uOttawa

L'Université canadienne
Canada's university

Semantics for a Higher-Order Functional Programming Language for Quantum Computation.

Benoît Valiron

Thesis Submitted to the Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements for the degree of Doctor of Philosophy in Mathematics[1]

Department of Mathematics and Statistics
Faculty of Science
University of Ottawa

© Benoît Valiron, Ottawa, Canada, 2008

[1]The Ph.D. program is a joint program with Carleton University, administered by the Ottawa-Carleton Institute of Mathematics and Statistics

Pour mon père.

Abstract

The objective of this thesis is to develop a semantics for higher-order quantum information.

Following the work done in the author's M.Sc. thesis, we study a lambda calculus for quantum computation with classical control. The language features two important properties. The first one, arising from the so-called no-cloning theorem of quantum computation, is the need for a distinction between duplicable and non-duplicable elements. For keeping track of duplicability at higher order, we use a type system inspired by the resource-sensitive linear logic. The second important aspect is the probability inherent to measurement, the only operation for retrieving classical data from quantum data. This forces us into choosing a reduction strategy for being able to define an operational semantics.

We address the question of a denotational semantics in two respects. First, we restrict the study to the strictly linear aspect of the language. Doing so, we suppress the need for distinguishing between duplicable and non-duplicable elements and we can focus on the description of quantum features at higher order. Using the category of completely positive maps as a framework, we build a fully abstract denotational model of the strictly linear fragment of the language.

The study of the full language is more demanding. For dealing with the probabilistic aspect of the language, we use a method inspired by Moggi and build a computational model with a distinction between values and computations. For the distinction between duplicability and non-duplicability in the calculus, we adapt Bierman's linear category, where the duplication is considered as a comonad with specific properties. The resulting model is what we call a linear category for duplication. Finally, we only focus on the fragment of the language that contains the aforementioned elements, and remove the classical Boolean and quantum Boolean features to get a generic computational linear lambda-calculus. In this idealized setting, we show that such a language have a full and complete interpretation in a linear category for duplication.

Acknowledgements

Numerous people allowed me to write this thesis. In particular, I would like to thank:

- Peter, who supported me for yet another degree;
- Caroline, who came all the way to Halifax;
- my parents, for being supportive all that time;
- Rob, for feeding me with coffee;
- the administrative staff of both Dalhousie and Ottawa University, for being so helpful;
- and finally Gilles, for printing more that 1000 pages of this opus of content.

Contents

Abstract **iii**

Acknowledgements **iv**

List of Figures **ix**

List of Tables **x**

1 Introduction **1**

2 Notions of Category Theory **6**
2.1 Categories and Functors . . . 6
2.2 Natural Transformations . . . 7
2.3 Adjoint Functors . . . 8
2.4 Products and Coproducts . . . 9
2.5 Monads . . . 10
2.6 Comonads . . . 11
2.7 Symmetric Monoidal Categories and Comonoids . . . 13
2.7.1 Monoidal Categories . . . 13
2.7.2 Commutative Comonoids . . . 13
2.8 Monoidal Comonads and Coalgebras . . . 14

3 Quantum Computation **17**
3.1 Mathematical Formalism . . . 17
3.1.1 Generalities on Finite Dimensional Hilbert Spaces . . . 17
3.1.2 Tensor Products of Hilbert Spaces . . . 19
3.1.3 Completely Positive Maps . . . 19
3.1.4 Superoperators . . . 20
3.1.5 The 2-Dimensional Hilbert Space . . . 21
3.2 Quantum Foundations . . . 23
3.2.1 Quantum Bits . . . 23
3.2.2 Operations . . . 23
3.2.3 Mixed States . . . 25
3.3 Quantum Effects . . . 26
3.3.1 No Cloning . . . 26
3.3.2 Entanglement . . . 26
3.3.3 Bell's Inequalities . . . 27
3.4 Some Algorithms and Uses of Quantum Effects . . . 28
3.4.1 Teleportation . . . 28

3.4.2 Dense Coding . . . 30
3.4.3 The Deutsch Algorithm . . . 31

4 A Tour in Existing Models of Quantum Computation 32
4.1 The Various Paradigms . . . 32
4.1.1 Unitary Gates as Computation. . . . 32
4.1.2 Concurrent Quantum Computation . . . 34
4.1.3 Measurement-Based Quantum Computation . . . 34
4.1.4 The QRAM Model . . . 35
4.2 Formalism of Hilbert Spaces . . . 35
4.2.1 Dagger Compact-Closed Categories . . . 36
4.2.2 Classical Objects . . . 37
4.3 A Flow-Chart Language . . . 37
4.3.1 The language . . . 37
4.3.2 The Category of Superoperators . . . 37
4.3.3 Interpretation of the Flow-Chart Language . . . 39
4.4 Extension to Higher Order . . . 40

5 Lambda Calculus and Semantics of Higher-Order Computation 41
5.1 Lambda Calculus . . . 41
5.1.1 The Language . . . 41
5.1.2 Free and Bound Variables . . . 42
5.1.3 Alpha-Equivalence . . . 42
5.1.4 Operational Meaning of Lambda Calculus . . . 42
5.1.5 Typed Lambda Calculus . . . 43
5.2 Proofs as Computations . . . 45
5.2.1 Intuitionistic Logic . . . 46
5.2.2 Curry-Howard Isomorphism . . . 46
5.3 Categorical Logic . . . 47
5.4 Lambda Calculus and Side Effects . . . 48
5.4.1 Pure Versus Impure Calculus. . . . 48
5.4.2 Reduction Strategies . . . 49
5.4.3 Towards a Semantics . . . 49
5.4.4 Computational Model for Call-By-Value . . . 50
5.5 Intuitionistic Linear Logic . . . 51
5.6 Linear Calculi and their Interpretations . . . 51
5.6.1 Earlier Models . . . 52
5.6.2 Bierman's Linear Category . . . 52

6 A Lambda Calculus for Quantum Computation 54
6.1 The Language . . . 54
6.2 Operational Semantics . . . 56
6.2.1 Abstract Machine . . . 56
6.2.2 Evaluation Strategy . . . 56
6.2.3 Probabilistic Reduction Systems . . . 58
6.2.4 Reduction System . . . 59
6.3 Type System . . . 60
6.3.1 Types . . . 60
6.3.2 Typing Rules . . . 61
6.4 Properties of the Type System . . . 62

6.4.1 Safety Properties . . . 62
6.4.2 Type Inference Algorithm . . . 63
6.5 Examples of Algorithms . . . 64
6.5.1 The Deutsch Algorithm . . . 64
6.5.2 The Teleportation Procedure . . . 64
6.5.3 Type Derivation of the Teleportation Protocol . . . 65
6.5.4 Reduction of the Teleportation Term . . . 67
6.5.5 Reduction of the Superdense Coding Term . . . 68
6.6 Towards a Denotational Semantics . . . 69

7 The Linear Fragment 70
7.1 A Linear Lambda Calculus for Quantum Computation . . . 70
7.2 Operational Semantics . . . 75
7.2.1 Small Step Semantics . . . 75
7.2.2 Safety Properties . . . 77
7.2.3 Normalization . . . 79
7.2.4 Quantum Context and Reduction . . . 81
7.2.5 Reduction to Values . . . 82
7.3 Denotational Semantics . . . 83
7.3.1 Modeling the Linear Quantum Lambda Calculus . . . 83
7.3.2 Fullness of the First-Order Fragment . . . 89
7.3.3 Fullness up to Scalar Multiple . . . 91
7.4 Equivalence Classes of Terms . . . 94
7.4.1 Axiomatic Equivalence . . . 94
7.4.2 Operational Context . . . 95
7.4.3 Operational Equivalence . . . 95
7.4.4 Denotational Equivalence . . . 96
7.5 Soundness and Full Abstraction . . . 96
7.5.1 Proof of the Soundness Theorem . . . 97
7.5.2 Full Abstraction: Preliminary Lemmas . . . 97
7.5.3 Proof of the Full Abstraction Theorem . . . 98

8 Structure of the Linear-Non-Linear Fragment 99
8.1 Computations and Values . . . 99
8.2 Duplicability Versus Non-Duplicability . . . 99
8.2.1 Computations as Proofs . . . 100
8.3 Structure of the Exponential . . . 100
8.3.1 Idempotency . . . 100
8.3.2 Coherence Property for Idempotent Comonads . . . 101
8.3.3 Duplicable Pairs and Pairs of Duplicable Elements . . . 106
8.4 Linear Category for Duplication . . . 107

9 A Computational Lambda Calculus for Duplication 108
9.1 An Indexed Lambda Calculus . . . 108
9.1.1 Type System . . . 108
9.1.2 Terms . . . 109
9.1.3 Typing Judgements . . . 110
9.1.4 Type Casting and Substitution Lemma . . . 115
9.2 Equational Logic of Typed Terms . . . 122
9.3 The Category $\mathcal{C}_\lambda$ . . . 127

9.3.1 Monoidal Structure 128
9.3.2 Monadic Structure 138
9.3.3 Comonadic Structure 144
9.3.4 The Category $\mathcal{C}_\lambda$ is a Linear Category for Duplication 146

10 Proof of Theorem 9.2.7 **147**
10.1 A Handy Tool: Neutral Terms 147
10.2 Term Rewriting System Number One 151
10.3 Term Rewriting System Number Two 157
10.3.1 The Rewriting System 157
10.3.2 Height of Terms 159
10.3.3 Degree of Terms 160
10.3.4 Last Notion of Measure: Number of a Term 162
10.3.5 Putting Everything Together 163
10.4 Proof of Theorem 9.2.7 171

11 Categorical Semantics **173**
11.1 Denotational Semantics 173
11.1.1 Interpretation of the Type System 173
11.1.2 Interpretation of the Language 173
11.2 Soundness of the Denotation 181
11.3 Completeness 197
11.4 Towards a Denotational Model 199

Bibliography **200**

Index **206**

List of Figures

3.1 The Bloch sphere. 22
3.2 Quantum teleportation protocol. 28
3.3 Dense coding protocol. 30

4.1 The QRAM model 35

List of Tables

4.1 Rules for constructing quantum flow-charts 38

5.1 Intuitionistic logic: natural deduction rules 46
5.2 Interpretation of the simply typed lambda calculus 48

6.1 Reductions rules of the quantum lambda calculus 60
6.2 Typing rules 62

7.1 Typing rules for the linear quantum lambda calculus 71
7.2 Reduction rules for the linear quantum lambda calculus 76
7.3 Denotational semantics for typing judgements. 84
7.4 Axiomatic equivalence 94

9.1 Typing rules for the linear-non-linear quantum lambda calculus 110
9.2 Axiomatic equivalence axioms: beta-eta-rules 123
9.3 Axiomatic equivalence axioms: commutation rules 123
9.4 Axiomatic equivalence: derived rules 124
9.5 Definitions of maps and operations on maps in $\mathcal{C}_\lambda$ 146

11.1 Interpretation of core values. 176
11.2 Interpretation of extended values. 177
11.3 Interpretation of computations. 178

Chapter 1

Introduction

Quantum computation is a paradigm of computation where information is potentially non-local, can be non-duplicable, and for which one of the core features is inherently probabilistic. However, using quantum computation, numerous hard problems, such as factoring, can be performed efficiently. Higher-order computation brings new light on these algorithms, but raises several challenging problems such as dealing with non-duplicable functions, or marrying duplicability and probabilistic behavior. By taking the problem both at the concrete level of completely positive maps and at the abstract level of categorical structures, this thesis addresses the issue of the semantics of higher-order quantum computation.

Background

Quantum computation. While classical computation uses data encoded on objects governed by the laws of classical physics, quantum computation uses data encoded on the state of particles governed by the laws of quantum physics. The basic unit of data in quantum computation is a quantum bit, which can be modeled by a normalized vector in a 2-dimensional Hilbert space $\mathcal{H}$. Choosing an orthonormal basis $\{|0\rangle, |1\rangle\}$, one can consider $|0\rangle$ and $|1\rangle$ as vectors of information, leading to potential *superposition of information*. The state of a system of two or more quantum bits is a normalized vector in $\mathcal{H} \otimes \mathcal{H} \otimes \ldots \otimes \mathcal{H}$. The permissible operations on quantum bits are determined by the laws of quantum physics. The only possible operations are initializations, unitary operations and measurements (and combinations thereof). Here, measurement is an operation that sends a quantum bit in state $\alpha|0\rangle + \beta|1\rangle$ to $|0\rangle$ with probability $|\alpha|^2$ and $|1\rangle$ with probability $|\beta|^2$, while simultaneously returning a Boolean 0 or 1, respectively.

Shor (1994) has shown that quantum computers can factor an integer in a time that is polynomial in its number of digits. It is not known whether any classical algorithm can solve the problem in polynomial time. The factoring problem has numerous implications in cryptography. This discovery has focused attention on quantum computing. Quantum computing is also able to bring change in other domains, such as database manipulation, with algorithms to query elements in databases, and such as numerical methods, with the ability to perform Fourier transform efficiently (Nielsen and Chuang, 2002).

From the point of view of quantum programming languages, there are several features of quantum computation that deserve special attention. First, unlike classical data, quantum data cannot in general be duplicated: there is no map $(\alpha|0\rangle + \beta|1\rangle) \mapsto (\alpha|0\rangle + \beta|1\rangle) \otimes (\alpha|0\rangle + \beta|1\rangle)$. This is known as the "no-cloning theorem" (Wootters and Zurek, 1982). Second, quantum data can be entangled: Consider the two-quantum bit state $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$. Although one can speak of "quantum

bit number 1" and "quantum bit number 2", there is no way of writing the system as $|\phi\rangle \otimes |\psi\rangle$. Finally, measurement, which is the only operation for converting quantum data to classical data, is inherently a probabilistic operation.

Semantics of programming languages. When dealing with a programming language, an important question is how to describe the behavior of programs, and to characterize the set of the possible behaviors. A *semantics* (Pierce, 2002) for a programming language serves as a description of properties that valid programs satisfy. This permits to get insights into the capabilities of the language and the logic underlying it. This can help for example in understanding the power and the limitations of the computational paradigm considered, or more pragmatically to build automated tools to check for the validity of a piece of code, along a set of specifications.

An *operational* semantics is a description of how valid programs behave by sequences of computational steps. Depending on the programming language considered, this can take the form of a rewriting system of the code itself (in the case, for example, of the simply-typed lambda calculus), or it can be more involved and require the description of an abstract machine (as, for example, in the case of the Turing machine).

An *axiomatic* semantics is a description of local syntactic equivalences of code. The goal is to provide an equational logic of programs, in order to exhibit general syntactic properties of the language. It is helpful to describe a system of normal forms in which every program can be rewritten. An important property of an axiomatic semantics is *soundness*. We say that an axiomatic semantics is *sound* if two axiomatically equivalent programs are operationally equivalent.

A *denotational* semantics is built by drawing a parallel between the programs in the language and a mathematically structured set. The properties of the language are understood as mathematical and logical properties, giving a handle to answer questions about the language. A very strong property for a denotational semantics is *full abstraction*: the semantics is fully abstract when any two programs have the same denotation precisely when they are operationally equivalent in every context. When the interpretation is also onto, the denotation is *fully complete*.

Sometimes, at first it is only possible to describe the axiomatic structure of the language. In this situation, by studying the relations between pieces of code it can be possible to describe the categorical structure underlying the logic and infer a *categorical semantics*.

Higher-order computation and lambda calculus. An essential paradigm in computation is higher-order computation. A higher-order function is a function that inputs or outputs a "black box", which is itself a function. The canonical formalism for expressing higher-order functions is the lambda calculus developed by Church (1936) and Kleene (1935a,b) to characterize computable functions. This endeavour was shown (Turing, 1936) to have the same computational power as Turing machines.

The lambda calculus not only permits the design of all computable functions, but it also allows one to draw a correspondence between proofs in logic and programs. This correspondence, known as the Curry-Howard isomorphism (Girard et al., 1990), is the one of the main tools used for passing from an axiomatic semantics to a categorical semantics (Lambek and Scott, 1989). It is thus a powerful handle to grasp the structure behind potentially intricate computational schemes.

Pure and impure lambda calculus. In its most general formulation, the lambda calculus is called *pure*, or *purely functional*. This means that programs are completely determined by their output result on each possible input. It is fairly restrictive and unfortunately rules out many interesting *side effects*, such as non-determinism, probabilistic outcomes, or input-outputs. To give a semantics to languages with side-effects, Moggi (1989, 1991) defines a lambda calculus, the *computational lambda calculus*, where two classes of terms are considered: the *values* and the *computations*. Using

an axiomatic equivalence and some categorical tools, he describes a general model of computations with side effects.

Semantics of Quantum Programming Languages

Besides a few existing algorithms, quantum computation is a relatively fresh area of research with no single best way of thinking about the processes in play. Understanding the semantics of quantum programming languages is therefore a useful tool in understanding the capabilities of quantum computation and for deciding on the most appropriate form to use as a computational scheme. Being in the intersection of physics, computer science and mathematics, several paradigms to process quantum computation exist, and a wide ranges of approaches are studied (Gay, 2006).

Some studies focus primarily on the unitary evolution of the quantum states, and view measurement as a meta-operation outside of the formalism. Benioff (1980) and Deutsch (1985) built a quantum Turing machine, where the tape, the head and the states are encoded as quantum information. Van Tonder (2003, 2004) described a lambda calculus encoded on strings of quantum bits. Altenkirch and Grattage (2005a,b) studied a first order functional language with a notion of quantum test compiling into a quantum circuit.

Other approaches consider measurements as the driving force in the computation. This is the one-way model of Raussendorf and Briegel (2001), further extended with an axiomatic and a denotational semantics in (Danos et al., 2007).

It is also possible to see measurements as an active part in the computation and to allow interleaved unitary operations and measurements. One example is the QRAM model of Knill (1996) and Bettelli et al. (2003). Here, a quantum computer consists of a classical computer with a quantum device attached to it. In this configuration, called "classical control" (Selinger, 2004c), the operation of the machine is controlled by a classical program which emits a sequence of instructions to the quantum device for performing measurements and unitary operations. Several programming languages have been proposed to deal with such a model, see for example the works of Sanders and Zuliani (2000), Bettelli et al. (2003), Selinger (2004b).

The denotational aspect of quantum computation has been studied in various ways. A first trend of research, started by Abramsky and Coecke (2004), focuses on the basic structures required for performing quantum computation, using a categorical framework. This work gave birth to a range of publications, such as (Coecke, 2004; Coecke and Pavlovic, 2007; Selinger, 2005) and (Coecke and Paquette, 2006). Other approaches focus on the description of quantum computation in term of superoperators. Selinger (2004b) defines a flow-chart language and provides a fully complete semantics for the language in term of superoperators. (Selinger, 2004c) is an attempt to generalize this semantics at higher order, by looking for a suitable category of normed vector spaces. However, none of the studied descriptions capture the required structure. Another related work is the one of Girard (2004), describing a model of linear logic in terms of normed vector spaces. However, although being a $*$-autonomous category, it does not yield the correct answer at base types, as pointed out by (Selinger, 2004c).

A Lambda Calculus for Quantum Computation

Selinger and Valiron (2006a, 2005) and Valiron (2004a,b) developed a higher-order language for quantum computation. The language is a typed lambda calculus together with classical and quantum data, featuring creations of quantum bits, measurements and unitary operations. For example, in such a language one can build a function that inputs a Boolean and returns a function from quantum bits to quantum bits based on the input bit.

The first important feature built into the language is the distinction between duplicable and non-duplicable elements. This arises from the no-cloning property of quantum computation. So if $x : qbit$ is a variable representing a quantum bit, and $y : bit$ is a variable representing a classical bit, then it is legal to write $f(y, y)$, but not $g(x, x)$. In order to keep track of duplicability at higher-order types we use a type system based on linear logic (Girard, 1987). We use the duplicability operator "!" to mark classical types.

The second feature of quantum computation included in the language is the probabilistic nature of the measurement. This was taken care of by defining an abstract machine together with an operational semantics in the form of a call-by-value reduction strategy.

The most significant results of (Valiron, 2004a) were the proof that the type system prevents a typed program from ending up in an error state, and the description a type inference algorithm for deciding whether a given program is typeable or not.

Semantics for Higher-Order Quantum Computation

We now turn to the question of a semantics for the quantum lambda calculus of (Valiron, 2004a).

As explained before, we use the duplicability operator "!" to mark classical types. In the categorical semantics, this operator gives rise to a comonad as in the work of Seely (1989) and Benton et al. (1992). Another account of mixing copyable and non-copyable data is Coecke and Pavlovic (2007), where the copyability is internal to objects. However, it is not clear whether this approach scales well to higher order.

To model the probabilistic effect of the measurement operator in our call-by-value setting, our semantics requires a computational monad in the sense of Moggi (1991). The coexistence of the computational monad and the duplicability comonad in the same category is what makes our semantics interesting and novel. It differs from the work of Benton and Wadler (1996), who considered a monad and a comonad one two different categories, arising from a single adjunction.

The computational aspects of linear logic have been extensively explored by many authors, including Abramsky (1993); Benton et al. (1992, 1993); Bierman (1993); Wadler (1992). However, these works contain explicit lambda terms to witness the structural rules of linear logic, for example, $x : !A \rhd \mathrm{derelict}(x) : A$. By contrast, in our language, structural rules are implicit at the term level, so that $!A$ is regarded as a subtype of A and one writes $x : !A \rhd x : A$. As it was shown in Selinger and Valiron (2006a), linearity information can automatically be inferred by the type checker. This allows the programmer to program as in a regular non-linear language.

This use of subtyping is the main technical complication in our proof of well-definedness of the semantics. This is because one has to show that the denotation is independent of the choice of a potentially large number of possible derivations of a given typing judgement. We are forced to introduce a Church-style typing system, and to prove that the semantics finally does not depend on the additional type annotations.

Another technical choice we made in our language concerns the relation between the exponential "!" and the pairing operation. Linear logic only requires $!A \otimes !B \rhd !(A \otimes B)$ and not the opposite implication. However, in our programming language setting, we find it natural to identify a classical pair of values with a pair of classical values, and therefore we will have an isomorphism $!A \otimes !B \cong !(A \otimes B)$.

Plan of the Thesis

The content of the thesis is summarized as follows.

In Chapter 2, we state the notions of category theory used in the thesis. In Chapter 3 we give a brief overview of quantum computation, and we describe in Chapter 4 the works in the

literature concerned with the semantics of quantum programming languages. In particular, we detail in Section 4.3 the flow-chart language of Selinger (2004b) and its semantics in term of superoperators, used in Chapter 7.

The last chapter on general background is Chapter 5, giving a brief introduction on lambda calculus and its categorical interpretation. We review the simply-typed lambda calculus, its computational extension for dealing with side-effects, and the computational interpretation of intuitionistic linear logic

Chapter 6 summarizes of the results contained in (Valiron, 2004a). The quantum lambda calculus that will be referred to all along the remainder of the thesis is defined here, together with its type system and its operational semantics.

The next chapters are the main contribution of this thesis.

Chapter 7 contains a precise study of the linear fragment of the quantum lambda calculus. This fragment is typed, its operational semantics is described in the same way as in Chapter 6, and a denotational semantics in term of completely positive maps is provided. Using the fact that the semantics is full at first order, the semantics is proved to be fully abstract.

Chapter 8 raises the issue of the structure of the full language. Using Moggi's computational model and Bierman's linear categories, a categorical structure candidate for being a model of the language, called *linear category for duplication* is provided.

Chapter 9 describes a version of the quantum lambda calculus stripped from the classical and the quantum Boolean structure. The language is reduced to its core features, namely the distinction between linearity and non-linearity, the higher order, the pairing and the computational aspect. An axiomatic semantics is described, yielding a syntactic category that is proved to be a linear category for duplication. The result requires Theorem 9.2.7, whose proof requires some technical machinery. Chapter 10 explains this machinery: We introduce two rewriting systems. They converge to a special class of terms, called the *neutral terms*, for which the theorem is trivial. The main difficulty is to prove a weak-normalization result for these rewriting systems. We use techniques from (Girard et al., 1990, Ch. 4) by defining a well-founded measure of convergence, decreasing with each reduction of the rewriting system.

Chapter 11 develops the interpretation of the computational lambda calculus for duplication into a general linear category for duplication, and the proof that the interpretation is sound and complete.

Chapter 2

Notions of Category Theory

In this chapter we give a brief exposition of the definitions and results of category theory used in the remainder of the thesis. For a complete introduction to category theory, (Mac Lane, 1998) is a solid reference. An approach to category theory oriented towards semantic aspects of computation can be found in (Lambek and Scott, 1989).

2.1 Categories and Functors

The content of this section follows (Lambek and Scott, 1989).

Definition 2.1.1. A *category* $\mathcal{C}$ consists of the following three elements.

- A class of *objects*, denoted by $|\mathcal{C}|$.
- A class of *morphisms*. To each morphism f is associated a *domain* A and a *codomain* B, where A, B are objects. We say f is a morphism (or an *arrow*, or a *map*) from A to B and we write $f : A \to B$. For any given A and B, we define the *homset* of A to B as the class of morphisms from A to B, and we denote it $\mathcal{C}(A, B)$, $\hom_{\mathcal{C}}(A, B)$, or $\hom(A, B)$ if there is no ambiguity. For each A there is a special morphism $id_A : A \to A$, called the *identity*.
- A binary operator $\circ$ called *composition* such that if $f : A \to B$ and $g : B \to C$ are morphisms then $g \circ f : A \to C$ is a morphism. We will also write $g \circ f$ as $f; g$.

The composition satisfies two properties: if $f : A \to B$, $g : B \to C$ and $h : C \to D$ are morphisms, then

$$(f; g); h = f; (g; h) \qquad \text{and} \qquad f; id_B = f = id_A; f.$$

We say that the category is *locally small* if for all objects A, B the class $\hom(A, B)$ is a set. We say it is *small* if moreover the class of objects is a set.

Definition 2.1.2. We say that a morphism $f : A \to B$ in a category $\mathcal{C}$ is an *isomorphism* if there exists a morphism $f^{-1} : B \to A$ such that $f; g = id_A$ and $g; f = id_B$.

Example 2.1.3. The trivial category $\mathbf{1}$. Its class of objects is the singleton $\{\star\}$ and its unique arrow is the identity on $\star$. Composition is defined trivially.

Example 2.1.4. The *category* **Set** *of sets*. Its objects are arbitrary sets and its morphisms arbitrary mappings between sets.

Example 2.1.5. Given a category $\mathcal{C}$, the *opposite category* $\mathcal{C}^{\mathrm{op}}$ consisting of the objects of $\mathcal{C}$ and the arrows of $\mathcal{C}$ in "opposite" direction: f is an arrow from B to A in $\mathcal{C}^{\mathrm{op}}$ if and only if $f : A \to B$ is an arrow of $\mathcal{C}$. We write f^{op} for the arrow in $\mathcal{C}^{\mathrm{op}}$ corresponding to f in $\mathcal{C}$.

Example 2.1.6. Given two categories $\mathcal{C}$ and $\mathcal{D}$ one defines a category $\mathcal{C} \times \mathcal{D}$ with the following data.

- Objects are pairs (A, B) where $A \in |\mathcal{C}|$ and $B \in |\mathcal{D}|$.
- An arrow $(A, B) \to (C, D)$ is a pair of arrows $(f : A \to C, g : B \to D)$, where $f \in \mathcal{C}$ and $g \in \mathcal{D}$. The identity arrow $id_{A,B}$ is the pair (id_A, id_B).
- The composition of two maps $(f, f') : (A, A') \to (B, B')$ and $(g, g') : (B, B') \to (C, C')$ is the map $(f; g, f; g') : (A, A') \to (C, C')$.

Definition 2.1.7. Given two categories $\mathcal{C}$ and $\mathcal{D}$, a *functor* F from $\mathcal{C}$ to $\mathcal{D}$ is a map from $|\mathcal{C}|$ to $|\mathcal{D}|$ and from $\hom_{\mathcal{B}}(A, B)$ to $\hom_{\mathcal{D}}(FA, FB)$ for all A, B in $\mathcal{C}$ preserving identities and composition:

$$F(id_A) = id_{FA} \qquad \text{and} \qquad F(f; g) = Ff; Fg.$$

A functor $F : \mathcal{C} \to \mathcal{D}$ is *full* if for all objects A and B of $\mathcal{C}$ the map

$$\begin{array}{rccc} F_{A,B} : & \hom_{\mathcal{C}}(A, B) & \to & \hom_{\mathcal{D}}(FA, FB) \\ & f & \mapsto & Ff \end{array}$$

is surjective. The functor F is *faithful* if the maps $F_{A,B}$ are injective, and *fully faithful* if they are bijective. A *full embedding* is a fully faithful functor that is also injective on objects.

Example 2.1.8. The *diagonal functor* on $\mathcal{C}$ is the functor $\triangle_{\mathcal{C}} : \mathcal{C} \to \mathcal{C} \times \mathcal{C}$ sending A to (A, A) and f to (f, f). It is faithful but not in general full.

Example 2.1.9. The *terminal functor* on $\mathcal{C}$ is the functor $\bigcirc_{\mathcal{C}} : \mathcal{C} \to \mathbf{1}$ sending every object on $\star$ and every morphism on $id_\star$. It is full but not in general faithful.

Example 2.1.10. Consider three categories $\mathcal{C}$, $\mathcal{D}$ and $\mathcal{E}$.

1. Given a functor $F : \mathcal{C} \to \mathcal{D}$ and a functor $G : \mathcal{D} \to \mathcal{E}$, the map $GF : \mathcal{C} \to \mathcal{E}$ defined as the composition of F and G is also a functor.
2. The map $id_{\mathcal{C}} : \mathcal{C} \to \mathcal{C}$ sending an object A to itself and a map $f : A \to B$ to itself is the *identity functor*. □

Definition 2.1.11. One defines the category **Cat** of small categories as the category where objects are small categories and morphisms are functors of small categories. The identity morphism on $\mathcal{C}$ is the identity functor $id_{\mathcal{C}}$ and the composition of arrows is the composition of functors.

2.2 Natural Transformations

The content of this section follows (Mac Lane, 1998).

Definition 2.2.1. Given two functors $F, G : \mathcal{C} \to \mathcal{D}$, a *natural transformation* n from F to G, denoted as $n : F \dot{\to} G$, is a collection of maps $n_A : FA \to GA$ in $\mathcal{D}$ for each object A of $\mathcal{C}$ such that for all morphism $f : A \to B$ in $\mathcal{C}$ the following diagram commutes:

$$\begin{tikzcd} FA \arrow[r, "n_A"] \arrow[d, "Ff"'] & GA \arrow[d, "Gf"] \\ FB \arrow[r, "n_B"'] & GB. \end{tikzcd}$$

We say that a diagram *commutes* when for every two vertices X, Y in the diagram, all the paths from X to Y (following arrows) yield equal morphisms. In this particular case, this means $n_A; Gf = Ff; n_B$. The diagrammatic notation for a natural transformation $n : F \dot{\to} G$ is

$$\begin{tikzcd} \mathcal{C} \arrow[r, bend left, "F"] \arrow[r, bend right, "G"'] \arrow[r, phantom, "\Downarrow n"] & \mathcal{D} \end{tikzcd} \tag{2.2.1}$$

We say that a natural transformation n is a *natural isomorphism* if for all objects A the map n_A is an isomorphism.

Definition 2.2.2. Given three functors $F, G, H : \mathcal{C} \to \mathcal{D}$ and two natural transformations $m : F \dot{\to} G$ and $n : G \dot{\to} H$, one defines the composition $m; n : F \dot{\to} H$ as the collection of maps $m_A; n_A : FA \to HA$. One defines the identity on F as the collection of arrows $id_{FA} : FA \to FA$.

Definition 2.2.3. Consider two categories $\mathcal{C}$ and $\mathcal{D}$. One defines the *functor category* $Func(\mathcal{C}, \mathcal{D})$, or $\mathcal{D}^{\mathcal{C}}$, as the category whose objects are functors $\mathcal{C} \to \mathcal{D}$ and arrows natural transformations $F \dot{\to} G$, where $F, G : \mathcal{C} \to \mathcal{D}$ are functors.

Definition 2.2.4. Consider four categories, four functors and one natural transformation in the following situation:

$$\begin{tikzcd} \mathcal{A} \arrow[r, "F"] & \mathcal{B} \arrow[r, bend left, "G"] \arrow[r, bend right, "H"'] \arrow[r, phantom, "\Downarrow n"] & \mathcal{C} \arrow[r, "K"] & \mathcal{D}. \end{tikzcd} \tag{2.2.2}$$

One defines the two natural transformations

$$nF : GF \dot{\to} HF, \qquad Kn : KG \dot{\to} KH$$

as $(nF)_A = n_{FA}$ and $(Kn)_A = K(n_A)$.

Lemma 2.2.5. *Consider the situation*

$$\begin{tikzcd} \mathcal{A} \arrow[r, "F"] & \mathcal{B} \arrow[r, bend left=60, "G"] \arrow[r, "H"] \arrow[r, bend right=60, "L"'] \arrow[r, phantom, shift left=3, "\Downarrow n"] \arrow[r, phantom, shift right=3, "\Downarrow m"] & \mathcal{C} \arrow[r, "K"] & \mathcal{D}. \end{tikzcd}$$

Then $K(nF) = (Kn)F$, $K(n; m) = (Kn; Km)$ *and* $(n; m)L = (nL; mL)$. □

Definition 2.2.6 (Mac Lane, 1998, p. 93)**.** Let $\mathcal{C}$ and $\mathcal{D}$ be two categories. We say that $\mathcal{C}$ and $\mathcal{D}$ are *equivalent* if there exists a pair of functors $F : \mathcal{C} \to \mathcal{D}$ and $G : \mathcal{D} \to \mathcal{C}$ together with natural isomorphisms $id_{\mathcal{C}} \cong G \circ F$ and $id_{\mathcal{D}} \cong F \circ G$. We say that F and G are *equivalences of category.*

2.3 Adjoint Functors

The following definitions and lemmas are taken directly from (Lambek and Scott, 1989, pp. 13–15)

Definition 2.3.1. An *adjunction* between categories $\mathcal{C}$ and $\mathcal{D}$ is given by a quadruple (F, U, η, ϵ), where $F : \mathcal{C} \to \mathcal{D}$ and $U : \mathcal{D} \to \mathcal{C}$ are functors and $\eta : id_{\mathcal{C}} \dot{\to} UF$ and $\epsilon : FU \dot{\to} id_{\mathcal{D}}$ are natural transformations such that $(\eta U); (U\epsilon) = id_U$ and $(F\eta); (\epsilon F) = id_F$. One says that U is *right adjoint* to F, or that F is *left adjoint* to U, and one calls η the *unit* and ϵ the *counit* of the adjunction.

Lemma 2.3.2. *An adjunction (F, U, η, ϵ) between locally small categories $\mathcal{C}$ and $\mathcal{D}$ gives rise to and is determined by a natural isomorphism τ*

$$\tau_{A,B} : \mathcal{D}(FA, B) \xrightarrow{\sim} \mathcal{C}(A, UB).$$

between the two functors $\mathcal{D}(F-,-)$ and $\mathcal{C}(-,U-)$ of type $\mathcal{C}^{\mathrm{op}} \times \mathcal{D} \to \mathbf{Set}$. □

Lemma 2.3.3 (Uniqueness)**.** *Adjoint functors determine each other uniquely up to natural isomorphism.* □

Lemma 2.3.4 (Composition)**.** *Given the two following adjunctions:*

$$(F : \mathcal{C} \to \mathcal{D},\ \ U : \mathcal{D} \to \mathcal{C},\ \ \eta : id_{\mathcal{C}} \dot{\to} UF,\ \ \ \ \epsilon : FU \dot{\to} id_{\mathcal{D}}),$$
$$(F' : \mathcal{D} \to \mathcal{E},\ U' : \mathcal{E} \to \mathcal{D},\ \eta' : id_{\mathcal{D}} \dot{\to} U'F'\ ,\epsilon' : F'U' \dot{\to} id_{\mathcal{E}}),$$

the quadruple

$$(F'F : \mathcal{C} \to \mathcal{E},\ UU' : \mathcal{E} \to \mathcal{C},\ \eta; U\eta'F : id_{\mathcal{C}} \dot{\to} UU'F'F,\ F'\epsilon U'; \epsilon' : F'FUU' \dot{\to} id_{\mathcal{E}})$$

is an adjunction between $\mathcal{C}$ and $\mathcal{E}$. □

Definition 2.3.5. We say that the adjunction $(F'F, UU')$ of Lemma 2.3.4 is the *composition* of (F, U) and (F', U').

Remark 2.3.6. The new structure is an adjunction because the two equations of Definition 2.3.1, namely

$$((\eta; U\eta'F)(UU')); ((UU')(F'\epsilon U'; \epsilon')) = id_{UU'}, \tag{2.3.1}$$
$$((F'F)(\eta; U\eta'F)); ((F'\epsilon U'; \epsilon')(F'F)) = id_{F'F}, \tag{2.3.2}$$

are satisfied.

2.4 Products and Coproducts

Definition 2.4.1 (Mac Lane, 1998)**.** An object T in a category $\mathcal{C}$ is called a *terminal object* if for each object A there exists a unique map $\bigcirc_A : A \to T$.

Definition 2.4.2. An object $\perp$ in a category $\mathcal{C}$ is called an *initial object* if for each object A there exists a unique map $\square_A : \perp \to A$.

Lemma 2.4.3 (Lambek and Scott, 1989)**.** *The category $\mathcal{C}$ has a terminal (respectively initial) object if and only if the functor $\bigcirc_{\mathcal{C}} : \mathcal{C} \to \mathbf{1}$ has a right (respectively left) adjoint.*

Definition 2.4.4 (Lambek and Scott, 1989)**.** Given a category $\mathcal{C}$ and two objects A and B, the *product* of A and B is, if it exists, the data consisting of an object $A \times B$ and two maps $\pi^1_{A,B} : A \times B \to A$ and $\pi^2_{A,B} : A \times B \to B$, such that for all maps $f : C \to A$ and $g : C \to B$ there exists a unique map $\langle f, g\rangle : C \to A \times B$ with the following equations holding for all $h : C \to A \times B$:

$$\langle f, g\rangle; \pi^1_{A,B} = f \tag{2.4.1}$$
$$\langle f, g\rangle; \pi^2_{A,B} = g \tag{2.4.2}$$
$$\langle h; \pi^1_{A,B}, h; \pi^2_{A,B}\rangle = h \tag{2.4.3}$$

We say that the category $\mathcal{C}$ *has binary products* if there is a product for all A and B.

Definition 2.4.5 (Mac Lane, 1998)**.** Given a category $\mathcal{C}$ and two objects A and B in $\mathcal{C}$, the *coproduct* of A and B is, if it exists, the data consisting of an object $A+B$ and two maps ${i^l}_{A,B} : A \to A+B$ and $i^r : B \to A+B$ such that for all maps $f : A \to C$ and $g : B \to C$ there exists a unique map $[f,g] : A+B \to C$ with the following equations holding for all $h : A+B \to C$:

$$ {i^l}_{A,B}; [f,g] = f \tag{2.4.4} $$

$$ {i^r}_{A,B}; [f,g] = g \tag{2.4.5} $$

$$ [{i^l}_{A,B}; h, {i^r}_{A,B}; h] = h \tag{2.4.6} $$

We say that the category $\mathcal{C}$ *has binary coproducts* if there is a coproduct for all A and B.

Lemma 2.4.6. *Binary products (respectively binary coproducts) in a category $\mathcal{C}$ induce a functor $\times$ (respectively $+$): $\mathcal{C} \times \mathcal{C} \to \mathcal{C}$.*

Lemma 2.4.7 (Lambek and Scott, 1989)**.** *The category $\mathcal{C}$ has binary products (respectively binary coproducts) if and only if the functor $\triangle_{\mathcal{C}} : \mathcal{C} \to \mathcal{C} \times \mathcal{C}$ has a right (respectively left) adjoint.*

Definition 2.4.8 (Lambek and Scott, 1989)**.** We say that a category $\mathcal{C}$ is *cartesian* if it has a terminal object T and binary products $\times$. The category is *cartesian closed* if for all A the functor $A \times (-)$ has a right adjoint $A \Rightarrow (-)$.

2.5 Monads

Definition 2.5.1 (Mac Lane, 1998)**.** A *monad* over a category $\mathcal{C}$ is a triple (T, η, μ) where $T : \mathcal{C} \to \mathcal{C}$ is a functor, $\eta : id \dot{\to} T$ and $\mu : T^2 \dot{\to} T$ are natural transformations and the following diagrams commute:

$$ \begin{tikzcd} T^3A \arrow[r, "T\mu_A"] \arrow[d, "\mu_{TA}"'] & T^2A \arrow[d, "\mu_A"] \\ T^2A \arrow[r, "\mu_A"] & TA, \end{tikzcd} \tag{2.5.1} \qquad \begin{tikzcd} TA \arrow[r, "\eta_{TA}"] \arrow[dr, "id_{TA}"'] & T^2A \arrow[d, "\mu_A"] & TA. \arrow[l, "T\eta_A"'] \arrow[dl, "id_{TA}"] \\ & TA & \end{tikzcd} \tag{2.5.2} $$

The natural transformation μ is called the *multiplication* of the monad and η the *unit* of the monad.

