\Gamma_2}^{0} \vdash \overline{e} :^{1} \mathbf{Unit}$ and $\overline{\Theta}^{\omega}, \overline{\Gamma_1}^{0}, \overline{\Gamma_2}^{1} \vdash \overline{f} :^{1} \overline{B}$.
  This case, then, follows by rule SLDC-LetUnit.

- Rule L-Pair. Have: $\Theta; \Gamma_1, \Gamma_2 \vdash_{\mathcal{L}} e \otimes f : A \otimes B$ where $\Theta; \Gamma_1 \vdash_{\mathcal{L}} e : A$ and $\Theta; \Gamma_2 \vdash_{\mathcal{L}} f : B$.
  Need to show: $\overline{\Theta}^{\omega}, \overline{\Gamma_1}^{1}, \overline{\Gamma_2}^{1} \vdash (\overline{e}^{1}, \overline{f}) :^{1} {}^{1}\overline{A} \times \overline{B}$.
  By IH, $\overline{\Theta}^{\omega}, \overline{\Gamma_1}^{1} \vdash \overline{e} :^{1} \overline{A}$ and $\overline{\Theta}^{\omega}, \overline{\Gamma_2}^{1} \vdash \overline{f} :^{1} \overline{B}$.
  By weakening, $\overline{\Theta}^{\omega}, \overline{\Gamma_1}^{1}, \overline{\Gamma_2}^{0} \vdash \overline{e} :^{1} \overline{A}$ and $\overline{\Theta}^{\omega}, \overline{\Gamma_1}^{0}, \overline{\Gamma_2}^{1} \vdash \overline{f} :^{1} \overline{B}$.
  This case, then, follows by rule SLDC-Pair.

- Rule NL-Pair. Have: $\Theta \vdash_{\mathcal{C}} (s, t) : X \times Y$ where $\Theta \vdash s : X$ and $\Theta \vdash t : Y$.
  Need to show: $\overline{\Theta}^{\omega} \vdash (\overline{s}^{\omega}, \overline{t}) :^{\omega} {}^{\omega}\overline{X} \times \overline{Y}$.
  By IH, $\overline{\Theta}^{\omega} \vdash \overline{s} :^{\omega} \overline{X}$ and $\overline{\Theta}^{\omega} \vdash \overline{t} :^{\omega} \overline{Y}$.
  This case, then, follows by rule SLDC-Pair.

- Rule L-LetPair. Have: $\Theta; \Gamma_1, \Gamma_2 \vdash_{\mathcal{L}} \mathbf{let}\ a \otimes b = e \,\mathbf{in}\, f : C$ where $\Theta; \Gamma_1 \vdash_{\mathcal{L}} e : A \otimes B$ and $\Theta; \Gamma_2, a : A, b : B \vdash_{\mathcal{L}} f : C$.
  Need to show: $\overline{\Theta}^{\omega}, \overline{\Gamma_1}^{1}, \overline{\Gamma_2}^{1} \vdash \mathbf{let}_1\ (a^{1}, b) = \overline{e} \,\mathbf{in}\, \overline{f} :^{1} \overline{C}$.
  By IH, $\overline{\Theta}^{\omega}, \overline{\Gamma_1}^{1} \vdash \overline{e} :^{1} {}^{1}\overline{A} \times \overline{B}$ and $\overline{\Theta}^{\omega}, \overline{\Gamma_2}^{1}, a :^{1} \overline{A}, b :^{1} \overline{B} \vdash \overline{f} :^{1} \overline{C}$.
  By weakening, $\overline{\Theta}^{\omega}, \overline{\Gamma_1}^{1}, \overline{\Gamma_2}^{0} \vdash \overline{e} :^{1} {}^{1}\overline{A} \times \overline{B}$ and $\overline{\Theta}^{\omega}, \overline{\Gamma_1}^{0}, \overline{\Gamma_2}^{1}, a :^{1} \overline{A}, b :^{1} \overline{B} \vdash \overline{f} :^{1} \overline{C}$.
  This case, then, follows by rule SLDC-LetPair.

- Rule NL-Fst. Have: $\Theta \vdash_{\mathcal{C}} \mathbf{fst}(s) : X$ where $\Theta \vdash_{\mathcal{C}} s : X \times Y$.
  Need to show: $\overline{\Theta}^{\omega} \vdash \mathbf{let}_1\ (x^{\omega}, y) = \overline{s} \,\mathbf{in}\, x :^{\omega} \overline{X}$, where x and y are fresh.
  By IH, $\overline{\Theta}^{\omega} \vdash \overline{s} :^{\omega} {}^{\omega}\overline{X} \times \overline{Y}$.
  By rule SLDC-Var, $\overline{\Theta}^{0}, x :^{\omega} \overline{X}, y :^{0} \overline{Y} \vdash x :^{\omega} \overline{X}$.
  Next, by rule SLDC-SubL, $\overline{\Theta}^{\omega}, x :^{\omega} \overline{X}, y :^{\omega} \overline{Y} \vdash x :^{\omega} \overline{X}$.
  This case, then, follows by rule SLDC-LetPair.

- Rule NL-Snd. Have: $\Theta \vdash_{\mathcal{C}} \mathbf{snd}(s) : Y$ where $\Theta \vdash_{\mathcal{C}} s : X \times Y$.
  Need to show: $\overline{\Theta}^{\omega} \vdash \mathbf{let}_1\ (x^{\omega}, y) = \overline{s} \,\mathbf{in}\, y :^{\omega} \overline{Y}$, where x and y are fresh.
  By IH, $\overline{\Theta}^{\omega} \vdash \overline{s} :^{\omega} {}^{\omega}\overline{X} \times \overline{Y}$.
  By rule SLDC-Var, $\overline{\Theta}^{0}, x :^{0} \overline{X}, y :^{\omega} \overline{Y} \vdash y :^{\omega} \overline{Y}$.
  Next, by rule SLDC-SubL, $\overline{\Theta}^{\omega}, x :^{\omega} \overline{X}, y :^{\omega} \overline{Y} \vdash y :^{\omega} \overline{Y}$.
  This case, then, follows by rule SLDC-LetPair.