Definition 2.5.2 (Manes, 1976)**.** A *Kleisli triple* over a category $\mathcal{C}$ is a triple $(T, \eta, -^*)$ where $T : |\mathcal{C}| \to |\mathcal{C}|$, $\eta_A : A \to TA$ for all A and $f^* : TA \to TB$ for all $f : A \to TB$, and the following equations hold:

$$ \eta_A^* = id_{TA}, \tag{2.5.3} $$

$$ \eta_A; f^* = f \text{ for } f : A \to TB, \tag{2.5.4} $$

$$ f^*; g^* = (f; g^*)^* \text{ for } f : A \to TB \text{ and } g : B \to TC. \tag{2.5.5} $$

Lemma 2.5.3 (Manes, 1976)**.** *There is a one-to-one correspondence between Kleisli triples and monads.*

Proof. Given a Kleisli triple $(T, \eta, -^*)$, we construct a monad (T, η, μ) by extending T to arrows: $T(f) = (f; \eta_B)^*$ for $f : A \to B$, and by defining $\mu_A = id_{TA}^*$. Conversely, given a monad (T, η, μ), $-^*$ is defined by $f^* = (Tf); \mu_B$ for $f : A \to TB$. □

Definition 2.5.4 (Manes, 1976)**.** Given a Kleisli triple $(T, \eta, -^*)$ over $\mathcal{C}$, the *Kleisli category* $\mathcal{C}_T$ is defined as follows:

- the objects of $\mathcal{C}_T$ are the objects of $\mathcal{C}$,

- the set $\mathcal{C}_T(A,B)$ of morphisms from A to B in $\mathcal{C}_T$ is $\mathcal{C}(A,TB)$,
- the identity on A is $\eta_A : A \to TA$,
- The composition of $f \in \mathcal{C}_T(A,B)$ and $g \in \mathcal{C}_T(B,C)$ in $\mathcal{C}_T$ is $f;g^* : A \to TC$.

Lemma 2.5.5. *Consider a monad (T,μ,η) over a category $\mathcal{C}$. There is an adjunction*

$$\begin{tikzcd} \mathcal{C} \arrow[r, bend left, "F"] \arrow[r, phantom, "\perp"] & \mathcal{C}_T \arrow[l, bend left, "U"] \end{tikzcd}$$

arising from the monad, where $F(A) = A$, $U(A) = T(A)$, and F and U perform the following operations on morphisms:

$$\begin{aligned} F: &\quad (A \xrightarrow{f} B) \longmapsto (A \xrightarrow{f} B \xrightarrow{\eta_B} TB), \\ U: &\ (A \xrightarrow{g} TB) \longmapsto (TA \xrightarrow{g^*} TB). \end{aligned}$$

The unit η of the adjunction is the unit of the monad, and the counit ϵ is the map $TA \xrightarrow{id_{TA}} TA$ in the category $\mathcal{C}$.

Lemma 2.5.6. *Conversely, given an adjunction (F,U,η,ϵ), there is a canonical monad arising from it, by setting $T = UF$. The unit of the monad is the unit of the adjunction, and the multiplication μ of the monad is $U\epsilon F : UFUF \dot{\to} UF$.*

2.6 Comonads

A dual notion to monad is the notion of comonad. We are going to use it extensively in the rest of the thesis.

Definition 2.6.1 (Mac Lane, 1998)**.** A *comonad* in a category $\mathcal{C}$ is a tuple (L,ϵ,δ) where $L : \mathcal{C} \to \mathcal{C}$ is a functor and $\epsilon : L \dot{\to} I$ and $\delta : L \dot{\to} L^2$ are natural transformations which render commutative the diagrams

$$\begin{tikzcd} LA \arrow[r, "\delta_A"] \arrow[d, "\delta_A"'] & L^2A \arrow[d, "L\delta_A"] \\ L^2A \arrow[r, "\delta_{LA}"'] & L^3A, \end{tikzcd} \tag{2.6.1}$$

$$\begin{tikzcd} & LA \arrow[dl, "id_{LA}"'] \arrow[d, "\delta_A"] \arrow[dr, "id_{LA}"] & \\ LA & L^2A \arrow[l, "\epsilon_{LA}"] \arrow[r, "L\epsilon_A"'] & LA. \end{tikzcd} \tag{2.6.2}$$

By analogy with monads, we define the map $(-)_*$

$$\frac{f : LA \to B}{f_* : LA \xrightarrow{\delta_A} L^2A \xrightarrow{Lf} LB}$$

Definition 2.6.2. Given a comonad (L,ϵ,δ) over a category $\mathcal{C}$, the *co-Kleisli category* $\mathcal{C}_L$ is defined as follows:

- the objects of $\mathcal{C}_L$ are the objects of $\mathcal{C}$,
- the set $\mathcal{C}_L(A,B)$ of morphisms from A to B in $\mathcal{C}_L$ is $\mathcal{C}(LA,B)$,

- the identity on A is $\epsilon_A : LA \to A$,
- The composition of $f \in \mathcal{C}_L(A, B)$ and $g \in \mathcal{C}_L(B, C)$ in $\mathcal{C}_L$ is $f_*; g : LA \to C$.

Definition 2.6.3. Given a comonad (L, δ, ϵ) on a category $\mathcal{C}$, an *L-coalgebra* is a pair $(A, h_A : A \to LA)$ where A is an object and h_A is an arrow of $\mathcal{C}$, such that both

$$\begin{tikzcd} A \arrow[r, "h_A"] \arrow[d, "h_A"'] & LA, \arrow[d, "\delta_A"] \\ LA \arrow[r, "Lh_A"'] & L^2A \end{tikzcd} \tag{2.6.3}$$

$$\begin{tikzcd} A \arrow[rr, "h_A"] \arrow[dr, "id_A"'] & & LA \arrow[dl, "\epsilon_A"] \\ & A & \end{tikzcd} \tag{2.6.4}$$

commute. A morphism $f : (A, h_A) \to (B, h_B)$ of L-coalgebras is an arrow $f : A \to B$ of $\mathcal{C}$ which renders commutative the diagram

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[d, "h_A"'] & B \arrow[d, "h_B"] \\ LA \arrow[r, "Lf"'] & LB. \end{tikzcd} \tag{2.6.5}$$

The category $\mathcal{C}^L$ of L-coalgebras of $\mathcal{C}$ is the category whose objects are L-coalgebras and whose maps are morphisms of L-coalgebras. It is called the *co-Eilenberg-Moore category* of the comonad L.

Lemma 2.6.4. *Given a comonad (L, δ, ϵ) on a category $\mathcal{C}$, for each object A in $\mathcal{C}$ the pair $(LA, \delta_A : LA \to L^2A)$ forms a coalgebra. Given any map $f : A \to B$, the map $Lf : (LA, \delta_A) \to (LB, \delta_B)$ is a coalgebra map.* □

Lemma 2.6.5. *Given a category $\mathcal{C}$ and a comonad (L, δ, ϵ) on it, the coalgebra (LA, δ_A) has the following universal property: If (X, h_X) is any L-coalgebra, and if $f : X \to A$ is any map in $\mathcal{C}$, there exists a unique coalgebra map $f^\sharp : (X, h_X) \to (LA, \delta_A)$ such that*

$$\begin{tikzcd} LA \arrow[r, "\epsilon_A"] & A. \\ X \arrow[u, dotted, "f^\sharp"] \arrow[ur, "f"'] & \end{tikzcd} \tag{2.6.6}$$

Moreover, $f^\sharp = h_X; Lf$. □

Corollary 2.6.6. *Given a category $\mathcal{C}$ and a comonad (L, δ, ϵ), there is a one-to-one correspondence between coalgebra morphisms $g : (LA, \delta_A) \to (LB, \delta_B)$ and maps of $\mathcal{C}$ of the form $f : LA \to B$.* □

Definition 2.6.7. The coalgebra (LA, δ_A) in Lemma 2.6.4 is called a *(co)free coalgebra.*

Corollary 2.6.8. *There is a full embedding of $\mathcal{C}_L$ into $\mathcal{C}^L$ sending the object A of $\mathcal{C}_L$ to (LA, δ_A) and sending the morphism $f : LA \to B$ into $f_* : LA \to LB$.* □

Lemma 2.6.9. *Given an adjunction (F, U, η, ϵ) there is a canonical comonad arising from it, by setting $T = FU$, the counit of the comonad is the counit of the adjunction, and the comultiplication δ of the comonad is $F\eta U : FUFU \dot{\to} FU$.* □

2.7 Symmetric Monoidal Categories and Comonoids

2.7.1 Monoidal Categories

Definition 2.7.1 (Mac Lane, 1998)**.** A *monoidal category* $\mathcal{C}$ is a tuple $(\mathcal{C}, \otimes, \top, \alpha, \lambda, \rho)$ where $\mathcal{C}$ is a category, $\otimes : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ is a functor, $\top$ is an object of $\mathcal{C}$ and α, λ, ρ are three natural isomorphisms

$$\alpha_{A,B,C} : A \otimes (B \otimes C) \to (A \otimes B) \otimes C,$$
$$\lambda_A : \top \otimes A \to A, \qquad \rho_A : A \otimes \top \to A$$

such that the diagrams

(2.7.1)
$$\begin{tikzcd}
 & (A \otimes B) \otimes (C \otimes D) \arrow[dr, "\alpha"] & \\
A \otimes (B \otimes (C \otimes D)) \arrow[ur, "\alpha"] \arrow[d, "id \otimes \alpha"'] & & ((A \otimes B) \otimes C) \otimes D \\
A \otimes ((B \otimes C) \otimes D) \arrow[rr, "\alpha"'] & & (A \otimes (B \otimes C)) \otimes D, \arrow[u, "\alpha \otimes id"']
\end{tikzcd}$$

(2.7.2)
$$\begin{tikzcd}
A \otimes (\top \otimes C) \arrow[rr, "\alpha"] \arrow[dr, "id \otimes \lambda"'] & & (A \otimes \top) \otimes C \arrow[dl, "\rho \otimes id"] \\
 & A \otimes C, &
\end{tikzcd}$$

(2.7.3)
$$\begin{tikzcd}
\top \otimes \top \arrow[d, bend right, "\rho"'] \arrow[d, bend left, "\lambda"] \\
\top
\end{tikzcd}$$

commute. We call the category *strict monoidal* when $\alpha_{A,B,C}$, λ_A and ρ_A are identity morphisms.

Definition 2.7.2 (Mac Lane, 1998)**.** A monoidal category is said to be *symmetric* when it is equipped with a natural isomorphism $\sigma_{A,B} : A \otimes B \to B \otimes A$ such that the diagrams

(2.7.4)
$$\begin{tikzcd}
A \otimes B \arrow[r, bend left, "\sigma_{A,B}"] & B \otimes A, \arrow[l, bend left, "\sigma_{B,A}"]
\end{tikzcd}$$

(2.7.5)
$$\begin{tikzcd}
B \otimes \top \arrow[r, "\sigma_{B,\top}"] \arrow[rr, bend right, "\rho_B"'] & \top \otimes B \arrow[r, "\lambda_B"] & B,
\end{tikzcd}$$

(2.7.6)
$$\begin{tikzcd}
A \otimes (B \otimes C) \arrow[r, "\alpha"] \arrow[d, "id \otimes \sigma"'] & (A \otimes B) \otimes C \arrow[r, "\sigma"] & C \otimes (A \otimes B) \arrow[d, "\alpha"] \\
A \otimes (C \otimes B) \arrow[r, "\alpha"] & (A \otimes C) \otimes B \arrow[r, "\sigma \otimes id"] & (C \otimes A) \otimes B
\end{tikzcd}$$

commute.

2.7.2 Commutative Comonoids

Definition 2.7.3. In a symmetric monoidal category $\mathcal{C}$, a *commutative comonoid object* is an object A of $\mathcal{C}$ equipped with two arrows $\diamondsuit_A : A \to \top$ and $\triangle_A : A \to A \otimes A$ such that the following diagrams commute

(2.7.7)
$$\begin{tikzcd}
 & A \arrow[dl, "\triangle_A"'] \arrow[dr, "\triangle_A"] & \\
A \otimes A \arrow[d, "A \otimes \triangle_A"'] & & A \otimes A \arrow[d, "\triangle_A \otimes A"] \\
A \otimes (A \otimes A) \arrow[rr, "\alpha"] & & (A \otimes A) \otimes A,
\end{tikzcd}$$

$$(2.7.8)\quad \begin{tikzcd} A\otimes\top \arrow[d, "\rho_A"'] & A\otimes A \arrow[l, "A\otimes\diamondsuit_A"'] \arrow[r, "\diamondsuit_A\otimes A"] & \top\otimes A \arrow[d, "\lambda_A"] \\ A \arrow[r, phantom, "="] & A \arrow[u, "\triangle_A"'] \arrow[r, phantom, "="] & A \end{tikzcd}, \qquad \begin{tikzcd} A\otimes A \arrow[rr, "\sigma_{A,A}"] & & A\otimes A. \\ & A \arrow[ul, "\triangle_A"] \arrow[ur, "\triangle_A"'] & \end{tikzcd} \quad (2.7.9)$$

A morphism $f : (A, \diamondsuit_A, \triangle_A) \to (B, \diamondsuit_A, \triangle_B)$ of commutative comonoids is an arrow $f : A \to B$ in $\mathcal{C}$ such that

$$(2.7.10)\quad \begin{tikzcd} A \arrow[r, "f"] \arrow[d, "\triangle_A"'] & B, \arrow[d, "\triangle_B"] \\ A\otimes A \arrow[r, "f\otimes f"'] & B\otimes B \end{tikzcd} \qquad \begin{tikzcd} A \arrow[rr, "f"] \arrow[dr, "\diamondsuit_A"'] & & B. \arrow[dl, "\diamondsuit_B"] \\ & \top & \end{tikzcd} \quad (2.7.11)$$

Definition 2.7.4 (Selinger, 2001)**.** We call a *diagonal structure* on a symmetric monoidal category $(\mathcal{C}, \otimes, \top)$ a family $\{(A, \diamondsuit_A, \triangle_A)\}_{A\in|\mathcal{C}|}$ of commutative comonoids that respects the symmetric monoidal structure of $\mathcal{C}$, that is, such that the following diagrams commute:

$$(2.7.12)\quad \begin{tikzcd} & A\otimes B \arrow[dl, "\diamondsuit_A\otimes\diamondsuit_B"'] \arrow[dr, "\diamondsuit_{A\otimes B}"] & \\ \top\otimes\top \arrow[rr, "\lambda_\top=\rho_\top"] & & \top, \end{tikzcd} \qquad \begin{tikzcd} \top \arrow[d, bend right, "id_\top"'] \arrow[d, bend left, "\diamondsuit_\top"] \\ \top, \end{tikzcd} \quad (2.7.13)$$

$$(2.7.14)\quad \begin{tikzcd} & A\otimes B \arrow[dl, "\triangle_A\otimes\triangle_B"'] \arrow[dr, "\triangle_{A\otimes B}"] & \\ (A\otimes A)\otimes(B\otimes B) \arrow[rr, "sw_{A,B}"] & & (A\otimes B)\otimes(A\otimes B), \end{tikzcd}$$

where sw is defined as the natural transformation constructed from α's and σ's swapping the two middle objects, that is $\alpha; (\alpha^{-1}\otimes id); ((id\otimes\sigma)\otimes id); (\alpha\otimes id); \alpha^{-1}$.

Theorem 2.7.5 (Selinger, 2001)**.** *Let $(\mathcal{C}, \otimes, \top, \alpha, \lambda, \rho, \sigma)$ be a symmetric monoidal category. It is a cartesian category if and only if there there exist two natural transformations $\triangle_A : A \to A\otimes A$ and $\diamondsuit_A : A \to \top$ such that $\{(A, \triangle_A, \diamondsuit_A)\}_{A\in|\mathcal{C}|}$ forms a diagonal structure. In this case the cartesian structure is given by*

$$\bigcirc_A : A \xrightarrow{\diamondsuit_A} \top, \qquad \langle f, g\rangle : C \xrightarrow{\triangle_C} C\otimes C \xrightarrow{f\otimes g} A\otimes B,$$

$$\pi^1_{A,B} : A\otimes B \xrightarrow{A\otimes\diamondsuit_B} A\otimes\top \xrightarrow{\rho_A} A, \qquad \pi^2_{A,B} : A\otimes B \xrightarrow{\diamondsuit_A\otimes B} \top\otimes B \xrightarrow{\lambda_B} B,$$

where $f : C \to A$ and $g : C \to B$. Given a cartesian structure, we construct the diagonal structure with $\triangle_A = \langle id_A, id_A\rangle$ and $\diamondsuit_A = \bigcirc_A$. □

2.8 Monoidal Comonads and Coalgebras

This section is taken from (Eilenberg and Kelly, 1965).

Definition 2.8.1. A *(lax) monoidal functor* F between two monoidal categories $(\mathcal{C}, \otimes, \top)$ and $(\mathcal{D}, \otimes', \top')$ consists of a functor $F : \mathcal{C} \to \mathcal{D}$ together with two natural transformations

$$d^F_{A,B} : FA\otimes' FB \to F(A\otimes B), \qquad d^F_\top : \top' \to F\top,$$

called *coherence maps*, with the following coherence equations:

$$\begin{tikzcd}
(FA \otimes' FB) \otimes' FC \arrow[r, "\alpha"] \arrow[d, "d^F_{A,B} \otimes' FC"'] & FA \otimes' (FB \otimes' FC) \arrow[d, "FA \otimes' d^F_{B,C}"] \\
F(A \otimes B) \otimes' FC \arrow[d, "d^F_{A\otimes B, C}"'] & FA \otimes' F(B \otimes C) \arrow[d, "d^F_{A, B\otimes C}"] \\
F((A \otimes B) \otimes C) \arrow[r, "F\alpha"'] & F(A \otimes (B \otimes C)),
\end{tikzcd} \tag{2.8.1}$$

$$\begin{tikzcd}
FA \otimes' \top' \arrow[r, "FA \otimes' d"] \arrow[d, "\rho"'] & FA \otimes' F\top \arrow[d, "d^F_{A,\top'}"] \\
FA & F(A \otimes \top), \arrow[l, "F\rho"]
\end{tikzcd} \tag{2.8.2}$$

$$\begin{tikzcd}
\top' \otimes' FB \arrow[r, "d \otimes' FB"] \arrow[d, "\lambda"'] & F\top \otimes' FB \arrow[d, "d^F_{\top', B}"] \\
FB & F(\top \otimes B). \arrow[l, "F\lambda"]
\end{tikzcd} \tag{2.8.3}$$

If the categories are symmetric, the functor is said to be *lax symmetric monoidal* if the following diagram also commutes:

$$\begin{tikzcd}
FA \otimes' FB \arrow[r, "\sigma"] \arrow[d, "d^F_{A,B}"'] & FB \otimes' FA \arrow[d, "d^F_{B,A}"] \\
F(A \otimes B) \arrow[r, "F\sigma"'] & F(B \otimes A).
\end{tikzcd} \tag{2.8.4}$$

We call the functor *strong monoidal* if the natural transformations $d^F_{A,B}$ and $d^F_\top$ are isomorphisms.

Theorem 2.8.2 (Mac Lane, 1998, p. 257)**.** *Any monoidal category $\mathcal{C}$ is categorically equivalent, via a strong monoidal functor $F : \mathcal{C} \to \mathcal{D}$ and a strong monoidal functor $G : \mathcal{D} \to \mathcal{C}$, to a strict monoidal category $\mathcal{D}$.* □

Lemma 2.8.3. *Let $(\mathcal{C}, \otimes_\mathcal{C}, \top_\mathcal{C})$, $(\mathcal{D}, \otimes_\mathcal{D}, \top_\mathcal{D})$ and $(\mathcal{E}, \otimes_\mathcal{E}, \top_\mathcal{E})$ be symmetric monoidal categories.*

1. *The identity functor $1_\mathcal{C} : \mathcal{C} \to \mathcal{C}$ together with $d^{1_\mathcal{C}}_{\top_\mathcal{C}} = id_{\top_\mathcal{C}} : \top_\mathcal{C} \to \top_\mathcal{C}$ and $d^{1_\mathcal{C}}_{A\otimes_\mathcal{C} B} = id_{A\otimes_\mathcal{C} B} : A \otimes_\mathcal{C} B \to A \otimes_\mathcal{C} B$ is a monoidal functor.*

2. *The functor $\diamondsuit_\mathcal{C}$ from $\mathcal{C}$ to $\mathcal{C}$ sending A to $\top_\mathcal{C}$ and f to $id_{\top_\mathcal{C}}$ is monoidal, with*
$$\begin{aligned} d^{\diamondsuit_\mathcal{C}}_{\top_\mathcal{C}} &= id_{\top_\mathcal{C}} : \top_\mathcal{C} \to \top_\mathcal{C}, \\ d^{\diamondsuit_\mathcal{C}}_{A,B} &= \lambda_{\top_\mathcal{C}} = \rho_{\top_\mathcal{C}} : \diamondsuit_\mathcal{C} A \otimes_\mathcal{C} \diamondsuit_\mathcal{C} B = \top_\mathcal{C} \otimes_\mathcal{C} \top_\mathcal{C} \to \top_\mathcal{C} = \diamondsuit_\mathcal{C}(A \otimes_\mathcal{C} B). \end{aligned}$$

3. *The diagonal functor $\triangle_\mathcal{C}$ from $\mathcal{C}$ to $\mathcal{C}$ sending A to $A \otimes_\mathcal{C} A$ and f to $f \otimes_\mathcal{C} f$ is monoidal with*
$$\begin{aligned} d^{\triangle_\mathcal{C}}_{\top_\mathcal{C}} &: \top \xrightarrow{\lambda_\top^{-1} = \rho_\top^{-1}} \top \otimes \top, \\ d^{\triangle_\mathcal{C}}_{A,B} &: \triangle_\mathcal{C} A \otimes \triangle_\mathcal{C} B = (A \otimes A) \otimes (B \otimes B), \\ &\xrightarrow{sw_{A,B}} (A \otimes B) \otimes (A \otimes B) = \triangle_\mathcal{C}(A \otimes B). \end{aligned}$$

4. *If $(F, d^F_{\top_\mathcal{C}}, d^F_{A,B}) : \mathcal{C} \to \mathcal{D}$ and $(G, d^G_{\top_\mathcal{D}}, d^G_{A,B}) : \mathcal{D} \to \mathcal{E}$ are any monoidal functor, so is $G \circ F : \mathcal{C} \to \mathcal{E}$ together with*
$$\begin{aligned} d^{G\circ F}_{\top_\mathcal{C}} &: \top_\mathcal{E} \xrightarrow{d^G_{\top_\mathcal{D}}} G\top_\mathcal{D} \xrightarrow{Gd^F_{\top_\mathcal{C}}} GF\top_\mathcal{C}, \\ d^{G\circ F}_{A,B} &: GFA \otimes_\mathcal{E} GFB \xrightarrow{d^G_{FA,FB}} G(FA \otimes_\mathcal{D} FB) \xrightarrow{Gd^F_{A,B}} GF(A \otimes_\mathcal{C} B). \end{aligned}$$

□

Definition 2.8.4. In the setting of Definition 2.8.1, if (F, d) and (F', d') are two symmetric monoidal functors, we say that a natural transformation $n : \mathcal{C} \dot{\to} \mathcal{D}$ is *monoidal* if the following diagrams commute:

$$\begin{tikzcd} FA \otimes' FB \arrow[r, "n_A \otimes' n_B"] \arrow[d, "d_{A,B}"'] & F'A \otimes' F'B \arrow[d, "d'_{A,B}"] \\ F(A \otimes B) \arrow[r, "n_{A\otimes B}"'] & F'(A \otimes B), \end{tikzcd} \tag{2.8.5}$$

$$\begin{tikzcd} & \top' \arrow[dl, "d"'] \arrow[dr, "d'"] & \\ F\top \arrow[rr, "n_\top"'] & & F'\top. \end{tikzcd} \tag{2.8.6}$$

Remark 2.8.5. The natural transformation n in Definition 2.8.4 could be called *symmetric* since the category is symmetric. However, since σ is not a structure but a property of monoidal functors, it does not yield any equation. In that sense the adjective "symmetric" does not apply to n.

Definition 2.8.6 (Benton et al., 1992; Bierman, 1993)**.** Let $(\mathcal{C}, \otimes, \top)$ be a (symmetric) monoidal category. A *(symmetric) monoidal comonad* on $\mathcal{C}$ is a comonad (L, δ, ϵ):

- equipped with two natural transformations $d_{A,B} : LA \otimes LB \to L(A \otimes B)$ and $d_\top : L\top \to \top$ making (L, d) a lax (symmetric) monoidal functor,
- such that δ and ϵ are monoidal natural transformations, i.e. such that the following diagrams commute:

$$\begin{tikzcd} LA \otimes LB \arrow[rr, "d_{A,B}"] \arrow[dr, "\epsilon_A \otimes \epsilon_B"'] & & L(A \otimes B), \arrow[dl, "\epsilon_{A\otimes B}"] \\ & A \otimes B & \end{tikzcd} \tag{2.8.7}$$

$$\begin{tikzcd} \top \arrow[rr, "d_\top"] \arrow[dr, "id_\top"'] & & L\top, \arrow[dl, "\epsilon_\top"] \\ & \top & \end{tikzcd} \tag{2.8.8}$$

$$\begin{tikzcd} LA \otimes LB \arrow[rr, "d_{A,B}"] \arrow[d, "\delta_A \otimes \delta_A"'] & & L(A \otimes B), \arrow[d, "\delta_{A\otimes B}"] \\ L^2A \otimes L^2B \arrow[r, "d_{LA,LB}"'] & L(LA \otimes LB) \arrow[r, "Ld_{A,B}"'] & L^2(A \otimes B) \end{tikzcd} \tag{2.8.9}$$

$$\begin{tikzcd} \top \arrow[r, "d_\top"] \arrow[d, "d_\top"'] & L\top. \arrow[d, "\delta_\top"] \\ L\top \arrow[r, "Ld_\top"'] & L^2\top \end{tikzcd} \tag{2.8.10}$$

Theorem 2.8.7. *Let $(\mathcal{C}, \otimes, \top)$ be a symmetric monoidal category with structure α, λ, ρ and σ. Let (L, δ, ϵ, d) be a symmetric monoidal comonad on $\mathcal{C}$. Then the co-Eilenberg-Moore category $\mathcal{C}^L$ is also symmetric monoidal when equipped with*

- *the functor $\otimes' : \mathcal{C}^L \times \mathcal{C}^L \to \mathcal{C}^L$, defined as $(A, h_A) \otimes' (B, h_B) = (A \otimes B, (h_A \otimes h_B); d_{A,B})$ on objects and $f \otimes' g = f \otimes g$ on arrows.*
- *the unit object $\top' = (\top, d_\top)$,*
- *the structure maps*

$$\alpha'_{(A,h_A),(B,h_B),(C,h_C)} = \alpha_{A,B,C}, \qquad \lambda'_{(A,h_A)} = \lambda_A,$$
$$\sigma'_{(A,h_A),(B,h_B)} = \sigma_{A,B}, \qquad \rho'_{(A,h_A)} = \rho_A.$$

□

Definition 2.8.8. In Theorem 2.8.7 we call $\otimes'$ the *induced tensor* and $(\otimes', \top', \alpha, \lambda, \rho, \sigma)$ the *induced symmetric monoidal structure*. We identify $\otimes'$ with $\otimes$ and $\top'$ with $\top$.

Chapter 3

Quantum Computation

Whether the notion of data is thought of concretely or abstractly, it is usually supposed to behave classically: a piece of data is supposed to be clonable, erasable, readable as many time as needed, and is not supposed to change when left untouched.

Quantum computation is a paradigm where data can be encoded with a medium governed by the law of quantum physics. Although only rudimentary quantum computers have been built so far, the laws of quantum physics are mathematically well described. It is therefore possible to try to understand the capabilities of quantum computation, and quantum information turns out to behave substantially differently from usual (classical) information.

In this chapter we will describe the mathematics needed for quantum computation, and we will have a brief overview of the theory of quantum computation. A more complete study of the subject can be found in (Nielsen and Chuang, 2002) and in (Preskill, 1999).

3.1 Mathematical Formalism

This section will cover the common wisdom we need about Hilbert spaces and their operators. All considered vector spaces are assumed to be finite-dimensional. For a more complete mathematical analysis of the subject, see e.g. (Lang, 1993, Ch. V).

3.1.1 Generalities on Finite Dimensional Hilbert Spaces

Definition 3.1.1 (Lang, 1993, p. 95)**.** Let $\mathcal{E}$ be a finite-dimensional vector space over the complex numbers $\mathbb{C}$. A *sesquilinear form* on $\mathcal{E}$ is a map $\phi : \mathcal{E} \times \mathcal{E} \to \mathbb{C}$, which is linear in y: $\phi(x, \alpha y + \beta z) = \alpha\phi(x, y) + \beta\phi(x, z)$, and semi-linear in x: $\phi(\alpha x + \beta t, y) = \overline{\alpha}\phi(x, y) + \bar{\beta}\phi(x, t)$, where $\overline{\alpha}$ is the complex conjugation of α: $\overline{a + bi} = a - bi$, for $a, b \in \mathbb{R}$. We say the form is *hermitian* if moreover $\phi(x, y) = \overline{\phi(y, x)}$. It is *positive definite* if $\phi(x, x) \geqslant 0$ with equality if and only if $x = 0$. If a form is hermitian and positive definite, we will call it a *scalar product*. In this case we often denote it as $\langle . | . \rangle$.

Definition 3.1.2. Let $\mathcal{E}$ be a complex vector space. A *norm* on $\mathcal{E}$ is a map $\|.\| : \mathcal{E} \to \mathbb{R}$ such that

1. $\|v\| \geqslant 0$ for all v, with equality if and only if $v = 0$;
2. $\|\alpha v\| = |\alpha|\|v\|$;
3. $\|v + w\| \leqslant \|v\| + \|w\|$.

Lemma 3.1.3. *Given a scalar product $\langle .|. \rangle$ on a complex vector space $\mathcal{E}$, the map $\|.\| : \mathcal{E} \to \mathbb{C}$ mapping v to $\sqrt{\langle v|v \rangle}$ is a norm.* □

Definition 3.1.4. Given a scalar product on a complex vector space $\mathcal{E}$. The norm in Lemma 3.1.3 is called the *induced norm*.

Definition 3.1.5. A vector space together with a norm is called a *normed vector space*.

Definition 3.1.6. Let $\mathcal{H}$ be a finite dimensional vector space over the field $\mathbb{C}$ of complex numbers. The space $\mathcal{H}$ is a *Hilbert space* if it comes equipped with a scalar product. When speaking about a given Hilbert space, we will always consider it to be normed by the induced norm.

Definition 3.1.7. An *operator* A on a Hilbert space $\mathcal{H}$ is a linear map $A : \mathcal{H} \to \mathcal{H}$. A *functional* on $\mathcal{H}$ is a linear map $\lambda : \mathcal{H} \to \mathbb{C}$. The space of functionals on $\mathcal{H}$ is denoted by $\mathcal{H}^*$.

Lemma 3.1.8. *Let λ be a functional on $\mathcal{H}$. Then there exists $x_0 \in \mathcal{H}$ such that for all $y \in \mathcal{H}$, $\lambda(y) = \langle x_0|y \rangle$.* □

Lemma 3.1.9. *Let ϕ be a sesquilinear form on a Hilbert space $\mathcal{H}$. Then the map $\lambda_x : y \mapsto \phi(x, y)$ is a functional.* □

Let A be an operator on $\mathcal{H}$. Define ϕ_A by $\phi_A(x, y) = \langle x|Ay \rangle$. Then ϕ_A is a sesquilinear form on $\mathcal{H}$. For each x, $\lambda_x(y) = \phi_A(x, y)$ is a functional. Therefore there exists some x_0 such that $\lambda_x(y) = \langle x_0|y \rangle$. Define A^* as $A^*x = x_0$: we have

$$\langle x|Ay \rangle = \langle A^*x|y \rangle.$$

Definition 3.1.10. One calls A^* the *adjoint* of A. If $A = A^*$, we say A is *self-adjoint*, or *hermitian*.

Lemma 3.1.11 (Lang, 1993, p. 106)**.** *The map $A \mapsto A^*$ satisfies the following properties:*

$$(A + B)^* = A^* + B^*, \qquad A^{**} = A,$$
$$(\alpha A)^* = \overline{\alpha} A^*, \qquad (AB)^* = B^* A^*.$$

□

Lemma 3.1.12 (Lang, 1993, p. 107)**.** *If for all x, $\langle x|Ax \rangle = 0$ then $A = 0$.*

Theorem 3.1.13 (Spectral Theorem, Lang, 2002)**.** *Let A be a hermitian operator on a finite dimensional Hilbert space $\mathcal{H}$. Then $\mathcal{H}$ has an orthogonal basis consisting of eigenvectors of A.* □

Definition 3.1.14. We say that an operator A on $\mathcal{H}$ is *positive* if it is hermitian and if all of its eigenvalues are positive. Equivalently, A is positive if the form ϕ_A is positive definite. We say A is a *density matrix* if moreover, the sum of the eigenvalues is less or equal to 1.

Lemma 3.1.15. *Let $\mathcal{H}$ be a finite dimensional Hilbert space, and A an invertible operator. Then the following conditions are equivalent:*

1. *$A^* = A^{-1}$;*
2. *$\|Ax\| = \|x\|$ for all $x \in \mathcal{H}$;*
3. *$\langle Ax|Ay \rangle = \langle x|y \rangle$ for all $x, y \in \mathcal{H}$;*
4. *$\|Ax\| = 1$ for every unit vector $x \in \mathcal{H}$.* □

Definition 3.1.16. An invertible operator satisfying the conditions of Lemma 3.1.15 is said to be *unitary*.

3.1.2 Tensor Products of Hilbert Spaces

Definition 3.1.17 (Dummit and Foote, 1999)**.** Consider two finite Hilbert spaces $\mathcal{V}$ and $\mathcal{W}$ with respective bases $(e_1, \ldots, e_n)$ and $(f_1, \ldots, f_k)$. We define the *tensor product of* $\mathcal{V}$ *and* $\mathcal{W}$, denoted $\mathcal{V} \otimes \mathcal{W}$, as the space generated by the basis of syntactic symbols:

$$(e_1 \otimes f_1, \ldots, e_1 \otimes f_k, \ldots\ldots, e_n \otimes f_1, \ldots, e_n \otimes f_k).$$

The tensor product of two elements $v = \sum_i \lambda_i \cdot e_i$ and $w = \sum_j \mu_j \cdot f_j$ is

$$v \otimes w = \sum_{i,j} \lambda_i \mu_j \cdot e_i \otimes f_j.$$

We define the scalar product in $V \otimes W$ as follows:

$$\langle e_i \otimes f_j | e_k \otimes f_l \rangle = \langle e_i | e_k \rangle \langle f_j | f_l \rangle.$$

Lemma 3.1.18. *If $\mathcal{U}$, $\mathcal{V}$ and $\mathcal{W}$ are finite-dimensional Hilbert spaces, there exists a one-to-one correspondence Φ between linear maps $\mathcal{U} \otimes \mathcal{V} \to \mathcal{W}$ and linear maps $\mathcal{U} \to \mathcal{V} \otimes \mathcal{W}$*[1]*.* □

Proof. Choose a basis $(e_i)_i$ for $\mathcal{V}$ and a basis $(f_j)_j$ for $\mathcal{W}$. Then if $f : \mathcal{U} \otimes \mathcal{V} \to \mathcal{W}$, we define $g = \Phi(f) : \mathcal{U} \to \mathcal{V} \otimes \mathcal{W}$ as the map

$$g(u) = \sum_i e_i \otimes f(u \otimes e_i).$$

If $g : \mathcal{U} \to \mathcal{V} \otimes \mathcal{W}$, and if

$$g(u) = \sum_{i,j} \rho_{i,j} \cdot e_i \otimes f_j,$$

we define $f = \Phi^{-1}(g) : \mathcal{U} \otimes \mathcal{V} \to \mathcal{W}$ as the map

$$f(u \otimes e_i) = \sum_j \rho_{i,j} \cdot e_i \otimes f_j.$$ □

Convention 3.1.19. If $A \in \mathbb{C}^{n \times n}$ and $B \in \mathbb{C}^{k \times k}$ are two square matrices of respective size $n \times n$ and $k \times k$ seen as operators respectively on $\mathbb{C}^n$ and on $\mathbb{C}^k$, the matrix $A \otimes B$ is an operator on $\mathbb{C}^{nk}$ and we represent it as the matrix of blocks

$$A \otimes B = \left(\begin{array}{c|c|c} a_{1,1}B & \cdots & a_{1,n}B \\ \hline \vdots & \ddots & \vdots \\ \hline a_{n,1}B & \cdots & a_{n,n}B \end{array} \right).$$

3.1.3 Completely Positive Maps

In this section, we focus primarily on Hilbert spaces of the form $\mathbb{C}^{n \times n}$, spaces of of $n \times n$-matrices. We choose as basis for $\mathbb{C}^{n \times n}$ the canonical basis $(E_{i,j})_{i,j \in \{1,\ldots,n\}}$, where $(E_{i,j})_{k,l} = 0$ for all k and l except when $k = i$ and $l = j$. We use the notations and definitions of (Selinger, 2004b).

Definition 3.1.20. A linear map $F : \mathbb{C}^{n \times n} \to \mathbb{C}^{k \times k}$ is said to be a *positive map* if for all positive matrices A, $F(A)$ is positive.

[1] Note that we identified $\mathcal{V}$ with $\mathcal{V}^*$. Although the result is correct since we are in finite dimension, we lose the basis Independence of the correspondence

Definition 3.1.21. A linear map $F : \mathbb{C}^{n\times n} \to \mathbb{C}^{k\times k}$ is said to be *completely positive* if for all i, if id_i is the identity on $i \times i$-matrices, $id_i \otimes F$ is a positive map $\mathbb{C}^{ni\times ni} \to \mathbb{C}^{ki\times ki}$.

Definition 3.1.22. The *characteristic matrix* of a linear map $F : \mathbb{C}^{n\times n} \to \mathbb{C}^{k\times k}$ is defined as the $nk \times nk$-matrix of blocks

$$\chi_F = \left(\begin{array}{c|c|c} F(E_{1,1}) & \cdots & F(E_{1,n}) \\ \hline \vdots & \ddots & \vdots \\ \hline F(E_{n,1}) & \cdots & F(E_{n,n}) \end{array}\right).$$

Lemma 3.1.23. *Let $F : \mathbb{C}^{n\times n} \to \mathbb{C}^{k\times k}$ be a linear operator, and let χ_F be its characteristic matrix. Then $F(A) = UAU^*$ for some $U \in \mathbb{C}^{k\times n}$ if and only if there exists a vector $v \in \mathbb{C}^{nk}$ such that $vv^* = \chi_F$.* □

Theorem 3.1.24 (Choi, 1975)**.** *Let $F : \mathbb{C}^{n\times n} \to \mathbb{C}^{k\times k}$ be a linear operator, and let χ_F be its characteristic matrix. Then the following are equivalent:*

1. *F is completely positive*
2. *χ_F is positive*
3. *F is of the form $F(A) = \sum_i U_i A U_i^*$ for some finite sequence of matrices $U_1, \ldots U_l$ in $\mathbb{C}^{k\times n}$.* □

3.1.4 Superoperators

The following definitions and results are from (Selinger, 2004b).

Definition 3.1.25 (Löwner order)**.** We define a relation $\sqsubseteq$ on $\mathbb{C}^{n\times n}$ as follows. Let A, B be matrices in $\mathbb{C}^{n\times n}$. Then $A \sqsubseteq B$ if and only if $B - A$ is positive.

Definition 3.1.26. A completely positive map $F : \mathbb{C}^{n\times n} \to \mathbb{C}^{k\times k}$ is called a *superoperator* if for all positive matrices $A \in \mathbb{C}^{n\times n}$, we have $\mathbf{tr}(F(A)) \leqslant \mathbf{tr}(A)$.

Theorem 3.1.27 (Kraus Representation Theorem)**.** *Let $F : \mathbb{C}^{n\times n} \to \mathbb{C}^{k\times k}$ be a completely positive operator. The following are equivalent:*

1. *F is a superoperator;*
2. *$(id_n \otimes \mathbf{tr}_k)(\chi_F) \sqsubseteq I_n$, where I_n is the identity matrix and $\mathbf{tr}_k : \mathbb{C}^{k\times k} \to \mathbb{C}$ is the trace operation;*
3. *F is of the form*
$$F(A) = \sum_i U_i A U_i^*$$
for matrices $U_1, \ldots, U_n$ with $\sum_i U_i^* U_i \sqsubseteq I_n$. □

Definition 3.1.28. A matrix $U \in \mathbb{C}^{k\times n}$ is called *contraction* if U is a sub-matrix of some unitary U', that is, if there exist matrices U_1, U_2, and U_3 such that

$$U' = \left(\begin{array}{c|c} U & U_1 \\ \hline U_2 & U_3 \end{array}\right)$$

is unitary. A linear function $F : \mathbb{C}^{n\times n} \to \mathbb{C}^{k\times k}$ is called a *contraction* if it is of the form $F(A) = UAU^*$ for some contraction U.