- Rule L-Lam. Have: $\Theta; \Gamma \vdash_{\mathcal{L}} \lambda a : A.e : A \multimap B$ where $\Theta; \Gamma, a : A \vdash_{\mathcal{L}} e : B$.
  Need to show: $\overline{\Theta}^{\omega}, \overline{\Gamma}^{1} \vdash \lambda^{1} a : \overline{A}.\overline{e} :^{1} {}^{1}\overline{A} \to \overline{B}$.
  By IH, $\overline{\Theta}^{\omega}, a :^{1} \overline{A} \vdash \overline{e} :^{1} \overline{B}$.
  This case, then, follows by rule SLDC-LamOmega.

- Rule NL-Lam. Have: $\Theta \vdash_{\mathcal{C}} \lambda x : X.s : X \to Y$ where $\Theta, x : X \vdash_{\mathcal{C}} s : Y$.
  Need to show: $\overline{\Theta}^{\omega} \vdash \lambda^{\omega} x : \overline{X}.\overline{s} :^{\omega} {}^{\omega}\overline{X} \to \overline{Y}$.
  By IH, $\overline{\Theta}^{\omega}, x :^{\omega} \overline{X} \vdash \overline{s} :^{\omega} \overline{Y}$.
  This case, then, follows by rule SLDC-LamOmega. (Observe that the constraint $q = \omega \Rightarrow r = \omega$ is satisfied!)

- Rule L-App. Have: $\Theta; \Gamma_1, \Gamma_2 \vdash_{\mathcal{L}} e\, f : B$ where $\Theta; \Gamma_1 \vdash_{\mathcal{L}} e : A \multimap B$ and $\Theta; \Gamma_2 \vdash_{\mathcal{L}} f : A$.
  Need to show: $\overline{\Theta}^{\omega}, \overline{\Gamma_1}^{1}, \overline{\Gamma_2}^{1} \vdash \overline{e}\, \overline{f}^{1} :^{1} \overline{B}$.
  By IH, $\overline{\Theta}^{\omega}, \overline{\Gamma_1}^{1} \vdash e :^{1} {}^{1}\overline{A} \to \overline{B}$ and $\overline{\Theta}^{\omega}, \overline{\Gamma_2}^{1} \vdash \overline{f} :^{1} \overline{A}$.
  By weakening, $\overline{\Theta}^{\omega}, \overline{\Gamma_1}^{1}, \overline{\Gamma_2}^{0} \vdash \overline{e} :^{1} {}^{1}\overline{A} \to \overline{B}$ and $\overline{\Theta}^{\omega}, \overline{\Gamma_1}^{0}, \overline{\Gamma_2}^{1} \vdash \overline{f} :^{1} \overline{A}$.
  This case, then, follows by rule SLDC-App.

- Rule NL-App. Have: $\Theta \vdash_{\mathcal{C}} s\, t : Y$ where $\Theta \vdash_{\mathcal{C}} s : X \to Y$ and $\Theta \vdash_{\mathcal{C}} t : X$.
  Need to show: $\overline{\Theta}^{\omega} \vdash \overline{s}\, \overline{t}^{\omega} :^{\omega} \overline{Y}$.
  By IH, $\overline{\Theta}^{\omega} \vdash \overline{s} :^{\omega} {}^{\omega}\overline{X} \to \overline{Y}$ and $\overline{\Theta}^{\omega} \vdash \overline{t} :^{\omega} \overline{X}$.
  This case, then, follows by rule SLDC-App.

- Rule L-FIntro. Have: $\Theta; \varnothing \vdash_{\mathcal{L}} F\, s : F\, X$ where $\Theta \vdash_{\mathcal{C}} s : X$.
  Need to show: $\overline{\Theta}^{\omega} \vdash (\overline{s}^{\omega}, \mathbf{unit}) :^{1} {}^{\omega}\overline{X} \times \mathbf{Unit}$.
  By IH, $\overline{\Theta}^{\omega} \vdash \overline{s} :^{\omega} \overline{X}$.
  Next, by rule SLDC-Unit, $\overline{\Theta}^{0} \vdash \mathbf{unit} :^{1} \mathbf{Unit}$.
  This case, then, follows by rule SLDC-Pair.

- Rule L-FElim. Have: $\Theta; \Gamma_1, \Gamma_2 \vdash_{\mathcal{L}} \mathbf{let}\, F(x) = e\, \mathbf{in}\, f : B$ where $\Theta; \Gamma_1 \vdash_{\mathcal{L}} e : F\, X$ and $\Theta, x : X; \Gamma_2 \vdash_{\mathcal{L}} f : B$.
  Need to show: $\overline{\Theta}^{\omega}, \overline{\Gamma_1}^{1}, \overline{\Gamma_2}^{1} \vdash \mathbf{let}_1\, (x^{\omega}, y) = \overline{e}\, \mathbf{in}\, \mathbf{let}_1\, \mathbf{unit} = y\, \mathbf{in}\, \overline{f} :^{1} \overline{B}$, where y is fresh.
  By IH, $\overline{\Theta}^{\omega}, \overline{\Gamma_1}^{1} \vdash \overline{e} :^{1} {}^{\omega}\overline{X} \times \mathbf{Unit}$.
  By weakening, $\overline{\Theta}^{\omega}, \overline{\Gamma_1}^{1}, \overline{\Gamma_2}^{0} \vdash \overline{e} :^{1} {}^{\omega}\overline{X} \times \mathbf{Unit}$.