Lemma 3.1.29. *Let $U \in \mathbb{C}^{n\times m}$ be a contraction. Then the matrix*

$$\left(\frac{U}{0}\right) \in \mathbb{C}^{(n+1)\times m}$$

is also a contraction.

Proof. With the notation of the definition, consider the unitary matrix

$$\left(\begin{array}{c|c|c} U & 0 & U_1 \\ \hline 0 & 1 & 0 \\ \hline U_2 & 0 & U_3 \end{array}\right). \qquad \square$$

Lemma 3.1.30. *A matrix $U \in \mathbb{C}^{k\times n}$ is a contraction if and only if $UU^* \sqsubseteq I_k$, and if and only if $U^*U \sqsubseteq I_n$.* $\square$

Definition 3.1.31. If $\mathbf{tr}_n$ is the trace on matrices in $\mathbb{C}^{n\times n}$, we define the *partial trace operator* $\mathbb{C}^{kn\times kn} \to \mathbb{C}^{k\times k}$ as

$$\mathbf{tr}_n^k = \mathbf{tr}_n \otimes id_{\mathbb{C}^{k\times k}}.$$

Theorem 3.1.32 (Naimark Theorem)**.** *Every superoperator $F : \mathbb{C}^{n\times n} \to \mathbb{C}^{m\times m}$ can be factored as $F = E \circ G$, where G is a contraction and E is a partial trace operator.* $\square$

3.1.5 The 2-Dimensional Hilbert Space

We will now study in more detail what we saw in the previous section, applied to the 2-dimensional Hilbert space. We identify it with $\mathbb{C}^2$ and we call the canonical basis $e_1 = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ and $e_2 = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$. The results of this section can be found in (Preskill, 1999).

Lemma 3.1.33. *The set of hermitian matrices on $\mathcal{H}$ is spanned as a real vector space by the following basis:*

$$\mathbf{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \boldsymbol{\sigma}_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \boldsymbol{\sigma}_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \boldsymbol{\sigma}_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \qquad \square$$

Definition 3.1.34. The matrices $\boldsymbol{\sigma}_x, \boldsymbol{\sigma}_y, \boldsymbol{\sigma}_z$ are called the *Pauli matrices*.

Remark 3.1.35. The eigenvectors and eigenvalues of the Pauli matrices are as follows:

$$\begin{array}{lll} \boldsymbol{\sigma}_x\left(\begin{smallmatrix}1\\1\end{smallmatrix}\right) = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right), & \boldsymbol{\sigma}_y\left(\begin{smallmatrix}1\\i\end{smallmatrix}\right) = \left(\begin{smallmatrix}1\\i\end{smallmatrix}\right), & \boldsymbol{\sigma}_z\left(\begin{smallmatrix}0\\1\end{smallmatrix}\right) = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right), \\ \boldsymbol{\sigma}_x\left(\begin{smallmatrix}1\\-1\end{smallmatrix}\right) = -\left(\begin{smallmatrix}1\\-1\end{smallmatrix}\right), & \boldsymbol{\sigma}_y\left(\begin{smallmatrix}1\\-i\end{smallmatrix}\right) = -\left(\begin{smallmatrix}1\\-i\end{smallmatrix}\right), & \boldsymbol{\sigma}_z\left(\begin{smallmatrix}1\\0\end{smallmatrix}\right) = -\left(\begin{smallmatrix}1\\0\end{smallmatrix}\right). \end{array}$$

Definition 3.1.36. Let $\mathcal{H}$ be a Hilbert space. A *ray* $[v]$ in $\mathcal{H}$ is an equivalence class of vectors in $\mathcal{H}$ that differ by multiplication by non-zero complex scalars: so $[v]$ contains all αv, $\alpha \in \mathbb{C} \setminus \{0\}$.

Lemma 3.1.37. *There is a group action from the group of linear operators of $\mathcal{H}$ on the set of rays of $\mathcal{H}$.* $\square$

Lemma 3.1.38. *If v and w are orthogonal, so are any two elements of their respective rays.* $\square$

Lemma 3.1.39. *The rays of $\mathbb{C}^2$ are in one-to-one correspondence with the points on the unit sphere of $\mathbb{R}^3$. In spherical coordinates, a point (θ, φ) with $\theta \in [0, \pi], \psi \in [-\pi, \pi]$ is in correspondence with the ray of representative*

$$r(\theta, \varphi) = \begin{pmatrix} \cos\frac{\theta}{2} \\ e^{i\varphi}\sin\frac{\theta}{2} \end{pmatrix}. \tag{3.1.1}$$

The correspondence is pictured in Figure 3.1. $\square$

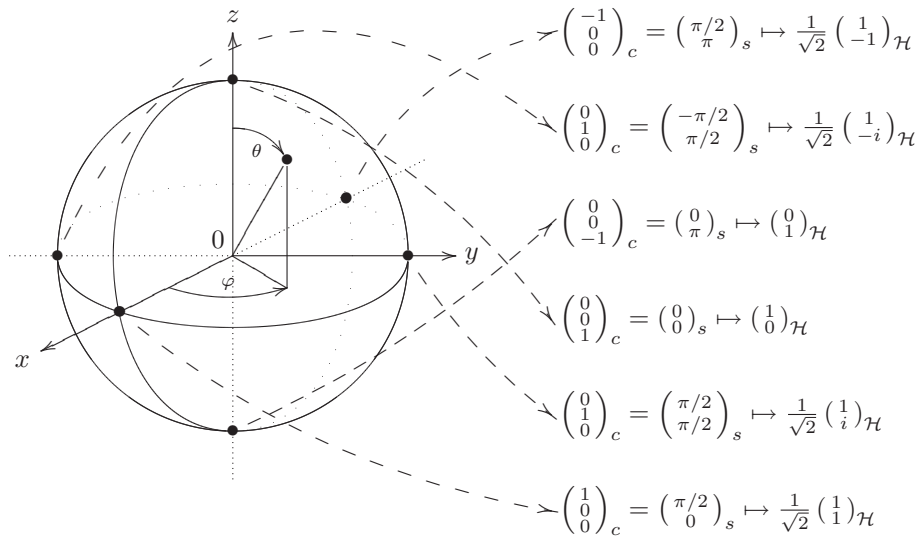


Figure 3.1: The Bloch sphere.

One calls this unit sphere the *Bloch sphere*. In this representation, the Pauli matrices acts on the rays as follows:

- $\boldsymbol{\sigma}_x$ sends $r(\theta,\varphi)$ onto
$$\begin{pmatrix} e^{i\varphi}\sin\frac{\theta}{2} \\ \cos\frac{\theta}{2} \end{pmatrix} \simeq \begin{pmatrix} \cos\frac{\pi-\theta}{2} \\ e^{-i\varphi}\sin\frac{\pi-\theta}{2}, \end{pmatrix}$$
thus sends the point (θ,φ) of the unit sphere of $\mathbb{R}^3$ (in spherical coordinate) to $(\pi-\theta,-\varphi)$, that is, in cartesian coordinates,
$$\begin{pmatrix} \sin\theta\cos\varphi \\ \sin\theta\sin\varphi \\ \cos\theta \end{pmatrix} \mapsto \begin{pmatrix} \sin\theta\cos\varphi \\ -\sin\theta\sin\varphi \\ -\cos\theta \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} x \\ -y \\ -z \end{pmatrix}.$$
The operator $\boldsymbol{\sigma}_x$ acts on the sphere as an axial symmetry around Ox. The two eigenvectors $\left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$ and $\left(\begin{smallmatrix}1\\-1\end{smallmatrix}\right)$ respectively correspond to the point $(1,0,0)$ and to the point $(-1,0,0)$ on the sphere.

- $\boldsymbol{\sigma}_y$ sends $r(\theta,\varphi)$ onto
$$\begin{pmatrix} e^{i\varphi}\sin\frac{\theta}{2} \\ \cos\frac{\theta}{2} \end{pmatrix} \simeq \begin{pmatrix} \cos\frac{\pi-\theta}{2} \\ e^{i(\pi-\varphi)}\sin\frac{\pi-\theta}{2}, \end{pmatrix}$$
i.e. maps (θ,φ) to $(\pi-\theta,\pi-\varphi)$, i.e. computes an axial symmetry around Oy. The two eigenvectors $\left(\begin{smallmatrix}1\\i\end{smallmatrix}\right)$ and $\left(\begin{smallmatrix}1\\-i\end{smallmatrix}\right)$ respectively correspond to the point $(0,1,0)$ and to the point $(0,-1,0)$ on the sphere.

- Finally, $\boldsymbol{\sigma}_z$ sends $r(\theta,\varphi)$ onto
$$\begin{pmatrix} e^{i\varphi}\sin\frac{\theta}{2} \\ \cos\frac{\theta}{2} \end{pmatrix} \simeq \begin{pmatrix} \cos\frac{\theta}{2} \\ e^{i(\pi-\varphi)}\sin\frac{\theta}{2}, \end{pmatrix}$$
i.e. maps (θ,φ) to $(\theta,\pi-\varphi)$, i.e. computes an axial symmetry around Oz. The two eigenvectors $\left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ and $\left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$ respectively correspond to the point $(0,0,1)$ and to the point $(0,0,1)$ on the sphere.

Lemma 3.1.40. *Two rays in $\mathbb{C}^2$ are orthogonal if and only if their representations on the Bloch sphere are antipodal.* □

3.2 Quantum Foundations

Quantum computation is somehow the art of making the theory of finite Hilbert spaces into a theory of computation: In quantum computation, information is encoded on states of quantum particles, states whose mathematical representation are rays in some Hilbert space.

Convention 3.2.1. In the following, we use the *Dirac convention* for writing vectors:

- vectors are written as *kets*: $|\phi\rangle$;
- functionals are written as *bras*: $\langle\psi|$;
- scalar products, which are by Lemma 3.1.9 the application of some functional to some vectors, are written as usual: $\langle\psi|\phi\rangle$.

3.2.1 Quantum Bits

One Quantum Bit

In classical computation, the smallest unit of data is the *bit*, element of the two-elements set $\{0,1\}$. In quantum computation, the smallest unit of data is a *quantum bit*, or *qubit*, defined as a ray in a 2-dimensional Hilbert space. We write the canonical basis of this space as $\{|0\rangle, |1\rangle\}$.

Using the results of Section 3.1.5, one says that

$$\begin{aligned}
(|0_z\rangle, |1_z\rangle) &= \Big(|0\rangle, |1\rangle\Big) && \text{is the } \textit{basis along } z,\\
(|0_x\rangle, |1_x\rangle) &= \Big(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \frac{1}{\sqrt{2}}(|1\rangle - |1\rangle)\Big) && \text{is the } \textit{basis along } x,\\
(|0_y\rangle, |1_y\rangle) &= \Big(\frac{1}{\sqrt{2}}(|0\rangle + i|1\rangle), \frac{1}{\sqrt{2}}(|1\rangle - i|1\rangle)\Big) && \text{is the } \textit{basis along } y.
\end{aligned}$$

Several Quantum Bits

When considering a system of n quantum bits, the state of the system is a ray in $\mathcal{H}^{\otimes n} = \mathcal{H} \otimes \cdots \otimes \mathcal{H}$, where $\mathcal{H}$ is the 2-dimensional Hilbert space. As stated in Definition 3.1.17, we take the canonical basis for $\mathcal{H}^{\otimes n}$ to be

$$\{|i_1 \dots i_n\rangle \mid i_1, \dots, i_n \in \{0,1\}\},$$

where we write $|i_1 \dots i_n\rangle$ for $|i_1\rangle \otimes \cdots \otimes |i_n\rangle$.

3.2.2 Operations

Measurements

Observables. In quantum physics, a measurement in a Hilbert space $\mathcal{H}$ is associated with a self-adjoint operator A of $\mathcal{H}$. Physicists call A an *observable*.

As we saw in Theorem 3.1.13, to a self-adjoint operator one can associate a set of orthogonal eigenspaces with their corresponding eigenvalues. If $|\phi\rangle$ is an eigenvector of A with eigenvalue a, the numerical outcome of the *measurement* of $|\phi\rangle$ with the observable A is the value a.

What does happen if ϕ is not an eigenvector? For simplicity assume that $\mathcal{H}$ is finite and that the basis of eigenvectors of A is $(|\phi_1\rangle, \ldots, |\phi_n\rangle)$ such that the eigenvalue corresponding to $|\phi_i\rangle$ is a_i. Then $|\phi\rangle$ can be re-written as $|\phi\rangle = \sum_{i=1}^{n} \alpha_i |\phi_i\rangle$, and measuring with A makes the system randomly collapse to a new state, namely a basis vector $|\phi_i\rangle$. This new state is $|\phi_i\rangle$ with probability $|\alpha_i|^2 = |\langle\phi|\phi_i\rangle|^2$. In this situation, the measurement outputs the numerical value a_i.

For the purpose of measurement the numerical eigenvalues have little importance. What is really important is that two eigenvalues corresponding to two distinct eigenspaces differ. When measuring a state of n quantum bits, we say that the *outcome* of the measurement of the state is $x_1 \ldots x_n$ if the state collapsed to $|x_1 \ldots x_n\rangle$. For example, we say:

> " The measurement of the state $|\phi\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$ in the standard basis results in the outcome 01 with probability $|\langle 01|\phi\rangle|^2 = |\beta|^2$. "

Pauli matrices. Given a quantum bit state $|\phi\rangle$, for $i = x, y, z$ the Pauli matrix $\boldsymbol{\sigma}_i$ provides an observable, whose measurement projects the state $|\phi\rangle$ in the basis along axis i.

From Section 3.1.5, the bases along x, y and z have special properties: if a quantum bit is in one of the basis element of the base along i, for some $i = x, y$ or z, then measuring the observable $\boldsymbol{\sigma}_j$, $j \neq i$ will yield 0 with probability $\frac{1}{2}$ and 1 with probability $\frac{1}{2}$. Thus somehow the information stored in the basis along i is not accessible from the two other ones.

Convention 3.2.2. In the remainder of the thesis, when speaking of "a measurement" without further precision, we always consider the measurement to be along the basis z.

Unitary gates

A quantum mechanical system, if not measured (by the operator or by the environment), evolves along a unitary path. That is, loosely speaking, all we can do is to apply rotations on the system. We call these actions *unitary gates*.

The ability to perform one gate or another depends on the physical implementation of the quantum bits. In general, only a few gates are needed to approximate any given unitary matrix, a result known as the *Solovay-Kitaev Theorem*. We already met the Pauli matrices in Lemma 3.1.33. The matrix $\boldsymbol{\sigma}_z$ is also called the *not gate*:

$$\mathbf{N} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

Two other usual gates are the *Hadamard gate* and the *phase-shift gate*:

$$\mathbf{H} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \qquad \mathbf{V}_\theta = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}.$$

Finally, written in the canonical basis $(|00\rangle, |01\rangle, |10\rangle, |11\rangle)$, we have the *exchange gate* and the *controlled-not gate*:

$$X = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \qquad \mathbf{N}_C = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

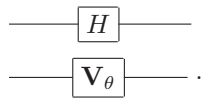
The $\mathbf{N}_C$-gate comes from a generic process: If U is a one-qubit gate, one constructs U_C, called *controlled* U, to be the two-qubits gate:

$$U_C\left(|0\rangle \otimes |\phi\rangle\right) = |0\rangle \otimes |\phi\rangle,$$
$$U_C\left(|1\rangle \otimes |\phi\rangle\right) = |1\rangle \otimes U|\phi\rangle.$$

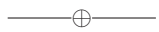
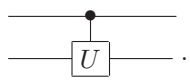
The gate U is applied on the second qubit depending on the value of the first qubit.

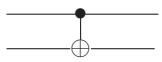
Quantum Circuit

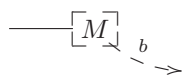
It is possible to write tensors and compositions of gates as wires and boxes on wires. For example, the map $\mathbf{H} \otimes \mathbf{V}_\theta$ can be written as



Some gates have specific representations:

- The gate $\mathbf{N}$ is written as 
- A controlled-U gate is written as 

- The gate $\mathbf{N}_C$ is written as 
- The gate X is simply
- Measurements will be represented by dashed boxes followed by a dashed wire, to indicate that we now have a classical bit, as follows:



3.2.3 Mixed States

The representation of a state of a system of quantum bits as a ray in some Hilbert space is a valid description of an isolated quantum system. It does not work when only part of the system is considered. Indeed, suppose we are given a two-qubit system $|\phi_{AB}\rangle$ in the state

$$\alpha|0_A\rangle \otimes |0_B\rangle + \beta|0_A\rangle \otimes |1_B\rangle + \gamma|1_A\rangle \otimes |0_B\rangle + \delta|1_A\rangle \otimes |1_B\rangle,$$

where $(|0_A\rangle, |1_A\rangle)$ and $(|0_B\rangle, |1_B\rangle)$ are some basis of $\mathbb{C}^2$ and suppose we only have access to qubit A.

Consider an observable M on A. Using the formalism developed with isolated systems, if we perform a measurement with this observable, the total expectation of the measurement is

$$\langle\phi_{AB}|(M \otimes \mathbf{I}_B)|\phi_{AB}\rangle = \begin{pmatrix} & \Big(\overline{\alpha}\langle 0_A| + \overline{\gamma}\langle 1_A|\Big) M \Big(\alpha|0_A\rangle + \gamma|1_A\rangle\Big) \\ + & \Big(\overline{\beta}\langle 0_A| + \overline{\delta}\langle 1_A|\Big) M \Big(\beta|0_A\rangle + \delta|1_A\rangle\Big) \end{pmatrix} = \mathbf{tr}(M\rho_A),$$

where ρ_A is the density matrix (as in Definition 3.1.14) equal to

$$\Big(\beta|0_A\rangle + \delta|1_A\rangle\Big)\Big(\overline{\beta}\langle 0_A| + \overline{\delta}\langle 1_A|\Big) + \Big(\alpha|0_A\rangle + \gamma|1_A\rangle\Big)\Big(\overline{\alpha}\langle 0_A| + \overline{\gamma}\langle 1_A|\Big),$$

that is

$$\begin{pmatrix} |\alpha|^2 & \overline{\gamma}\alpha \\ \overline{\alpha}\gamma & |\gamma|^2 \end{pmatrix} + \begin{pmatrix} |\beta|^2 & \overline{\delta}\beta \\ \overline{\beta}\delta & |\delta|^2 \end{pmatrix}$$

in the basis $(|0_A\rangle, |1_A\rangle)$. Note that if M is the projection onto $|0_A\rangle$, as expected we get $|\alpha|^2 + |\beta|^2$. And if it is the projection onto $|1_A\rangle$, we get $|\delta|^2 + |\gamma|^2$.

Thus, if we are only considering the quantum bit A, one can safely work with the density matrix ρ_A in place of the whole system. This representation is called a *mixed state*: Since $|\alpha|^2 + |\gamma|^2 + |\beta|^2 + |\delta|^2 = 1$, if

$$\alpha' = \frac{\alpha}{|\alpha|^2 + |\gamma|^2}, \quad \gamma' = \frac{\gamma}{|\alpha|^2 + |\gamma|^2}, \quad \beta' = \frac{\beta}{|\beta|^2 + |\delta|^2}, \quad \delta' = \frac{\delta}{|\beta|^2 + |\delta|^2},$$

then the matrix ρ_A represents a probability distribution over pure states

$$(|\alpha|^2 + |\gamma|^2)\Big\{\alpha'|0_A\rangle + \gamma'|1_A\rangle\Big\} + (|\beta|^2 + |\delta|^2)\Big\{\beta'|0_A\rangle + \delta'|1_A\rangle\Big\}.$$

We can re-encode the state of an n-quantum bit system $|\phi\rangle$ as the $2^n \times 2^n$ density matrix $\rho = |\phi\rangle\langle\phi|$. A unitary operation U on this state yields the density matrix $U\rho U^*$, and the measurement of an observable M yields the expected value $\mathbf{tr}(M\rho)$.

3.3 Quantum Effects

We list in this section a few specifics of quantum computation, following (Preskill, 1999).

3.3.1 No Cloning

Consider a quantum bit in some unknown state $|\phi\rangle = \alpha|0\rangle + \beta|1\rangle$. Then it is not possible to "clone" the state and get the two-qubit state $|\phi\rangle \otimes |\phi\rangle$. The reason is simple: the only operations we are allowed to perform are unitaries and measurements, which are linear maps. The cloning operation being non-linear, it is therefore not implementable.

It is however possible to build a "copying" map G

$$\alpha|0\rangle + \beta|1\rangle \longmapsto \alpha|00\rangle + \beta|11\rangle.$$

But it does not satisfy the usual commutativity rule for duplication: if $f : \mathcal{H} \to \mathcal{H}$ is any linear map, then $G \circ f \neq (f \otimes f) \circ G$.

3.3.2 Entanglement

Another special property of quantum information is the *superposition*: the 2-qubit state

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{3.3.1}$$

is a valid state, but cannot be written as $|\phi\rangle \otimes |\psi\rangle$. We say the two quantum bits are *entangled*.

The interesting aspect of entanglement is that the measurements of the qubits will be correlated. For example, measuring the first qubit in the state (3.3.1) with respect to the standard basis yields 0 with probability $\frac{1}{2}$ and 1 with probability $\frac{1}{2}$. In the former case, the state of the 2-qubit system is $|00\rangle$, thus measuring the second qubit yields 0 with probability 1. In the latter, the state of the system is $|11\rangle$, and measuring the second qubit yields 1 with probability 1.

A useful basis for 2-qubit systems is the following set of entangled states:

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle), \quad \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle),$$
$$\frac{1}{\sqrt{2}}(|01\rangle + |10\rangle), \quad \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle).$$

3.3.3 Bell's Inequalities

This section is taken from (Preskill, 1999).

From a classical perspective, a measurement is a special form of coin-toss. A *hidden variable theory* is a theory where a probability λ is associated to each state and to each axis such that the measurement along this axis outputs λ. The question is then whether it is possible to build a hidden variable theory that would agree with the prediction of quantum theory while still satisfying the strong *locality criterion* (Einstein et al., 1935), stating:

> " Suppose that A and B are space-like separated systems. Then in a *complete* description of physical reality an action performed on system A must not modify the description of system B. "

Bell (1964) shows that it is not possible. The argument goes as follows. Consider a quantum machine that maximally entangles two quantum bits A and B

$$|\phi_{AB}\rangle = \frac{1}{\sqrt{2}}(|0\rangle \otimes |1\rangle - |1\rangle \otimes |0\rangle)$$

and sends qubit A to Alice and qubit B to Bob. Suppose that they can independently choose one of the following axes (in the Bloch sphere, see Figure 3.1) to measure:

$$a = (0,0,1), \qquad b = \left(\frac{\sqrt{3}}{2}, 0, -\frac{1}{2}\right), \qquad c = \left(-\frac{\sqrt{3}}{2}, 0, -\frac{1}{2}\right).$$

They live in the xz-plane of the Bloch sphere and correspond respectively to the bases in $\mathbb{C}^2$:

$$\begin{aligned} |0_a\rangle &= |0\rangle, & |0_b\rangle &= \frac{1}{2}|0\rangle + \frac{\sqrt{3}}{2}|1\rangle, & |0_c\rangle &= \frac{1}{2}|0\rangle - \frac{\sqrt{3}}{2}|1\rangle, \\ |1_a\rangle &= |1\rangle & |1_b\rangle &= \frac{\sqrt{3}}{2}|0\rangle - \frac{1}{2}|1\rangle, & |1_c\rangle &= \frac{\sqrt{3}}{2}|0\rangle + \frac{1}{2}|1\rangle. \end{aligned}$$

What is the probability of obtaining the same output when measuring A and B with respect to two different bases?

In a local hidden variable theory, the result of measuring A and B along each of the possible axis is predetermined. Let $P_{x,y}$ be the probability of obtaining the same output while measuring A along x and B along y. For any local hidden variable theory,

$$P_{a,b} + P_{b,c} + P_{c,a} \geqslant 1, \tag{3.3.2}$$

since for any possible distribution of measurement values, there will always be two values that will be equal (see (Preskill, 1999) for a complete discussion). However, the quantum theory provides the value

$$P_{x,y} = \langle\phi_{AB}|(|0_x\rangle\langle 0_x| \otimes |0_y\rangle\langle 0_y|)|\phi_{AB}\rangle + \langle\phi_{AB}|(|1_x\rangle\langle 1_x| \otimes |1_y\rangle\langle 1_y|)|\phi_{AB}\rangle.$$

The computation shows that $P_{x,y} = \frac{1}{4}$ for $x \neq y$, $x, y = a, b, c$. In particular,

$$P_{a,b} + P_{b,c} + P_{c,a} = \frac{3}{4} < 1,$$

which violates Equation (3.3.2).

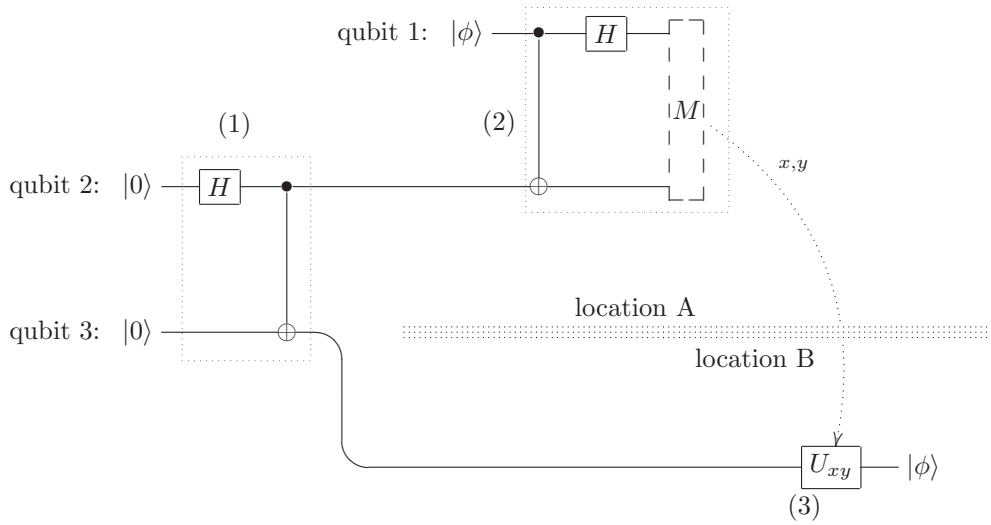


Figure 3.2: Quantum teleportation protocol.

3.4 Some Algorithms and Uses of Quantum Effects

The main power of quantum computation comes from entanglement. We describe in the following three algorithms that we will consider in the remainder of the thesis. For more algorithms, such as the quantum fast Fourier transform, or Shor's factorization algorithm (Shor, 1994), a good reference is (Aharonov, 1999).

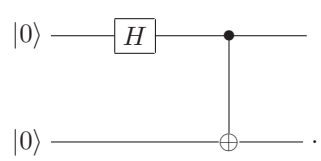
3.4.1 Teleportation

One use of entanglement is the "teleportation" of a quantum bit state via a classical channel. The idea is the following: Alice and Bob share an entangled pair of qubits

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle).$$

Alice wants to send a qubit in an unknown state $|\phi\rangle = \alpha|0\rangle + \beta|1\rangle$ to Bob. The *teleportation procedure*, summarized in Figure 3.2, can be described in three steps:

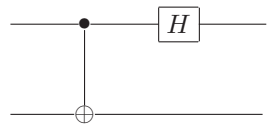
1. At location A, create the initial entangled state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ with the two last qubits using the circuit



   Alice keeps qubit 2 and stays at location A, while Bob takes qubit 3 and goes to location B.

2. Alice, to send qubit 1 in state $|\phi\rangle$ to Bob, applies a rotation on her two qubits 1 and 2, using

the circuit



.

She then measures the 2-qubit resulting state, gets two classical bits (x, y) and sends them to Bob.

3. Bob, to transform qubit 3 to state $|\phi\rangle$ applies on it the transformation U_{xy} defined as follows:

$$\begin{array}{ll} \text{if } M \text{ outputs } 00, & U_{00} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \\ \text{if } M \text{ outputs } 01, & U_{01} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \\ \text{if } M \text{ outputs } 10, & U_{10} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \\ \text{if } M \text{ outputs } 11, & U_{11} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}. \end{array} \tag{3.4.1}$$

Note that the entanglement of qubits 2 and 3 can be done ahead of time (so long as they stay entangled).

Proof of the correctness of the protocol. The rotation of step (2) performs the following computation

$$\begin{array}{ccccc} & CNOT & & H \otimes id & \\ |00\rangle & \mapsto & |00\rangle & \mapsto & \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle), \\ |01\rangle & \mapsto & |01\rangle & \mapsto & \frac{1}{\sqrt{2}}(|01\rangle + |11\rangle), \\ |10\rangle & \mapsto & |11\rangle & \mapsto & \frac{1}{\sqrt{2}}(|01\rangle - |11\rangle), \\ |11\rangle & \mapsto & |10\rangle & \mapsto & \frac{1}{\sqrt{2}}(|00\rangle - |10\rangle). \end{array}$$

If we apply it to the two first qubits of

$$(\alpha|0\rangle + \beta|1\rangle) \otimes \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) = \frac{1}{\sqrt{2}}(\alpha|000\rangle + \alpha|011\rangle + \beta|100\rangle + \beta|111\rangle)$$

we get

$$\frac{1}{2}\Big(\alpha(|000\rangle + |100\rangle) + \alpha(|011\rangle + |111\rangle) + \beta(|010\rangle - |110\rangle) + \beta(|001\rangle - |101\rangle)\Big)$$
$$= \frac{1}{2}\begin{pmatrix} & |00\rangle \otimes (\alpha|0\rangle + \beta|1\rangle) \\ + & |01\rangle \otimes (\beta|0\rangle + \alpha|1\rangle) \\ + & |10\rangle \otimes (\alpha|0\rangle - \beta|1\rangle) \\ + & |11\rangle \otimes (\alpha|1\rangle - \beta|0\rangle) \end{pmatrix}$$

If we measure the two first qubits, the third qubit becomes

$$\begin{array}{ll} \alpha|0\rangle + \beta|1\rangle & \text{if 00 was measured,} \\ \beta|0\rangle + \alpha|1\rangle & \text{if 01 was measured,} \\ \alpha|0\rangle - \beta|1\rangle & \text{if 10 was measured,} \\ \alpha|1\rangle - \beta|0\rangle & \text{if 11 was measured.} \end{array}$$

Finally note that if U_{xy} is applied in the case where x, y was measured, then the state of the last qubit is $\alpha|0\rangle + \beta|1\rangle$. □

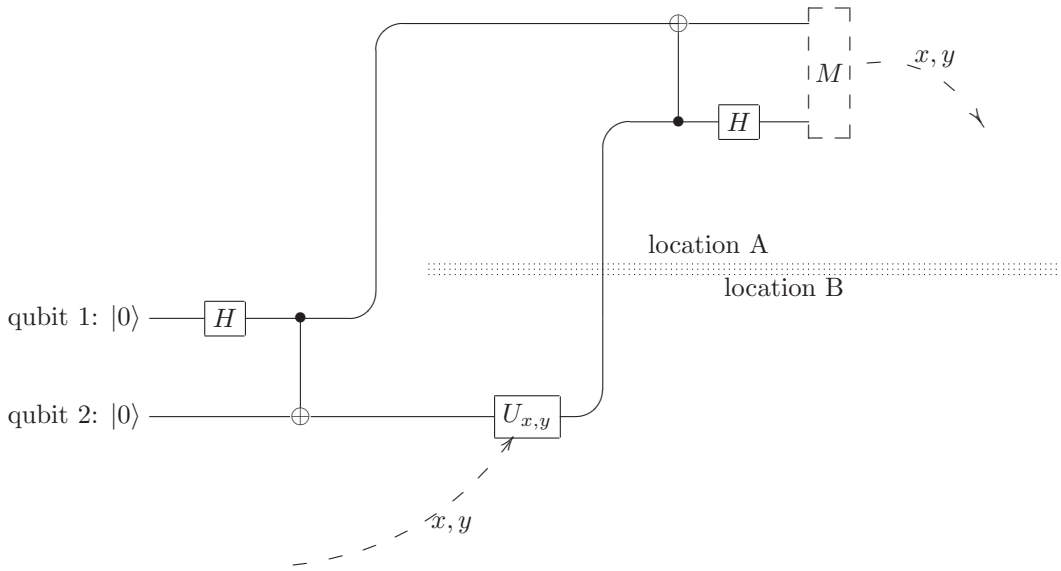


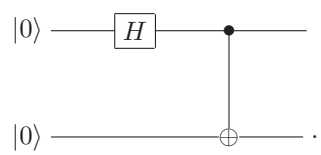
Figure 3.3: Dense coding protocol.

3.4.2 Dense Coding

The teleportation algorithm allows to "send" one qubit using two classical bits. It is possible to perform the reverse operation, namely to send two classical bits using an entangled pair and a quantum channel.

The situation is similar. Alice and Bob share an entangled state of 2 qubits, Alice has qubit 1, Bob has qubit 2. This time, Bob wants to send the two classical bits (x, y) to Alice. He can use the protocol summarized in Figure 3.3, called *dense coding*. Like the teleportation algorithm, the procedure is again in three steps:

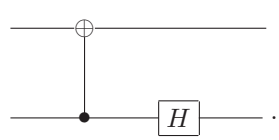
1. At location B, Alice and Bob create an entangled state of two quantum bits 1 and 2, using the circuit



   Then, Alice goes to location A with qubit 1, while Bob stays at location B with qubit 2.

2. If Bob wants to send two classical bits (x, y) to Alice, he needs to apply the gate $U_{x,y}$ as defined in the teleportation protocol (Equation (3.4.1)), and send the modified qubit 2 to Alice.

3. Alice, to retrieve the two classical bits, applies a rotation on qubits 1 and 2 using the gate



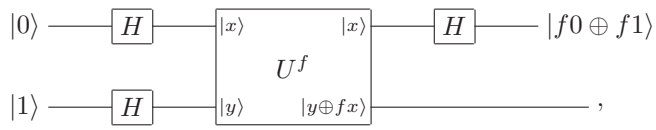
   and a measurement, retrieving the two bits (x, y).

3.4.3 The Deutsch Algorithm

The Deutsch algorithm is for finding out whether a boolean function is balanced or constant. In classical computation, two calls to the function are needed. In quantum computation, one can find it out in only one call. The algorithm takes as input an unknown circuit U^f computing

$$U^f(|x\rangle \otimes |y\rangle) = |x\rangle \otimes |y \oplus f(x)\rangle$$

for some boolean function f, where $\oplus$ stands for the boolean addition. The quantum circuit for the algorithm is the following:



where we write fx in place of $f(x)$. To retrieve the answer, we have to measure the first qubit: it is 0 when the function is balanced and 1 when it is not. Note that the input of this algorithm is a "black-box", in other terms a function from two qubits to two qubits.

Proof that the procedure is correct. This will compute the following thing:

$$\begin{aligned}
&(H \otimes id)(U^f)(H \otimes H)(|0\rangle \otimes |1\rangle) \\
=&(H \otimes id)(U^f)\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \\
=&\frac{1}{2}(H \otimes id)(U^f)(|00\rangle + |10\rangle - |01\rangle - |11\rangle) \\
=&\frac{1}{2}(H \otimes id)\begin{pmatrix} |0\rangle \otimes |0 \oplus f0\rangle + |1\rangle \otimes |0 \oplus f1\rangle \\ -|0\rangle \otimes |1 \oplus f0\rangle - |1\rangle \otimes |1 \oplus f1\rangle \end{pmatrix} \\
=&\frac{1}{2\sqrt{2}}\begin{pmatrix} (|0\rangle + |1\rangle) \otimes |f0\rangle \quad +(|0\rangle - |1\rangle) \otimes |f1\rangle \\ -(|0\rangle + |1\rangle) \otimes |1 \oplus f0\rangle - (|0\rangle - |1\rangle) \otimes |1 \oplus f1\rangle) \end{pmatrix} \\
=&\frac{1}{2\sqrt{2}}\begin{pmatrix} |0\rangle \otimes (|f0\rangle + |f1\rangle - |1 \oplus f0\rangle - |1 \oplus f1\rangle) \\ +|1\rangle \otimes (|f0\rangle - |f1\rangle - |1 \oplus f0\rangle + |1 \oplus f1\rangle) \end{pmatrix}.
\end{aligned}$$

If $f0 = f1$, then $|f0\rangle - |f1\rangle - |1 \oplus f0\rangle + |1 \oplus f1\rangle = 0$ and the result is

$$\frac{1}{\sqrt{2}}|0\rangle \otimes (|f1\rangle - |1 \oplus f1\rangle).$$

If $f0 = 1 \oplus f1$, then $|f0\rangle + |f1\rangle - |1 \oplus f0\rangle - |1 \oplus f1\rangle = 0$ and the result is

$$\frac{1}{\sqrt{2}}|1\rangle \otimes (|1 \oplus f1\rangle - |f1\rangle).$$

So the value of the measurement of the first qubit is 0 if the function f is balanced, and 1 in the other case. □

Chapter 4

A Tour in Existing Models of Quantum Computation

Chapter 3 developed the formalism and the techniques needed to perform quantum computation. This chapter surveys a few selected models for manipulating quantum programs and protocols. A more complete set can be found in e.g. (Gay, 2006).

The goal is twofold. First, we want to be able to write programs and codes manipulating quantum data. Numerous attempts have been done in the past years to provide a more easy way to write (and read) quantum algorithms. Second, we aim at a deeper understanding of the structure of quantum information by developing a semantics for quantum computation.

4.1 The Various Paradigms

For building languages and understanding algorithms, several paradigms of computation have been studied.

4.1.1 Unitary Gates as Computation.

Families of Quantum Circuits

One formal system for writing quantum algorithms is the *quantum circuit* formalism described by Deutsch (1989) and sketched in Section 3.2.2. In this formalism, the driving forces in the computation are the unitary operations. An algorithm is composed of three steps:

1. allocation of n quantum bits;
2. application of a list of unitary gates;
3. measurement of all or a part of the n quantum bits.

The description of an algorithm is a family of quantum circuits, one for each possible number of input qubits. The family needs to be uniform, i.e. efficiently computable by a classical computer. As for classical boolean circuits, it is then possible to speak of the *complexity* of an algorithm by taking the limit of the ratio of the number of gates used in the circuit versus the number of input quantum bits.

The Quantum Turing Machine

A Turing machine (Turing, 1936) is an automaton consisting of the following elements:

1. an infinite tape, divided into cells, each cell containing a letter from a given alphabet;
2. a head, able to read and write into cells, and to move to the left or to the right along the tape;
3. a finite set of states and a state register, recording the state of the machine. One of the state is called *initial* and a subset of the states are called *final*;
4. a transition table, mapping each possible pair (letter, state) to a triple consisting of a new state, a letter and a direction (left or right).

At the beginning the state register is in the initial state. At each time-step, the automaton reads the cell of tape below the head. Based on the letter read and its state, using the transition table, it writes on the tape the corresponding letter and moves in the given direction. It iterates this course of action until a final state is reached. A program consists of a given alphabet, a set of states and a transition table.

Benioff (1980) and Deutsch (1985) describe a Turing machine where the tape, the head, and the states of the automaton are encoded as a quantum state. The transition table is described in a unitary operation. This model provides a tool to reason about expressiveness and complexity of quantum algorithms: As was shown by Nishimura and Ozawa (2002), the resulting notion of computation is equivalent to that of the quantum circuit model.

QML or the Quantum Test

Altenkirch and Grattage (2005a,b) describe a first-order language for manipulating quantum bits. This calculus understands duplication as sharing, or copying:

$$\alpha|0\rangle + \beta|1\rangle \mapsto \alpha|00\rangle + \beta|11\rangle.$$

The language is written on a classical device and compiles into a quantum circuit. Due to the copying operation, contraction can be done blindly but weakening (that is, measurement) has to be kept track of.

The language features superposition of quantum states, leading to a "quantum" test. For example, the gate $\mathbf{N}_C$ is defined as:

$$\begin{array}{ll} \mathbf{N}\ x = & \mathit{if}^q x\ \mathit{then}\ 0\ \mathit{else}\ 1, \\ \mathbf{N}_C\ \langle c, x\rangle = & \mathit{if}^q c\ \mathit{then}\ \langle c, \mathbf{N}x\rangle\ \mathit{else}\ \langle c, x\rangle. \end{array}$$

The main drawback of this formalism is that the quantum test

$$\mathit{if}^q x\ \mathit{then}\ M\ \mathit{else}\ N$$

is well defined if and only if M and N have orthogonal representations. Therefore, in general one cannot say before running the computation whether a program is valid or not.