  Again, by IH, $\overline{\Theta}^{\omega}, x :^{\omega} \overline{X}, \overline{\Gamma_2}^{1} \vdash \overline{f} :^{1} \overline{B}$.
  By exchange, $\overline{\Theta}^{\omega}, \overline{\Gamma_2}^{1}, x :^{\omega} \overline{X} \vdash \overline{f} :^{1} \overline{B}$.
  By weakening, $\overline{\Theta}^{\omega}, \overline{\Gamma_1}^{0}, \overline{\Gamma_2}^{1}, x :^{\omega} \overline{X}, y :^{0} \mathbf{Unit} \vdash \overline{f} :^{1} \overline{B}$.
  Next, by rule SLDC-Var, $\overline{\Theta}^{0}, \overline{\Gamma_1}^{0}, \overline{\Gamma_2}^{0}, x :^{0} \overline{X}, y :^{1} \mathbf{Unit} \vdash y :^{1} \mathbf{Unit}$.
  Then, by rule SLDC-LetUnit, $\overline{\Theta}^{\omega}, \overline{\Gamma_1}^{0}, \overline{\Gamma_2}^{1}, x :^{\omega} \overline{X}, y :^{1} \mathbf{Unit} \vdash \mathbf{let}_1\, \mathbf{unit} = y\, \mathbf{in}\, \overline{f} :^{1} \overline{B}$.
  This case, then, follows by rule SLDC-LetPair.

- Rule NL-GIntro. Have: $\Theta \vdash_{\mathcal{C}} G\, e : G\, A$ where $\Theta; \varnothing \vdash_{\mathcal{L}} e : A$.
  Need to show: $\overline{\Theta}^{\omega} \vdash \overline{e} :^{\omega} \overline{A}$.

By IH, $\overline{\Theta}^{\omega} \vdash \overline{e} :^1 \overline{A}$.

This case, then, follows by the multiplication lemma.

- Rule L-GElim. Have: $\Theta; \varnothing \vdash_{\mathcal{L}} \textbf{derelict}\, s : A$ where $\Theta \vdash_{\mathcal{C}} s : G\, A$.
  Need to show: $\overline{\Theta}^{\omega} \vdash \overline{s} :^1 \overline{A}$.
  By IH, $\overline{\Theta}^{\omega} \vdash \overline{s} :^{\omega} \overline{A}$.
  This case, then, follows by rule SLDC-SubR.

Next, to show the third clause of the lemma, we do case analysis on the β-rules of the LNL λ-calculus.

- Rule Beta-LetUnitL. Have: $\textbf{let}\, * = * \,\textbf{in}\, f \to_{\beta} f$.
  Need to show: $\textbf{let}_1\, \textbf{unit} = \textbf{unit}\, \textbf{in}\, \overline{f} =_{\beta} \overline{f}$.
  Follows by rule Step-LetUnitBeta.

- Rule Beta-LetPairL. Have: $\textbf{let}\, a_1 \otimes a_2 = e_1 \otimes e_2\, \textbf{in}\, f \to_{\beta} f\{e_1/a_1\}\{e_2/a_2\}$.
  Need to show: $\textbf{let}_1\, (a_1^1, a_2) = (\overline{e_1}^1, \overline{e_2})\, \textbf{in}\, \overline{f} =_{\beta} \overline{f}\{\overline{e_1}/a_1\}\{\overline{e_2}/a_2\}$.
  Follows by rule Step-LetPairBeta.

- Rule Beta-Fst. Have: $\textbf{fst}((s,t)) \to_{\beta} s$.
  Need to show: $\textbf{let}_1\, (x^{\omega}, y) = (\overline{s}^{\omega}, \overline{t})\, \textbf{in}\, x =_{\beta} \overline{s}$.
  Follows by rule Step-LetPairBeta.

- Rule Beta-Snd. Have: $\textbf{snd}((s,t)) \to_{\beta} t$.
  Need to show: $\textbf{let}_1\, (x^{\omega}, y) = (\overline{s}^{\omega}, \overline{t})\, \textbf{in}\, y =_{\beta} \overline{t}$.
  Follows by rule Step-LetPairBeta.

- Rule Beta-AppL. Have: $(\lambda a : A.e)\, f \to_{\beta} e\{f/a\}$.
  Need to show: $(\lambda^1 a : \overline{A}.\overline{e})\, \overline{f}^1 =_{\beta} \overline{e}\{\overline{f}/a\}$.
  Follows by rule Step-AppBeta.

- Rule Beta-AppNL. Have: $(\lambda x : X.s)\, t \to_{\beta} s\{t/x\}$.
  Need to show: $(\lambda^{\omega} x : \overline{X}.\overline{s})\, \overline{t}^{\omega} =_{\beta} \overline{s}\{\overline{t}/x\}$.
  Follows by rule Step-AppBeta.

- Rule Beta-F. Have: $\textbf{let}\, F(x) = F\, s\, \textbf{in}\, e \to_{\beta} e\{s/x\}$.
  Need to show: $\textbf{let}_1\, (x^{\omega}, y) = (\overline{s}^{\omega}, \textbf{unit})\, \textbf{in}\, \textbf{let}_1\, \textbf{unit} = y\, \textbf{in}\, \overline{e} =_{\beta} \overline{e}\{\overline{s}/x\}$ (y fresh).
  Now, $\vdash \textbf{let}_1\, (x^{\omega}, y) = (\overline{s}^{\omega}, \textbf{unit})\, \textbf{in}\, \textbf{let}_1\, \textbf{unit} = y\, \textbf{in}\, \overline{e} \leadsto \textbf{let}_1\, \textbf{unit} = \textbf{unit}\, \textbf{in}\, \overline{e}\{\overline{s}/x\}$, by rule Step-LetPairBeta.
  This case, then, follows by rule Step-LetUnitBeta.

- Rule Beta-G. Have: $\textbf{derelict}\, (G\, e) \to_{\beta} e$.
  Need to show: $\overline{e} =_{\beta} \overline{e}$.
  Follows by reflexivity.