Van Tonder's Quantum Lambda Calculus

(Van Tonder, 2004) describes a lambda calculus where the terms are encoded on strings of quantum bits. The language is built using the terms

$$\begin{array}{lll} \mathit{Term}\ M, N & ::= & x \mid c \mid MN \mid \lambda x.M, \\ \mathit{Constant}\ c & ::= & 0 \mid 1 \mid H \mid \cdots, \end{array}$$

where the constants c contains booleans 0 and 1, as representatives of the quantum bits $|0\rangle$ and $|1\rangle$, and representatives for a chosen set of unitary gates. For example, the identity is $|\lambda x.x\rangle$. The unitaries apply on 0 and 1 in the naive way: $|H0\rangle$ should reduce to $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. This language is purely quantum, and the reduction is encoded as a unitary map.

A trick needs to be used to overcome the fact that the reduction is not in general a reversible operation (Consider for example $(\lambda x.1)0 \to 1$). Given any function $\beta : X \to Y$, it is possible to produce a reversible map $X \to X \times Y$ invertible on its range by mapping $x \mapsto (x, \beta(x))$. If β is a reduction of terms, we can make it reversible by appending to the successive terms in the reduction its history:

$$M \to (M, \beta(M)) \to (M, \beta(M), \beta^2(M)) \to \cdots$$

However, a naive implementation of the lambda calculus into quantum bits does not provide a consistent equational theory. Indeed the term

$$\frac{1}{\sqrt{2}}(|(\lambda x.0)0\rangle + |(\lambda x.0)1\rangle) = |\lambda x.0\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

should be β-equivalent to $\frac{1}{\sqrt{2}}(|0\rangle + |0\rangle) = \sqrt{2}|0\rangle$, which is not valid since its norm is greater than 1.

As it turns out, one needs to distinguish between terms that are in superposition (such as $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and the ones that are not. For the system to be consistent, terms in superposition should not be duplicated nor erased, and should be the same, modulo the 0's and the 1's.

4.1.2 Concurrent Quantum Computation

Quantum computation has important implications in cryptography. The design of quantum protocols to securely communicate data is a field of ongoing research.

From a theoretic side, a question one might ask is whether the tools that were developed to analyse the safety of classical distributed systems can be used in the context of quantum systems.

The simultaneous works of Lalire and Jorrand (2004) and Gay and Nagarajan (2005) develop a process algebraic approach to concurrent quantum computation. Starting from the π-calculus, quantum features, such as quantum bits, unitaries and measurements, are added to the syntax, and an operational semantics is developed. (Gay and Nagarajan, 2005) gives a typed language and prove some safety properties for well-typed protocols; for example the fact that a quantum bit cannot be shared among several processes, and the soundness of the type system.

4.1.3 Measurement-Based Quantum Computation

The unitary operations are one side of the quantum coin. The other side is the measurement. It turns out that it is possible to perform quantum computation with "only" measurements, as it was described first by Briegel and Raussendorf (2002); Raussendorf and Briegel (2001) and then computationally by Danos et al. (2007).

The idea is the following: A state of highly entangled quantum bits is prepared and then a series of measurements are performed. The computation is "run" by the projections occurring during the measurements. In its most general formulation, the calculus happens to be as powerful as the regular quantum-circuit computational model, and is slightly better in terms of time complexity.

Formally, one works with a *pattern*, composed of a finite set V, a set of inputs $I \subset V$, a set of outputs $O \subset V$, and a sequence of operations of the following kinds (where $i, j \in V$):

- 1-quantum measurements M_i^α: measurement of qubit i in the xy-plane, along the axes

$$\frac{1}{\sqrt{2}}(|0\rangle + e^{i\alpha}|1\rangle), \qquad \frac{1}{\sqrt{2}}(|0\rangle - e^{i\alpha}|1\rangle).$$

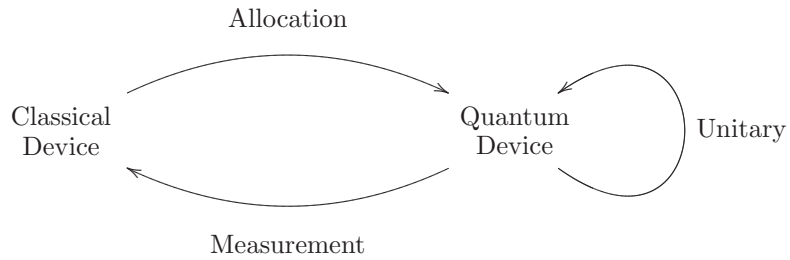


Figure 4.1: The QRAM model

- 2-qubit entanglement operators $E_{i,j}$: where $E_{i,j}$ is $(\boldsymbol{\sigma}_z)_C$ (controlled-$\boldsymbol{\sigma}_z$) applied to qubits i, j.
- 1-qubit Pauli corrections X_i, Z_i: The corresponding Pauli matrix applied on qubit i.

(Danos et al., 2007) describe an operational semantics and a compositional denotational semantics in terms of completely positive, trace preserving maps. More importantly, it gives a rewrite theory for patterns and proves the confluence of the rewriting system as well as a standardization theorem allowing patterns to be put in semantically equivalent form.

4.1.4 The QRAM Model

One of the first formal descriptions of a quantum programming language is the pseudo-code underlying the quantum random access machine (QRAM) model of Knill (1996). While it does not strictly speaking present a language, the paper provides a basis for formally describing algorithms. The idea (see Figure 4.1) is to consider a computation as done by a classical process that has access to a quantum device. A series of built-in operations are allowed, such as allocation of quantum bits, measurements, and unitary operations. One can summarize this approach with the slogan "classical control, quantum data" (Selinger, 2004b).

Ömer (2000) describes a rich, architecture-independent programming language called QCL following the QRAM model. The language is sequential, classical based, and comes with powerful routines to deal with quantum effects: quantum register allocation, scratch space management, syntactic reversibility of user-defined quantum operators.

Building up on top of these works, (Bettelli et al., 2003) describe a language in a C++ style for manipulating quantum constructs and concepts more efficiently. Its main characteristic is to treat quantum operators as objects, thus allowing construction and manipulation at run-time.

However, none of these works contains a semantics. A language somehow different is the language qGCL of Sanders and Zuliani (2000). The paper develops a guarded-command language for probabilistic operation together with an operational semantics based on probability distributions, and extend it to quantum constructs. The described semantics is very close to the actual implementation: the interpretation of a program assigns to each possible input state a probability distribution on the output space.

Another language-based approach with semantics in mind is the work of Selinger (2004b), developing a flow-chart language. The language has a sound and complete interpretation in the category of superoperators. We detail this semantics in Section 4.3.

4.2 Formalism of Hilbert Spaces

A recent research thread initiated by Abramsky and Coecke (2004) aims at providing an abstract framework in which one could interpret quantum computation. The goal is to extract from the

category of Hilbert spaces the required structures for performing quantum computation.

4.2.1 Dagger Compact-Closed Categories

Abstract notions of adjointness, self-adjointness, unitaries, bases and scalar products arise in the context of dagger compact-closed categories with biproducts:

Definition 4.2.1 (Selinger, 2005)**.** A *dagger category* is a category $\mathcal{C}$ together with a functor $\dagger : \mathcal{C}^{\mathrm{op}} \to \mathcal{C}$ such that for every $f : A \to B$, $f^{\dagger\dagger} = f$ and for every object A, $A^\dagger = A$.

Definition 4.2.2 (Selinger, 2005)**.** A *dagger compact-closed category* is a dagger symmetric monoidal category that is also compact closed (see e.g. Kelly and Laplaza, 1980), and such that the following diagram commutes for all A:

$$\begin{tikzcd} \top \arrow[r, "\epsilon_A^\dagger"] \arrow[dr, "\eta_A"'] & A \otimes A^* \arrow[d, "\sigma_{A,A^*}"] \\ & A^* \otimes A. \end{tikzcd}$$

Various notions of Hilbert spaces can be defined in the context of dagger compact-closed categories with biproducts:

- A *scalar* is a morphism $s : \top \to \top$.
- A *vector* is a morphism $v : \top \to A$.
- The *adjoint* of a morphism $f : A \to B$ is the morphism $f^\dagger : B \to A$.
- The *scalar product* of $v, w : \top \to A$ is $v; w^\dagger : \top \to \top$.
- A *unitary morphism* is a morphism $f : A \to B$ such that $f^\dagger$ is its inverse.
- A *basis* for an object A is a unitary morphism

  $$b_A : n{\cdot}\top \to A,$$

  where $n{\cdot}\top$ is a sum of n $\top$'s. In this case, the *dimension* of A is said to be n.
- We can define the *matrix* of a morphism $f : A \to B$ where A and B are finite dimensional as the map

  $$n_A{\cdot}\top \xrightarrow{b_A} A \xrightarrow{f} B \xrightarrow{b_B} n_B{\cdot}\top.$$

These definitions satisfy various equations related to Hilbert spaces and hermitian positive operators. For example, (Selinger, 2005) shows that completely positive maps can be defined in this context. The paper provides a categorical construction for building a category of completely positive maps out of any dagger compact-closed category.

Although no formal syntax comes with the semantics, a colourful graphical language is provided, making proofs of properties relatively easy to follow. However, this graphical language does not mix well with the need for biproducts: in the desired quantum computational applications, the notion of bases is heavily used.

4.2.2 Classical Objects

In an attempt to remove biproducts, Coecke and Pavlovic (2007) describe a new notion called *classical objects*. A classical object is an object X in the category together with a map $\delta_X : X \to X \otimes X$, a map $\epsilon_X : X \to \top$, and a few equations. In particular, these equations make $(X, \epsilon_X, \delta_X)$ a commutative comonoid object, as in Definition 2.7.3. The notion of classical object captures the notion of basis without the needs for biproducts. The idea is the following. A generic object X in the category is like a generic finite dimensional Hilbert space, with no associated basis. Associating a structure (δ_X, ϵ_X) to the object X is the same thing as associating a basis to X. In Hilbert spaces, the maps δ_X and ϵ_X are

$$\delta_X : e_i \mapsto e_i \otimes e_i, \qquad \epsilon_X : e_i \mapsto 1,$$

for some basis $(e_1, \dots e_n)$ of X. With enough structure, it is possible to simulate what was done with biproducts. In particular, as was shown by Coecke and Paquette (2006), an abstract version of the Naimark theorem can be proved using the graphical language.

4.3 A Flow-Chart Language

(Selinger, 2004b) describes a semantics for QRAM-based first-order quantum computation. Instead of starting from the category of Hilbert spaces, the paper describes computations as maps in the category of superoperators.

4.3.1 The language

The language is described by a flow-chart notation: it is a super-set of a classical flow-chart language. A program is a graph together with a cursor that follows the wires, with data attached to it. A graph is constructed from the rules in Table 4.1. The box *new qbit* $b = i$ creates a new quantum bit of name b with value $|i\rangle$. The box *new bit* $b = i$ creates a new boolean variable named b with value i. The gate $q_1, \dots q_n *= U_n$ applies the unitary gate U_n on the corresponding variables. The gate *permute* ϕ permutes the context Δ according to the permutation ϕ. The gate *branch* b performs a test on the boolean b and *meas* q measures the quantum bit q and performs a test on the output.

4.3.2 The Category of Superoperators

The language has a full and complete representation in the category **Q** of superoperators. This category is defined as the refinement of the category **CPM**.

Definition 4.3.1. The category **CPM** is defined as follows:

- The objects are signatures $\sigma = n_1, \dots, n_k$, i.e., finite tuples of positive integers,
- the morphisms $\sigma \to \tau$ are completely positive maps $V_\sigma \to V_\tau$, where $V_{n_1,\dots,n_k}$ is the Hilbert space $\mathbb{C}^{n_1 \times n_1} \times \cdots \times \mathbb{C}^{n_k \times n_k}$.

We use the notation B_σ for the canonical basis of V_σ.

The objects in this category can be seen as spaces of block matrices. Following the construction in Lemma 3.1.18, one can define a tensor and a coproduct structure, as follows:

Definition 4.3.2. Given two signatures $\sigma = (n_1, \dots, n_k)$ and $\tau = (m_1, \dots, m_l)$, we define $\sigma \otimes \tau$ and $\sigma \oplus \tau$ to be respectively

$$\sigma \otimes \tau = (n_1 m_1, \dots, n_1 m_l, n_2 m_1, \dots, n_k m_l), \qquad \sigma \oplus \tau = (n_1, \dots, n_k, m_1, \dots, m_l).$$

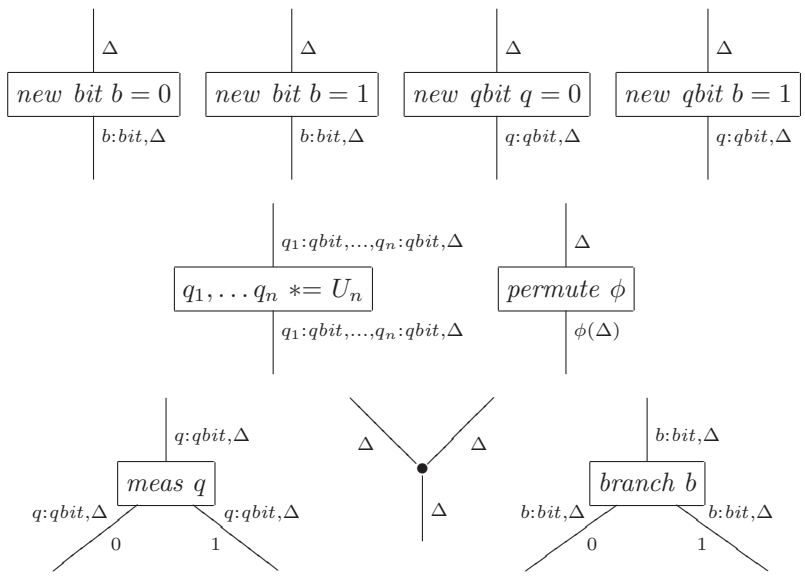


Table 4.1: Rules for constructing quantum flow-charts

Lemma 4.3.3. *Let σ and τ be two signatures. Then we have $V_{\sigma\otimes\tau} \cong V_\sigma \otimes V_\tau$ and $V_{\sigma\oplus\tau} \cong V_\sigma \oplus V_\tau$.* □

Definition 4.3.4. We define a map between arrows

$$\Phi : \mathbf{CPM}(\sigma \otimes \tau, \sigma') \xrightarrow{\sim} \mathbf{CPM}(\sigma, \tau \otimes \sigma')$$

as in Lemma 3.1.18, using the canonical basis.

Lemma 4.3.5. *The category $(\mathbf{CPM}, \otimes, (1))$ is a symmetric monoidal category. The map Φ makes* **CPM** *symmetric monoidal closed. The structure $\oplus$ and the signature (0) provide a structure of biproducts.* □

Definition 4.3.6. Given a signature $\sigma = (n_1, \ldots, n_k)$, a tuple $A = (A_1, \ldots, A_k)$ in V_σ is *hermitian* (respectively *positive*) if each matrix A_i is hermitian (respectively positive). We extend the Löwner order of Definition 3.1.25 to tuples of matrices by letting $A \sqsubseteq B$ if $B - A$ is positive. We define the *trace* of the matrix tuple A to be the sum of the traces of its components: $\mathbf{tr}(A) = \sum_i \mathbf{tr}(A_i)$.

Definition 4.3.7. Let $\sigma = (n_1, \ldots, n_k)$ and $\tau = (m_1, \ldots, m_l)$ be two signatures. A linear function $F : V_\sigma \to V_\tau$ is a tuple $(F_{1,1}, \ldots, F_{1,k}, \ldots\ldots, F_{n,1}, \ldots, F_{n,k})$, where $F_{i,j}$ is the component $\mathbb{C}^{n_i} \to \mathbb{C}^{m_j}$. The *characteristic matrix tuple* χ_F of the function F is

$$(\chi_{F_{1,1}}, \ldots, \chi_{F_{1,k}}, \ldots\ldots, \chi F_{n,1}, \ldots, \chi_{F_{n,k}}).$$

Note that since $F : V_\sigma \to V_\tau$, $\chi_F \in V_{\sigma\otimes\tau}$

Definition 4.3.8. The category **Q** has the same objects as **CPM** and as arrows superoperators. A superoperator is, as in Definition 3.1.26, a completely positive map $f : \sigma \to \tau$ such that for all $A \in V_\sigma$, $\mathbf{tr}(f(A)) \leqslant \mathbf{tr}(A)$.

Lemma 4.3.9. *The category* $\mathbf{Q}$ *is symmetric monoidal and has coproducts, with the same construct as for* $\mathbf{CPM}$*. It is not, however, closed.* □

Definition 4.3.10. The *trace characteristic matrix tuple* of a linear function $F : V_\sigma \to V_\tau$ is defined to be $\chi_F^{(\mathbf{tr})} = \chi_{\mathbf{tr}_\tau \circ F} \in V_\sigma$, that is, the characteristic matrix tuple of $\mathbf{tr}_\tau \otimes F$. Equivalently, $\chi_F^{(\mathbf{tr})} = (id_\sigma \otimes \mathbf{tr}_\tau)(\chi_F)$.

Completely positive maps and superoperators in this generalized context still satisfy a version of Theorem 3.1.27.

Theorem 4.3.11 (Generalized Kraus Representation Theorem)**.** *Consider the signatures* σ *of the form* $(n_1, \ldots, n_s)$ *and* τ *of the form* $(m_1, \ldots, m_t)$*. Let* $F : V_\sigma \to V_\tau$ *be a linear function. Then the following holds:*

1. *F is completely positive if and only if χ_F is positive.*
2. *F is a superoperator if and only if χ_F is positive and $\chi_F^{(\mathbf{tr})} \sqsubseteq I_\sigma$, where $I_\sigma \in V_\sigma$ is the tuple consisting of identity matrices.*
3. *F is a superoperator if and only if it can be written in the form*

$$F(A_1, \ldots, A_s) = \big(\sum_{i,l} U_{i1l} A_i U_{i1l}^*, \ldots, \sum_{i,l} U_{isl} A_i U_{itl}^*\big),$$

for matrices $U_{ijl} \in \mathbb{C}^{m_j \times n_i}$ where $\sum_{j,l} U_{ijl}^* U_{ijl} \sqsubseteq I_{n_i}$ for all i. Here, l ranges over a finite set. □

4.3.3 Interpretation of the Flow-Chart Language

It is possible to interpret a flow chart diagram as a superoperator. We state in this section the few results of (Selinger, 2004b) used in Chapter 7.

Definition 4.3.12. Let $\sigma = (n_1, \ldots, n_s)$ be a signature, and let $\sigma' = n_1 + \ldots + n_s$ be an integer regarded as a simple signature. The *measurement operator* $\mu_\sigma : V_{\sigma'} \to V_\sigma$ is defined as

$$\mu_{\sigma'} \begin{pmatrix} A_{1,1} & \cdots & A_{1,s} \\ \vdots & \ddots & \vdots \\ A_{s,1} & \cdots & A_{s,s} \end{pmatrix} = (A_{1,1}, A_{2,2}, \ldots, A_{s,s}),$$

where $A_{i,j} \in \mathbb{C}^{n_i \times n_j}$.

Definition 4.3.13. We extend the definition of partial trace of Definition 3.1.31 to any map of the form $(\mathbf{tr}_\sigma \otimes id_\tau) : V_{\sigma \otimes \tau} \to V_\tau$.

Definition 4.3.14. We extend the definition of contractions of Definition 3.1.28 to linear maps. We say that $f : V_\sigma \to V_\tau$ is a contraction if it if of the form

$$f(A_1, \ldots, A_s) = (U_1 A_1 U_1^*, \ldots, U_s A_s U_s^*)\,,$$

for contractions $U_i \in \mathbb{C}^{m_i \times n_i}$, where $\sigma = (n_1, \ldots, n_s)$ and $\tau = (m_1, \ldots, m_s)$.

Theorem 4.3.15. *Every superoperator $F : V_\sigma \to V_\tau$ can be factored as $F = M \circ E \circ G$, where G is a contraction, E is a partial trace operator, and M is a measurement operator.* □

Theorem 4.3.16. *The interpretation of any flow-chart in* $\mathbf{CPM}$ *yields a superoperator. Conversely, any superoperator is the image of a flow chart diagram under the interpretation.* □

4.4 Extension to Higher Order

A natural avenue of research for finding a model of higher-order quantum computation is the generalization of the category of superoperators. Selinger (2004c) investigated possible categories of normed vector spaces as model of higher-order quantum computation. A challenging problem turns out to be the matching between the norm for the tensor of two spaces and the required norm on the space of functions. This study did not find any satisfactory model.

Another work in this direction is the paper of Girard (2004). In this paper, the author describes a models of linear logic based on normed vector spaces, following the same constructions as for coherent spaces (Girard, 1987). The resulting category, although based on the right sort of spaces, does not yield the correct answer at base types, as pointed out by (Selinger, 2004c).

Chapter 5

Lambda Calculus and Semantics of Higher-Order Computation

In this chapter I will briefly describe the semantics for classical higher-order computation. The canonical tool for this purpose is a *semantics*, a mathematical model that will allow us to exhibit underlying structures and invariants.

For an complete introduction on lambda calculus, see (Barendregt, 1984). An in-depth survey of categorical logic can be found in (Lambek and Scott, 1989). Finally, references on linear logic and its semantics can be found in (Bierman, 1995; Girard, 1987; Troelstra, 1992).

5.1 Lambda Calculus

The origin of higher-order computation lies in the 1930's when Kleene (1935a,b) described a formal system called *lambda calculus*, in order to capture the notion of computable function. Lambda calculus was shown (Church, 1936) universal in the sense that any computable function can be interpreted as an expression in the language.

In this section we briefly describe the core lambda calculus. For a thoughtful development of the theory see for example (Barendregt, 1984).

5.1.1 The Language

In the lambda calculus, functions are described explicitly as formulae. The function $f : x \mapsto f(x)$ is written as the *lambda-abstraction* $\lambda x.f(x)$. The application of a function to an argument is simply the juxtaposition of the two expressions. Thus

" let $f : x \mapsto x^2$ in $f(2)$ "

becomes $(\lambda x.x^2)(2)$ in lambda calculus.

Formally, a lambda-term is an expression made of the following grammar, written in BNF[1] form:

$$\textit{Term } M, N \ ::= \ x \mid (\lambda x.M) \mid (MN) \mid \langle M, N\rangle \mid \mathit{fst}(M) \mid \mathit{snd}(M) \mid *,$$

where x ranges over an infinite set of *variables*. The term $\lambda x.M$ stands for the function that inputs x and outputs M, the term MN represents the application of a function M to an argument N, the term $\langle M, N\rangle$ represents the pair composed of M and N, $*$ is the 0-tuple, and $\mathit{fst}(M)$ and $\mathit{snd}(M)$

[1] The BNF formulation was first introduced by Naur et al. (1960) for a formal definition of the language ALGOL'60.

the left and right projections of the pairing. We speak of *term combinator* or *term operator* when referring to each token in the grammar definition.

The formalism extends transparently to higher-order functions: In lambda calculus, we can simply write $\lambda f.\lambda g.\lambda x.f(g(x))$ for the composition operator $f, g \mapsto f \circ g$. And the expression

" let $H : (f, g) \mapsto (x \mapsto f(g(x)))$ in
let $a : x \mapsto x^2$ in
let $b : x \mapsto x + 3$ in $(H(a, b))(z)$ "

for some variable z simply becomes

$$(\lambda f.\lambda g.\lambda x.f(g(x)))(\lambda x.x^2)(\lambda x.x + 3)(z). \tag{5.1.1}$$

5.1.2 Free and Bound Variables

In (5.1.1), in the term $(\lambda f.\lambda g.\lambda x.f(g(x))$ it should be understood that x is local, and that the x in $(\lambda x.x^2)$ does not interfere with the one of the previous formula. We say that the variable x in $\lambda x.x^2$ is *bound* by the lambda-abstraction. On the other hand, the variable z is not bound by anything: we say it is *free*. We define the set of free variables of a given term by induction as follows:

$$\begin{aligned} FV(x) &= \{x\}, & FV(\lambda x.M) &= FV(M) \setminus \{x\}, \\ FV(MN) &= FV(M) \cup FV(N), & FV(\langle M, N\rangle) &= FV(M) \cup FV(N), \\ FV(\mathit{fst}(M)) &= FV(\mathit{snd}(M)) = FV(M), & FV(*) &= \{\}. \end{aligned}$$

A variable that is not free is said to be bound. A term is called *closed* if it does not have any free variable, and *open* if it has at least one free variable.

5.1.3 α-Equivalence

It should be understood that $\lambda x.x^2$ is the same term as, say, $\lambda t.t^2$. We say that the two terms are α-equivalent. However, $\lambda f.\lambda g.\lambda x.f(g(x))$ and $\lambda g.\lambda g.\lambda x.g(g(x))$ are not α-equivalent: the g in the expression refers to the innermost λg.

Definition 5.1.1 (Barendregt, 1984)**.** For any term M, we define the *syntactic variable substitution* $M[x := y]$ as the term M where all syntactic occurrences of x (whether they are free or not) are replaced by y. We define a *change of bound variables in* M to be the replacement of a subterm $\lambda x.N$ by $\lambda t.N[x := t]$, provided that t does not occur anywhere in N.

We say that M *is α-equivalent to* N, written $M \approx_{ax} N$, if N results from M by a series of changes of bound variables.

Convention 5.1.2. Terms are considered up to α-equivalence from now on.

5.1.4 Operational Meaning of Lambda Calculus

We now discuss the operational meaning of the calculus by defining a rewriting system.

Substitution

A term of the form $(\lambda x.M)N$ is understood as "M with N in place of x". The process of replacing N by x is called substitution, and we write the resulting term by $M[N/x]$. Of course, we are only allowed to modify the free occurrences of x, and one should check that the process does not bind any free variables. For example, the term $(\lambda x.yx)[(fx)/y]$ is not the same as $\lambda x.(fx)x$: indeed, the variable x in fx is free. Instead, using the α-equivalent formulation $(\lambda t.yt)[(fx)/y]$ one sees that the term is in fact $\lambda t.(fx)t$.

Definition 5.1.3. Given a term M and a term P, we define the *substitution of x in M by P*, written $M[P/x]$, by the following:

$$\begin{aligned}
x[P/x] &= P,\\
*[P/x] &= *,\\
(MN)[P/x] &= (M[P/x])(N[P/x]),\\
(\langle M, N\rangle)[P/x] &= \langle M[P/x], N[P/x]\rangle,\\
(\mathit{fst}(M)[P/x] &= \mathit{fst}(M[P/x]),\\
(\mathit{snd}(M)[P/x] &= \mathit{snd}(M[P/x]),\\
(\lambda y.M)[P/x] &= \lambda y.(M[P/x]) \qquad (\text{if } y \neq x \text{ and } y \notin FV(P)).
\end{aligned}$$

We write $M[M_i/x_i, \ldots, M_n/x_n]$ for $M[M_i/x_i] \ldots [M_n/x_n]$.

Reduction of Terms

The simply-typed lambda calculus is a language describing operations: application of a function to an argument, projection of a pair. To express these operations we develop a set of rules, called *reduction rules*. They come in two flavors: the *β-reductions*, which perform operations and enforce the implicit meaning of the term:

$$(\lambda x.M)N \to_\beta M[N/x], \tag{β_λ}$$

$$\mathit{fst}\langle M, N\rangle \to_\beta M, \tag{$\beta^1_{\langle\rangle}$}$$

$$\mathit{snd}\langle M, N\rangle \to_\beta N, \tag{$\beta^2_{\langle\rangle}$}$$

and *η-reductions* simplifying terms:

$$\lambda x.(Mx) \to_\eta M, \tag{η_λ}$$

$$\langle \mathit{fst}(M), \mathit{snd}(M)\rangle \to_\eta M. \tag{$\eta_{\langle\rangle}$}$$

Each term on the left side of these rule is called a *redex*.

A *context* is a "term with a hole", defined formally by the syntax

$$C[-] ::= \quad [-] \mid (\lambda x.C[-]) \mid (C[-]M) \mid (MC[-]) \mid \langle M, C[-]\rangle \mid \langle C[-], M\rangle \mid \mathit{fst}(C[-]) \mid \mathit{snd}(C[-]),$$

where M stands for any term. Given a term N and a context $C[-]$, we write $C[N]$ for the context $C[-]$ where $[-]$ is syntactically replaced with N. In this setting, we naturally extend the notion of reduction, as follows: Given any context $C[-]$, if N is a redex reducing to N', then the reduction $\to_i$ extends to $C[N] \to_i C[N']$, for $i = \beta, \eta$. A term with no subterm in redex form is said to be in *normal form*.

We denote by $\to^*$ the reflexive transitive closure of the relation $\to$. The relation $\to^*_\beta$ satisfies a property called the *Church-Rosser theorem* that we only mention here. For a proof of this result consult e.g. (Barendregt, 1984).

Theorem 5.1.4 (Church-Rosser theorem)**.** *Suppose that M, N and P are lambda-terms such that $M \to^*_\beta N$ and such that $N \to^*_\beta P$. Then there exists a lambda-term Z such that $N \to^*_\beta Z$ and $P \to^*_\beta Z$.* □

5.1.5 Typed Lambda Calculus

In mathematics, to a function one usually associates a domain and a codomain. So for example, when we mention the identity function *id*, it is always associated to a given space. Although one may lose some generality, we gain in legibility.

Type System

Similarly, in order to be able to manipulate lambda-terms more easily, one can associate a *type* to each lambda-term. A type is an expression providing some information about the term. One can associate a *type system* to the untyped lambda calculus defined in Section 6.1 as follows:

$$\textit{Type} \quad A, B \quad ::= \quad \iota \mid \top \mid A \times B \mid A \Rightarrow B.$$

We choose an arbitrary constant type ι, and types are constructed by induction. If A and B are types, then $A \times B$ is the type for binary products, and $A \Rightarrow B$ is the type for functions from A to B. Finally, $\top$ stands for the type of the 0-tuple.

A *typed term* is the data consisting of the term and its type. The type of an open term is a function of the type of its free variables. For example, the type of $\lambda x.(yx)$ should be a function type $A \Rightarrow B$, depending on the type of y. We define the notion of a typing judgement to formalize the concept. A *typing judgement* is an expression

$$x_1 : A_1, \ldots x_n : A_n \rhd M : B$$

where the free variables of M are contained in $\{x_1, \ldots x_n\}$, and B is the type of M when each x_i has type A_i. The set $\{x_1 : A_1, \ldots x_n : A_n\}$ is called the *typing context* of the judgement, and it might be empty.

Typing Rules

The typed version of the lambda calculus presented in Section 6.1 is called the *simply-typed lambda calculus*. A typing judgement is said to be *valid* if it is derived from a *typing derivation*, or *typing tree*. A valid typing derivation is constructed using the following inductive rules:

$$\frac{}{\Delta, x : A \rhd x : A}\ (ax), \quad \frac{\Delta, x : A \rhd M : B}{\Delta \rhd \lambda x.M : A \Rightarrow B}\ (\lambda), \quad \frac{\Delta \rhd M : A \Rightarrow B \quad \Delta \rhd N : A}{\Delta \rhd MN : B}\ (app),$$

$$\frac{\Delta \rhd M : A \quad \Delta \rhd N : B}{\Delta \rhd \langle M, N\rangle : A \times B}\ (\times.I), \qquad \frac{}{x : A \rhd * : \top}\ (\top.I),$$

$$\frac{\Delta \rhd M : A \times B}{\Delta \rhd \mathit{fst}(M) : A}\ (\times.E_1), \qquad \frac{\Delta \rhd M : A \times B}{\Delta \rhd \mathit{snd}(M) : B}\ (\times.E_2).$$

Properties of Typed Terms

On one hand, we have a description of some intended properties of the language through a type system: for example, a term M of type $A \Rightarrow B$ can only by applied to an argument of type A. On the other hand, we have a rewriting system by means of reduction rules, accounting for the operational behavior of the language. Are these two descriptions compatible?

The main results that should be satisfied (and are verified in the case of the simply-typed lambda calculus) are the following.

Lemma 5.1.5 (Substitution)**.** *If $\Delta \rhd N : A$ and $\Delta, x : A \rhd M : B$ are valid typing judgements, then $\Delta \rhd M[N/x] : B$ is valid.* □

Theorem 5.1.6 (Subject reduction)**.** *If $\Delta \rhd M : A$ and if $M \to_{\beta\eta} M'$ then $\Delta \rhd M' : A$ is valid.* □

They state that the type of a term remains constant along the reduction process. A third important result states the consistency of well-typed closed terms, in the sense that a well-typed closed term will reduce until reaching a normal form. The result is stated as follows:

Theorem 5.1.7 (Progress)**.** *If M is a closed, well-typed term, then either M is a normal form or $M \to_\beta M'$.*

Axiomatic Equivalence

We say that two valid typing judgements $\Delta \rhd M : A$ and $\Delta \rhd N : A$ are *axiomatically equivalent*, written $\Delta \rhd M \approx_{ax} N : A$, if the relation can be derived from

$$\begin{array}{llll}
\Delta \rhd (\lambda x.M)N & \approx_{ax} M[N/x] : A, & \Delta \rhd \lambda x.(Mx) & \approx_{ax} M : A \multimap B, \\
\Delta \rhd \mathit{fst}\langle M, N\rangle & \approx_{ax} M : A, & \Delta \rhd \mathit{snd}\langle M, N\rangle & \approx_{ax} N : B, \\
\Delta \rhd \langle \mathit{fst}(M), \mathit{snd}(M)\rangle & \approx_{ax} M : A \times B, & \Delta \rhd x & \approx_{ax} * : \top,
\end{array}$$

and from one congruence rule per term construct, such as

$$\frac{\Delta \rhd M \approx_{ax} M' : A \Rightarrow B \quad \Delta \rhd N \approx_{ax} N' : A}{\Delta \rhd MN \approx_{ax} M'N' : B.}$$

Curry Style Versus Church Style

While defining the type system, we wrote

> " A typed term is the data consisting of the term and its type. "

There are actually two distinct ways of typing a term. One can adopt the Church-style typing convention, and consider that all of its subterms are typed. Or one can adopt the Curry-style typing convention, and see only the outside term as typed. From the point of view of the programmer, it is simpler to type a lambda-term in a Curry-style fashion. However, to be able to check whether a typed term is valid or not it is easier to have it Church-style typed.

These two ways of typing terms might or might not be equivalent: it is a question of understanding the *meaning* of the term. For example, consider the Curry-style typing judgement

$$x : A \rhd (\lambda f.x)(\lambda y.y) : A.$$

The type of the variable x is given. However, the type of the variable y is not mentioned anywhere and might be anything. For each possible type B, the typing derivation

$$\dfrac{\dfrac{\dfrac{}{x : A, f : B \Rightarrow B \rhd x : A}}{x : A \rhd \lambda f.x : (B \Rightarrow B) \Rightarrow A} \quad \dfrac{\dfrac{}{x : A, y : B \rhd y : B}}{x : A \rhd \lambda y.y : B \Rightarrow B}}{x : A \rhd (\lambda f.x)(\lambda y.y) : A}$$

is valid. Therefore the data consisting of a set of typed variables, a term and a type does not uniquely determine a typing derivation. In a Curry-style type system, the axiomatic equivalence is really defined on typing derivations, and not typing judgements.

Obviously, in this simple example the term reduces to x, and thus all possible typing derivations live in the same axiomatic equivalence class. But in some more complicated language, this might not be the case, and the correspondence between typing derivations and typing judgements might not be one-to-one.

5.2 Proofs as Computations

Let us consider a typing derivation. We can concentrate on the meaning of terms, and understand the typing tree as a computation. But it is also possible to focus on the types, and see the typing tree as a proof in some logical system.

$$\frac{}{A_1, \ldots, A_n \rhd A_k}\ Ax \qquad \frac{}{\Gamma \rhd \top}\ \top_I$$

$$\frac{\Gamma \rhd A \quad \Gamma \rhd B}{\Gamma \rhd A \times B}\ \times_I \qquad \frac{\Gamma \rhd A \times B}{\Gamma \rhd A}\ \times_{E1} \qquad \frac{\Gamma \rhd A \times B}{\Gamma \rhd B}\ \times_{E2}$$

$$\frac{\Gamma, A \rhd B}{\Gamma \rhd A \Rightarrow B}\ \Rightarrow_I \qquad \frac{\Gamma \rhd A \Rightarrow B \quad \Gamma \rhd A}{\Gamma \rhd B}\ \Rightarrow_E$$

$$\frac{\Gamma \rhd B}{\Gamma, A \rhd B}\ W \qquad \frac{\Gamma, A, A \rhd B}{\Gamma, A \rhd B}\ C$$

Table 5.1: Intuitionistic logic: natural deduction rules

5.2.1 Intuitionistic Logic

The logic behind the simply typed lambda calculus described in Section 5.1.5 is called *intuitionistic logic*. Formulae are constructed as expressions coming from the grammar

$$\textit{Formula } A, B \ ::= \ \iota \mid \top \mid A \times B \mid A \Rightarrow B,$$

where ι ranges over a set of constants. $\times$ is the *conjunction*, and $\Rightarrow$ is the *implication*.

A *sequent* in intuitionistic logic is a pair (Γ, B), written as $\Gamma \rhd B$, where Γ is a multiset (possibly empty) of formulae $\{A_1, \ldots, A_n\}$ and B is a formula. We read the sequent as "A_1 and A_2 and... and A_n implies B", and we call Γ the *context* of the sequent.

A sequent is called *valid* it can be derived from a *proof*. Proofs are trees of formulae defined inductively. In intuitionistic logic a proof is said to be *constructive*: the proof of $A \times B$ should include the proof of A and the proof of B. The proof of $A \multimap B$ should be a map from the proofs of A to the proofs of B.

We will concentrate on *natural deduction* style proof rules, first introduced by by Gentzen (1934), where we mainly deal with the right side of the sequents. The proofs associated with the formulae presented above are constructed using the rules in Table 5.1. The rules come in two flavors:

1. the axiom Ax and two structural rules: the *weakening* rule W and the *contraction* rule C,
2. *introduction* and *elimination* for each connective (except for the null-ary connective $\top$ which has only an introduction rule).

5.2.2 Curry-Howard Isomorphism

Being able to write proofs, we now turn to the question of an *equivalence relation* of proofs. Given two proofs of the same sequent, when can we say that they are equal ? For example, it is reasonable to expect that

$$\frac{\dfrac{\dfrac{}{A, B \rhd A}\ Ax \quad \dfrac{}{A, B \rhd B}\ Ax}{A, B \rhd A \times B}\ \times_I}{A, B \rhd A}\ \times_{E1} \qquad \text{and} \qquad \frac{}{A, B \rhd A}\ Ax \tag{5.2.1}$$

are the same proof. Indeed, in the first case we do not use $A, B \rhd B$: the proof only "forgets" it in the last step.

By comparing the rules for building proofs and the typing rules of the simply-typed lambda calculus, one can give a computational interpretation of proofs. It is known as the *Curry-Howard isomorphism*. Under this scheme, one can say that two proofs are equivalent if they represent two axiomatically equivalent lambda-terms.

If we re-interpret the reduction rules for the lambda calculus in the context of proof of intuitionistic linear logic, one finds a formal definition of the "similar look" of the two proofs of Equation 5.2.1. The first proof correspond to the typing judgement $x : A, y : B \rhd \pi_1(\langle x, y\rangle) : A$ and the second to the typing judgement $x : A, y : B \rhd x : A$.

Interpreted in the context of proofs, the β-reduction rules correspond to the removal of an introduction followed by an elimination. We can construct the following rewrite system of proofs:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ \Gamma \rhd A\end{array} \quad \begin{array}{c}\pi_2\\ \vdots\\ \Gamma \rhd B\end{array}}{\Gamma \rhd A \times B}\times_I}{\Gamma \rhd A}\times_{E1} \quad \rightsquigarrow \quad \begin{array}{c}\pi_1\\ \vdots\\ \Gamma \rhd A,\end{array}$$

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ \Gamma \rhd A\end{array} \quad \begin{array}{c}\pi_2\\ \vdots\\ \Gamma \rhd B\end{array}}{\Gamma \rhd A \times B}\times_I}{\Gamma \rhd B}\times_{E2} \quad \rightsquigarrow \quad \begin{array}{c}\pi_2\\ \vdots\\ \Gamma \rhd A,\end{array}$$

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ \Gamma, A \rhd B\end{array}}{\Gamma \rhd A \Rightarrow B}\Rightarrow_I \quad \begin{array}{c}\pi_2\\ \vdots\\ \Gamma \rhd A\end{array}}{\Gamma \rhd B}\Rightarrow_{E2} \quad \rightsquigarrow \quad \begin{array}{c}\pi_1^*\\ \vdots\\ \Gamma \rhd B,\end{array}$$

where π_1^* is π_1 without the A in the context and whenever an axiom of π_1 is of the form $\Gamma, A, \Delta \rhd A$ it is replaced by the proof

$$\begin{array}{c}\pi_2\\ \vdots\\ \Gamma, \Delta \rhd A.\end{array}$$

5.3 Categorical Logic

A proof can be set in relation with a term in a suitable language. However, while this describes the equivalence classes of proofs, it does not give out the internal structure of the theory.