□

Bibliography

Martín Abadi. 2006. Access Control in a Core Calculus of Dependency. In *Proceedings of the Eleventh ACM SIGPLAN International Conference on Functional Programming* (Portland, Oregon, USA) *(ICFP '06)*. Association for Computing Machinery, New York, NY, USA, 263–273. https://doi.org/10.1145/1159803.1159839

Martín Abadi, Anindya Banerjee, Nevin Heintze, and Jon G. Riecke. 1999. A Core Calculus of Dependency. In *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Antonio, Texas, USA) *(POPL '99)*. Association for Computing Machinery, New York, NY, USA, 147–160. https://doi.org/10.1145/292540.292555

Martín Abadi, Butler Lampson, and Jean-Jacques Lévy. 1996. Analysis and Caching of Dependencies. In *Proceedings of the First ACM SIGPLAN International Conference on Functional Programming* (Philadelphia, Pennsylvania, USA) *(ICFP '96)*. Association for Computing Machinery, New York, NY, USA, 83–91. https://doi.org/10.1145/232627.232638

Edwin A. Abbott. 2008. *Flatland*. Oxford University Press, London, UK.

Andreas Abel and Jean-Philippe Bernardy. 2020. A Unified View of Modalities in Type Systems. *Proc. ACM Program. Lang.* 4, ICFP, Article 90 (Aug. 2020), 28 pages. https://doi.org/10.1145/3408972

Andreas Abel and Gabriel Scherer. 2012. On Irrelevance and Algorithmic Equality in Predicative Type Theory. *Logical Methods in Computer Science* 8, 1 (mar 2012). https://doi.org/10.2168/lmcs-8(1:29)2012

Samson Abramsky. 1993. Computational interpretations of linear logic. *Theoretical Computer Science* 111, 1 (1993), 3–57. https://doi.org/10.1016/0304-3975(93)90181-R

The Agda Team. 2021. *Agda Standard Library*. Retrieved October 23, 2022 from https://agda.github.io/agda-stdlib/Data.Fin.Base.html

Danel Ahman, Neil Ghani, and Gordon D. Plotkin. 2016. Dependent Types and Fibred Computational Effects. In *Foundations of Software Science and Computation Structures*, Bart Jacobs and Christof Löding (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 36–54.

Maximilian Algehed. 2018. A Perspective on the Dependency Core Calculus. In *Proceedings of the 13th Workshop on Programming Languages and Analysis for Security* (Toronto, Canada) *(PLAS '18)*. Association for Computing Machinery, New York, NY, USA, 24–28. https://doi.org/10.1145/3264820.3264823

Maximilian Algehed and Jean-Philippe Bernardy. 2019. Simple Noninterference from Parametricity. *Proc. ACM Program. Lang.* 3, ICFP, Article 89 (July 2019), 22 pages. https://doi.org/10.1145/3341693

Robert Atkey. 2018. Syntax and Semantics of Quantitative Type Theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science* (Oxford, United Kingdom) *(LICS '18)*. Association for Computing Machinery, New York, NY, USA, 56–65. https://doi.org/10.1145/3209108.3209189

Andrew Barber. 1996. *Dual Intuitionistic Linear Logic*. Technical Report. Edinburgh, Scotland.

Hendrik Pieter Barendregt. 1984. *The Lambda Calculus: Its Syntax and Semantics*. Elsevier.

H. P. Barendregt. 1993. *Lambda Calculi with Types*. Oxford University Press, Inc., USA, 117–309.

Bruno Barras and Bruno Bernardo. 2008. The Implicit Calculus of Constructions as a Programming Language with Dependent Types. In *Foundations of Software Science and Computational Structures*, Roberto Amadio (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 365–379.

N. Benton and P. Wadler. 1996. Linear logic, monads and the lambda calculus. In *Proceedings 11th Annual IEEE Symposium on Logic in Computer Science*. 420–431. https://doi.org/10.1109/LICS.1996.561458

P. N. Benton. 1994. A Mixed Linear and Non-Linear Logic: Proofs, Terms and Models (Extended Abstract). In *Selected Papers from the 8th International Workshop on Computer Science Logic (CSL '94)*. Springer-Verlag, Berlin, Heidelberg, 121–135.

P. N. Benton, Gavin M. Bierman, Valeria de Paiva, and Martin Hyland. 1993. A Term Calculus for Intuitionistic Linear Logic. In *Proceedings of the International Conference on Typed Lambda Calculi and Applications (TLCA '93)*. Springer-Verlag, Berlin, Heidelberg, 75–90.

Jean-Philippe Bernardy, Mathieu Boespflug, Ryan R. Newton, Simon Peyton Jones, and Arnaud Spiwack. 2018. Linear Haskell: practical linearity in a higher-order polymorphic language. In *Principles of Programming Languages 2018 (POPL 2018)*.

Jean-Philippe Bernardy and Moulin Guilhem. 2013. Type-Theory in Color. *SIGPLAN Not.* 48, 9 (Sept. 2013), 61–72. https://doi.org/10.1145/2544174.2500577

G. Birkhoff. 1967. *Lattice Theory* (3rd ed.). American Mathematical Society, Providence.

William J. Bowman and Amal Ahmed. 2015. Noninterference for Free. *SIGPLAN Not.* 50, 9 (Aug. 2015), 101–113. https://doi.org/10.1145/2858949.2784733

Edwin C. Brady. 2021. Idris 2: Quantitative Type Theory in Practice. In *35th European Conference on Object-Oriented Programming, ECOOP 2021, July 11-17, 2021, Aarhus, Denmark (Virtual Conference) (LIPIcs, Vol. 194)*, Anders Møller and Manu Sridharan (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 9:1–9:26. https://doi.org/10.4230/LIPIcs.ECOOP.2021.9

Joachim Breitner, Richard A. Eisenberg, Simon Peyton Jones, and Stephanie Weirich. 2014. Safe Zero-Cost Coercions for Haskell. *SIGPLAN Not.* 49, 9 (aug 2014), 189–202. https://doi.org/10.1145/2692915.2628141

Stephen Brookes and Shai Geva. 1992. Computational Comonads and Intensional Semantics. In *Applications of Categories in Computer Science (Proceedings of the LMS Symposium, Vol. 177)*. Cambridge University Press.