The structure of the proof system can be extracted using a categorical description of the situation. In categorical logic, we consider a category whose objects are formulae and whose arrows $A \to B$ are equivalence classes of proofs of the sequent $A \rhd B$. Equivalently, objects are types and arrows are equivalence classes of typing judgements $x : A \rhd M : B$.

In this section we describe the category generated by the language of Section 5.1.5 (or, equivalently, of the proof theory defined in Section 5.2.1).

Definition 5.3.1. Given a cartesian closed category $(\mathcal{C}, \times, \Rightarrow, T)$, let

$$\Phi : \hom(A \times B, C) \xrightarrow{\sim} \hom(A, B \Rightarrow C)$$

be the canonical isomorphism and $\varepsilon_{A,B} : (A \Rightarrow B) \times A \to B$ the counit of the adjunction.

Given a map ψ from type variables to objects of $\mathcal{C}$, one defines an interpretation of any type by induction:

$$[\![A \times B]\!]_\psi = [\![A]\!]_\psi \times [\![B]\!]_\psi, \qquad [\![A \Rightarrow B]\!]_\psi = [\![A]\!]_\psi \Rightarrow [\![B]\!]_\psi,$$

and one can interpret valid typing judgements of the simply-typed lambda calculus as morphisms in $\mathcal{C}$:

$$[\![x_1 : A_1, \dots x_n : A_n \rhd M : B]\!]_\psi : [\![A_1]\!]_\psi \times \cdots [\![A_n]\!]_\psi \to [\![B]\!]_\psi.$$

Axiom:

$$\frac{}{\Delta, x : A \rhd x : A} \qquad \frac{}{\pi^2 : [\![\Delta]\!] \times [\![A]\!] \to [\![A]\!]}$$

Unit:

$$\frac{}{\Delta \rhd * : \top} \qquad \frac{}{\bigcirc_{[\![\Delta]\!]} : [\![\Delta]\!] \to T}$$

Product:

$$\frac{\Delta \rhd M : A \quad \Delta \rhd N : A}{\Delta \rhd \langle M, N\rangle : A \times B} \qquad \frac{f : [\![\Delta]\!] \to [\![A]\!] \quad g : [\![\Delta]\!] \to [\![B]\!]}{\langle f, g\rangle : [\![\Delta]\!] \to [\![A]\!] \times [\![B]\!]}$$

Application:

$$\frac{\Delta \rhd M : A \Rightarrow B \quad \Delta \rhd N : A}{\Delta \rhd MN : B} \qquad \frac{f : [\![\Delta]\!] \to [\![A]\!] \Rightarrow [\![B]\!] \quad g : [\![\Delta]\!] \to [\![A]\!]}{[\![\Delta]\!] \xrightarrow{\langle f,g\rangle} ([\![A]\!] \Rightarrow [\![B]\!]) \times [\![A]\!] \xrightarrow{\varepsilon_{A,B}} [\![B]\!],}$$

Abstraction:

$$\frac{\Delta, x : A \rhd M : B}{\Delta \rhd \lambda x.M : A \Rightarrow B} \qquad \frac{f : [\![\Delta]\!] \times [\![A]\!] \to [\![B]\!]}{\Phi(f) : [\![\Delta]\!] \to [\![A]\!] \Rightarrow [\![B]\!]}$$

Table 5.2: Interpretation of the simply typed lambda calculus

They are defined by induction as in Table 5.2. The denotation of a context $\Delta = x_1 : A_1, \ldots x_n : A_n$ is defined by

$$[\![\Delta]\!]_\psi = [\![A_1]\!]_\psi \times \cdots [\![A_n]\!]_\psi.$$

If the context Δ is empty, then $[\![\Delta]\!] = \top$. For legibility one drops the ψ on the symbol $[\![-]\!]_\psi$.

Lemma 5.3.2. *The interpretation is sound: If $\Delta \rhd M \approx_{ax} M' : A$ then*

$$[\![\Delta \rhd M : A]\!] = [\![\Delta \rhd M' : A]\!].$$

Definition 5.3.3. Let $\mathcal{C}_\lambda^{st}$ be the category defined as follows:

- Objects are types of the simply typed lambda calculus.
- Arrows from A to B are β-equivalence classes of typing judgements $x : A \rhd M : B$.
- The identity map on A is $x : A \rhd x : A$, and the composition of $x : A \rhd M : B$ and $y : B \rhd N : C$ is $x : A \rhd (\lambda y.N)M : C$.

Lemma 5.3.4. *The category $\mathcal{C}_\lambda^{st}$ is cartesian closed.*

Theorem 5.3.5. *The simply-typed lambda calculus is an internal language for cartesian categories: In $\mathcal{C}_\lambda^{st}$, we have*

$$[\![x : A \rhd M : B]\!]_{\mathcal{C}_\lambda^{st}} \approx_{ax} (x : A \rhd M : B).$$

5.4 Lambda Calculus and Side Effects

5.4.1 Pure Versus Impure Calculus.

The simply typed lambda calculus as model of computation is slightly restrictive: It allow only *pure* computations, that is, computations with no side effects. A program is said to produce some *side effect* if the returned value does not entirely describe the behaviour of the program. Here are several examples of side effects:

1. Probabilistic choice. A program that tosses a coin along the computation will have side effects: two runs of the program might not give the same output value.

2. Non-termination. A program that can diverge (i.e. not return anything) will have side effects, since no value can describe the behaviour of the program.

3. External state. A program that read and/or write some data on some external state is subject to side-effects: the value the program output might depend on the external state.

5.4.2 Reduction Strategies

In the event of side effects, the order of reduction of the redexes might influence the output, or even the behaviour of the program (in the case of divergence, for example). The description of a program allowing side effects has to come with a description of *values* and a choice of *reduction strategy*. The *values* are special terms that do not reduce anymore under the chosen strategy. For a simply-typed lambda calculus with side effects, the values are defined as

$$\textit{Value} \quad V, W \quad ::= x \mid \lambda x.M \mid \langle V, W \rangle \mid *.$$

Two main strategies are the following:

1. Call-by-name. In this strategy, one first applies functions to arguments before evaluating them. The rules for the application is

$$\frac{M \to M'}{MN \to M'N}\ , \qquad \frac{}{(\lambda x.M)N \to M[N/x]}\ .$$

2. Call-by-value. In this strategy, one reduces arguments to values before applying a function to them. Rules for application can be written as follows:

$$\frac{N \to N'}{MN \to MN'}\ , \qquad \frac{M \to M'}{MV \to M'V}\ , \qquad \frac{}{(\lambda x.M)V \to M[V/x]}\ .$$

In the following we will concentrate on a model for call-by-value based computation.

5.4.3 Towards a Semantics

The categorical analysis developed in Section 5.3 does not apply to a lambda calculus with side effects. In order to make categorical sense of the calculus, one has to separate the analysis of the *values* from the analysis of the *computations* (that is, the arbitrary terms).

The semantics is due to Moggi (1988, 1989, 1991). A categorical model for a lambda calculus with side effects is a cartesian category $(\mathcal{C}, \times, T)$ together with a strong monad (T, μ, η, t).

Definition 5.4.1 (Moggi, 1991)**.** A *strong monad* over a monoidal category $\mathcal{C}$ is a monad (T, η, μ) together with a natural transformation $t_{A,B} : A \otimes TB \to T(A \otimes B)$, call the *tensorial strength*, such

that the diagrams

$$\begin{tikzcd}
\top \otimes TA \arrow[r, "\lambda"] \arrow[dr, "t"] & TA \\
A \otimes B \arrow[d, "id\otimes\eta"'] \arrow[dr, "\eta"] & T(\top \otimes A), \arrow[u, "T\lambda"'] \\
A \otimes TB \arrow[r, "t"] & T(A \otimes B) \\
A \otimes T^2B \arrow[u, "id\otimes\mu"] \arrow[r, "t"] & T(A \otimes TB) \arrow[r, "Tt"] & T^2(A \otimes B) \arrow[ul, "\mu"']
\end{tikzcd} \tag{5.4.1}$$

$$\begin{tikzcd}
(A \otimes B) \otimes TC \arrow[d, "t"] & A \otimes (B \otimes TC) \arrow[l, "\alpha"'] \arrow[d, "id\otimes t"] \\
T((A \otimes B) \otimes C) & A \otimes T(B \otimes C) \arrow[d, "t"] \\
 & T(A \otimes (B \otimes C)), \arrow[ul, "T(\alpha)"]
\end{tikzcd} \tag{5.4.5}$$

commute. (5.4.4)

Remark 5.4.2 (Moggi, 1991)**.** If the category $\mathcal{C}$ is symmetric, the tensorial strength t induces two natural transformations $TA \otimes TB \to T(A \otimes B)$, namely

$$\Psi_1 : TA \otimes TB \xrightarrow{\sigma_{TA,TB}} TB \otimes TA \xrightarrow{t_{TB,A}} T(TB \otimes A) \xrightarrow{(\sigma_{TB,A};t_{A,B})^*} T(A \otimes B),$$
$$\Psi_2 : TA \otimes TB \xrightarrow{t_{TA,B}} T(TA \otimes B) \xrightarrow{(\sigma_{TA,B};t_{B,A})^*} T(B \otimes A) \xrightarrow{T\sigma_{B,A}} T(A \otimes B).$$

Note that since $\sigma_{TA,B};t_{B,A} : TA \otimes B \to T(B \otimes A)$, $(\sigma_{TA,B};t_{B,A})^*$ is a morphism $T(TA \otimes B) \to T(B \otimes A)$. Note also that Ψ_1 and Ψ_2 might not be equal: the map Ψ_1 "evaluates" the second variable and then the first one. The map Ψ_2 does the opposite. The strong monad is called *commutative* if $\Psi_1 = \Psi_2$.

Lemma 5.4.3. *If T is a commutative strong monad on a symmetric monoidal category $\mathcal{C}$, then the two natural transformations Ψ_1 and Ψ_2 make both T monoidal.*

The category $\mathcal{C}$ is not quite cartesian closed. Indeed, from a computation $x : A \rhd M : B$ one retrieve a value $\lambda x.M : A \Rightarrow B$. We ask the category to have exponentials in the following sense.

Definition 5.4.4. A symmetric monoidal category $(\mathcal{C}, \otimes, \top)$ together with a strong monad (T, η, μ) is said to have *T-exponentials* (Moggi, 1991), or *Kleisli exponentials*, if it is equipped with a bifunctor $\multimap : \mathcal{C}^{op} \times \mathcal{C} \to \mathcal{C}$, and a natural isomorphism

$$\Phi : \mathcal{C}(A, B \multimap C) \xrightarrow{\cong} \mathcal{C}(A \otimes B, TC).$$

Lemma 5.4.5. *The map Φ induces a natural transformation $\varepsilon_{A,B} : (A \multimap B) \otimes A \to TB$ defined by $\Phi(id_{A \multimap B})$.* □

5.4.4 Computational Model for Call-By-Value

The typed lambda calculus developed by Moggi (1991) for dealing with side-effects is called a *computational lambda calculus*. It is interpreted in a *computational model*, that is:

- a cartesian category $(\mathcal{C}, \times, T)$,
- with a strong monad (T, μ, η, t) satisfying the mono requirement,
- and Kleisli exponentials $A \Rightarrow B$ for every A, B objects in $\mathcal{C}$.

The values are interpreted as maps in $\mathcal{C}$, and the computations in $\mathcal{C}_T$. The category $\mathcal{C}$ is called the *category of values* and the category $\mathcal{C}_T$ the *category of computations*.

5.5 Intuitionistic Linear Logic

Linear logic is a resource-sensitive logic developed by Girard (1987). It tries to address the fact that in classical logic, the introduction rule for the conjunction $\times$

$$\frac{\Delta \rhd A \quad \Delta \rhd B}{\Delta \rhd A \times B} \times_I$$

can be derived from two distinct rules, respectively an *additive* and a *multiplicative* one:

$$\frac{\Delta \rhd A \quad \Delta \rhd B}{\Delta \rhd A \times B} \times_I^a, \qquad \frac{\Delta \rhd A \quad \Gamma \rhd B}{\Delta, \Gamma \rhd A \times B} \times_I^m,$$

where Δ and Γ are disjoint. If these two rules are "the same" in classical logic, it is because of the existence of the *weakening* and *contraction* rules

$$\frac{\Delta \rhd B}{\Delta, A \rhd B} W, \qquad \frac{\Delta, A, A \rhd B}{\Delta, A \rhd B} C.$$

Using (W) and $(\times_I^m)$ one can recover $(\times_I^a)$:

$$\frac{\dfrac{\Delta, \Gamma \rhd A \quad \Delta, \Gamma \rhd B}{\Delta, \Gamma, \Delta, \Gamma \rhd A \times B} \times_I^m}{\Delta, \Gamma \rhd A \times B} W,$$

and using (C) and $(\times_I^a)$ one can recover $(\times_I^m)$:

$$\frac{\dfrac{\Delta \rhd A}{\Delta, \Gamma \rhd A} W \quad \dfrac{\Gamma \rhd A}{\Delta, \Gamma \rhd B} W}{\Delta, \Gamma \rhd A \times B} \times_I^a.$$

In linear logic, the contraction and weakening rules are not valid in general, so for example A and $A \times A$ are not isomorphic, and the product is split into its additive and multiplicative components $\times$ and $\otimes$.

In the *intuitionistic multiplicative fragment*, formulae are defined by

$$A, B \quad ::= \quad \iota \mid \top \mid A \otimes B \mid A \multimap B \mid !A,$$

where ι ranges over a set of constants, $\otimes$ is the multiplicative conjunction, called the *tensor*, $\top$ stands for the unit of the tensor, and $A \multimap B$ is the implication corresponding to the tensor. $!(-)$ is a unary constructor for allowing weakening and contraction:

$$\frac{!A, !A \rhd B}{!A \rhd B} (C), \qquad \frac{\rhd B}{!A \rhd B} (W).$$

5.6 Linear Calculi and their Interpretations

Numerous interpretations of intuitionistic linear logic have been proposed, both at an abstract level and at a computational level.

Very good references for this section are the Ph.D. theses of Bierman (1993) and Maneggia (2004) and the review of Melliès (2002).

5.6.1 Earlier Models

One of the first well-known categorical descriptions of the system of proofs of linear logic was (Seely, 1989). The paper discusses the required structures for a model of linear logic, where the structure of the operator "!" is captured by a comonad arising in the context of an adjunction between a cartesian category and a symmetric monoidal category.

In his thesis, Lafont (1988a,b) also describes a model of intuitionistic linear logic where the modality ! is this time defined in terms of a comonoid structure. The logic is thought of as a type system. In that spirit, the type $!A$ is defined via its constructors and destructors, as would be the types of integers or booleans.

Abramsky (1993) and Wadler (1992) describe a computational interpretation of intuitionistic linear logic, with a typed lambda calculus whose typing rules follows closely the inference rules of the logic. The calculus is a lambda calculus with special term constructs to account for duplicable terms.

5.6.2 Bierman's Linear Category

The formulation generalizing all of the above descriptions is the one proposed by Bierman (1993) in his Ph.D. thesis, simplified afterward by Benton et al. (1992, 1993). The categorical description comes with a complete lambda calculus that we skip for the purpose of this thesis.

For the definition of the category, we prefer here the terminology given in Schalk (2004), and use the concept of *linear exponential comonad*.

Definition 5.6.1. Let $(\mathcal{C}, \otimes, \top, \alpha, \lambda, \rho, \sigma)$ be a symmetric monoidal category. Let $(L, \delta, \epsilon, d^L, d^L)$ be a monoidal comonad. We say that L is a *linear exponential comonad* provided that

1. each object in $\mathcal{C}$ of the form LA is equipped with a commutative comonoid $(LA, \triangle_A, \diamondsuit_A)$, where $\triangle_A : LA \to LA \otimes LA$ and $\diamondsuit_A : LA \to \top$;

2. $\triangle_A$ and $\diamondsuit_A$ are monoidal natural transformations, i.e. the following diagram

$$\begin{tikzcd}
LA \otimes LB \arrow[r, "\triangle_A \otimes \triangle_B"] \arrow[dd, "d^L_{A,B}"'] & (LA \otimes LA) \otimes (LB \otimes LB) \arrow[d, "sw"] \\
 & (LA \otimes LB) \otimes (LA \otimes LB) \arrow[d, "d^L_{A,B} \otimes d^L_{A,B}"] \\
L(A \otimes B) \arrow[r, "\triangle_{(A \otimes B)}"'] & L(A \otimes B) \otimes L(A \otimes B)
\end{tikzcd} \tag{5.6.1}$$

and the diagrams

$$\begin{tikzcd}
\top \arrow[r, "\lambda_\top^{-1}"] \arrow[d, "d^L_\top"'] & \top \otimes \top \arrow[d, "d^L_\top \otimes d^L_\top"] \\
L\top \arrow[r, "\triangle_\top"'] & L\top \otimes L\top
\end{tikzcd} \tag{5.6.2}$$

$$\begin{tikzcd}
LA \otimes LB \arrow[r, "\diamondsuit_A \otimes \diamondsuit_B"] \arrow[d, "d^L_{A,B}"'] & \top \otimes \top \arrow[d, "\lambda_\top"] \\
L(A \otimes B) \arrow[r, "\diamondsuit_{A \otimes B}"'] & \top
\end{tikzcd} \tag{5.6.3}$$

$$\begin{tikzcd}
\top \arrow[rr, "id"] \arrow[dr, "d^L_\top"'] & & \top; \\
 & L\top \arrow[ur, "\diamondsuit_\top"'] &
\end{tikzcd} \tag{5.6.5}$$

commutes.

3. The maps

$$\triangle_A : (LA, \delta_A) \to (LA \otimes LA, (\delta_A \otimes \delta_A); d_A),$$
$$\diamondsuit_A : (LA, \delta_A) \to (\top, d^L_\top)$$

are L-coalgebra morphisms, i.e.

$$\begin{tikzcd} LA \arrow[r, "\triangle_A"] \arrow[dd, "\delta_A"'] & LA \otimes LA \arrow[d, "\delta_A \otimes \delta_A"] \\ & L^2A \otimes L^2A \arrow[d, "d^L_{LA,LA}"] \\ L^2A \arrow[r, "L\triangle_A"'] & L(LA \otimes LA), \end{tikzcd} \tag{5.6.6}$$

$$\begin{tikzcd} LA \arrow[r, "\diamondsuit_A"] \arrow[dd, "\delta_A"'] & \top \arrow[dd, "d^L_\top"] \\ & \\ L^2A \arrow[r, "L\diamondsuit_A"'] & L\top; \end{tikzcd} \tag{5.6.7}$$

4. Every map δ_A is a comonoid morphism $(LA, \diamondsuit_A, \triangle_A) \to (L^2A, \diamondsuit_{LA}, \triangle_{LA})$, i.e.

$$\begin{tikzcd} LA \arrow[r, "\delta_A"] \arrow[d, "\triangle_A"'] & L^2A, \arrow[d, "\triangle_{LA}"] \\ LA \otimes LA \arrow[r, "\delta_A \otimes \delta_A"'] & L^2A \otimes L^2A \end{tikzcd} \tag{5.6.8}$$

$$\begin{tikzcd} LA \arrow[rr, "\delta_A"] \arrow[dr, "\diamondsuit_A"'] & & L^2A. \arrow[dl, "\diamondsuit_{LA}"] \\ & \top & \end{tikzcd} \tag{5.6.9}$$

Definition 5.6.2 (Bierman, 1993)**.** A *linear category* is a symmetric monoidal category $(\mathcal{C}, \otimes, \top)$ with finite products $(\times, 1)$, together with a linear exponential comonad L.

Benton (1994) gave a clean formulation of the above category in terms of adjunctions. We follow Barber (1997) by not mentioning the cartesian closedness, and use the exposition of Melliès (2002) as reference.

Definition 5.6.3. A *linear-non-linear category* consists of

- A symmetric monoidal closed category $(\mathcal{C}, \otimes, \top)$;
- a category $(\mathcal{M}, \times, 1)$ with finite products;
- a symmetric monoidal adjunction

$$\begin{tikzcd} \mathcal{M} \arrow[rr, bend left, "U"] \arrow[rr, phantom, "\perp"] & & \mathcal{C}. \arrow[ll, bend left, "F"] \end{tikzcd}$$

Lemma 5.6.4. *Every linear category defines a linear-non-linear category, where* $(\mathcal{M}, \times, 1)$ *is the category of coalgebras of the comonad.* □

Theorem 5.6.5. *Every linear-non-linear category is a linear category.* □

Chapter 6

A Lambda Calculus for Quantum Computation

The question this thesis tries to address is the question of the *meaning* and the *description* of higher-order quantum information, in the paradigm of the QRAM model. That is, programs are written on a classical device but are given access to quantum information.

In this chapter we will describe a typed lambda calculus for quantum computation, together with an operational semantics. The requirements we have in mind for the language are the following:

1. The language should have classical control. That is, the structural aspect of the programs (pairing, tests, etc.) should be classical. Such a language will be described in Section 6.1.

2. We want the program to manipulate quantum data. So quantum information should be thought of as part of the code as much as possible. We give in Section 6.2 an abstract machine and an operational semantics for giving sense to the language.

3. We want the language to be as natural as possible for the programmer. In particular, the compiler should be the one to make sure that valid programs do not duplicate quantum variables. We do not want to have special term constructs for this. In Section 6.3 we define a type system to rule out invalid programs.

The work presented in this chapter appeared in my M.Sc. thesis (Valiron, 2004a). It was also published in (Selinger and Valiron, 2006a, 2005).

6.1 The Language

We adapt the lambda calculus described in Section 5.1 to manipulate classical data and quantum data. We focus primarily on the classical booleans 0 and 1, on the measurement, the creation of quantum bits and the unitary operations.

Definition 6.1.1. A lambda-term is an expression made of the following core grammar, written in BNF form:

$$\begin{array}{lll} \mathit{Term} & M, N, P & ::= \; x \mid MN \mid \lambda x.M \mid \mathit{if}\, M \;\mathit{then}\; N \;\mathit{else}\; P \mid 0 \mid 1 \mid \mathit{meas} \mid \mathit{new} \mid U \mid \\ & & \quad * \mid \langle M, N\rangle \mid \mathit{let}\;\; \langle x, y\rangle = M \;\mathit{in}\; N, \end{array}$$

where x ranges over an infinite set of *term variables* and U a set of unitary gates; the term $\mathit{if}\, M \;\mathit{then}\; N \,\mathit{else}\, P$ stands for a test on P with output M if true, N else; the constant terms 0 and 1 stand for the

boolean values false and true respectively; the constant terms *meas*, *new* and U stand respectively for the measurement, the creation and the application of a unitary gate on a quantum bit; the terms $\mathit{let}\ \langle x, y\rangle = M\ \mathit{in}\ N$ stands for the retrieval of the content of a pair.

We identify terms up to α-equivalence, defined as in Section 5.1.3.

Convention 6.1.2. We sometimes use the shorthand notations

$$\begin{array}{rcl}
\langle M_1, \ldots, M_n\rangle & = & \langle M_1, \langle M_2, \ldots\rangle\rangle, \\
M_1 M_2 \ldots M_n & = & (\ldots((M_1 M_2) M_3) \ldots M_n) \\
\mathit{let}\ x = M\ \mathit{in}\ N & = & (\lambda x.N)M, \\
\lambda\langle x, y\rangle.M & = & \lambda z.(\mathit{let}\ \langle x, y\rangle = z\ \mathit{in}\ N), \\
\lambda x_1 \ldots x_n.M & = & \lambda x_1.\lambda x_2. \ldots \lambda x_n.M, \\
\mathit{let}\ x\ y_1 \ldots y_n = M\ \mathit{in}\ N & = & \mathit{let}\ x = (\lambda y_1 \ldots y_n.M)\ \mathit{in}\ N.
\end{array}$$

The notions of free and bound variables and of substitution are defined as in Chapter 5. We give the full definition for further reference.

Definition 6.1.3. The set $FV(M)$ of *free variables* of a term M is defined as follows:

$$\begin{array}{rcl}
FV(*) = \emptyset, & & FV(x) = \{x\}, \\
FV(U) = \emptyset, & & FV(\lambda x.M) = FV(M) \setminus \{x\}, \\
FV(\mathit{new}) = \emptyset, & & FV(MN) = FV(M) \cup FV(N), \\
FV(\mathit{meas}) = \emptyset, & & FV(\langle M, N\rangle) = FV(M) \cup FV(N), \\
FV(0) = \emptyset, & & FV(\mathit{if}\, M\ \mathit{then}\ N\ \mathit{else}\ P) = FV(M) \cup FV(N) \cup FV(P), \\
FV(1) = \emptyset, & & FV(\mathit{let}\ \langle x, y\rangle = M\ \mathit{in}\ N) = FV(M) \cup (FV(N) \setminus \{x, y\}).
\end{array}$$

Definition 6.1.4. Given a term M and a term P, we define the *substitution of x in M by P*, written $M[P/x]$, by the following:

$$\begin{array}{rl}
x[P/x] = P & \\
c[P/x] = c & \\
*[P/x] = * & \\
(MN)[P/x] = (M[P/x])(N[P/x]) & \\
(\langle M, N\rangle)[P/x] = \langle M[P/x], N[P/x]\rangle & \\
(\mathit{if}\, M\ \mathit{then}\ N_1\ \mathit{else}\ N_2)[P/x] = \mathit{if}\, M[P/x]\ \mathit{then}\ N_1[P/x]\ \mathit{else}\ N_2[P/x] & \\
(\lambda x.M)[P/x] = \lambda x.M & \\
(\lambda y.M)[P/x] = \lambda y.(M[P/x]) & \quad (\text{if } y \neq x \text{ and } y \notin FV(P))
\end{array}$$

If $y = x$ or $z = x$:

$$(\mathit{let}\ \langle y, z\rangle = M\ \mathit{in}\ N)[P/x] = (\mathit{let}\ \langle y, z\rangle = (M[P/x])\ \mathit{in}\ N)$$

If $y \neq x$ and $z \neq x$ (and $y, z \notin FV(P)$):

$$(\mathit{let}\ \langle y, z\rangle = M\ \mathit{in}\ N)[P/x] = (\mathit{let}\ \langle y, z\rangle = (M[P/x])\ \mathit{in}\ (N[P/x])).$$

Example 6.1.5. Using this calculus, one can produce terms that compute quantum algorithms. For example, a fair coin can be implemented as follows:

$$\mathit{meas}(H(\mathit{new}\, 0)),$$

where H is the Hadamard gate.

6.2 Operational Semantics

Although we want the language to perform quantum computation, no quantum bit was added into the terms of the language. For example we might want to represent the constant function outputting a quantum bit, as follows:

$$\lambda x.(\alpha|0\rangle + \beta|1\rangle).$$

The reason why we do not allow such a notation is the problem of entanglement. As noted in Section 3.3.2, it is not always possible to write a two quantum bit system in the form $|\phi\rangle \otimes |\psi\rangle$. Therefore, if x is the variable corresponding to the first quantum bit and y the one corresponding to the second in an entangled state, it is not possible to write classically the term

$$(\lambda f.fx)(\lambda t.(gy)t),$$

as there is no means of writing x and y.

6.2.1 Abstract Machine

This non-local nature of quantum information forces us to introduce a level of indirection for representing the state of a quantum program. Following the scheme of the QRAM model, we describe an abstract machine consisting of a lambda-term M, a array of quantum bits Q (the QRAM), and a map linking variables in M to quantum bits in Q:

Definition 6.2.1. A *quantum closure* is represented by a triple $[Q, L, M]$, where

- Q is a normalized vector of $\otimes_{i=1}^{n}\mathbb{C}^2$, for some $n \geqslant 0$, called the *quantum array*.
- M is a lambda term,
- L is an injective function from a set $|L|$ of term variables to $\{1, \ldots, n\}$, where $FV(M) \subseteq |L|$. L is called the *linking function*.

We write $|Q| = n$. If $L(x_i) = i$, we will sometimes write L as the ordered list $|x_1 \cdots x_n\rangle$. The idea is that the variable x_i is bound in the term M to qubit number $L(x_i)$ of the state Q. We also call the pair (Q, L) a *quantum context*. If there are no quantum bits, i.e. if $Q \in \mathbb{C}$, we write $Q = |\rangle$. Similarly, if L is empty we write $L = |\rangle$.

The purpose of the linking function is to assign specific free variables of M to specific quantum bits in Q. The notion of α-equivalence extends naturally to quantum closure, for instance, the states $[|1\rangle, |x\rangle, \lambda y.x]$ and $[|1\rangle, |z\rangle, \lambda y.z]$ are equivalent. We therefore extend the notion of α-equivalence to quantum closures:

$$[\ Q, |x \cdots y \cdots z\rangle, M\ ] =_\alpha [\ Q, |x \cdots y' \cdots z\rangle, M[y'/y]\ ]$$

if $y' \notin FV(M) \cup \{x, \ldots, y, \ldots, z\}$.

6.2.2 Evaluation Strategy

Although we solved the problem of entanglement, there is another issue that prevents us from blindly using substitution, namely the probabilistic nature of the measurement.

Consider the term **plus** to be the boolean addition function, defined as

$$\mathbf{plus} = \lambda xy.\mathit{if}\, x\ \mathit{then}\ (\mathit{if}\, y\ \mathit{then}\ 0\ \mathit{else}\ 1)\ \mathit{else}\ (\mathit{if}\, y\ \mathit{then}\ 1\ \mathit{else}\ 0).$$

Now, consider the term $M = (\lambda x.\mathbf{plus}\ x\ x)(\mathit{meas}(H(\mathit{new}\ \ 0)))$. Depending of the choice made in reducing the term, we obtain a different answer, as shown below, using an intuitive reduction system.

Call-By-Value.

Reducing this in the empty environment, using the call-by-value reduction strategy (as defined in Section 5.4.2), we obtain the following reductions:

$$\begin{aligned}
[\ |\rangle, |\rangle, M\ ] &\to [\ |0\rangle, |p\rangle, (\lambda x.\mathbf{plus}\ x\ x)(\mathit{meas}(H\ p))\ ] \\
&\to [\ \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), |p\rangle, (\lambda x.\mathbf{plus}\ x\ x)(\mathit{meas}\ \ p)\ ] \\
&\to \begin{cases} [\ |0\rangle, |p\rangle, (\lambda x.\mathbf{plus}\ x\ x)(0)\ ] \\ [\ |1\rangle, |p\rangle, (\lambda x.\mathbf{plus}\ x\ x)(1)\ ] \end{cases} \\
&\to \begin{cases} [\ |0\rangle, |p\rangle, \mathbf{plus}\ 0\ 0\ ] \\ [\ |1\rangle, |p\rangle, \mathbf{plus}\ 1\ 1\ ] \end{cases} \qquad \to \begin{cases} [\ |0\rangle, |p\rangle, 0\ ] \\ [\ |1\rangle, |p\rangle, 0\ ] \end{cases}
\end{aligned} \tag{6.2.1}$$

where the two branches are taken with probability $\frac{1}{2}$ each. Thus, under call-by-value reduction, this program produces the boolean value 0 with probability 1.

Call-By-Name

Reducing the same term under the call-by-name strategy, we obtain in one step

$$[\ |\rangle, |\rangle, M\ ] \to [\ |\rangle, |\rangle, \mathbf{plus}\ (\mathit{meas}(H(\mathit{new}\ 0)))\ (\mathit{meas}(H(\mathit{new}\ 0)))\ ],$$

and then with probability $\frac{1}{4}$,

$$[\ |01\rangle, |pq\rangle, 1\ ], \quad [\ |10\rangle, |pq\rangle, 1\ ], \quad [\ |00\rangle, |pq\rangle, 0\ ], \quad [\ |11\rangle, |pq\rangle, 0\ ].$$

Therefore, the boolean output of this function is 0 or 1 with equal probability.

Mixed Strategy

Finally, if we mix the two reduction strategies, the program can even reduce to an ill-formed term. Namely, reducing by call-by-value until we reach the term

$$[\ \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), |p\rangle, (\lambda x.\mathbf{plus}\ x\ x)(\mathit{meas}\ p)\ ],$$

and then changing to call-by-name, we obtain in one step the term

$$[\ \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), |p\rangle, \mathbf{plus}(\mathit{meas}\ p)(\mathit{meas}\ p)\ ],$$

which is not a valid program since there are two occurrences of the quantum bit p.

Remark 6.2.2. In call-by-name, a measurement of the form $\mathit{meas}\ M$ is carried over along β-reductions. There is no possible way to "force" the measurement to happen. Therefore, one cannot obtain the behavior of the reductions in (6.2.1) using a call-by-name procedure. However, it is possible to simulate a call-by-name reduction in call-by-value by encapsulating terms in lambda-abstractions. For example, to carry over the term $\mathit{meas}\ M$ using a call-by-value reduction strategy, one can write the term as $\lambda y.\ \mathit{meas}\ M$, with y a fresh variable.

Convention 6.2.3. Following Remark 6.2.2, we consider the call-by-value reduction strategy in the remainder of this chapter.

6.2.3 Probabilistic Reduction Systems

In order to formalize the operational semantics of the quantum lambda calculus, we introduce the notion of a probabilistic reduction system. Note that several notions of probabilistic rewrite systems have been studied in the literature (see e.g. Bournez and Hoyrup, 2003; Sen et al., 2003). However, these works study rewrite system in a very general setting. For the purpose of this work, where our probabilistic reduction is quite simple, we define an abstract notion of probabilistic reduction only to capture the notion of value and error state and not to study the system itself.

Definition 6.2.4. A *probabilistic reduction system* is a tuple $(X, U, R, prob)$ where X is a set of *states*, $U \subseteq X$ is a subset of *value states*, $R \subseteq (X \setminus U) \times X$ is a set of *reductions*, and $prob : R \to [0, 1]$ is a *probability function*, where $[0, 1]$ is the real unit interval. Moreover, we impose the following conditions:

- For any $x \in X$, $R_x = \{ x' \mid (x, x') \in R \}$ is finite.
- $\sum_{x' \in R_x} prob(x, x') \leqslant 1$

We call *prob* the one-step reduction, and denote $x \to_p y$ to be $prob(x, y) = p$. Let us extend *prob* to the n-step reduction

$$prob^0(x, y) = \begin{cases} 0 & \text{if } x \neq y \\ 1 & \text{if } x = y \end{cases}$$

$$prob^1(x, y) = \begin{cases} prob(x, y) & \text{if } (x, y) \in R \\ 0 & \text{else} \end{cases}$$

$$prob^{n+1}(x, y) = \sum_{z \in R_x} prob(x, z) prob^n(z, y),$$

and the notation is extended to $x \to_p^n y$ to mean $prob^n(x, y) = p$.

Definition 6.2.5. We say that y is *reachable in one step with non-zero probability* from x, denoted $x \to_{>0} y$ when $x \to_p y$ with $p > 0$. We say that y is *reachable with non-zero probability* from x, denoted $x \to_{>0}^* y$ when there exists $n \geqslant 0$ such that $x \to_p^n y$ with $p > 0$.

Definition 6.2.6. We can then compute the probability to reach $u \in U$ from x: It is a function from $X \times U$ to $\mathbb{R}$ defined by $prob_U(x, u) = \sum_{n=0}^{\infty} prob^n(x, u)$. The total probability for reaching U from x is

$$prob_U(x) = \sum_{n=0}^{\infty} \sum_{u \in U} prob^n(x, u).$$

On the other hand, there is also the probability to *diverge* from x, or never reaching anything. This value is

$$prob_\infty(x) = \lim_{n \to \infty} \sum_{y \in X} prob^n(x, y).$$

We define the *error probability of* x to be the number

$$prob_{err}(x) = 1 - prob_U(x) - prob_\infty(x).$$

Lemma 6.2.7. *For all $x \in X$, $prob_U(x) + prob_\infty(x) \leqslant 1$. That is, $0 \leqslant prob_{err}(x) \leqslant 1$.* □

Definition 6.2.8. In addition to the notion of reachability with non-zero probability, there is also a weaker notion of reachability, given by R: We will say that y is *reachable in one step* from x, written $x \rightsquigarrow y$, if xRy. By the properties of R, $x \to_{>0} y$ implies $x \rightsquigarrow y$. As usual, $\rightsquigarrow^*$ denotes the transitive reflexive closure of $\rightsquigarrow$, and we say that y is *reachable* from x if $x \rightsquigarrow^* y$.

Definition 6.2.9. In a probabilistic reduction system, a state x is called an *error-state* if $x \notin U$ and $\sum_{x' \in X} prob(x, x') < 1$. An element $x \in X$ is *consistent* if there is no error-state e such that $x \rightsquigarrow^* e$.

Lemma 6.2.10. *If x is consistent, then $prob_{err}(x) = 0$.* □

Remark 6.2.11. We need the weaker notion of reachability $x \rightsquigarrow y$, in addition to reachability with non-zero probability $x \rightarrow_{>0}{}^* y$, because a null probability of getting a certain result is not an absolute warranty of its impossibility. In the QRAM, suppose we have a qubit in state $|0\rangle$. Measuring it cannot theoretically yield the value 1, but in practice, this might happen with small probability, due to imprecision of the physical operations and decoherence. Therefore, when we prove subject reduction (see Theorem 6.4.5), we will use the stronger notion. In short: a type-safe program should not crash, even in the event of random QRAM errors.

Remark 6.2.12. The converse of Lemma 6.2.10 is false. For instance, if $X = \{a, b\}$, $U = \emptyset$, $a \rightarrow_1 a$, and $a \rightarrow_0 b$, then b is an error state, and b is reachable from a, but only with probability zero. Hence $prob_{err}(a) = 0$, although a is inconsistent.

6.2.4 Reduction System

We define a probabilistic call-by-value reduction procedure for the quantum lambda calculus. Note that, although the reduction itself is probabilistic, the choice of which redex to reduce at each step is deterministic.

Definition 6.2.13. A *value* is a term of the following form:

$$\textit{Value} \quad V, W \quad ::= \quad x \mid \lambda x.M \mid 0 \mid 1 \mid \textit{meas} \mid \textit{new} \mid U \mid * \mid \langle V, W \rangle.$$

A quantum closure of the form $[Q, L, V]$ where V is a value is called a *quantum value-state*.

The reduction rules are shown in Table 6.1. We write $[Q, L, M] \rightarrow_p [Q', L', M']$ for a single-step reduction of states which takes place with probability p. In the rule for reducing the term $U\langle x_{j_1}, \ldots, x_{j_n} \rangle$, U is an n-ary built-in unitary gate, $j_1, \ldots, j_n$ are pairwise distinct, and Q' is the quantum state obtained from Q by applying this gate to qubits $j_1, \ldots, j_n$. In the rule for measurement, $|Q_0\rangle$ and $|Q_1\rangle$ are normalized states of the form

$$|Q_0\rangle = \sum_j \alpha_j |\phi_j^0\rangle \otimes |0\rangle \otimes |\psi_j^0\rangle, \qquad |Q_1\rangle = \sum_j \beta_j |\phi_j^1\rangle \otimes |1\rangle \otimes |\psi_j^1\rangle,$$

where ϕ_j^0 and ϕ_j^1 are i-qubit states (so that the measured qubit is the one pointed to by x_i). In the rule for *new*, Q is an n-qubit state, so that $Q \otimes |i\rangle$ is an $(n+1)$-qubit state.

Definition 6.2.14. We define a weaker relation $\rightsquigarrow$. This relation models the transformations that can happen in the presence of decoherence and imprecision of physical operations. We define $[Q, M] \rightsquigarrow [Q', M']$ to be $[Q, M] \rightarrow_p [Q', M']$, even when $p = 0$, plus the additional rule, if Q and Q' are vectors of equal dimensions: $[Q, M] \rightsquigarrow [Q', M]$.