Aloïs Brunel, Marco Gaboardi, Damiano Mazza, and Steve Zdancewic. 2014. A Core Quantitative Coeffect Calculus. In *Proceedings of the 23rd European Symposium on Programming Languages and Systems - Volume 8410*. Springer-Verlag, Berlin, Heidelberg, 351–370. https://doi.org/10.1007/978-3-642-54833-8_19

Luís Caires and Frank Pfenning. 2010. Session Types as Intuitionistic Linear Propositions. In *CONCUR 2010 - Concurrency Theory*, Paul Gastin and François Laroussinie (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 222–236.

Cristiano Calcagno, Walid Taha, Liwen Huang, and Xavier Leroy. 2003. Implementing Multi-stage Languages Using ASTs, Gensym, and Reflection. In *Generative Programming and Component Engineering*, Frank Pfenning and Yannis Smaragdakis (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 57–76.

Iliano Cervesato and Frank Pfenning. 2002. A Linear Logical Framework. *Information and Computation* 179, 1 (2002), 19–75. https://doi.org/10.1006/inco.2001.2951

Jawahar Chirimar, Carl Gunter, and Jon Riecke. 2000. Reference Counting as a Computational Interpretation of Linear Logic. *Journal of Functional Programming* 6 (03 2000). https://doi.org/10.1017/S0956796800001660

Pritam Choudhury. 2022a. Monadic and Comonadic Aspects of Dependency Analysis. *Proc. ACM Program. Lang.* 6, OOPSLA2, Article 172 (Oct. 2022), 29 pages. https://doi.org/10.1145/3563335

Pritam Choudhury. 2022b. Monadic and Comonadic Aspects of Dependency Analysis. https://doi.org/10.48550/ARXIV.2209.06334

Pritam Choudhury. 2023. Unifying Linearity and Dependency Analyses. arXiv:2304.03175 [cs.PL]

Pritam Choudhury, Harley Eades, Richard A. Eisenberg, and Stephanie C Weirich. 2020a. A graded dependent type system with a usage-aware semantics (extended version). https://doi.org/10.48550/ARXIV.2011.04070

Pritam Choudhury, Harley Eades, and Stephanie Weirich. 2022a. A Dependent Dependency Calculus. In *Programming Languages and Systems*, Ilya Sergey (Ed.). Springer International Publishing, Cham, 403–430.

Pritam Choudhury, Harley Eades, and Stephanie Weirich. 2022b. A Dependent Dependency Calculus (Extended Version). https://doi.org/10.48550/ARXIV.2201.11040

Pritam Choudhury, Harley Eades III, Richard A. Eisenberg, and Stephanie Weirich. 2021. A Graded Dependent Type System with a Usage-Aware Semantics. *Proc. ACM Program. Lang.* 5, POPL, Article 50 (Jan. 2021), 32 pages. https://doi.org/10.1145/3434331

Pritam Choudhury, Harley Eades III, Richard A. Eisenberg, and Stephanie Weirich. 2020b. *Artifact for "A Graded Dependent Type System with a Usage-Aware Semantics"*. https://doi.org/10.1145/3410265

Pritam Choudhury, Harley Eades III, and Stephanie Weirich. 2022c. *Artifact associated with "A Dependent Dependency Calculus"*. https://doi.org/10.5281/zenodo.5903727

E. B. Cowell (Ed.). 1895–1907. *The Jātaka or Stories of the Buddha's Former Births. 6 vols.* Vol. 2. Cambridge University Press.

Karl Crary. 2004. Logical Relations and a Case Study in Equivalence Checking. In *Advanced Topics in Types and Programming Languages*, Benjamin C. Pierce (Ed.). The MIT Press, Massachusetts, USA, Chapter 6.

Rowan Davies. 2017. A Temporal Logic Approach to Binding-Time Analysis. *J. ACM* 64, 1, Article 1 (mar 2017), 45 pages. https://doi.org/10.1145/3011069

Dorothy E. Denning. 1976. A Lattice Model of Secure Information Flow. *Commun. ACM* 19, 5 (May 1976), 236–243. https://doi.org/10.1145/360051.360056

Samuel Eilenberg and G. Max Kelly. 1966. Closed Categories. In *Proceedings of the Conference on Categorical Algebra*, S. Eilenberg, D. K. Harrison, S. MacLane, and H. Röhrl (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 421–562.

Richard A. Eisenberg, Guillaume Duboc, Stephanie Weirich, and Daniel Lee. 2021. An Existential Crisis Resolved: Type Inference for First-Class Existential Types. *Proc. ACM Program. Lang.* 5, ICFP, Article 64 (aug 2021), 29 pages. https://doi.org/10.1145/3473569

Matthew Fluet, Greg Morrisett, and Amal Ahmed. 2006. Linear Regions Are All You Need. In *Proceedings of the 15th European Conference on Programming Languages and Systems* (Vienna, Austria) *(ESOP'06)*. Springer-Verlag, Berlin, Heidelberg, 7–21. https://doi.org/10.1007/11693024_2

Soichiro Fujii. 2019. A 2-Categorical Study of Graded and Indexed Monads. arXiv:1904.08083 [math.CT]

Marco Gaboardi, Shin-ya Katsumata, Dominic Orchard, Flavien Breuvart, and Tarmo Uustalu. 2016. Combining Effects and Coeffects via Grading. *SIGPLAN Not.* 51, 9 (sep 2016), 476–489. https://doi.org/10.1145/3022670.2951939

Dan R. Ghica and Alex I. Smith. 2014. Bounded Linear Types in a Resource Semiring. In *Programming Languages and Systems*, Zhong Shao (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 331–350.

David K. Gifford and John M. Lucassen. 1986. Integrating Functional and Imperative Programming. In *Proceedings of the 1986 ACM Conference on LISP and Functional Programming* (Cambridge, Massachusetts, USA) *(LFP '86)*. Association for Computing Machinery, New York, NY, USA, 28–38. https://doi.org/10.1145/319838.319848

Jean-Yves Girard. 1987. Linear logic. *Theoretical Computer Science* 50, 1 (1987), 1–101. https://doi.org/10.1016/0304-3975(87)90045-4

Jean-Yves Girard. 1993. On the unity of logic. *Annals of Pure and Applied Logic* 59, 3 (1993), 201–217. https://doi.org/10.1016/0168-0072(93)90093-S

Jean-Yves Girard. 1995. Linear Logic: Its Syntax and Semantics. In *Proceedings of the Workshop on Advances in Linear Logic*. Cambridge University Press, USA, 1–42.

Jean-Yves Girard, Andre Scedrov, and Philip J. Scott. 1992. Bounded linear logic: a modular approach to polynomial-time computability. *Theoretical Computer Science* 97, 1 (1992), 1–66. https://doi.org/10.1016/0304-3975(92)90386-T

J. A. Goguen and J. Meseguer. 1982. Security Policies and Security Models. In *1982 IEEE Symposium on Security and Privacy*. 11–11.

Jonathan S. Golan. 1999. *Semirings and their Applications*. Springer Netherlands. https://doi.org/10.1007/978-94-015-9333-5

Carsten K. Gomard and Neil D. Jones. 1991. A partial evaluator for the untyped lambda-calculus. *Journal of Functional Programming* 1, 1 (1991), 21–69. https://doi.org/10.1017/S0956796800000058

Robert Harper, Furio Honsell, and Gordon Plotkin. 1993. A Framework for Defining Logics. *J. ACM* 40, 1 (jan 1993), 143–184. https://doi.org/10.1145/138027.138060

John Hatcliff and Olivier Danvy. 1997. A computational formalization for partial evaluation. *Mathematical Structures in Computer Science* 7, 5 (1997), 507–541. https://doi.org/10.1017/S0960129597002405

Nevin Heintze and Jon G. Riecke. 1998. The SLam Calculus: Programming with Secrecy and Integrity. In *Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Diego, California, USA) *(POPL '98)*. Association for Computing Machinery, New York, NY, USA, 365–377. https://doi.org/10.1145/268946.268976

Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. 1998. Language primitives and type discipline for structured communication-based programming. In *Programming Languages and Systems*, Chris Hankin (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 122–138.

Bart Jacobs. 1999. *Categorical Logic and Type Theory*. Elsevier, Amsterdam, The Netherlands.

Peter T. Johnstone. 2002. *Sketches of an Elephant*. Vol. 1. Oxford University Press, Inc.

Anita K. Jones and Richard J. Lipton. 1975. The Enforcement of Security Policies for Computation. *SIGOPS Oper. Syst. Rev.* 9, 5 (nov 1975), 197–206. https://doi.org/10.1145/1067629.806538

Neil D. Jones, Carsten K. Gomard, and Peter Sestoft. 1993. Partial evaluation and automatic program generation. In *Prentice Hall international series in computer science*.

Shin-ya Katsumata. 2014. Parametric Effect Monads and Semantics of Effect Systems. *SIGPLAN Not.* 49, 1 (jan 2014), 633–645. https://doi.org/10.1145/2578855.2535846

Shin-ya Katsumata. 2018. A Double Category Theoretic Analysis of Graded Linear Exponential Comonads. In *Foundations of Software Science and Computation Structures*, Christel Baier and Ugo Dal Lago (Eds.). Springer International Publishing, Cham, 110–127.

G. A. Kavvos. 2019. Modalities, Cohesion, and Information Flow. *Proc. ACM Program. Lang.* 3, POPL, Article 20 (jan 2019), 29 pages. https://doi.org/10.1145/3290333

Anders Kock. 1970. Monads on symmetric monoidal closed categories. 21 (1970), 1–10. https://doi.org/10.1007/BF01220868

Anders Kock. 1972. Strong Functors and Monoidal Monads. 23 (1972), 113–120. https://doi.org/10.1007/BF01304852

Neelakantan R. Krishnaswami, Pierre Pradic, and Nick Benton. 2015. Integrating Linear and Dependent Types. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Mumbai, India) *(POPL '15)*. Association for Computing Machinery, New York, NY, USA, 17–30. https://doi.org/10.1145/2676726.2676969

Y. Lafont. 1988. The linear abstract machine. *Theoretical Computer Science* 59, 1 (1988), 157–180. https://doi.org/10.1016/0304-3975(88)90100-4

Paul Blain Levy. 2003. *Call-by-Push-Value: A Functional-Imperative Synthesis*. Springer. https://doi.org/10.1007/978-94-007-0954-6

P. Lincoln and J. Mitchell. 1992. Operational aspects of linear lambda calculus. In *1992 Proceedings of the Seventh Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society, Los Alamitos, CA, USA, 235,236,237,238,239,240,241,242,243,244,245,246. https://doi.org/10.1109/LICS.1992.185536

Luísa Lourenço and Luís Caires. 2015. Dependent Information Flow Types. *SIGPLAN Not.* 50, 1 (Jan. 2015), 317–328. https://doi.org/10.1145/2775051.2676994

J. M. Lucassen and D. K. Gifford. 1988. Polymorphic Effect Systems. In *Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Diego, California, USA) *(POPL '88)*. Association for Computing Machinery, New York, NY, USA, 47–57. https://doi.org/10.1145/73560.73564

Saunders MacLane. 1971. *Categories for the Working Mathematician*. Springer-Verlag, New York. ix+262 pages. Graduate Texts in Mathematics, Vol. 5.

Daniel Marino and Todd Millstein. 2009. A Generic Type-and-Effect System. In *Proceedings of the 4th International Workshop on Types in Language Design and Implementation* (Savannah, GA, USA) *(TLDI '09)*. Association for Computing Machinery, New York, NY, USA, 39–50. https://doi.org/10.1145/1481861.1481868

Daniel Marshall, Michael Vollmer, and Dominic Orchard. 2022. Linearity and Uniqueness: An Entente Cordiale. In *Programming Languages and Systems*, Ilya Sergey (Ed.). Springer International Publishing, Cham, 346–375.