Lemma 6.2.15. *Let X be the state of all quantum closures and U the set of quantum value-states, Let prob be the function such that for $x, y \in X$, $prob(x, y) = p$ if $x \rightarrow_p y$ and 0 else. Then $(X, U, \rightsquigarrow, \rightarrow)$ is a probabilistic reduction system.* □

Remark 6.2.16. This probabilistic reduction system has error states, for example, the states $[Q, L, H(\lambda x.x)]$ or $[Q, |xyz\rangle, U\langle x, x\rangle]$. Such error states correspond to run-time errors. In the next section, we introduce a type system designed to rule out such error states.

$$[\, Q, L, (\lambda x.M)V \,] \to_1 [\, Q, L, M[V/x] \,]$$

$$\frac{[\, Q, N \,] \to_p [\, Q'L, , N' \,]}{[\, Q, L, MN \,] \to_p [\, Q', L, MN' \,]}$$

$$\frac{[\, Q, M \,] \to_p [\, Q', L, M' \,]}{[\, Q, L, MV \,] \to_p [\, Q', L, M'V \,]}$$

$$\frac{[\, Q, L, M_1 \,] \to_p [\, Q', L, M_1' \,]}{[\, Q, L, \langle M_1, M_2 \rangle \,] \to_p [\, Q', L, \langle M_1', M_2 \rangle \,]}$$

$$\frac{[\, Q, L, M_2 \,] \to_p [\, Q', L, M_2' \,]}{[\, Q, L, \langle V_1, M_2 \rangle \,] \to_p [\, Q', L, \langle V_1, M_2' \rangle \,]}$$

$$[\, Q, L, \mathit{if}\, 0 \ \mathit{then}\ M \ \mathit{else}\ N \,] \to_1 [\, Q, L, N \,]$$

$$[\, Q, L, \mathit{if}\, 1 \ \mathit{then}\ M \ \mathit{else}\ N \,] \to_1 [\, Q, L, M \,]$$

$$[\, Q, L, U\langle x_{j_1}, \ldots, x_{j_n} \rangle \,] \to_1 [\, Q', L, \langle x_{j_1}, \ldots, x_{j_n} \rangle \,]$$

$$[\, \alpha|Q_0\rangle + \beta|Q_1\rangle, L, \mathit{meas}\ \ x_i \,] \to_{|\alpha|^2} [\, |Q_0\rangle, L, 0 \,]$$

$$[\, \alpha|Q_0\rangle + \beta|Q_1\rangle, L, \mathit{meas}\ \ x_i \,] \to_{|\beta|^2} [\, |Q_1\rangle, L, 1 \,]$$

$$[\, Q, |x_1 \ldots x_n\rangle, \mathit{new}\ \ 0 \,] \to_1 [\, Q \otimes |0\rangle, |x_1 \ldots x_{n+1}\rangle, x_{n+1} \,]$$

$$[\, Q, |x_1 \ldots x_n\rangle, \mathit{new}\ \ 1 \,] \to_1 [\, Q \otimes |1\rangle, |x_1 \ldots x_{n+1}\rangle, x_{n+1} \,]$$

$$\frac{[\, Q, L, P \,] \to_p [\, Q', L, P' \,]}{[\, Q, L, \mathit{if}\, P \ \mathit{then}\ M \ \mathit{else}\ N \,] \to_p [\, Q', L, \mathit{if}\, P' \ \mathit{then}\ M \ \mathit{else}\ N \,]}$$

$$\frac{[\, Q, L, M \,] \to_p [\, Q', L, M' \,]}{[\, Q, L, \mathit{let}\ \langle x_1, x_2 \rangle = M \ \mathit{in}\ N \,] \to_p [\, Q', L, \mathit{let}\ \langle x_1, x_2 \rangle = M' \ \mathit{in}\ N \,]}$$

$$[\, Q, L, \mathit{let}\ \langle x_1, x_2 \rangle = \langle V_1, V_2 \rangle \ \mathit{in}\ N \,] \to_1 [\, Q, L, N[V_1/x_1, V_2/x_2] \,]$$

Table 6.1: Reductions rules of the quantum lambda calculus

6.3 Type System

We will now define a type system designed to eliminate all run-time errors arising from the reduction system of the previous section, such as the ones shown in Remark 6.2.16.

6.3.1 Types

In the lambda calculus we just defined, we need to be able to account for higher order, products (and unit), classical booleans and quantum booleans. Since we do not want to have specific term construct for dealing with duplication and non-duplication, the information has to be encoded into the type system. We use a type system inspired from linear logic. By default, a term of type A is assumed to be non-duplicable, and duplicable terms are given the type $!A$ instead.

Definition 6.3.1. Formally, we define the following type system for the lambda calculus of Section 6.1 as follows:

$$\mathit{qType}\ A, B \quad ::= \quad \mathit{bit} \mid \mathit{qbit} \mid !A \mid (A \multimap B) \mid \top \mid (A \otimes B).$$

The constant types *bit* and *qbit* stand respectively for the classical and the quantum booleans; the type $!A$ is for duplicable elements of type A; the type $A \multimap B$ for functions from A to B; the type $A \otimes B$ for pairs of elements of types A and B; finally $\top$ stands for the type of the term $*$.

We call the operator "!" the *exponential*.

Convention 6.3.2. We write $!^n A$ for $!!!\ldots!!A$, with n repetitions of !. We also use the notation $A^{\otimes n}$ for the n-fold tensor product $A \otimes \ldots \otimes A = (\ldots(A \otimes A)\ldots \otimes A)$.

The typing rules will ensure that any value of type $!A$ is duplicable. However, there is no harm in using it only once; thus, such a value should also have type A. We use a subtyping relation (Breazu-Tannen et al., 1989; Pierce, 2002) to capture this notion.

Definition 6.3.3. We define a subtyping relation $<:$ on types as follows, using the overall condition on n and m that $(m = 0) \vee (n \geqslant 1)$:

$$\frac{}{!^n bit <: !^m bit}\ (bit), \quad \frac{}{!^n qbit <: !^m qbit}\ (qbit), \quad \frac{}{!^n \top <: !^m \top}\ (\top),$$

$$\frac{A_1 <: B_1 \quad A_2 <: B_2}{!^n(A_1 \otimes A_2) <: !^m(B_1 \otimes B_2)}\ (\otimes), \quad \frac{A <: A' \quad B <: B'}{!^n(A' \multimap B) <: !^m(A \multimap B')}\ (\multimap).$$

Lemma 6.3.4. *$(qType, <:)$ is reflexive and transitive. If we define an equivalence relation $\doteqdot$ by $A \doteqdot B$ iff $A <: B$ and $B <: A$, $(qType/\doteqdot, <:)$ is a poset.*

Proof. The proof uses the fact that $(m = 0) \vee (n \geqslant 1)$ can be rewritten as $(n = 0) \Rightarrow (m = 0)$ (where $\Rightarrow$ means "implies"). Then the result follows from the reflexivity and the transitivity of the implication. □

Lemma 6.3.5. *If $A <: !B$, then $A = !A'$ for some type A'. Dually, if A is not of the form $!A'$ and if $A <: B$, then B is not of the form $!B'$.*

Proof. By inspection of the cases. □

6.3.2 Typing Rules

In their definition in Section 6.1, we implicitly considered the terms to be typed in a Church-style fashion. Considering terms this way is closer to the programmer approach: the types of the internal subterms have to be figured out by the compiler.

Definition 6.3.6. A *typing judgement* is the given of a set $\Delta = \{x_1 : A_1, \ldots, x_n : A_n\}$ of *typed variables*, of a term M and of a type B, written as $\Delta \rhd M : B$. We call the set Δ the *typing context*, or simply the *context*.

Given two contexts Δ_1 and Δ_2, we write (Δ_1, Δ_2) for the context consisting of the variables in Δ_1 and in Δ_2. When described in this form, it is assumed that $|\Delta_1| \cap |\Delta_2| = \emptyset$.

A typing derivation is called *valid* if it can be inferred from the rules of Table 6.2. We write $!\Delta$ for a context of the form $\{x_1 : !A_1, \ldots, x_n : !A_n\}$. We use the notation $|\Delta|$ to represent the set of (untyped) variables $\{x_1, \ldots, x_n\}$ contained in Δ. In the table, the symbol c spans the set of term constants $\{meas, new, U, 0, 1\}$. To each constant c we associate a term A_c, as follows:

$$A_0, A_1 = bit, \qquad A_{new} = bit \multimap qbit, \qquad A_U = qbit^{\otimes n} \multimap qbit^{\otimes n}, \qquad A_{meas} = qbit \multimap !bit.$$

Remark 6.3.7. The type $!A_c$ is understood as being the "most generic" one for c, as enforced by the typing rule (ax_2). For example, we defined A_{new} to be $bit \multimap qbit$. Since *new* can take any type B such that $!A_c <: B$, the term *new* can be typed with all the types in the poset:

$$\begin{tikzcd}[column sep=small] & !(!bit \multimap qbit) \arrow[dl, dash, "<:"'] \arrow[dr, dash, "<:"] & \\ !(bit \multimap qbit) \arrow[dr, dash, "<:"'] & & !bit \multimap qbit. \arrow[dl, dash, "<:"] \\ & bit \multimap qbit & \end{tikzcd}$$

This implies, as expected, that no created quantum bit can have the type $!qbit$.

Remark 6.3.8. Note that the type system enforces the requirement that variables holding quantum data cannot be freely duplicated; thus $\lambda x.\langle x, x\rangle$ is not a valid term of type $qbit \multimap qbit \otimes qbit$. On the other hand, we allow variables to be discarded freely.

Note also that due to rule (λ_2) the term $\lambda x.M$ is duplicable only if all the free variables of M (other that x) are duplicable. This follows the call-by-value approach: since $\lambda x.M$ is a value, if it is duplicated it will be duplicated as a syntactic string of symbols, thus duplicating every single free variable of it (other than x).

$$\frac{A <: B}{\Delta, x : A \rhd x : B}\ (ax_1) \qquad \frac{!A_c <: B}{\Delta \rhd c : B}\ (ax_2)$$

$$\frac{\Gamma_1, !\Delta \rhd P : bit \quad \Gamma_2, !\Delta \rhd M : A \quad \Gamma_2, !\Delta \rhd N : A}{\Gamma_1, \Gamma_2, !\Delta \rhd \mathit{if}\, P \text{ then } M \text{ else } N : A}\ (\mathit{if})$$

$$\frac{\Gamma_1, !\Delta \rhd M : A \multimap B \quad \Gamma_2, !\Delta \rhd N : A}{\Gamma_1, \Gamma_2, !\Delta \rhd MN : B}\ (\mathit{app})$$

$$\frac{x : A, \Delta \rhd M : B}{\Delta \rhd \lambda x.M : A \multimap B}\ (\lambda_1) \qquad \frac{\begin{array}{c}\text{If } FV(M) \cap |\Gamma| = \emptyset: \\ \Gamma, !\Delta, x : A \rhd M : B\end{array}}{\Gamma, !\Delta \rhd \lambda x.M : !^{n+1}(A \multimap B)}\ (\lambda_2)$$

$$\frac{!\Delta, \Gamma_1 \rhd M_1 : !^n A_1 \quad !\Delta, \Gamma_2 \rhd M_2 : !^n A_2}{!\Delta, \Gamma_1, \Gamma_2 \rhd \langle M_1, M_2 \rangle : !^n (A_1 \otimes A_2)}\ (\otimes.I) \qquad \frac{}{\Delta \rhd * : !^n \top}\ (\top)$$

$$\frac{!\Delta, \Gamma_1 \rhd M : !^n (A_1 \otimes A_2) \quad !\Delta, \Gamma_2,\ x_1{:}!^n A_1,\ x_2{:}!^n A_2 \rhd N : A}{!\Delta, \Gamma_1, \Gamma_2 \rhd \text{let } \langle x_1, x_2 \rangle = M \text{ in } N : A}\ (\otimes.E)$$

Table 6.2: Typing rules

Definition 6.3.9. A quantum closure $[\ Q, L, M\ ]$ is *well-typed* (or *valid*) of type A in the typing context Γ, written $\Gamma \vDash [\ Q, L, M\ ] : A$, if

- $|L| \cap |\Gamma| = \emptyset$,
- $FV(M) \setminus |\Gamma| \subseteq |L|$, and
- $\Gamma, x_1 : \mathit{qbit}, \ldots, x_k : \mathit{qbit} \rhd M : A$ is a valid typing judgement, where

$$\{x_1, \ldots, x_k\} = FV(M) \setminus |\Gamma|.$$

A well-typed quantum closure is *closed* if $|\Gamma| = \emptyset$, and a closed well-typed quantum closure is also called a *program*.

6.4 Properties of the Type System

We briefly state the properties of the type system that are similar to the properties of the simply-typed lambda calculus of Section 5.1.5. For a full development, we refer the reader to (Valiron, 2004a).

6.4.1 Safety Properties

Lemma 6.4.1. *If $\Gamma <: \Delta$ and $A <: B$, and if $\Delta \rhd M : A$ is a valid typing judgement, then $\Gamma \rhd B$.* □

Lemma 6.4.2. *If V is a value and if $\Delta \rhd V : A$ is a valid typing judgement, where for all $x \in FV(V)$ then x is duplicable (i.e. there exists a type B such that $\{x : !B\} \subseteq \Delta$), then $\Delta \rhd V : !A$ is valid.* □

Lemma 6.4.3 (Substitution)**.** *If V is a value, and if $!\Delta, \Gamma_1, x : A \rhd M : B$ and $!\Delta, \Gamma_2 \rhd V : A$ are two valid typing judgements, then $!\Delta, \Gamma_1, \Gamma_2 \rhd M[V/x] : B$ is valid.* □

Remark 6.4.4. Note that all the rules of affine intuitionistic logic (Troelstra, 1992) are derived rules of our type system *except* for a general promotion rule: Indeed, although the typing judgement $\rhd$ *meas* $0 : qbit$ is valid, the judgement $\rhd$ *meas* $0 : !qbit$ is not valid. However, as stated in Lemma 6.4.2, the promotion rule is true for values:

$$\frac{!\Delta \rhd V : A}{!\Delta \rhd V : !A} \ .$$

This point will be further developed in Chapter 8 when we will look for a categorical model of the language.

Theorem 6.4.5 (Subject reduction)**.** *Given a program* $[\ Q, L, M\ ]$ *of type* B *such that there exists* $[\ Q', L', M'\ ]$ *with* $[\ Q, L, M\ ] \rightsquigarrow^* [\ Q', L', M'\ ]$*, then* $[\ Q', L', M'\ ]$ *is a program of type* B*.* □

Theorem 6.4.6 (Progress)**.** *Let* $[\ Q, L, M\ ]$ *be a program of type* B*. Then* $[\ Q, L, M\ ]$ *is not in an error state in the sense of Definition 6.2.9. In particular, either it is a value, or there exists some state* $[\ Q', L', M'\ ]$ *such that* $[\ Q, L, M\ ] \rightarrow_p [\ Q', L', M'\ ]$*. Moreover, the total probability of all possible single-step reductions from* $[\ Q, L, M\ ]$ *is* 1*.* □

6.4.2 Type Inference Algorithm

The fact that a well-typed term behaves well is a good property. However, it still remains to provide a type for the term. For a large class of classical lambda calculus, such a task can be automated using a *type inference algorithm*.

In these cases, the algorithm is built on the fact that each term admits a *principal type*, that is, a most general type from which all the other types can be derived. For example, in simply typed lambda calculus, the term $M = \lambda fx.fx$ admits a principal type of the form $(X \Rightarrow Y) \Rightarrow (X \Rightarrow Y)$. Here, X and Y are type variables, and any valid type can be written in this form, for some value of X and Y.

The quantum lambda calculus we defined does not satisfy the principal type property. Indeed, consider again the term $M = \lambda fx.fx$. Naively, a principal type would be (1) the most general and (2) the smallest with respect to the subtyping relation. The most general type would have a structure of the form $(X \multimap Y) \multimap (X \multimap Y)$. Since $!((X \multimap Y) \multimap !(X \multimap Y))$ and $(X \multimap Y) \multimap !(X \multimap Y)$ are not valid types for M, the valid types for the term M, ordered by the subtyping relation (from left to right) are as follows:

$$\begin{tikzcd}
!((X\multimap Y)\multimap(X\multimap Y)) \arrow[r] \arrow[dr] & (X\multimap Y)\multimap(X\multimap Y) \arrow[dr] & \\
& !(!(X\multimap Y)\multimap(X\multimap Y)) \arrow[r] & !(X\multimap Y)\multimap(X\multimap Y). \\
!(!(X\multimap Y)\multimap!(X\multimap Y)) \arrow[r] \arrow[ur] & !(X\multimap Y)\multimap!(X\multimap Y) \arrow[ur] &
\end{tikzcd}$$

Note that there is no smallest element.

It is however possible to find a type inference algorithm for the quantum lambda calculus. Indeed, well-typed terms in the quantum lambda calculus are well-typed in a simply-typed version of the language. That is to say, a valid typing derivation in the quantum lambda calculus is an intuitionistic typing derivation *indexed* with "!". Following this idea it is possible to extract a type inference algorithm for the quantum lambda calculus. The algorithm proceeds by first finding the most general intuitionistic type for the given term, and then placing "!" whenever it is necessary. We refer the reader to (Valiron, 2004a) for the full discussion.

6.5 Examples of Algorithms

It turns out that some quantum algorithms can be efficiently understood as elements of higher-order types. We give in this section the implementation of two non-trivial quantum algorithm, the Deutsch algorithm and the teleportation procedure.

The latter one is of particular interest, since the usual formulation (see Section 3.4.1) is completely sequential.

6.5.1 The Deutsch Algorithm

The simplest example to interpret in a higher-order setting is the Deutsch algorithm. We refer the reader to Section 3.4.3 for its definition. It can be implemented in the quantum lambda calculus in the following way:

$$\begin{array}{l} \mathit{let}\ \mathbf{Deutsch}\ U_f\ = \\ \quad \mathit{let}\ \mathbf{tens}\ f\ g\ = \lambda\langle x,y\rangle.\langle fx, gy\rangle\ \mathit{in} \\ \quad \mathit{let}\ \langle x,y\rangle = (\mathbf{tens}\ H\ (\lambda x.x))(U_f\langle H(\mathit{new}\ 0), H(\mathit{new}\ 1)\rangle)\ \mathit{in} \\ \qquad \mathit{meas}\ x, \end{array}$$

using the notations of Convention 6.1.2. Note that U_f is a variable that stands for a function from a two-qubit state to a two-qubit state. And indeed the function **Deutsch** is a higher-order function:

$$\triangleright\ \mathbf{Deutsch}{:}!((\mathit{qbit} \otimes \mathit{qbit} \multimap \mathit{qbit} \otimes \mathit{qbit}) \multimap \mathit{bit})$$

is a well-typed typing judgment. Note that **Deutsch** is duplicable, and that U_f does not need to be duplicable, since it is used only once.

Also note that ideally, instead of U_f the algorithm takes the Boolean function f. Although one can write the type of the function $U_- : f \mapsto U_f$ as

$$(\mathit{bit} \multimap \mathit{bit}) \multimap (\mathit{qbit} \otimes \mathit{qbit} \multimap \mathit{qbit} \otimes \mathit{qbit}),$$

this function is not representable in the quantum lambda calculus.

6.5.2 The Teleportation Procedure

Consider the teleportation algorithm as it is described in Section 3.4.1. We can embed each quantum circuit part of the procedure in a function. There is a function $\mathbf{EPR} : !(\top \multimap (\mathit{qbit} \otimes \mathit{qbit}))$ that creates an entangled state, as in the step (1):

$$\mathbf{EPR} = \lambda x.\mathit{CNOT}\langle H(\mathit{new}\,0), \mathit{new}\,0\rangle.$$

There is a function $\mathbf{BellMeasure} : !(\mathit{qbit} \multimap (\mathit{qbit} \multimap \mathit{bit} \otimes \mathit{bit}))$ that takes two qubits, rotates and measures them, as in step (2):

$$\mathbf{BellMeasure} = \lambda q_2.\lambda q_1.(\mathit{let}\ \langle x,y\rangle = \mathit{CNOT}\langle q_1, q_2\rangle\ \mathit{in}\ \langle \mathit{meas}(Hx), \mathit{meas}\,y\rangle$$

We also can define a function $\mathbf{U} : !(\mathit{qbit} \multimap (\mathit{bit} \otimes \mathit{bit} \multimap \mathit{qbit}))$ that takes a qubit q and two bits x, y and returns $U_{xy}q$, as in step (3):

$$\begin{array}{rl} \mathbf{U} = \lambda q.\lambda\langle x,y\rangle.\mathit{if}\ x\ \mathit{then}\ (\mathit{if}\ y\ \mathit{then}\ U_{11}q\ \mathit{else}\ U_{10}q) \\ \qquad\qquad\qquad\qquad \mathit{else}\ (\mathit{if}\ y\ \mathit{then}\ U_{01}q\ \mathit{else}\ U_{00}q), \end{array}$$

where the operators U_{xy} are defined as in Section 3.4.1. The teleportation procedure can be seen as the creation of two non-duplicable functions f and g

$$\begin{aligned} f &: qbit \multimap bit \otimes bit, \\ g &: bit \otimes bit \multimap qbit, \end{aligned}$$

such that $(g \circ f)(x) = x$ for an arbitrary qubit x. We can construct such a pair of functions by the following code:

$$\begin{aligned} &let\ \langle x, y\rangle = \mathbf{EPR} * in \\ &\quad let\ f = \mathbf{BellMeasure}\ x\ in \\ &\quad let\ g = \mathbf{U}\ y \\ &\quad\ in\ \langle f, g\rangle. \end{aligned}$$

Note that, since f and g depend on the state of the qubits x and y, respectively, these functions cannot be duplicated, which is reflected in the fact that the types of f and g do not contain a top-level "!".

The Dense Coding Procedure

Note that the two functions f and g defined in the previous section also satisfy a dual property, namely that $(f \circ g)\langle x, y\rangle = \langle x, y\rangle$ for an arbitrary pair of classical bits. It turns out that this is precisely the higher-order description of the dense coding protocol of Section 3.4.2.

The pair of functions $\langle f, g\rangle$ creates an isomorphism between $qbit$ and $bit \otimes bit$. This is not in contradiction with the non-duplicability of quantum information since the functions are non-duplicable: one can use the function g only once to "retrieve" the quantum bit encoded into two classical booleans.

Tensor and Space-Like Separation

These two protocols have in common the fact that two parts of the algorithm take place in two different places. This space-like separation is modeled in the type system by the tensor product. It is interesting to note that an interpretation of these two complex protocols can be efficiently interpreted as one single higher-order term.

6.5.3 Type Derivation of the Teleportation Protocol

To illustrate the linear type system from Section 6.3.2, we give a complete derivation of the type of the quantum teleportation term. The notation $(L.x.y.z)$ means that Lemma.$x.y.z$ is used. Computing some subtypes:

1	α_2	$!^n\alpha <: \alpha$
2	α_2	$!^m\beta <: \beta$
3	$\multimap, 1, 2$	$!^k(\alpha \multimap !^m\beta) <: (!^n\alpha \multimap \beta)$
4	$(L.6.3.4)$	$A <: A$
5	$D, 4$	$!A <: A$

Computing the type of **EPR**:

6	$const, 3$	$\triangleright\ new : bit \multimap qbit$
7	$const, 5$	$\triangleright\ 0{:}bit$

8	$app, 6, 7$	$\rhd new\,0{:}qbit$
9	$const, 3$	$\rhd H{:}qbit \multimap qbit$
10	$app, 9, 8$	$\rhd H(new\,0){:}qbit$
11	$\otimes.I, 10, 9$	$\rhd \langle H(new\,0), new\,0\rangle : qbit \otimes qbit$
12	$const, 3$	$x{:}\top \rhd CNOT{:}(qbit \otimes qbit) \multimap (qbit \otimes qbit)$
13	$app, 12, 11$	$x{:}\top \rhd CNOT\langle H(new\,0), new\,0\rangle{:}qbit \otimes qbit$
14	$\lambda_2, 13$	$\rhd \lambda x.CNOT\langle H(new\,0), new\,0\rangle{:}!(\top \multimap (qbit \otimes qbit))$

Computing the type of **BellMeasure**:

15	$var, 1$	$y{:}qbit \rhd y{:}qbit$
16	$const, 3$	$\rhd meas : qbit \multimap bit$
17	$app, 16, 15$	$y{:}qbit \rhd meas\,y{:}bit$
18	$var, 1$	$x{:}qbit \rhd x{:}qbit$
19	$app, 9, 18$	$x{:}qbit \rhd Hx{:}qbit$
20	$app, 16, 19$	$x{:}qbit \rhd meas(Hx){:}bit$
21	$var, 1$	$q_1{:}qbit \rhd q_1{:}qbit$
22	$var, 1$	$q_2{:}qbit \rhd q_2{:}qbit$
23	$\otimes.I, 21, 22$	$q_2{:}qbit, q_1{:}qbit \rhd \langle q_1, q_2\rangle{:}qbit \otimes qbit$
24	$const, 3$	$\rhd CNOT{:}(qbit \otimes qbit) \multimap (qbit \otimes qbit)$
25	$app, 24, 23$	$q_2{:}qbit, q_1{:}qbit \rhd CNOT\langle q_1, q_2\rangle{:}qbit \otimes qbit$
26	$\otimes.I, 20, 17$	$x{:}qbit, y{:}qbit \rhd \langle meas(Hx), meas\,y\rangle{:}bit \otimes bit$
27	$\otimes.E, 25, 26$	$q_2{:}qbit, q_1{:}qbit \rhd let\ \langle x, y\rangle = CNOT\langle q_1, q_2\rangle$ $in\langle meas(Hx), meas\,y\rangle{:}bit \otimes bit$
28	$\lambda_1, 27$	$q_2{:}qbit \rhd \lambda q_1.(let\ \langle x, y\rangle = CNOT\langle q_1, q_2\rangle$ $in\ \langle meas(Hx), meas\,y\rangle){:}qbit \multimap bit \otimes bit$
29	$\lambda_2, 28$	$\rhd \lambda q_2.\lambda q_1.(let\ \langle x, y\rangle = CNOT\langle q_1, q_2\rangle$ $in\ \langle meas(Hx), meas\,y\rangle){:}!(qbit \multimap (qbit \multimap bit \otimes bit))$

Computing the type of **U**:

30	$var, 1$	$q{:}qbit \rhd q{:}qbit$
31	$const, 3$	$\rhd U_{ij}{:}qbit \multimap qbit$
32	$app, 30, 31$	$q{:}qbit \rhd U_{ij}q{:}qbit$
33	$var, 1$	$y{:}bit \rhd y{:}!bit$
34	$var, 1$	$x{:}bit \rhd x{:}!bit$
35	$if, 33, 32, 32$	$q{:}qbit, y{:}bit \rhd \text{if } y \text{ then } U_{i1}q \text{ else } U_{i0}q{:}qbit$
36	$if, 34, 35, 35$	$q{:}qbit, x{:}bit, y{:}bit \rhd if\ x\ then\ (if\ y\ then\ U_{11}q\ else\ U_{10}q)$ $else\ (if\ y\ then\ U_{01}q\ else\ U_{00}q){:}qbit$
37	$\lambda_1, 36$	$q{:}qbit \rhd \lambda\langle x, y\rangle.if\ x\ then\ (if\ y\ then\ U_{11}q\ else\ U_{10}q)$ $else\ (if\ y\ then\ U_{01}q\ else\ U_{00}q){:}bit \otimes bit \multimap qbit$
38	$\lambda_2, 37$	$\rhd \lambda q.\lambda\langle x, y\rangle.if\ x\ then\ (if\ y\ then\ U_{11}q\ else\ U_{10}q)$

$$else\ (if\ y\ then\ U_{01}q\ else\ U_{00}q):!(qbit \multimap (bit \otimes bit \multimap qbit))$$

Finally, computing the type of the pair $\langle f, g\rangle$:

39	$\top$	$\rhd *:\top$
40	$(L.6.4.2), 14, 5$	$\rhd \mathbf{EPR}:\top \multimap (qbit \otimes qbit)$
41	$app, 40, 39$	$\rhd \mathbf{EPR}*:qbit \otimes qbit$
42	$(L.6.4.2), 29, 5$	$\rhd \mathbf{BellMeasure}:qbit \multimap (qbit \multimap bit \otimes bit)$
43	$var, 1$	$x:qbit \rhd x:qbit$
44	$app, 42, 43$	$x:qbit \rhd \mathbf{BellMeasure}\ x:qbit \multimap bit \otimes bit$
45	$var, 1$	$y:qbit \rhd y:qbit$
46	$(L.6.4.2), 38, 5$	$\rhd \mathbf{U}:qbit \multimap (bit \otimes bit \multimap qbit)$
47	$app, 46, 45$	$y:qbit \rhd \mathbf{U}\ y:bit \otimes bit \multimap qbit$
48	$var, 1$	$f:qbit \multimap bit \otimes bit \rhd f:qbit \multimap bit \otimes bit$
49	$var, 1$	$g:bit \otimes bit \multimap qbit \rhd g:bit \otimes bit \multimap qbit$
50	$\otimes, 48, 49$	$g:bit \otimes bit \multimap qbit, f:qbit \multimap bit \otimes bit \rhd \langle f, g\rangle:$ $(qbit \multimap bit \otimes bit) \otimes (bit \otimes bit \multimap qbit)$
51	$let, 47, 50$	$f:qbit \multimap bit \otimes bit, y:qbit \rhd letg = \mathbf{U}\ y\ in\ \langle f, g\rangle:$ $(qbit \multimap bit \otimes bit) \otimes (bit \otimes bit \multimap qbit)$
52	$let, 44, 51$	$x:qbit, y:qbit \rhd letf = \mathbf{BellMeasure}\ x\ in\ letg = \mathbf{U}\ y$ $in\ \langle f, g\rangle):(qbit \multimap bit \otimes bit) \otimes (bit \otimes bit \multimap qbit)$
53	$let, 41, 52$	$\rhd let\langle x, y\rangle = \mathbf{EPR}*\ in\ letf = \mathbf{BellMeasure}\ x$ $in\ letg = \mathbf{U}\ y\ in\ \langle f, g\rangle)):$ $(qbit \multimap bit \otimes bit) \otimes (bit \otimes bit \multimap qbit)$

6.5.4 Reduction of the Teleportation Term

As an illustration of the reduction rules of the quantum lambda calculus we show the detailed reduction of the term from the teleportation example from Section 6.5.2. The reduction of the teleportation term corresponds to the equality $g \circ f = id$. We use the following abbreviations:

$$\begin{aligned}
M_{p,p'} &:= let\ f = \mathbf{BellMeasure}\ p\ in\ let\ g = \mathbf{U}\ p'\ in\ g(f\ p_0) \\
B_{p_1} &:= \lambda q_1.(let\ \langle p, p'\rangle = CNOT\langle q_1, p_1\rangle\ in\ \langle meas(Hp), meas\, p'\rangle) \\
U_{p_2} &:= \lambda\langle x, y\rangle.\ (if\ x\ then\ (if\ y\ then\ U_{11}p_2\ else\ U_{10}p_2) \\
&\qquad\qquad else\ (if\ y\ then\ U_{01}p_2\ else\ U_{00}p_2))
\end{aligned}$$

The reduction of the term is then as follows:

$$\begin{aligned}
&\left[\alpha|0\rangle + \beta|1\rangle, \begin{array}{l} let\ \langle p, p'\rangle = \mathbf{EPR}* \\ \quad let\ f = \mathbf{BellMeasure}\ p \\ \quad in\ let\ g = \mathbf{U}\ p' \\ \quad in\ g(f\ p_0) \end{array}\right] \\
&\to_1 [\alpha|0\rangle + \beta|1\rangle, let\ \langle p, p'\rangle = CNOT\langle H(new\, 0), new\, 0\rangle\ in\ M_{p,p'}] \\
&\to_1 [(\alpha|0\rangle + \beta|1\rangle) \otimes |0\rangle, let\ \langle p, p'\rangle = CNOT\langle Hp_1, new\, 0\rangle\ in\ M_{p,p'}]
\end{aligned}$$

$$\rightarrow_1 \left[(\alpha|0\rangle + \beta|1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \mathit{let}\ \langle p, p'\rangle = \mathit{CNOT}\langle p_1, \mathit{new}\ 0\rangle\ \mathit{in}\ M_{p,p'}\right]$$

$$\rightarrow_1 \left[(\alpha|0\rangle + \beta|1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |0\rangle, \mathit{let}\ \langle p, p'\rangle = \mathit{CNOT}\langle p_1, p_2\rangle\ \mathit{in}\ M_{p,p'}\right]$$

$$\rightarrow_1 \left[(\alpha|0\rangle + \beta|1\rangle) \otimes \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle), \mathit{let}\ \langle p, p'\rangle = \langle p_1, p_2\rangle\ \mathit{in}\ M_{p,p'}\right]$$

$$\rightarrow_1 \left[(\alpha|0\rangle + \beta|1\rangle) \otimes \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle), \begin{array}{l} \mathit{let}\ f = \mathbf{BellMeasure}\ p_1 \\ \quad \mathit{in}\ \mathit{let}\ g = \mathbf{U}\ p_2 \\ \quad \mathit{in}\ g(f\ p_0) \end{array}\right]$$

$$\rightarrow_1{}^* \left[(\alpha|0\rangle + \beta|1\rangle) \otimes \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle), U_{p_2}(B_{p_1} p_0)\right]$$

$$\rightarrow_1 \left[(\alpha|0\rangle + \beta|1\rangle) \otimes \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle), U_{p_2}\left(\begin{array}{c} \mathit{let}\ \langle p, p'\rangle = \mathit{CNOT}\langle p_0, p_1\rangle \\ \mathit{in}\ \langle \mathit{meas}(Hp), \mathit{meas}\ p'\rangle \end{array}\right)\right]$$

$$\rightarrow_1 \left[\frac{1}{\sqrt{2}}\left(\begin{array}{c} \alpha|000\rangle + \alpha|011\rangle \\ +\beta|110\rangle + \beta|101\rangle \end{array}\right), U_{p_2}\left(\begin{array}{c} \mathit{let}\ \langle p, p'\rangle = \langle p_0, p_1\rangle \\ \mathit{in}\ \langle \mathit{meas}(Hp), \mathit{meas}\ p'\rangle \end{array}\right)\right]$$

$$\rightarrow_1 \left[\frac{1}{\sqrt{2}}\left(\begin{array}{c} \alpha|000\rangle + \alpha|011\rangle \\ +\beta|110\rangle + \beta|101\rangle \end{array}\right), U_{p_2}\langle \mathit{meas}(Hp_0), \mathit{meas}\ p_1\rangle\right]$$

$$\rightarrow_1 \left[\frac{1}{2}\left(\begin{array}{c} \alpha|000\rangle + \alpha|011\rangle \\ +\alpha|100\rangle + \alpha|111\rangle \\ +\beta|010\rangle + \beta|001\rangle \\ -\beta|110\rangle - \beta|101\rangle \end{array}\right), U_{p_2}\langle \mathit{meas}\ p_0, \mathit{meas}\ p_1\rangle\right]$$

$$\left\{\begin{array}{l} \xrightarrow{\frac{1}{2}} \left[\frac{1}{\sqrt{2}}\left(\begin{array}{c} \alpha|000\rangle + \alpha|011\rangle \\ +\beta|010\rangle + \beta|001\rangle \end{array}\right), U_{p_2}\langle 0, \mathit{meas}\ p_1\rangle\right] \\ \xrightarrow{\frac{1}{2}} \left[\frac{1}{\sqrt{2}}\left(\begin{array}{c} \alpha|100\rangle + \alpha|111\rangle \\ -\beta|110\rangle - \beta|101\rangle \end{array}\right), U_{p_2}\langle 1, \mathit{meas}\ p_1\rangle\right] \end{array}\right.$$

$$\left\{\begin{array}{lll} \xrightarrow{1/2} \left[\big(\alpha|000\rangle + \beta|001\rangle\big), U_{p_2}\langle 0, 0\rangle\right] & \rightarrow_1{}^* & \left[\big(\alpha|000\rangle + \beta|001\rangle\big), U_{00} p_2\right] \\ \xrightarrow{1/2} \left[\big(\alpha|011\rangle + \beta|010\rangle\big), U_{p_2}\langle 0, 1\rangle\right] & \rightarrow_1{}^* & \left[\big(\alpha|011\rangle + \beta|010\rangle\big), U_{01} p_2\right] \\ \xrightarrow{1/2} \left[\big(\alpha|100\rangle - \beta|101\rangle\big), U_{p_2}\langle 1, 0\rangle\right] & \rightarrow_1{}^* & \left[\big(\alpha|100\rangle - \beta|101\rangle\big), U_{10} p_2\right] \\ \xrightarrow{1/2} \left[\big(\alpha|111\rangle - \beta|110\rangle\big), U_{p_2}\langle 1, 1\rangle\right] & \rightarrow_1{}^* & \left[\big(\alpha|111\rangle - \beta|110\rangle\big), U_{11} p_2\right] \end{array}\right.$$

$$\left\{\begin{array}{lll} \rightarrow_1 & [\ (\alpha|000\rangle + \beta|001\rangle), p_2\ ] = & [\ |00\rangle \otimes (\alpha|0\rangle + \beta|1\rangle), p_2\ ] \\ \rightarrow_1 & [\ (\alpha|010\rangle + \beta|011\rangle), p_2\ ] = & [\ |01\rangle \otimes (\alpha|0\rangle + \beta|1\rangle), p_2\ ] \\ \rightarrow_1 & [\ (\alpha|100\rangle + \beta|101\rangle), p_2\ ] = & [\ |10\rangle \otimes (\alpha|0\rangle + \beta|1\rangle), p_2\ ] \\ \rightarrow_1 & [\ (\alpha|110\rangle + \beta|111\rangle), p_2\ ] = & [\ |11\rangle \otimes (\alpha|0\rangle + \beta|1\rangle), p_2\ ] \end{array}\right.$$

6.5.5 Reduction of the Superdense Coding Term

As another example of the reduction rules, we give the reduction of the superdense coding example from Section 6.5.2. This reduction shows the equality $f \circ g = id$. Of the four possible cases, we only give one case, namely $(f \circ g)\langle 0, 1\rangle = \langle 0, 1\rangle$; the remaining cases are similar. We use the same abbreviations as above.

$$\left[\begin{array}{l} \mathit{let}\ \langle p, p'\rangle = \mathbf{EPR}* \\ \quad \mathit{let}\ f = \mathbf{BellMeasure}\ p \\ |\rangle, \quad \mathit{in}\ \mathit{let}\ g = \mathbf{U}\ p' \\ \quad \mathit{in}\ f(g\langle 0, 1\rangle) \end{array}\right]$$

$$
\begin{aligned}
&\to_1{}^* \left[\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle), \begin{array}{l} let\ f = \mathbf{BellMeasure}\ p_0 \\ \quad in\ let\ g = \mathbf{U}\ p_1 \\ \quad in\ f(g\langle 0,1\rangle) \end{array}\right] \\
&\to_1{}^* \left[\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle), B_{p_0}(U_{p_1}\langle 0,1\rangle)\right] \\
&\to_1{}^* \left[\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle), B_{p_0}(U_{01}p_1)\right] \\
&\to_1 \left[\frac{1}{\sqrt{2}}(|01\rangle + |10\rangle), B_{p_0}p_1\right] \\
&\to_1 \left[\frac{1}{\sqrt{2}}(|01\rangle + |10\rangle), let\ \langle p, p'\rangle = CNOT\langle p_1, p_0\rangle\ in\ \langle meas(Hp), meas\, p'\rangle\right] \\
&\to_1 \left[\frac{1}{\sqrt{2}}(|11\rangle + |10\rangle), let\ \langle p, p'\rangle = \langle p_1, p_0\rangle\ in\ \langle meas(Hp), meas\, p'\rangle\right] \\
&\to_1 \left[\frac{1}{\sqrt{2}}(|11\rangle + |10\rangle), \langle meas(Hp_1), meas\, p_0\rangle\right] \\
&\to_1 [|10\rangle, \langle meas\, p_1, meas\, p_0\rangle] \\
&\to_1{}^* [|10\rangle, \langle 0,1\rangle]
\end{aligned}
$$

6.6 Towards a Denotational Semantics

In this chapter, we described a lambda calculus for dealing with quantum information. Given a typing judgement

$$x : A \rhd M : B,$$

we can now give an operational description of the behavior of the term M.

Another approach for understanding the language is to understand this expression as the syntactic representation of a function from some space A to some space B. A *denotational semantics* is the description of a theory of mathematically structured spaces, where types are interpreted as spaces and typing judgements as structure-preserving maps between these spaces.

The difficulty in the case of the quantum lambda calculus is two-fold:

1. We need to account for probabilistic and quantum effects,
2. while keeping track of duplicable and non-duplicable data.

The remainder of this thesis is devoted to the study of these problems.

Chapter 7

The Linear Fragment

In this chapter, we restrict the quantum lambda calculus to the purely linear case, namely the fragment of the language where each value, classical and quantum, must be used exactly once. The linear quantum lambda calculus differs from its nonlinear cousin in that it is less sensitive to the evaluation order of terms. We give a denotational semantics for this language in a category of completely positive maps, and we show that it is fully abstract with respect to the operational semantics.

The plan of the chapter is as follows. First we briefly describe the language and the type system. Then we develop for it an operational semantics similar to the one developed in Section 6.2.4, and we define a notion of axiomatic equivalence of terms. Finally we build a denotational semantics for the language, and we show the soundness of the axiomatic equivalence with respect to the operational semantics, as well as the full abstraction of the denotational semantics. An extended abstract of the results presented in this chapter has been published in Selinger and Valiron (2006b).

7.1 A Linear Lambda Calculus for Quantum Computation

In Chapter 6, we defined a lambda calculus for quantum computation with classical control. This language was defined in a Curry-style manner. Although this makes sense from the point of view of the programmer, this typing paradigm is harder to tackle for the semantics. Indeed, we shall define the interpretation of a valid typing judgement by induction on its typing derivation. A one-to-one correspondence between typing judgements and typing derivation is therefore required.

We begin by re-adapting the definitions and results from Chapter 6 to the requirements of this study.