Per Martin-Löf. 1982. Constructive Mathematics and Computer Programming. In *Logic, Methodology and Philosophy of Science VI*, L. Jonathan Cohen, Jerzy Łoś, Helmut Pfeiffer, and Klaus-Peter Podewski (Eds.). Studies in Logic and the Foundations of Mathematics, Vol. 104. Elsevier, 153–175. https://doi.org/10.1016/S0049-237X(09)70189-2

Conor McBride. 2016. *I Got Plenty o' Nuttin'*. Springer International Publishing, Cham, 207–233. https://doi.org/10.1007/978-3-319-30936-1_12

Robin Milner, Joachim Parrow, and David Walker. 1992. A calculus of mobile processes, I. *Information and Computation* 100, 1 (1992), 1–40. https://doi.org/10.1016/0890-5401(92)90008-4

Alexandre Miquel. 2001. The Implicit Calculus of Constructions Extending Pure Type Systems with an Intersection Type Binder and Subtyping. In *Typed Lambda Calculi and Applications*, Samson Abramsky (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 344–359.

Nathan Mishra-Linger and Tim Sheard. 2008. Erasure and Polymorphism in Pure Type Systems. In *Proceedings of the Theory and Practice of Software, 11th International Conference on Foundations of Software Science and Computational Structures* (Budapest, Hungary) *(FOSSACS'08/ETAPS'08)*. Springer-Verlag, Berlin, Heidelberg, 350–364.

Richard Nathan Mishra-Linger. 2008. *Irrelevance, Polymorphism, and Erasure in Type Theory*. Ph.D. Dissertation. Portland State University, Department of Computer Science. https://doi.org/10.15760/etd.2669

Eugenio Moggi. 1991. Notions of computation and monads. *Information and Computation* 93, 1 (1991), 55–92. https://www.sciencedirect.com/science/article/pii/0890540191900524 Selections from 1989 IEEE Symposium on Logic in Computer Science.

Benjamin Moon, Harley Eades III, and Dominic Orchard. 2021. Graded Modal Dependent Type Theory. In *Programming Languages and Systems*, Nobuko Yoshida (Ed.). Springer International Publishing, Cham, 462–490.

Andrew C. Myers. 1999. JFlow: Practical Mostly-Static Information Flow Control. In *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Antonio, Texas,

USA) *(POPL '99)*. Association for Computing Machinery, New York, NY, USA, 228–241. https://doi.org/10.1145/292540.292561

A. Nerode, J.B. Remmel, and A. Scedrov. 1989. Polynomially graded logic I. A graded version of system T. In *[1989] Proceedings. Fourth Annual Symposium on Logic in Computer Science*. 375–385. https://doi.org/10.1109/LICS.1989.39192

Andreas Nuyts and Dominique Devriese. 2018. Degrees of Relatedness: A Unified Framework for Parametricity, Irrelevance, Ad Hoc Polymorphism, Intersections, Unions and Algebra in Dependent Type Theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science* (Oxford, United Kingdom) *(LICS '18)*. Association for Computing Machinery, New York, NY, USA, 779–788. https://doi.org/10.1145/3209108.3209119

Dominic Orchard, Vilem-Benjamin Liepelt, and Harley Eades III. 2019. Quantitative Program Reasoning with Graded Modal Types. *Proc. ACM Program. Lang.* 3, ICFP, Article 110 (July 2019), 30 pages. https://doi.org/10.1145/3341714

Jens Palsberg and Peter Ørbæk. 1995. Trust in the Lambda-Calculus. In *Proceedings of the Second International Symposium on Static Analysis (SAS '95)*. Springer-Verlag, Berlin, Heidelberg, 314–329.

Christine Paulin-Mohring. 2015. Introduction to the Calculus of Inductive Constructions. In *All about Proofs, Proofs for All*, Bruno Woltzenlogel Paleo and David Delahaye (Eds.). Studies in Logic (Mathematical logic and foundations), Vol. 55. College Publications. https://hal.inria.fr/hal-01094195

Tomas Petricek, Dominic Orchard, and Alan Mycroft. 2013. Coeffects: Unified Static Analysis of Context-Dependence. In *Automata, Languages, and Programming*. Springer Berlin Heidelberg, Berlin, Heidelberg, 385–397.

Tomas Petricek, Dominic Orchard, and Alan Mycroft. 2014. Coeffects: A calculus of context-dependent computation. In *Proceedings of International Conference on Functional Programming* (Gothenburg, Sweden) *(ICFP 2014)*.

F. Pfenning. 2001. Intensionality, extensionality, and proof irrelevance in modal type theory. In *Proceedings 16th Annual IEEE Symposium on Logic in Computer Science*. 221–230. https://doi.org/10.1109/LICS.2001.932499

Andrew M. Pitts. 2013. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge University Press, USA.

Frédéric Prost. 2000. A Static Calculus of Dependencies for the λ-Cube. In *Proceedings of the 15th Annual IEEE Symposium on Logic in Computer Science (LICS '00)*. IEEE Computer Society, USA, 267.

W. V. O. Quine. 1960. *Word & Object*. MIT Press.

Greg Restall. 1999. *An Introduction to Substructural Logics*. Routledge.

Peter Schroeder-Heister and Kosta Došen. 1994. *Substructural Logics*. Clarendon Press, Oxford, UK.

Naokata Shikuma and Atsushi Igarashi. 2006. Proving Noninterference by a Fully Complete Translation to the Simply Typed λ-Calculus. In *Proceedings of the 11th Asian Computing Science Conference on Advances in Computer Science: Secure Software and Related Issues* (Tokyo, Japan) *(ASIAN'06)*. Springer-Verlag, Berlin, Heidelberg, 301–315.

Geoffrey Smith and Dennis Volpano. 1998. Secure Information Flow in a Multi-Threaded Imperative Language. In *Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Diego, California, USA) *(POPL '98)*. Association for Computing Machinery, New York, NY, USA, 355–364. https://doi.org/10.1145/268946.268975

Deian Stefan, David Mazières, John C. Mitchell, and Alejandro Russo. 2017. Flexible dynamic information flow control in the presence of exceptions. *Journal of Functional Programming* 27 (2017), e5. https://doi.org/10.1017/S0956796816000241

Martin Sulzmann, Manuel M. T. Chakravarty, Simon Peyton Jones, and Kevin Donnelly. 2007. System F with Type Equality Coercions. In *Proceedings of the 2007 ACM SIGPLAN International Workshop on Types in Languages Design and Implementation* (Nice, Nice, France) *(TLDI '07)*. Association for Computing Machinery, New York, NY, USA, 53–66. https://doi.org/10.1145/1190315.1190324

M. Takahashi. 1995. Parallel Reductions in λ-Calculus. *Information and Computation* 118, 1 (1995), 120–127. https://doi.org/10.1006/inco.1995.1057

J.P. Talpin and P. Jouvelot. 1994. The Type and Effect Discipline. *Information and Computation* 111, 2 (1994), 245–296. https://doi.org/10.1006/inco.1994.1046

Yan Mei Tang and Pierre Jouvelot. 1995. Effect Systems with Subtyping. In *In ACM Conference on Partial Evaluation and Program Manipulation*. ACM Press, 45–53.

Matúš Tejiščák. 2020. A Dependently Typed Calculus with Pattern Matching and Erasure Inference. *Proc. ACM Program. Lang.* 4, ICFP, Article 91 (Aug. 2020), 29 pages. https://doi.org/10.1145/3408973

Peter Thiemann and Vasco T. Vasconcelos. 2019. Label-Dependent Session Types. *Proc. ACM Program. Lang.* 4, POPL, Article 67 (dec 2019), 29 pages. https://doi.org/10.1145/3371135

Frank Tip. 1995. A Survey of Program Slicing Techniques. *Journal of Programming Languages* 3 (1995), 121–189.

Mads Tofte and Jean-Pierre Talpin. 1997. Region-Based Memory Management. *Inf. Comput.* 132, 2 (feb 1997), 109–176. https://doi.org/10.1006/inco.1996.2613

Bernardo Toninho, Luís Caires, and Frank Pfenning. 2011. Dependent Session Types via Intuitionistic Linear Type Theory. In *Proceedings of the 13th International ACM SIGPLAN Symposium on Principles and Practices of Declarative Programming* (Odense, Denmark) *(PPDP '11)*. Association for Computing Machinery, New York, NY, USA, 161–172. https://doi.org/10.1145/2003476.2003499

Bernardo Toninho and Nobuko Yoshida. 2018. Depending on Session-Typed Processes. In *Foundations of Software Science and Computation Structures*, Christel Baier and Ugo Dal Lago (Eds.). Springer International Publishing, Cham, 128–145.

Stephen Tse and Steve Zdancewic. 2004. Translating Dependency into Parametricity. In *Proceedings of the Ninth ACM SIGPLAN International Conference on Functional Programming* (Snow Bird, UT, USA) *(ICFP '04)*. Association for Computing Machinery, New York, NY, USA, 115–125. https://doi.org/10.1145/1016850.1016868

David N. Turner and Philip Wadler. 1999. Operational Interpretations of Linear Logic. *Theor. Comput. Sci.* 227, 1–2 (sep 1999), 231–248. https://doi.org/10.1016/S0304-3975(99)00054-7

Tarmo Uustalu and Varmo Vene. 2008. Comonadic Notions of Computation. *Electronic Notes in Theoretical Computer Science* 203, 5 (2008), 263–284. https://doi.org/10.1016/j.entcs.2008.05.029 Proceedings of the Ninth Workshop on Coalgebraic Methods in Computer Science (CMCS 2008).

Matthijs Vákár. 2015. Syntax and Semantics of Linear Dependent Types. arXiv:1405.0033 [cs.LO]

Matthijs Vákár. 2016. A Framework for Dependent Types and Effects. arXiv:1512.08009 [cs.LO]

Dennis Volpano, Cynthia Irvine, and Geoffrey Smith. 1996. A Sound Type System for Secure Flow Analysis. *J. Comput. Secur.* 4, 2–3 (jan 1996), 167–187.

Philip Wadler. 1989. Theorems for Free!. In *Proceedings of the Fourth International Conference on Functional Programming Languages and Computer Architecture* (Imperial College, London, United Kingdom) *(FPCA '89)*. Association for Computing Machinery, New York, NY, USA, 347–359. https://doi.org/10.1145/99370.99404

Philip Wadler. 1990. Linear Types Can Change the World!. In *Programming Concepts and Methods*. North.

Philip Wadler. 1994. A syntax for linear logic. In *Mathematical Foundations of Programming Semantics*, Stephen Brookes, Michael Main, Austin Melton, Michael Mislove, and David Schmidt (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 513–529.

Philip Wadler and Peter Thiemann. 2003. The Marriage of Effects and Monads. *ACM Trans. Comput. Logic* 4, 1 (jan 2003), 1–32. https://doi.org/10.1145/601775.601776

Stephanie Weirich, Pritam Choudhury, Antoine Voizard, and Richard A. Eisenberg. 2019. A Role for Dependent Types in Haskell. *Proc. ACM Program. Lang.* 3, ICFP, Article 101 (jul 2019), 29 pages. https://doi.org/10.1145/3341705

Stephanie Weirich, Antoine Voizard, Pedro Henrique Azevedo de Amorim, and Richard A. Eisenberg. 2017. A Specification for Dependent Types in Haskell. *Proc. ACM Program. Lang.* 1, ICFP, Article 31 (aug 2017), 29 pages. https://doi.org/10.1145/3110275

Stephanie Weirich, Dimitrios Vytiniotis, Simon Peyton Jones, and Steve Zdancewic. 2011. Generative Type Abstraction and Type-Level Computation. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Austin, Texas, USA) *(POPL '11)*. Association for Computing Machinery, New York, NY, USA, 227–240. https://doi.org/10.1145/1926385.1926411