Definition 7.1.1. The linear quantum lambda calculus has the following (explicitly typed) terms:

$$\begin{aligned}
&\textit{Types } A, B && ::= \textit{bit} \mid \textit{qbit} \mid A \multimap B \mid \top \mid A \otimes B,\\
&\textit{Term } M, N, P && ::= x^A \mid MN \mid \lambda x^A.M \mid \langle M, N\rangle \mid \textit{ let } \langle x^A, y^B\rangle = M \textit{ in } N \mid\\
& && \quad \textit{if } M \textit{ then } N \textit{ else } P \mid 0 \mid 1 \mid \textit{ new } \mid \textit{ meas } \mid U \mid\\
& && \quad \textit{let } * = M \textit{ in } N \mid * \mid \Omega^A_{x_n^{A_1},\ldots,x_n^{A_n}}.
\end{aligned}$$

Ω is a non-terminating term and *let* $* = M$ *in* N is used to match 0-tuples. The remaining terms are as in Definition 6.1.1. We identify terms up to α-equivalence. The set of free variables of a term M is written $FV(M)$. In the following, we often write c for an arbitrary constant of the language, i.e., 0, 1, *meas*, *new*, U, or $*$.

$$\frac{}{x : A \rhd x^A : A}\ (ax_1) \qquad \frac{}{\rhd c : A_c}\ (ax_2) \qquad \frac{}{\rhd * : \top}\ (\top.I)$$

$$\frac{\Gamma_1 \rhd P : bit \quad \Gamma_2 \rhd M : A \quad \Gamma_2 \rhd N : A}{\Gamma_1, \Gamma_2 \rhd \mathit{if}\, P\ \mathit{then}\ M\ \mathit{else}\ N : A}\ (\mathit{if})$$

$$\frac{\Gamma_1 \rhd M : A \multimap B \quad \Gamma_2 \rhd N : A}{\Gamma_1, \Gamma_2 \rhd MN : B}\ (\mathit{app}) \qquad \frac{\Delta, x : A \rhd M : B}{\Delta \rhd \lambda x^A.M : A \multimap B}\ (\lambda)$$

$$\frac{\Gamma_1 \rhd M_1 : A_1 \quad \Gamma_2 \rhd M_2 : A_2}{\Gamma_1, \Gamma_2 \rhd \langle M_1, M_2 \rangle : A_1 \otimes A_2}\ (\otimes.I) \qquad \frac{}{\rhd * : \top}\ (\top.I) \qquad \frac{}{\Delta \rhd \Omega^A_\Delta : A}\ (\Omega)$$

$$\frac{\Gamma_1 \rhd M : A_1 \otimes A_2 \quad \Gamma_2,\ x_1{:}A_1,\ x_2{:}A_2 \rhd N : A}{\Gamma_1, \Gamma_2 \rhd \text{let } \langle x_1, x_2 \rangle = M \text{ in } N : A}\ (\otimes.E)$$

$$\frac{\Gamma_1 \rhd M : \top \quad \Gamma_2 \rhd N : A}{\Gamma_1, \Gamma_2 \rhd \mathit{let} * = M\ \mathit{in}\ N : A}\ (\top.E)$$

Table 7.1: Typing rules for the linear quantum lambda calculus

We use the same shortcuts as in Conventions 6.1.2 and 6.3.2. Also, as we did in the previous chapter, to each term constant c, we associate a fixed type A_c, namely

$$\mathit{meas} : \mathit{qbit} \multimap \mathit{bit}, \qquad 0, 1 : \mathit{bit}, \qquad \mathit{new} : \mathit{bit} \multimap \mathit{qbit}, \qquad U : \mathit{qbit}^{\otimes n} \multimap \mathit{qbit}^{\otimes n}.$$

The only difference is the suppression of the type constructor "!", and therefore of subtyping.

As before, a *typing context* Δ is a set of typed variables, written as $\Delta = \{x_1 : A_1, \ldots, x_n : A_n\}$. A *typing judgement* is an expression $\Delta \rhd M : A$, where Δ is a list of distinct typed variables called a *typing context*, M is a term, and A is a type. We say that a typing judgement is *valid* if it follows from the typing rules given in Table 7.1. In this table, if $\Delta = \{x_1 : A_1, \ldots, x_n : A_n\}$ we write Ω^A_Δ in place of $\Omega^A_{x_n^{A_1}, \ldots, x_n^{A_n}}$.

Remark 7.1.2. Note that, unlike the language of the previous chapter, there is no type constructor $!A$. Also note that we added the term construct $\mathit{let} * = M\ \mathit{in}\ N$. Operationally, if we understand a "command" as a function that outputs the unit term $*$, this term is used for being able to "use" a command in a program. Semantically, it is simply the destructor associated to $\top$.

Definition 7.1.3. Given a term M, we define the set of its free variables using the same definition as in Definition 6.1.3, modulo the indexing on the term. We extend the definition of FV to the terms Ω and $\mathit{let} * = M\ \mathit{in}\ N$ as follows:

$$FV(\Omega^A_{x_1^{A_1}, \ldots, x_n^{A_n}}) = \{x_1, \ldots, x_n\}, \qquad FV(\mathit{let} * = M\ \mathit{in}\ N) = FV(M) \cup FV(N).$$

Lemma 7.1.4. *Suppose that $\Delta \rhd M : A$ is a valid typing judgement. Then the set of free variables $FV(M)$ is equal to $|\Delta|$.*

Proof. Consider a typing derivation π of $\Delta \rhd M : A$. We prove the result by induction on π.

(ax_1). The derivation is of the form $x : A \rhd x : A$, and $FV(x) = \{x\}$.

(ax_2). The derivation is of the form $\rhd c : A$, and $FV(c) = \emptyset$.

$(\otimes.I)$. The derivation is of the form $\rhd * : \top$, and $FV(*) = \emptyset$.

(if). The derivation is of the form $\Delta, \Gamma \rhd if\, P\ then\ N_1\ else\ N_2 : A$, where

$$\Delta \rhd P : bit, \quad \Delta \rhd N_1 : A, \quad \Delta \rhd N_2 : A.$$

By induction hypothesis, $FV(P) = |\Delta|$, $FV(N_1) = |\Gamma|$ and $FV(N_2) = |\Gamma|$. Therefore,

$$FV(if\, P\ then\ N_1\ else\ N_2) = FV(P) \cup FV(N_1) \cup FV(N_2) = |\Delta| \cup |\Gamma| = |\Delta, \Gamma|.$$

(Ω). The derivation is of the form $\Delta \rhd \Omega^A_\Delta : A$, and $FV(\Omega^A_\Delta) = |\Delta|$.

(λ). The derivation is of the form $\Delta \rhd \lambda x.N : A \multimap B$, where $\Delta, x : A \rhd N : B$. By induction, $FV(N) = |\Delta, x : A|$, which is $|\Delta| \cup \{x\}$. Thus $FV(\lambda x.N) = |\Delta|$.

(app). The derivation is of the form $\Gamma_1, \Gamma_2 \rhd N_1 N_2 : A$, where $\Gamma_1 \rhd N_1 : A$ and $\Delta_2 \rhd N_2 : A$. By induction hypothesis, $FV(N_1) = |\Gamma_1|$ and $FV(N_2) = |\Gamma_2|$. Therefore, $FV(N_1 N_2) = FV(N_1) \cup FV(N_2) = |\Gamma_1, \Gamma_2|$.

$(\otimes.I)$. The derivation is of the form $\Gamma_1, \Gamma_2 \rhd \langle N_1, N_2 \rangle : B \otimes c$, where $\Gamma_1 \rhd N_1 : C$ and $\Delta_2 \rhd N_2 : D$. By induction hypothesis, $FV(N_1) = |\Gamma_1|$ and $FV(N_2) = |\Gamma_2|$. Therefore, $FV\langle N_1, N_2 \rangle = FV(N_1) \cup FV(N_2) = |\Gamma_1, \Gamma_2|$.

$(\top.E)$. The derivation is of the form $\Gamma_1, \Gamma_2 \rhd let * = N_1\ in\ N_2 : A$, where $\Gamma_1 \rhd N_1 : A$ and $\Delta_2 \rhd N_2 : A$. By induction hypothesis, $FV(N_1) = |\Gamma_1|$ and $FV(N_2) = |\Gamma_2|$. Therefore, $FV(let * = N_1\ in\ N_2) = FV(N_1) \cup FV(N_2) = |\Gamma_1, \Gamma_2|$.

$(\otimes.E)$. The derivation is of the form $\Gamma_1, \Gamma_2 \rhd let\ \langle x^B, y^C \rangle = N_1\ in\ N_2 : A$, where $\Gamma_1 \rhd N_1 : A$ and $\Delta_2, x : B, y : C \rhd N_2 : A$. By induction hypothesis, $FV(N_1) = |\Gamma_1|$ and $FV(N_2) = |\Gamma_2, x : B, y : C|$. Therefore, $FV(let\ \langle x^B, y^C \rangle = N_1\ in\ N_2) = FV(N_1) \cup (FV(N_2) \setminus \{x, y\}) = |\Gamma_1, \Gamma_2|$. □

Lemma 7.1.5. *Consider a valid typing judgement $\Delta \rhd M : A$. Then the type A is completely determined by Δ and M, and there exists only one derivation for the typing judgement.*

Proof. The proof is done by induction on the size of the term M, using Lemma 7.1.4 and noticing that for every term construct only one possible typing rule does apply. □

Definition 7.1.6. Let $\Gamma \rhd N : A$ be a valid typing judgement, let M be a term such that $FV(M) \cap |\Gamma| = \emptyset$, and let x be a term variable. We define the *substitution of x in M by N*, written $M[N/x]$, as we did it in Definition 6.1.4 (modulo the type indexation). We extend this definition to the two term constructs Ω^C_Δ and $let * = M\ in\ N$ as follows. If $\Delta = (\Delta', x : B)$ for some type B, we define $\Omega^C_{\Delta', x:B}[N/x]$ as the term $\Omega^C_{\Delta', \Gamma}$. If $x \notin |\Delta|$, the substitution $\Omega^C_\Delta[N/x]$ is simply equal to Ω^C_Δ. Finally, the rule for the last construct is $(let * = M_1\ in\ M_2)[N/x] = (let * = M_1[N/x]\ in\ M_2[N/x])$. Note that we only make use of the typing context of N in the rule concerning Ω.

Lemma 7.1.7. *In the context of Definition 7.1.6, suppose that the variable x does not belong to $FV(M)$. Then $M[N/x] = M$.*

Proof. The proof is done by induction on M.

$M \equiv y$. Then $y \neq x$, and for all N, $y[N/x] = y$.

$M \equiv c$ *and* $M \equiv *$. In both cases, $M[N/x] = N$ by definition.

$M \equiv \Omega^C_\Delta$. By definition, $FV(M) = |\Delta|$. Since x does not belong to $FV(M)$, it does neither belong to $|\Delta|$. Therefore, from the definition of the substitution, $\Omega^C_\Delta[N/x] = \Omega^C_\Delta$.

$M \equiv \lambda y.P$. If $y = x$, then $x \notin FV(\lambda y.P)$ and by definition $(\lambda y.P)[N/x] = \lambda y.P$. If $y \neq x$, for x not being a free variable of M one needs x not to be a free variable of P. In which case induction hypothesis applies: $P[N/x] = P$. Therefore, $M[N/x] = N$.

$M \equiv P_1 P_2$. Since $FV(P_1 P_2) = FV(P_1) \cup FV(P_2)$, if $x \notin FV(M)$, x is neither in $FV(P_1)$ nor in $FV(P_2)$. Since $(P_1 P_2)[N/x] = (P_1[N/x])(P_2[N/x])$, by induction hypothesis this means that $(P_1 P_2)[N/x] = P_1 P_2$.

$M \equiv \mathit{if}\, P_1\ \mathit{then}\ P_2\ \mathit{else}\ P_3$. Since $FV(\mathit{if}\, P_1\ \mathit{then}\ P_2\ \mathit{else}\ P_3) = FV(P_1) \cup FV(P_2) \cup FV(P_3)$, if $x \notin FV(M)$, x is neither in $FV(P_1)$ nor in $FV(P_2)$ nor in $FV(P_3)$. Since $(\mathit{if}\, P_1\ \mathit{then}\, P_2\ \mathit{else}\, P_3)[N/x] = \mathit{if}\, P_1[N/x]\ \mathit{then}\ P_2[N/x]\ \mathit{else}\ P_3[N/x])$, by induction hypothesis this means that $(\mathit{if}\, P_1\ \mathit{then}\ P_2\ \mathit{else}\ P_3)[N/x] = (\mathit{if}\, P_1\ \mathit{then}\ P_2\ \mathit{else}\ P_3)$.

$M \equiv (\mathit{let} * = P_1\ \mathit{in}\ P_2)$. Since $FV(\mathit{let} * = P_1\ \mathit{in}\ P_2) = FV(P_1) \cup FV(P_2)$, if $x \notin FV(M)$, x is neither in $FV(P_1)$ nor in $FV(P_2)$. Since

$$(\mathit{let} * = P_1\ \mathit{in}\ P_2)[N/x] = (\mathit{let} * = P_1[N/x]\ \mathit{in}\ P_2[N/x]),$$

by induction hypothesis this means that $(\mathit{let} * = P_1\ \mathit{in}\ P_2)[N/x] = (\mathit{let} * = P_1\ \mathit{in}\ P_2)$.

$M \equiv \langle P_1, P_2 \rangle$. Since $FV(\langle P_1, P_2 \rangle) = FV(P_1) \cup FV(P_2)$, if $x \notin FV(M)$, x is neither in $FV(P_1)$ nor in $FV(P_2)$. Since $(\langle P_1, P_2 \rangle)[N/x] = \langle P_1[N/x], P_2[N/x] \rangle)$, by induction hypothesis this means that $\langle P_1, P_2 \rangle[N/x] = \langle P_1, P_2 \rangle$.

$M \equiv (\mathit{let}\ \langle y^C, z^D \rangle = P_1\ \mathit{in}\ P_2)$. Since

$$FV(\mathit{let}\ \langle y^C, z^D \rangle = P_1\ \mathit{in}\ P_2) = FV(P_1) \cup (FV(P_2) \setminus \{y, z\}),$$

if $x \notin FV(M)$, x is neither in $FV(P_1)$ nor in $FV(P_2) \setminus \{y, z\}$. By induction hypothesis $P_1[N/x] = P_1$. There are two cases:

If $x = y$ or $x = z$. Then $(\mathit{let}\ \langle y^C, z^D \rangle = P_1\ \mathit{in}\ P_2)[N/x] = (\mathit{let}\ \langle y^C, z^D \rangle = P_1[N/x]\ \mathit{in}\ P_2)$, and we have the result.

If $x \neq y$ and $x \neq z$. Then since x is not in $FV(P_2) \setminus \{y, z\}$, it is not in $FV(P_2)$. By induction hypothesis $P_2[N/x] = P_2$. Since $(\mathit{let}\ \langle y^C, z^D \rangle = P_1\ \mathit{in}\ P_2)[N/x] = (\mathit{let}\ \langle y^C, z^D \rangle = P_1[N/x]\ \mathit{in}\ P_2[N/x])$, we have the result. □

Lemma 7.1.8 (Substitution)**.** *Suppose that $\Delta, x : A \rhd M : B$ and $\Gamma \rhd N : A$ are two valid typing judgements such that $|\Delta| \cap |\Gamma| = \emptyset$. Then $\Delta, \Gamma \rhd M[N/x] : B$ is a valid typing judgement.*

Proof. We prove the result by induction on the size of M.

Case $M \equiv y$. The typing rule for obtaining $\Delta, x : A \rhd M : B$ is (ax_1). That is $A = B$, $x = y$ and Δ is empty. Thus $\Delta, \Gamma \rhd M[N/x] : B$ becomes $\Gamma \rhd N : A$, which is valid by hypothesis.

Case $M \equiv c$. The typing rule for obtaining $\Delta, x : A \rhd M : B$ is (ax_2). Thus $\Delta, x : A \rhd c : B$ is valid if $(\Delta, x : A)$ is empty, and it is not. This case is therefore not to be considered.

Case $M \equiv \Omega^B_{\Delta'}$. The typing rule for obtaining $\Delta, x : A \rhd M : B$ is (Ω). Thus $\Delta' = (\Delta, x : A)$. Since $\Omega^B_{\Delta, x:A}[N/x] = \Omega^B_{\Delta,\Gamma}$, the result is obtained by applying back the typing rule (Ω).

Case $M \equiv \lambda y^C.P$. The typing rule for obtaining $\Delta, x : A \rhd M : B$ is (λ). Thus $B = C \multimap D$ and $\Delta, x : A, y : C \rhd P : D$. A corollary is that $y \neq x$, thus that $M[N/x] = \lambda y^C.(P[N/x])$.

By induction hypothesis, $\Delta, \Gamma, y : C \rhd P[N/x] : D$ is valid. Applying rule (λ), we get $\Delta, \Gamma \rhd \lambda y^C.(P[N/x]) : C \multimap D$, which is the desired result.

Case $M \equiv \mathit{if}\, P\ \mathit{then}\ N_1\ \mathit{else}\ N_2$. The only possible typing rule for obtaining $\Delta, x : A \rhd M : B$ is (if). Then $(\Delta, x : A)$ splits into (Δ_1, Δ_2), where

$$\Delta_1 \rhd P : \mathit{bit}, \quad \Delta_2 \rhd N_1 : B, \quad \Delta_2 \rhd N_2 : B.$$

Note that the contexts Δ_1 and Δ_2 do not intersect. There are two possible cases:

$x \in |\Delta_1|$. In this case $\Delta_0 = (\Delta_1', x : A)$. We are in the setting of the lemma: by induction hypothesis, $\Delta_1', \Gamma \rhd P[V/x] : \mathit{bit}$.

From Lemma 7.1.4, $FV(N_1) = |\Delta_2|$ and $FV(N_2) = |\Delta_2|$. Therefore x is neither in $FV(N_1)$ nor in $FV(N_2)$. From Lemma 7.1.7, $M[N/x] = (\mathit{if}\, P[V/x]\ \mathit{then}\ N_1\ \mathit{else}\ N_2)$.

Thus, applying (if), we get that $\Delta, \Gamma \rhd \mathit{if}\, P[V/x]\ \mathit{then}\ N_1\ \mathit{else}\ N_2 : B$ is valid.

$x \in |\Delta_2|$. In this case $\Delta_1 = (\Delta_2', x : A)$. We are in the setting of the lemma: by induction hypothesis, $\Delta_2', \Gamma \rhd N_1[V/x] : B$ and $\Delta_2', \Gamma \rhd N_2[V/x] : B$

From Lemma 7.1.4, $FV(P) = |\Delta_1|$. Therefore x is not in $FV(P)$. From Lemma 7.1.7, $M[N/x] = (\mathit{if}\, P\ \mathit{then}\ N_1[N/x]\ \mathit{else}\ N_2[N/x])$.

Thus, applying (if), we get that $\Delta, \Gamma \rhd \mathit{if}\, P\ \mathit{then}\ N_1[N/x]\ \mathit{else}\ N_2[N/x] : B$ is valid.

Cases $M \equiv N_1 N_2$, $\langle N_1, N_2 \rangle$ *and* $(\mathit{let} * = N_1\ \mathit{in}\ N_2)$. In each case the term M can be written in the form $f(N_1, N_2)$, and there is only one possible typing rule for obtaining $\Delta, x : A \rhd M : B$, namely respectively (app), $(\otimes.I)$ and $(\top.E)$. Then $(\Delta, x : A)$ splits into (Δ_1, Δ_2), where $\Delta_1 \rhd N_1 : C_1$ and $\Delta_2 \rhd N_2 : C_2$, for some types C_1 and C_2 depending on the rule used:

If the rule used is...	$f(N_1, N_2)$ is...	C_1 is...	C_2 is...	and B is...
(app)	$N_1 N_2$	$C \multimap D$	C	D
$(\otimes.I)$	$\langle N_1, N_2 \rangle$	C	D	$C \otimes D$
$(\top.E)$	$\mathit{let} * = N_1\ \mathit{in}\ N_2$	$\top$	C	C

Note that for $i = 1, 2$, the contexts Δ_i and Γ do not intersect. There are two possible cases: $x \in |\Delta_1|$, or $x \in |\Delta_2|$. Since they are symmetric by exchanging the role of the indices 1 and 2, we only prove one of them.

Suppose that $x \in |\Delta_1|$ In this case $\Delta_1 = (\Delta_1', x : A)$. We are in the setting of the lemma: by induction hypothesis, $\Delta_1', \Gamma \rhd N_1[V/x] : C_1$. From Lemma 7.1.4, $FV(N_2) = |\Delta_2|$. Therefore x is not in $FV(N_2)$. From Lemma 7.1.7, $M[N/x] = f(N_1[N/x], N_2)$.

Applying back the typing rule used, the judgement $\Delta, \Gamma \rhd f(N_1[N/x], N_2) : B$ is valid.

Case $M \equiv (\mathit{let}\ \langle y^C, z^D \rangle = N_1\ \mathit{in}\ N_2)$. This case is similar to the previous ones: There is only one possible typing rule for obtaining $\Delta, x : A \rhd M : B$, namely $(\otimes.E)$. Then $(\Delta, x : A)$ splits into (Δ_1, Δ_2), where $\Delta_1 \rhd N_1 : C \otimes D$ and $\Delta_2, y : C, y : D \rhd N_2 : B$.

Note that for $i = 1, 2$, the contexts Δ_i and Γ do not intersect. There are two possible cases: $x \in |\Delta_1|$, or $x \in |\Delta_2|$.

$x \in |\Delta_1|$. In this case $\Delta_1 = (\Delta_1', x : A)$. We are in the setting of the lemma: by induction hypothesis, $\Delta_1', \Gamma \rhd N_1[V/x] : C \otimes D$. From Lemma 7.1.4, $FV(N_2) = |\Delta_2, y : C, z : D|$. Therefore x is not in $FV(N_2)$. From Lemma 7.1.7, $M[N/x] = (\mathit{let}\ \langle y^C, z^D \rangle = N_1[N/x]\, \mathit{in}\, N_2)$.

Applying back the typing rule $(\otimes.E)$, the judgement $\Delta, \Gamma \rhd \mathit{let}\ \langle y^C, z^D \rangle = N_1[N/x]\, \mathit{in}\, N_2 : B$ is valid.

$x \in |\Delta_2|$. Since $(\Delta_2, y : C, y : D)$ is a typing context, x is different from y and z: We have then $\Delta_2 = (\Delta_2', x : A)$. We are in the setting of the lemma, and by induction hypothesis, $\Delta_2', \Gamma, y : C, z : D \rhd N_2[V/x] : B$.

From Lemma 7.1.4, $FV(N_1) = |\Delta_1|$. Therefore x is not in $FV(N_1)$. From Lemma 7.1.7, $M[N/x] = (\mathit{let}\ \langle y^C, z^D\rangle = N_1\ \mathit{in}\ N_2[N/x])$.

Applying back the typing rule $(\otimes.E)$, the judgement $\Delta, \Gamma \rhd \mathit{let}\ \langle y^C, z^D\rangle = N_1\ \mathit{in}\ N_2[N/x] : B$ is valid. □

Remark 7.1.9. Note that, in the non-linear case, the Substitution Lemma only holds when $N = V$ is a value. However, in the linear calculus considered here, it holds for general N.

7.2 Operational Semantics

We develop in this section an operational semantics similar to the one given in Section 6.2, and prove some of its properties.

7.2.1 Small Step Semantics

We adopt the definitions of quantum closures and programs from Section 6.2.1 to the linear case.

Definition 7.2.1. A *quantum closure* is a triple $[\ Q, L, M\ ]$ where Q is a normalized vector in $\otimes_{i=1}^{n}\mathbb{C}^2$, for some $n \geqslant 0$, L is a bijective function from a set $|L|$ of term variables to $\{0, \ldots, n-1\}$, and M is a term. As before, Q is called a *quantum array*, and L is called a *linking function*. Note that the only difference from Definition 6.2.1 lies in the fact that L is a *bijective* function. We use the notations of Definition 6.2.1, and consider quantum closure up to α-equivalence. Finally, we use the notion of *well-typed quantum closure* and *program* of Definition 6.3.9.

Definition 7.2.2. We define the call-by-value reduction strategy on program by structural induction. For this purpose we need the notion of a value. As in Definition 6.2.13, we define a *value term* to be of the form

$$V, W \ ::= \ x \mid c \mid * \mid \lambda x.M \mid \langle V, W\rangle.$$

A *value program* is a program of the form $[Q, L, V]$, where V is a value term. We set the rules to be the "classical" ones found in Table 7.2, plus the following "quantum" rules. In the first two rules, decompose Q into

$$Q = \sum_j \alpha_j |\psi_j^0\rangle \otimes |0\rangle \otimes |\tilde{\psi}_j^0\rangle + \sum_j \beta_j |\psi_j^1\rangle \otimes |1\rangle \otimes |\tilde{\psi}_j^1\rangle,$$

where $|\psi_j^0\rangle, |\psi_j^1\rangle \in \mathbb{C}^{2^{i-1}}$ and $|\tilde{\psi}_j^0\rangle, |\tilde{\psi}_j^1\rangle \in \mathbb{C}^{2^{n-i}}$ are basis vectors. Then, if

$$\alpha = \sum_j |\alpha_j|^2, \quad \beta = \sum_j |\beta_j|^2,$$

$$[\ Q, |x_1 \cdots x_n\rangle, \mathit{meas}\ \ x_i\ ] \rightarrow_{\alpha}^{m_0} [\ \textstyle\sum_j |\psi_j^0\rangle \otimes |\tilde{\psi}_j^0\rangle, |x_1 \cdots x_{i-1}x_{i+1} \cdots x_n\rangle, 0\ ],$$
$$[\ Q, |x_1 \cdots x_n\rangle, \mathit{meas}\ \ x_i\ ] \rightarrow_{\beta}^{m_1} [\ \textstyle\sum_j |\psi_j^1\rangle \otimes |\tilde{\psi}_j^1\rangle, |x_1 \cdots x_{i-1}x_{i+1} \cdots x_n\rangle, 1\ ].$$

In the second set of rules, if w is a fresh term variable not yet in use:

$$[\ Q, |x_1 \cdots x_n\rangle, \mathit{new}\ \ 0\ ] \rightarrow_1^{n_0} [\ Q \otimes |0\rangle, |x_1 \cdots x_n w\rangle, w\ ],$$

$$[\, Q, L, \Omega \,] \to_1^{\omega} [\, Q, L, \Omega \,] \qquad [\, Q, L, (\lambda x^B.M)V \,] \to_1^{\beta} [\, Q, L, M[V/x] \,]$$

$$[\, Q, L, \mathit{let}\ \langle x^B, y^C \rangle = \langle V, W \rangle\ \mathit{in}\ N \,] \to_1^{\otimes} [\, Q, L, N[V/x, W/y] \,]$$

$$[\, Q, L, \mathit{let}\ * = *\ \mathit{in}\ N \,] \to_1^{\top} [\, Q, L, N \,]$$

$$[\, Q, L, \mathit{if}\, 0\ \mathit{then}\ M\ \mathit{else}\ N \,] \to_1^{\mathit{if}_0} [\, Q, L, N \,]$$

$$[\, Q, L, \mathit{if}\, 1\ \mathit{then}\ M\ \mathit{else}\ N \,] \to_1^{\mathit{if}_1} [\, Q, L, M \,]$$

$$\frac{[\, Q, L, N \,] \to_p^{\kappa} [\, Q', L', N' \,]}{[\, Q, L, MN \,] \to_p^{\kappa} [\, Q', L', MN' \,]} \qquad \frac{[\, Q, L, M \,] \to_p^{\kappa} [\, Q', L', M' \,]}{[\, Q, L, MV \,] \to_p^{\kappa} [\, Q', L', M'V \,]}$$

$$\frac{[\, Q, L, M_1 \,] \to_p^{\kappa} [\, Q', L', M_1' \,]}{[\, Q, L, \langle M_1, M_2 \rangle \,] \to_p^{\kappa} [\, Q', L', \langle M_1', M_2 \rangle \,]} \qquad \frac{[\, Q, L, M_2 \,] \to_p^{\kappa} [\, Q', L', M_2' \,]}{[\, Q, L, \langle V_1, M_2 \rangle \,] \to_p^{\kappa} [\, Q', L', \langle V_1, M_2' \rangle \,]}$$

$$\frac{[\, Q, L, P \,] \to_p^{\kappa} [\, Q', L', P' \,]}{[\, Q, L, \mathit{if}\, P\ \mathit{then}\ M\ \mathit{else}\ N \,] \to_p^{\kappa} [\, Q', L', \mathit{if}\, P'\ \mathit{then}\ M\ \mathit{else}\ N \,]}$$

$$\frac{[\, Q, L, M \,] \to_p^{\kappa} [\, Q', L', M' \,]}{[\, Q, L, \mathit{let}\ \langle x_1, x_2 \rangle = M\ \mathit{in}\ N \,] \to_p^{\kappa} [\, Q', L', \mathit{let}\ \langle x_1, x_2 \rangle = M'\ \mathit{in}\ N \,]}$$

$$\frac{[\, Q, L, M \,] \to_p^{\kappa} [\, Q', L', M' \,]}{[\, Q, L, \mathit{let}\ * = M\ \mathit{in}\ N \,] \to_p^{\kappa} [\, Q', L', \mathit{let}\ * = M'\ \mathit{in}\ N \,]}$$

Table 7.2: Reduction rules for the linear quantum lambda calculus

$$[\, Q, |x_1 \cdots x_n\rangle, \mathit{new}\ \ 1 \,] \to_1^{n_1} [\, Q \otimes |1\rangle, |x_1 \cdots x_n w\rangle, w \,].$$

Finally, if Q' is the result of applying U to the quantum bits $L(x_1), \ldots, L(x_n)$ in Q:

$$[\, Q, L, U\langle x_1, \ldots, x_n \rangle \,] \to_1^{U} [\, Q', L, \langle x_1, \ldots, x_n \rangle \,].$$

Note that each reduction rule is named, and has a probability associated to it.

Remark 7.2.3. Note that since we want a linear language, we modified the measurement rule from the rule in Definition 6.2.13 by deleting the quantum bit measured from the quantum array. The two other differences from Table 6.1 are the (classical) rules concerning Ω and *let* $* = M$ *in* N. The former was defined to be a non-terminating term: it remains unchanged along the reduction. The latter matches 0-tuples. The rule follows the fact that the only 0-tuple is $*$.

Definition 7.2.4. A program $[\, Q, L, M \,]$ reducing with the ω-rule is called a *fixed point*.

Lemma 7.2.5. *Suppose that P is a value program. Then it does not reduce. Now suppose that P is a fixed point. Then the only possible programs P' such that $P \to_p^{\kappa} P'$ are fixed points.*

Proof. For the former, proof by case distinction. For the latter, proof by induction on the reduction $P \to^{\kappa} P'$. □

Lemma 7.2.6. *Suppose that V is a value such that $x_1 : \mathit{qbit}, \ldots, x_n : \mathit{qbit} \rhd V : A \multimap B$. Then V is either of the form $\lambda x^A.N$ or it is one of the three term constants new, meas and U. If V is of type $A \otimes B$, then it is of the form $\langle V_1, V_2 \rangle$. If it is of type bit, it is either 0 or 1. Finally, if it is of type qbit, it is a term variable.*

Proof. The proof for the type $A \multimap B$ is done by case distinction: V cannot be a variable x_i since *qbit* is not of the form $A \multimap B$, it cannot be $*$ and it cannot be of the form $\langle V_1, V_2 \rangle$. Therefore it is

either a lambda-abstraction or a term constant. The only term constants with the right types are the one offered in the lemma. The proof for the type $A \otimes B$ is done similarly. The proofs for the last two cases are done by case distinction. □

7.2.2 Safety Properties

Theorem 7.2.7 (Subject reduction)**.** *If P is a program of type A such that $P \to_\rho P'$, then P' is also a program of type A.*

Proof. We prove the result by induction on the derivation of the relation $P \to_\rho P'$. If $P \equiv [\, Q, L, M \,]$, we suppose that $FV(M) = \{x_1, \ldots, x_n\} \subseteq |L|$.

Case $(\to_1^\beta)$. The only possible typing rule yielding $x_1 : qbit, \ldots, x_n : qbit \rhd (\lambda x^B.M)V : A$ is (app). It means that there exists Δ and Γ such that $(\Delta, \Gamma) = (x_1 : qbit, \ldots, x_n : qbit)$ and such that $\Delta \rhd (\lambda x^B.M) : B \multimap A$ and $\Gamma \rhd P : B$ are valid. The former can only be originated from typing rule (λ). It means that $\Delta, x : B \rhd M : A$ is valid. From Lemma 7.1.8, $\Delta, \Gamma \rhd M[V/x] : B$ is valid. Therefore, $[\, Q, |x_1 \ldots x_n\rangle, M[V/x] \,]$ is a program of type A.

Case $(\to_1^\otimes)$. The only possible typing rule for getting $x_1 : qbit, \ldots, x_n : qbit \rhd let\ \langle x^B, y^C\rangle = \langle V, W\rangle\ in\ M : A$ is $(\otimes.E)$. It means that there exists Δ and Γ such that $(\Delta, \Gamma) = (x_1 : qbit, \ldots, x_n : qbit)$ and such that $\Gamma \rhd \langle V, W\rangle : B \otimes C$ and $\Delta, x : B, y : C \rhd M : A$ are valid. The former can only be originated from typing rule $(\otimes)$. Thus $\Gamma = (\Gamma_1, \Gamma_2)$ with $\Gamma_1 \rhd V : B$ and $\Gamma_2 \rhd W : C$.

From Lemma 7.1.8, $\Delta, x : B, \Gamma_2 \rhd M[W/y] : A$ is valid. From Lemma 7.1.8, $\Delta, \Gamma_1, \Gamma_2 \rhd M[W/y][V/x] : A$ is valid. Therefore the typing judgement $\Delta, \Gamma \rhd M[V/x, W/y] : A$ is valid and the quantum context $[\, Q, |x_1 \ldots x_n\rangle, M[V/x, W/y] \,]$ is a program of type A.

Case $(\to_1^\top)$. The only possible typing rule for getting $x_1 : qbit, \ldots, x_n : qbit \rhd let\ * = *\ in\ N : A$ is $(\top.E)$. It means that there exists Δ and Γ such that $(\Delta, \Gamma) = (x_1 : qbit, \ldots, x_n : qbit)$ and such that $\Delta \rhd * : \top$ and $\Gamma \rhd M : A$ are valid.

The typing judgement $\Delta \rhd * : \top$ being originated from $(\top.E)$, one has $|\Delta| = \emptyset$. Therefore $\Gamma = (x_1 : qbit, \ldots, x_n : qbit)$

This makes $[\, Q, |x_1 \ldots x_n\rangle, N \,]$ a program of type A.

Cases $(\to_1^{if_0})$ *and* $(\to_1^{if_1})$. The only possible typing rule yielding $x_1 : qbit, \ldots, x_n : qbit \rhd if 0\ then\ M\, else N : A$ is (if). It means that there exists Δ and Γ such that $(\Delta, \Gamma) = (x_1 : qbit, \ldots, x_n : qbit)$ and such that $\Delta \rhd 1 : bit$, $\Gamma \rhd M : A$ and $\Gamma \rhd N : A$ are valid.

The typing judgement $\Delta \rhd 1 : bit$ being originated from (ax_2), one has $|\Delta| = \emptyset$. Therefore $\Gamma = (x_1 : qbit, \ldots, x_n : qbit)$

Therefore, $[\, Q, |x_1 \ldots x_n\rangle, M \,]$ and $[\, Q, |x_1 \ldots x_n\rangle, N \,]$ are programs of type A.

Case $(\to_1^\omega)$. In this case, the term P and the term P' are the same: therefore, if P is a program of type A, so is P'.

Cases $(\to_\alpha^{m_0})$ *and* $(\to_\beta^{m_1})$. In these cases, $FV(meas\, x_i) = \{x_i\}$. The typing judgement $x_i : qbit \rhd meas\, x_i : A$ being valid, since $meas : qbit \multimap bit$, by typing rule (app) we have $A = bit$. Therefore $\rhd 0 : A$ and $\rhd 1 : A$ are valid: in both cases, since one removes x_i from $|L|$, the program P' is of type A.

Cases $(\to_1^{n_0})$ *and* $(\to_1^{n_1})$. In these case, $FV(new\, 0) = FV(new\, 1) = \emptyset$. The judgements $\rhd new\, 0 : A$ and $\rhd new\, 1 : A$ are valid, thus since $A_{new} = bit \multimap qbit$, the type A is equal to $qbit$. Thus P' is a program of type A.

Case $(\rightarrow_1^U)$. Since $A_U = qbit^{\otimes n} \multimap qbit^{\otimes n}$ for some n, one has

$$x_1 : qbit, \ldots, x_n : qbit \rhd U\langle x_1, \ldots, x_n\rangle : qbit^{\otimes n}.$$

Thus $A = qbit^{\otimes n}$. But we also have $x_1 : qbit, \ldots, x_n : qbit \rhd \langle x_1, \ldots, x_n\rangle : qbit^{\otimes n}$. Therefore, P' is a program of type A.

The cases corresponding to the induction steps are directly coming from Lemma 7.1.4. □

Theorem 7.2.8 (Progress)**.** *If P is a program of type A, then either P is a value program, or P reduces using the reduction system of Definition 7.2.2.*

Proof. Consider a valid program $[\, Q, |x_1 \ldots x_n\rangle, M \,]$ of type A. Then $x_1 : qbit, \ldots, x_n : qbit \rhd M : A$ is valid. Let π be a typing derivation of this typing judgement. We prove the result by induction on π.

Cases (ax_1), (ax_2), $(\top.I)$, (λ). In all of these cases, the term M is a value. Thus P is a value program.

Case (if). The term M is of the form $if\, M'$ *then* N_1 *else* N_2. There exist contexts Δ and Γ such that $(\Delta, \Gamma) = (x_1 : qbit, \ldots, x_n : qbit)$ and such that $\Delta \rhd M' : bit$ and $\Gamma \rhd N_1, N_2 : A$. From Lemma 7.1.4, $|\Delta| = FV(M')$. Thus the quantum context $[\, Q, |x_1 \ldots x_n\rangle, M' \,]$ is a program of type bit. By induction hypothesis, either it is a value program or it reduces to some closure $[\, Q', L', M'' \,]$.

The only value of type bit being 0 and 1, if it is a value, then M' is respectively the term constant 0 and 1, and P reduces using respectively rules $(\rightarrow_{if_0})$ and $(\rightarrow_{if_1})$. If it is not a value, then by congruence P reduces to $[\, Q', L', if\, M'' N_1 N_2 \,]$.

Thus, in any case the program P reduces.

Case (app). The term M is of the form $N_1 N_2$. There exist contexts Δ and Γ such that $(\Delta, \Gamma) = (x_1 : qbit, \ldots, x_n : qbit)$ and such that $\Delta \rhd N_1 : B \multimap A$ and $\Gamma \rhd N_2 : B$ for some type B.

If N_2 is not a value: then the program $[\, Q, |x_1 \ldots x_n\rangle, N_2 \,]$ reduces by induction hypothesis, and by congruence we are done.

If N_2 is a value, and if N_1 is not a value, then this time $[\, Q, |x_1 \ldots x_n\rangle, N_1 \,]$ reduces by induction hypothesis, and by congruence we are done.

If N_2 and N_1 are values, then from Lemma 7.2.6, N_1 can only be one of the following:

$N_1 \equiv \lambda x^B.N$. Then P reduces using rule $(\rightarrow_\beta)$

$N_1 \equiv new$. Then $B = bit$. The only values of type bit being 0 and 1, the program P reduces using either rule $(\rightarrow_{n_0})$ or $(\rightarrow_{n_1})$.

$N_1 \equiv meas$. Then $B = qbit$. The only values of type $qbit$ being variables, N_2 must be a term variable. Since $|\Gamma| \subseteq \{x_1 \ldots x_n\}$, $N_2 = x_j$ for some j. Then P reduces using either rule $(\rightarrow_{m_0})$ or $(\rightarrow_{m_1})$.

$N_1 \equiv U^k$. Then $B = qbit^{\otimes k}$. The only values of type $qbit^{\otimes k}$ being of the form $\langle y_1, \ldots y_k\rangle$, and since $|\Gamma| \subseteq \{x_1 \ldots x_n\}$, $N_2 = \langle x_{\sigma(1)}, \ldots, x_{\sigma(k)}\rangle$ for some injective map $\sigma : \{1, \ldots, k\} \rightarrow \{1, \ldots, n\}$. Then P reduces using rule $(\rightarrow_U)$.

Case $(\otimes.I)$. The term M is of the form $\langle N_1, N_2\rangle$ and $A = B_1 \otimes B_2$. There exist contexts Δ and Γ such that $(\Delta, \Gamma) = (x_1 : qbit, \ldots, x_n : qbit)$ and such that $\Delta \rhd N_1 : B_1$ and $\Gamma \rhd N_2 : B_2$.

If N_1 is not a value: then the program $[\, Q, |x_1 \ldots x_n\rangle, N_1 \,]$ reduces by induction hypothesis, and by congruence we are done.

If N_1 is a value but not N_2, then this time $[\ Q, |x_1 \dots x_n\rangle, N_2\ ]$ reduces by induction hypothesis, and by congruence we are done.

If they are both value, so is M and by definition P is a value program.

Case $(\otimes.E)$. The term M is of the form *let* $\langle x^B, y^C\rangle = N_1$ *in* N_2. There exist contexts Δ and Γ such that $(\Delta, \Gamma) = (x_1 : qbit, \dots, x_n : qbit)$ and such that $\Delta \rhd N_1 : B \otimes C$ and $\Gamma, x : B, y : C \rhd N_2 : A$ are valid.

If N_1 is not a value: then since $|\Delta| \subseteq \{x_1 \dots x_n\}$, the program $[\ Q, |x_1 \dots x_n\rangle, N_1\ ]$ reduces by induction hypothesis and P reduces using congruence.

If N_1 is a value, then it is of the form $\langle V, W\rangle$, and then P reduces using rule $(\rightarrow_\otimes)$.

Case $(\top.E)$. The term M is of the form *let* $* = N_1$ *in* N_2. There exist contexts Δ and Γ such that $(\Delta, \Gamma) = (x_1 : qbit, \dots, x_n : qbit)$ and such that $\Delta \rhd N_1 : \top$ and $\Gamma \rhd N_2 : A$ are valid.

If N_1 is not a value: then since $|\Delta| \subseteq \{x_1 \dots x_n\}$, the program $[\ Q, |x_1 \dots x_n\rangle, N_1\ ]$ reduces by induction hypothesis and P reduces using congruence.

If N_1 is a value, then it is equal to $*$, and then P reduces using rule $(\rightarrow_\top)$.

Case (Ω). The term M is of the form $\Omega^A_{x_1,\dots x_n}$. Therefore P reduces using the reduction rule $(\rightarrow_\omega)$. □

7.2.3 Normalization

The language being linear, the reduction has a strong relation on the length of the terms.

Definition 7.2.9. Let $l(M)$ be the length of a term M, defined recursively as an integer greater or equal to 1 as follows:

$$\begin{aligned}
l(x^A) = l(c) = l(*) = l(\Omega) &= 1,\\
l(\lambda x^A.M) &= 1 + l(M),\\
l(\mathit{let}\ \langle x^B, y^C\rangle = M\ \mathit{in}\ N) &= 1 + l(M) + l(N),\\
l(\mathit{let}\ * = M\ \mathit{in}\ N) &= 1 + l(M) + l(N),\\
l(\mathit{if}\, P\ \mathit{then}\ M\ \mathit{else}\ N) &= 1 + l(P) + \max(l(M), l(N)),\\
l(MN) &= 1 + l(M) + l(N),\\
l(\langle M, N\rangle) &= 1 + l(M) + l(N).
\end{aligned}$$

Lemma 7.2.10. *Suppose that* $\Delta, x : A \rhd M : B$ *and* $\Gamma \rhd N : A$ *are valid. Then* $l(M[N/x]) < l(M) + l(N)$.

Proof. The proof is done by structural induction on M.

$M \equiv x^C$. The computation goes like this: $l(x[N/x]) = l(N) < l(x) + l(N)$.

$M \equiv c$, $*$, Ω, *and* y^C *(with* $y \neq x$*).* In all of these cases, $M[N/x] = M$. Therefore $l(M[N/x]) = l(M)$, which by definition is strictly smaller than $l(M) + l(N)$.

$M \equiv \lambda y^C.M'$. If $y = x$ then $M[N/x] = M$. As in the previous case, we obtain the result using the definition.

If $y \neq x$, then $M[N/x] = \lambda y^A.(M'[N/x])$. By induction hypothesis, $l(M'[N/x]) < l(M') + l(N)$. Therefore, since $l(M) = 1 + l(M')$, $l(M[N/x]) = 1 + l(M'[N/x]) < 1 + l(M') + l(N) = l(M) + l(N)$.

$M \equiv \mathit{if}\, P\ \mathit{then}\ N_1\ \mathit{else}\ N_2$. We have $l(M) = 1 + l(P) + \max(l(N_1), l(N_2))$ and

$$M[N/x] = \mathit{if}\, P[N/x]\ \mathit{then}\ N_1[N/x]\ \mathit{else}\ N_2[N/x].$$

The valid judgement $\Delta, x : A \rhd M : B$ can only come from the typing rule (if). That is, $(\Delta, x : A)$ splits into (Γ_1, Γ_2), where $\Gamma_1 \rhd P : \mathit{bit}$ and $\Gamma_2 \rhd N_1, N_2 : A$.

If $x \in |\Gamma_1|$, then $x \notin |\Gamma_2|$, and from Lemma 7.1.4 x is neither in $FV(N_1)$ nor in $FV(N_2)$, therefore from Lemma 7.1.7 $N_2[N/x] = N_2$ and $N_1[N/x] = N_1$. The length of $M[N/x]$ is then $1 + l(P[N/x]) + \max(l(N_1), l(N_2))$. By induction hypothesis, $l(P[N/x]) < l(P) + l(N)$, thus $l(M[N/x]) < l(M) + l(N)$.

If $x \in |\Gamma_2|$, then $x \notin |\Gamma_1|$, and from Lemma 7.1.4 $x \notin FV(P)$, therefore from Lemma 7.1.7 $P[N/x] = N_1$. The length of $M[N/x]$ is then

$$1 + l(P) + \max(l(N_2[N/x]), l(N_1[N/x])).$$

By induction hypothesis, $l(N_1[N/x]) < l(N_1) + l(N)$ and $l(N_2[N/x]) < l(N_2) + l(N)$, thus as expected $l(M[N/x]) < l(M) + l(N)$.

In the four remaining cases. The term M is respectively of the form

$$\begin{array}{ll} \mathit{let}\ \langle y^C, z^D \rangle = N_1\ \mathit{in}\ N_2), & N_1 N_2, \\ \mathit{let}\ * = N_1\ \mathit{in}\ N_2, & \langle N_1, N_2 \rangle. \end{array}$$

In all of these cases, we have $l(M) = l(N_1) + l(N_2) + 1$. In the first cases, by α-equivalence one can assume that $x \neq x, y$. Therefore, $M[N/x]$ becomes respectively:

$$\begin{array}{ll} \mathit{let}\ \langle y^C, z^D \rangle = N_1[N/x]\ \mathit{in}\ N_2[N/x]), & (N_1[N/x])(N_2[N/x]), \\ \mathit{let}\ * = N_1[N/x]\ \mathit{in}\ N_2[N/x], & \langle N_1[N/x], N_2[N/x] \rangle. \end{array}$$

The valid judgement $\Delta, x : A \rhd M : B$ can only come from respectively the typing rule $(\otimes.E)$, (app), $(\top.E)$, $(\otimes.I)$. That is, $(\Delta, x : A)$ splits into (Γ_1, Γ_2), where

$$\Gamma_1 \rhd N_1 : C_1, \qquad \Gamma_2, \Lambda \rhd N_2 : C_2,$$

for some types C_1, C_2 and some context Λ, with $x \notin |\Lambda|$.

If $x \in |\Gamma_1|$, then $x \notin |\Gamma_2|$, and from Lemma 7.1.4 $x \notin FV(N_2)$, therefore from Lemma 7.1.7 $N_2[N/x] = N_2$. The length of $M[N/x]$ is then $1 + l(N_1[N/x]) + l(N_2)$. By induction hypothesis, $l(N_1[N/x]) < l(N_1) + l(N)$, thus $l(M[N/x]) < l(M) + l(N)$.

If $x \in |\Gamma_2|$, then $x \notin |\Gamma_1|$, and from Lemma 7.1.4 $x \notin FV(N_1)$, therefore from Lemma 7.1.7 $N_1[N/x] = N_1$. The length of $M[N/x]$ is then $1 + l(N_2[N/x]) + l(N_1)$. By induction hypothesis, $l(N_2[N/x]) < l(N_2) + l(N)$, thus as expected $l(M[N/x]) < l(M) + l(N)$. □

Lemma 7.2.11. *If $[\, Q, L, M \,] \to_\kappa [\, Q', L', M' \,]$, then either $x \neq \omega$ and $l(M') < l(M)$ or $\kappa = \omega$ and $l(M') = l(M)$. In the latter case, $[\, Q, L, M \,] = [\, Q', L', M' \,]$.*

Proof. The proof is done by induction on the derivation of the reduction $[\, Q, L, M \,] \to_\kappa [\, Q', L', M' \,]$.

Case $(\to_\beta)$. The term M and the term M' are respectively of the form $(\lambda x^A.N)N'$ and $N'[N/x]$. Since P is a program, M is well-typed: By Lemma 7.2.10, since $l(M) = l(N) + l(N') + 2$, the length $l(M')$ verifies $l(M') < l(N') + l(N) < l(M)$.

Case ($\rightarrow_{\otimes}$). The term M and the term M' are respectively of the form ($\mathit{let}\ \langle x^A, y^B\rangle = \langle V, W\rangle\ \mathit{in}\ N$) and $N[W/y, V/x]$. Since P is a program, M is well-typed: By Lemma 7.2.10, since $l(M) = l(N) + l(V) + l(W) + 2$, the length of M' verifies

$$l(M') = l(N[W/y][V/x]) < l(N[W/y]) + l(V) < l(N) + l(V) + l(W) < l(M).$$

Case ($\rightarrow_{\top}$). The term M and the term M' are respectively of the form ($\mathit{let}\ * = *\ \mathit{in}\ N$) and N. Since $l(M) = 2 + l(N)$, as requested $l(M') < l(M)$.

Cases ($\rightarrow_{if_0}$) *and* ($\rightarrow_{if_1}$). The term M and the term M' are respectively of the form $\mathit{if}\,0\,\mathit{then}\,N\,\mathit{else}\,N'$ and N', or $\mathit{if}\,1\,\mathit{then}\,N\,\mathit{else}\,N'$ and N. In both cases $l(M) = 2 + \max(l(N), l(N'))$, this strictly larger than both $l(N)$ and $l(N')$: the inequality $l(M') < l(M)$ is verified.

Case ($\rightarrow_{\omega}$). In this case, $\kappa = \omega$ and $[\ Q, L, M\ ] = [\ Q', L', M'\ ]$.

Cases ($\rightarrow_{m_0}$) *and* ($\rightarrow_{m_1}$). The term M has the form $\mathit{meas}\,x$: $l(M) = 3$. The term M' is the term constant 0 or the term constant 1: $l(M') = 1 < l(M)$.

Cases ($\rightarrow_{n_0}$) *and* ($\rightarrow_{n_1}$). The term M has the form $\mathit{new}\,0$ or $\mathit{new}\,1$: $l(M) = 3$. The term M' is a term variable: $l(M') = 1 < l(M)$.

Case ($\rightarrow_U$). The term M is of the form $U\langle x_1, \ldots, x_n\rangle$ and the term M' the form $\langle x_1, \ldots, x_n\rangle$. Therefore

$$l(M') = l\langle x_1, \ldots, x_n\rangle < 2 + l\langle x_1, \ldots, x_n\rangle = 1 + l(U) + l\langle x_1, \ldots, x_n\rangle = l(M).$$

The cases corresponding to the congruence are straightforward using the induction hypothesis. □

Theorem 7.2.12 (Normalization)**.** *Let $P = [\ Q, L, M\ ]$ be a program, and let $(P_n)_n$ be a sequence of programs $P_n = [\ Q_n, L_n, M_n\ ]$ such that $P_0 = P$ and such that for all n, either $P_n \rightarrow^{\kappa_n}_{\rho_n} P_{n+1}$ or $P_n = P_{n+1}$ if P_n does not reduce. Then $P_{l(M)}$ is either a value or a fixed point, and for all $n > l(M)$, $P_n = P_{l(M)}$.*

Proof. If $(P_n)_n$ is the sequence described in the hypothesis, we show by induction on n that either P_n is a value, or it is a fixed point, or $l(M_n) \leqslant l(M) - n$: If $n = 0$: since $P_0 = P$, we are done. Suppose the result is true for n. Then either P_n is a value, or from Theorem 7.2.8 there exists P' such that $P_n \rightarrow P'$, and thus P_{n+1} is such that $P_n \rightarrow_{\kappa_n} P_{n+1}$. From Lemma 7.2.11, either $\kappa_n = \omega$ or $l(M_{n+1}) < l(M_n)$. Since by induction hypothesis $l(M_n) \leqslant l(M) - n$, by definition we have $l(M_{n+1}) \leqslant l(M) - (n + 1)$.

We have $l(M_n) \geqslant 1$ for all n. Therefore the n for which P_n reduce are smaller than $l(M)$. And for $n = l(M)$, P_n is either a fixed point (in which case it will remain so for all larger n by Lemma 7.2.5) or a value (in which case it will remain so for all larger n since values do not reduce, also by Lemma 7.2.5). □

7.2.4 Quantum Context and Reduction

When describing the reduction rules, we carefully separated the quantum context from the lambda-term. In this subsection we show that the precise order of quantum bits in the quantum array does not matter.

Definition 7.2.13. If σ is a permutation of $\{1, \ldots, n\}$, we extend σ to $\mathbb{N}$ with $\sigma(j) = j$ for $j > n$, and we define $\bar{\sigma}$ to be the corresponding permutation of quantum bits

$$\bar{\sigma}|x_1 \cdots x_n \cdots x_{n+k}\rangle = |x_{\sigma(1)} \cdots x_{\sigma(n)} x_{n+1} \cdots x_{n+k}\rangle.$$

We say that (Q_1, L_1) is σ-equivalent to (Q_2, L_2) if Q_1 and Q_2 have the same size, $Q_2 = \bar{\sigma}(Q_1)$ and $L_2 = \sigma^{-1} \circ L_1$. We write $(Q_1, L_1) =_\alpha{}^\sigma (Q_2, L_2)$. We define an equivalence relation called *alpha-equivalence on quantum contexts* by $(Q_1, L_1) =_\alpha (Q_2, L_2)$ if there exists a σ such that $(Q_1, L_1) =_\alpha{}^\sigma (Q_2, L_2)$.

The alpha-equivalence is sound with respect to the semantics:

Lemma 7.2.14. *Suppose that* $[\ Q, |x_1 \dots x_n\rangle, M\ ] \to_p [\ Q', |y_1 \dots y_m\rangle, M'\ ]$. *Then for all permutations* σ *on* $\{1, \dots, n\}$, *there exists a permutation* τ *on* $\{1, \dots, m\}$ *such that* $[\ \bar{\sigma}Q, \sigma|x_1 \dots x_n\rangle, M\ ] \to_p [\ \bar{\tau}Q', \tau|y_1 \dots y_m\rangle, M'\ ]$.

Proof. The proof is done by induction on the derivation of the reduction $[\ Q, |x_1 \dots x_n\rangle, M\ ] \to_p [\ Q', |y_1 \dots y_m\rangle, M'\ ]$. In the classical cases of Table 7.2, since the quantum context is not touched, choose $\tau = \sigma$ and apply the same rule that was used at first. For the rules $(\to_1^{n_0})$ and $(\to_1^{n_1})$, again, since σ only act on the n first quantum bits of the context, applying the permutation before or after the addition of a new quantum bit to the quantum array does not change the output. Thus, choosing $\tau = \sigma \cup \{n + 1 \mapsto n + 1\}$ solves the problem.

Now, suppose that we are in the case of the rule $(\to_U 1)$. In this situation we apply a unitary map on some quantum bits in the array, referred to by L. Since we perform σ both on L and on Q, it does not modify the actual unitary map performed. Therefore, again choosing $\tau = \sigma$ gives us the answer.

In the cases $(\to_\alpha^{m_0})$ and $(\to_\beta^{m_1})$, the sizes of the quantum array and of the linking function change along the reduction by the removal of the entity number i. However, in the program $[\ \bar{\sigma}Q, \sigma|x_1 \dots x_n\rangle, \mathit{meas}\, x_{\sigma(i)}\ ]$ the reduction operation will remove entity number $\sigma(i)$. Therefore, if f and g are the maps

$$f(j) = \begin{cases} j & \text{if } j < i, \\ j+1 & \text{if } j \geqslant i, \end{cases} \qquad g(j) = \begin{cases} j & \text{if } j < \sigma(i), \\ j-1 & \text{if } j \geqslant i, \end{cases}$$

define τ as $g \circ \sigma \circ f$.

Finally, the induction cases are easy application of the induction hypothesis. □

7.2.5 Reduction to Values

In the reduction process, what we are really seeking is the final result of the computation. In this section we explicate the relation of programs to values.

Definition 7.2.15. We use the notion defined in Section 6.2.3: If X is the set of closed valid programs and U the set of values, let $\mathit{prob}_U : X \times U \to [0, 1]$ be the map $\mathit{prob}_U(P, V)$ that returns the total probability for a program P to end up on the value V in zero or more steps. This function is called the *big-step reduction*.

We also define the *small-step reduction* operation $\mathit{prob} : X \times X \to [0, 1]$: for closed programs P, P', we define $\mathit{prob}(P, P') = p$ if there is a single-step probabilistic reduction $P \to_p P'$, $\mathit{prob}(V, V) = 1$ if V is a value program, and $\mathit{prob}(P, P') = 0$ in all other cases. Note that for all well-typed P, $\sum_{P' \in X} \mathit{prob}(P, P') = 1$.

Lemma 7.2.16. *The reduction system defined in Definition 7.2.2 is a probabilistic reduction system.*

Proof. By case distinction. □

Definition 7.2.17. Given a set Z, let $\mathcal{C}Z$ be the set of probability distributions over it. We curry the small-step reduction $\mathit{prob} : X \times X \to [0, 1]$ to the map $\mathit{prob}' : \mathcal{C}X \to \mathcal{C}X$: $\mathit{prob}'(\sum_i \alpha_i P_i) = \sum_i \alpha_i \sum_{P' \in X} \mathit{prob}(P_i, P')P'$. Similarly, we curry the function $\mathit{prob}_U : X \times U \to [0, 1]$ to the map $\mathit{prob}'_U : \mathcal{C}X \to \mathcal{C}U$: $\mathit{prob}'_U(\sum_i \alpha_i P_i) = \sum_i \alpha_i \mathit{prob}_U(P_i, V)V$.

Definition 7.2.18. If P is a closed well-typed program of type *bit*, and $b \in \{0,1\}$, we define $(P \Downarrow b) = \sum_{V \in U_b} prob'(P,V)$, where U_b is the set of valid programs with term the value b. We say that P *evaluates to b with probability $P \Downarrow b$*. The definition of $P \Downarrow b$ can be extended in the same way to probability distributions of programs.

Lemma 7.2.19. *Let $P : A$ be a program such that $P \to^{\kappa}_{\rho} P'$. Then either $prob'(P) = P'$, $\rho = 1$ and $\kappa \neq m_0, m_1$, or $prob'(P) = \alpha P_0 + \beta P_1$, with $P \to^{m_0}_{\alpha} P_0$ and $P \to^{m_1}_{\beta} P_1$*

Proof. By induction on the reduction $P \to^{\kappa}_{\rho} P'$. □

Definition 7.2.20. We define a *formal probability distribution* of quantum closures to be $\Gamma \vDash \sum_i \rho_i [Q_i, L_i, M_i] : A$, where each $\Gamma \vDash [Q_i, L_i, M_i] : A$ is valid and $\sum \rho_i \leqslant 1$. The distribution is said to be closed if $|\Gamma| = \emptyset$.

Lemma 7.2.21. *If $\vDash P : A$ is valid, so is $\vDash prob' P : A$ and $\vDash prob'_U P : A$.*

Proof. The lemma is a corollary of Theorem 7.2.7. □

Due to the strong normalization theorem, applying the map $prob'_U$ is applying the map $prob'$ finitely many times.

Theorem 7.2.22. *If $[\ Q, L, M\ ]$ is a program, then the reduction satisfies*

$$prob'^{l(M)}[\ Q, L, M\ ] = \sum_{i=1}^{n} \rho_i [\ Q_i, L_i, V_i\ ] + \sum_{j=1}^{m} \rho'_j P_j,$$

with $prob'_U[\ Q, L, M\ ] = \sum_{i=1}^{n} \rho_i [\ Q_i, L_i, V_i\ ]$, the P_j being fixed points, $\sum_i \rho_i + \sum_j \rho'_j = 1$, and $n + m < 2^{l(M)}$.

Proof. Corollary of Theorem 7.2.12. □

7.3 Denotational Semantics

In Section 4.3 the notion of completely positive map is used to model the notion of quantum computation. We aim to show that the linear subset of the quantum lambda calculus has the category **CPM** as a fully abstract model. Note that the interpretation will not be "onto" all completely positive maps, but will only be "onto up to scalar multiplies".

7.3.1 Modeling the Linear Quantum Lambda Calculus

We interpret the language in the category defined in Section 4.3.2.

Lemma 7.3.1. *The following statements are valid.*

1. *Let σ, τ be any signatures. Then $0 : \sigma \to \tau$ is completely positive.*
2. *The maps*

$$\iota : \begin{array}{ccc} 1,1 & \to & 2 \\ (a,b) & \mapsto & \left(\begin{smallmatrix} a & 0 \\ 0 & b \end{smallmatrix}\right), \end{array} \qquad p : \begin{array}{ccc} 2 & \to & 1,1 \\ \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right) & \mapsto & (a,d) \end{array}$$

are completely positive.

$$
\begin{array}{rcl}
[\![x : A \rhd x : A]\!](v) & = & v \\
[\![\rhd * : \top]\!](x) & = & x
\end{array}
\qquad
\begin{array}{rcl}
[\![\rhd 0 : bit]\!](x) & = & (x, 0) \\
[\![\rhd 1 : bit]\!](x) & = & (0, x)
\end{array}
$$

$$
[\![\Delta \rhd \Omega : A]\!] = \quad 0 : \quad [\![\Delta]\!] \to [\![A]\!]
$$

$$
[\![\rhd new : bit \multimap qbit]\!] = \quad \Phi(\iota) : \quad 1 \to [\![bit]\!] \otimes [\![qbit]\!]
$$

$$
[\![\rhd meas : qbit \multimap bit]\!] = \quad \Phi(p) : \quad 1 \to [\![qbit]\!] \otimes [\![bit]\!]
$$

$$
[\![\rhd U : qbit^{\otimes n} \multimap qbit^{\otimes n}]\!] = \quad \Phi(U) : \quad 1 \to 2^{2n}
$$

$$
\frac{[\![\Delta, x : A \rhd M : B]\!] \quad = \quad f : [\![\Delta]\!] \otimes [\![A]\!] \to [\![B]\!]}{[\![\Delta \rhd \lambda x.M : A \multimap B]\!] \quad = \quad \Phi(f) : [\![\Delta]\!] \to [\![A]\!] \otimes [\![B]\!]}
$$

$$
\frac{\begin{array}{rcl} [\![\Delta \rhd M : A \multimap B]\!] & = & \Phi(g) : [\![\Delta]\!] \to [\![A]\!] \otimes [\![B]\!] \\ [\![\Gamma \rhd N : A]\!] & = & f : [\![\Gamma]\!] \to [\![A]\!] \end{array}}{[\![\Delta, \Gamma \rhd MN : B]\!] \quad : \quad x \otimes y \mapsto g(x \otimes (fy)) : [\![\Delta]\!] \otimes [\![\Gamma]\!] \to [\![B]\!]}
$$

$$
\frac{\begin{array}{rcl} [\![\Delta \rhd P : bit]\!] & : & x \mapsto (px, qx) : [\![\Delta]\!] \to [\![bit]\!] \\ [\![\Gamma \rhd M : A]\!] & = & f : [\![\Gamma]\!] \to [\![A]\!] \\ [\![\Gamma \rhd N : A]\!] & = & g : [\![\Gamma]\!] \to [\![A]\!] \end{array}}{[\![\Delta, \Gamma \rhd if P \ then \ M \ else \ N : A]\!] \quad : \quad x \otimes y \mapsto (px)(fy) + (qx)(gy)}
$$

$$
\frac{\begin{array}{rcl} [\![\Delta \rhd M : A]\!] & = & f : [\![\Delta]\!] \to [\![A]\!] \\ [\![\Gamma \rhd N : B]\!] & = & g : [\![\Gamma]\!] \to [\![B]\!] \end{array}}{[\![\Delta, \Gamma \rhd \langle M, N \rangle : A \otimes B]\!] \quad : \quad x \otimes y \mapsto fx \otimes gy : [\![\Delta]\!] \otimes [\![\Gamma]\!] \to [\![A]\!] \otimes [\![B]\!]}
$$

$$
\frac{\begin{array}{rcl} [\![\Delta \rhd M : A \otimes B]\!] & = & f : [\![\Delta]\!] \to [\![A]\!] \otimes [\![B]\!] \\ [\![\Gamma, x : A, y : B \rhd N : C]\!] & = & g : [\![\Gamma]\!] \otimes [\![A]\!] \otimes [\![B]\!] \to [\![C]\!] \end{array}}{[\![\Delta, \Gamma \rhd let \ \langle x, y \rangle = M \ in \ N : C]\!] \quad : \quad u \otimes v \mapsto g(v \otimes (fu)) : [\![\Delta]\!] \otimes [\![\Gamma]\!] \to [\![C]\!]}
$$

$$
\frac{\begin{array}{rcl} [\![\Delta \rhd M : \top]\!] & = & f : [\![\Delta]\!] \to 1 \\ [\![\Gamma \rhd N : C]\!] & = & g : [\![\Gamma]\!] \to [\![C]\!] \end{array}}{[\![\Delta, \Gamma \rhd let * = M \ in \ N : C]\!] \quad : \quad u \otimes v \mapsto (fu)(gv) : [\![\Delta]\!] \otimes [\![\Gamma]\!] \to [\![C]\!]}
$$

Table 7.3: Denotational semantics for typing judgements.

3. The maps

$$
\begin{array}{rcl} 1 & \to & 1, 1 \\ a & \mapsto & (a, 0), \end{array}
\qquad
\begin{array}{rcl} 1 & \to & 1, 1 \\ a & \mapsto & (0, a). \end{array}
$$

are completely positive.

Proof. The maps defined in the lemma are completely positive since they have positive characteristic matrices. □

Definition 7.3.2. We set the denotation of types to be

$$
[\![bit]\!] = (1, 1), \qquad [\![A \otimes B]\!] = [\![A]\!] \otimes [\![B]\!],
$$
$$
[\![qbit]\!] = (2), \qquad [\![A \multimap B]\!] = [\![A]\!] \otimes [\![B]\!],
$$

and the denotation of contexts to be

$$
[\![x_1 : A_1, \ldots, x_n : A_n]\!] = [\![A_1]\!] \otimes \cdots \otimes [\![A_n]\!], \quad [\![\emptyset]\!] = 1.
$$

The denotation of a typing judgement of the form $\Delta \rhd M : A$ is a linear map

$$
[\![\Delta \rhd M : A]\!] : [\![\Delta]\!] \to [\![A]\!],
$$

defined inductively on the typing derivation of $\Delta \rhd M : A$ as in Table 7.3. Note that from Lemma 7.1.7 this typing derivation is unique, so the definition makes sense. Here, $\Phi : \hom(\sigma \otimes \sigma', \tau) \xrightarrow{\sim} \hom(\sigma, \sigma' \otimes \tau)$ is the bijection of Definition 4.3.4 and ι and p are respectively the quantum bits creation and the measurement operation of Lemma 7.3.1.

Definition 7.3.3. We also define the denotation of a valid quantum closure. Consider a valid quantum closure $\Delta \vDash [\ Q, L, M\ ] : A$ where $L = |x_1 \cdots x_n y_1 \cdots y_m\rangle$ and $|Q| = n + m$. The quantum context (Q, L) can be seen as a map $g : 1 \to 2^{\otimes n} \otimes 2^{\otimes m}$ such that $g(1) = Q$. Then if f is the morphism

$$[\![\Delta, x_1 : qbit, \ldots, x_n : qbit \rhd M : A]\!] : [\![\Delta]\!] \otimes 2^{\otimes n} \to [\![A]\!],$$

one defines $[\![\Delta \vDash [\ Q, L, M\ ] : A]\!]$ as the composition

$$[\![\Delta]\!] \xrightarrow{id_{[\![\Delta]\!]} \otimes g} [\![\Delta]\!] \otimes 2^{\otimes n} \otimes 2^{\otimes m} \xrightarrow{f \otimes id_{2^{\otimes m}}} [\![A]\!] \otimes 2^{\otimes m} \xrightarrow{id_{[\![A]\!]} \otimes \mathbf{tr}} [\![A]\!].$$

One extends this definition to probabilistic distributions of valid quantum closures using linearity:

$$[\![\Delta \vDash \sum_i \rho_i P_i : A]\!] = \sum_i \rho_i [\![\Delta \vDash P_i : A]\!].$$

Lemma 7.3.4. *Let $\Delta \rhd M : A$ be a valid typing judgement. Then $[\![\Delta \rhd M : A]\!]$ is a completely positive map. Let $\Gamma \vDash [\ Q, L, M\ ] : A$ be a valid quantum closure. Then $[\![\Gamma \vDash [\ Q, L, M\ ] : A]\!]$ is a completely positive map.*

Proof. We prove that $[\![\Delta \rhd M : A]\!]$ is a completely positive map by induction on the typing derivation of $\Delta \rhd M : A$.

(ax_1) *and* $(\top.I)$. The identity map is completely positive since **CPM** is a category.

(ax_2). The denotations of the term constants *meas*, *new*, U, 0 and 1 are completely positive map from Lemma 7.3.1.

(Ω). The map 0 is completely positive from Lemma 7.3.1.

(λ). From the monoidal closedness of **CPM**, the image of f by Φ is completely positive.

(app). Since Φ^{-1} sends completely positive matrices to completely positive matrices. Thus g is completely positive. The final denotation is completely positive since composition and tensors of completely positive maps are completely positive.

(if). The denotation can be reformulated as

$$[\![\Delta]\!] \otimes [\![\Gamma]\!] \xrightarrow{[\![P]\!] \otimes id} (1 \oplus 1) \otimes [\![\Gamma]\!] = [\![\Gamma]\!] \oplus [\![\Gamma]\!] \xrightarrow{f \oplus g} [\![A]\!] \oplus [\![A]\!] \xrightarrow{\mathbf{tr} \otimes id_{[\![A]\!]}} [\![A]\!].$$

$(\otimes.I)$, $(\otimes.E)$ *and* $(\top.E)$. In these three cases, the denotation is the tensor of two completely positive maps: it is therefore a completely positive map.

This proves that the denotation of a valid term is a completely positive map. The fact that the denotation of $\Gamma \vDash [\ Q, L, M\ ] : A$ is also completely positive comes from the fact that completely positive maps are closed under composition and tensor, and that the trace operator and the identity are themselves completely positive maps. □

Lemma 7.3.5 (Substitution)**.** *Suppose that* $|\Gamma| \cap |\Delta| = \emptyset$ *and*

$$\begin{aligned} [\![\Delta, x : A \rhd M : B]\!] = &\quad G : [\![\Delta]\!] \otimes [\![A]\!] \to [\![B]\!], \\ [\![\Gamma \rhd N : A]\!] = &\quad F : [\![\Gamma]\!] \to [\![A]\!], \\ [\![\Delta, \Gamma \rhd M[N/x] : B]\!] = &\quad H : [\![\Delta]\!] \otimes [\![\Gamma]\!] \to [\![B]\!]. \end{aligned}$$

Then for all $d \in V_{[\![\Delta]\!]}$ *and all* $g \in V_{[\![\Gamma]\!]}$*, we have* $H(d \otimes g) = G(d \otimes (Fg))$*.*

Proof. We show that $H = (id_{[\![\Delta]\!]} \otimes F); G$ by induction on the typing derivation of $\Delta, x : A \rhd M : B$.

(ax_1). We are considering the judgement $x : A \rhd x : A$. We have $|\Delta| = \emptyset$, $A = B$ and $F = id_{[\![A]\!]}$. Since $x[N/x] = N$, we have as required $H = G = F; G$.

(ax_2) *and* $(\top.I)$. These cases are not to be considered: the typing contexts must contain $(x : A)$.

(Ω). Since $\Omega[N/x] = \Omega$, and since $(id_{[\![\Delta]\!]} \otimes 0); G = 0$, the result is true.

(λ). In this case, M is of the form $\lambda y^C.M'$ and A is of the form $C \multimap D$. Also, the typing judgement $\Delta, y : C, x : A \rhd M' : D$ is valid. Note that $y \neq x$. By induction hypothesis, if G' is the map $[\![\Delta, y : C, x : A \rhd M' : D]\!]$, the map $[\![\Delta, y : C, \Gamma \rhd M'[N/x] : D]\!]$ is H' verifying for all $d \in V_{[\![\Delta]\!]}$, for all $g \in V_{[\![\Gamma]\!]}$, for all $c \in V_{[\![C]\!]}$, $H'(d \otimes c \otimes g) = G'(d \otimes c \otimes F(g))$. Applying Φ, we get

$$\Phi(H')(d \otimes g) = \sum_{c \in B_{[\![C]\!]}} c \otimes H'(d \otimes c \otimes g) = \sum_{c \in B_{[\![C]\!]}} c \otimes G'(d \otimes c \otimes F(g)) = \Phi(G')(d \otimes F(g)),$$

that is, $[\![\Delta, \Gamma \rhd (\lambda y^C.M')[N/x] : C \multimap D]\!]$.

(app). The term M is of the form $M_1 M_2$ and the context $(\Delta, x : A)$ splits into (Δ_1, Δ_2), such that $\Delta_1 \rhd M_1 : C \multimap B$ and $\Delta_2 \rhd M_2 : C$ for some type C. Let

$$G_1 = [\![\Delta_1 \rhd M_1 : C \multimap B]\!], \qquad G_2 = [\![\Delta_2 \rhd M_2 : C]\!].$$

There are two cases

$x \in |\Delta_1|$. The context Δ_1 splits into $(\Delta'_1, x : A)$. From Lemma 7.1.4, the variable x belongs to $FV(M_1)$ but not to $FV(M_2)$. From Lemma 7.1.7, $M[N/x] = (M_1[N/x])M_2$. Let G'_1 be the denotation $[\![\Delta'_1, \Gamma \rhd M_1[N/x] : C \multimap D]\!]$. By induction hypothesis, for all $d \in V_{\Delta'_1}$, for all $g \in V_\Gamma$, $G'_1(d \otimes g) = G_1(d \otimes F(g))$. If $d' \in V_{[\![\Delta_2]\!]}$, this means that $H = [\![\Delta, \Gamma \rhd M_1[N/x]M_2 : B]\!]$ is

$$H(d \otimes d' \otimes g) = \Phi^{-1}(G_1)(d \otimes G_2(d') \otimes F(g)).$$

Since $G(d \otimes a \otimes d') = \Phi^{-1}(G_1)(d \otimes G_2(d') \otimes a)$, we have as required

$$H(d \otimes d' \otimes g) = G(d \otimes d' \otimes F(g)).$$

$x \in |\Delta_2|$. The context $\Delta_1 frm{-}e$ splits into $(\Delta'_2, x : A)$. From Lemma 7.1.4, x belongs to $FV(M_2)$ but not to $FV(M_1)$. From Lemma 7.1.7, $M[N/x] = M_1(M_2[N/x])$. Let G'_2 be the denotation $[\![\Delta'_2, \Gamma \rhd M_2[N/x] : C]\!]$. By induction hypothesis, for all $d \in V_{\Delta'_2}$, for all $g \in V_\Gamma$, $G'_2(d \otimes g) = G_2(d \otimes F(g))$. If $d' \in V_{[\![\Delta_1]\!]}$, this means that $H = [\![\Delta, \Gamma \rhd M_1 M_2[N/x] : B]\!]$ is

$$H(d' \otimes d \otimes g) = \Phi^{-1}(G_1)(d' \otimes G_2(d \otimes F(g))).$$

Since $G(d \otimes a \otimes d') = \Phi^{-1}(G_1)(d' \otimes G_2(d \otimes a))$, we have as required

$$H(d \otimes d' \otimes g) = G(d' \otimes d \otimes F(g)).$$

(if). The term M is of the form *if* P *then* M_1 *else* M_2 and the context $(\Delta, x : A)$ splits into (Δ_1, Δ_2), such that $\Delta_1 \rhd P : bit$ and $\Delta_2 \rhd M_1, M_2 : A$. Let

$$(p, q) = [\![\Delta_1 \rhd P : bit]\!], \qquad G_1 = [\![\Delta_2 \rhd M_1 : A]\!], \qquad G_2 = [\![\Delta_2 \rhd M_2 : A]\!].$$

There are two cases.

$x \in |\Delta_1|$. The context Δ_1 splits into $(\Delta_1', x : A)$. From Lemma 7.1.4, $x \in FV(P)$ and x is either in $FV(M_1)$ nor in $FV(M_2)$. From Lemma 7.1.7, $M[N/x] = if\, P[N/x]\, then\, M_1\, else\, M_2$. Suppose that $g \in V_{[\![\Gamma]\!]}$, $d' \in V_{[\![\Delta_1']\!]}$, $d \in V_{[\![\Delta_2]\!]}$. Let $(p', q') = [\![\Delta_1', \Gamma \rhd P[N/x] : bit]\!]$. By induction hypothesis, this is

$$(p'(d' \otimes g), q'(d' \otimes g)) = (p', q')(d' \otimes g) = (p, q)(d' \otimes F(g)) = (p(d' \otimes F(g)), q(d' \otimes F(g))),$$

thus $[\![\Delta_1', \Gamma, \Delta_2 \rhd if\, P[N/x]\, then\, M_1\, else\, M_2 : A]\!]$ is

$$H(d' \otimes g \otimes d) = p(d' \otimes F(g))G_1(d) + q(d' \otimes F(g))G_2(d) = G(d' \otimes d \otimes F(g)),$$

that is the requested form.

$x \in |\Delta_2|$. The context Δ_2 splits into $(\Delta_2', x : A)$. From Lemma 7.1.4, $x \in FV(P)$ and x is either in $FV(M_1)$ nor in $FV(M_2)$. From Lemma 7.1.7, $M[N/x] = if\, P[N/x]\, then\, M_1\, else\, M_2$. Suppose that $g \in V_{[\![\Gamma]\!]}$, $d' \in V_{[\![\Delta_2']\!]}$, $d \in V_{[\![\Delta_1]\!]}$. Let

$$G_1' = [\![\Delta_2', \Gamma \rhd M_1[N/x] : A]\!], \quad G_2' = [\![\Delta_2', \Gamma \rhd M_2[N/x] : A]\!].$$

By induction hypothesis, this is

$$G_1' = G_1(d' \otimes F(g)), \quad G_2' = G_2(d' \otimes F(g)).$$

thus $[\![\Delta_1, \Delta_2', \Gamma \rhd if\, P\, then\, M_1[N/x]\, else\, M_2[N/x] : A]\!]$ is

$$H(d \otimes d' \otimes g) = p(d)G_1(d' \otimes F(g)) + q(d)G_2(d' \otimes F(g)) = G(d \otimes d' \otimes F(g)),$$

which is in the requested form.

The three last cases are proved similarly. □

Lemma 7.3.6. *Suppose that $P : A$ is a program of type A such that $prob' P = \sum_i \rho_i P_i$ then*

$$[\![P : A]\!] = \sum_i \rho_i [\![P_i : A]\!]$$

Proof. Suppose that $P = [\, Q, L, M\,]$, where $L = |x_1 \ldots x_n\rangle$. If P does not reduce, then $prob' P = P$ and we are done. If P reduce to some P' via some rule κ, we use Lemma 7.2.19, and proceed by case distinction:

First, suppose that $\kappa \neq m_0, m_1$. We prove the result by induction on a derivation of the reduction.

($\rightarrow_1^\beta$). In this case, $M \equiv (\lambda x^B.N)N'$, and then $P' = [\, Q, L, N[N'/x]\,]$. By definition and Theorem 7.2.7, since P is of type A we have

$$x_1 : qbit, \ldots, x_n : qbit \rhd (\lambda x^B.N)N' : A, \quad x_1 : qbit, \ldots, x_n : qbit \rhd N[N'/x] : A.$$

The former being valid, its context splits as (Δ, Γ) where $\Delta, x : B \rhd N : A$ and $\Gamma \rhd N' : B$. Let $F = [\![\Delta, x : B \rhd N : A]\!]$ and $G = [\![\Gamma \rhd N' : B]\!]$. By definition, $[\![\Delta, \Gamma \rhd (\lambda x^B.N)N' : A]\!]$ is equal to $d \otimes g \mapsto F(d \otimes G(g))$, if $d \in V_{[\![\Delta]\!]}$ and $g \in V_{[\![\Gamma]\!]}$. By Lemma 7.3.5, this is precisely the denotation of the latter. Thus, $[\![\vDash P' : A]\!] = 1 \cdot [\![\vDash P : A]\!]$.

($\rightarrow_1^{\otimes}$). In this case, $M \equiv (\mathit{let}\ \langle x^B, y^C\rangle = \langle V, W\rangle\ \mathit{in}\ N)$, and then $P' = [\ Q, L, N[V/x, W/y]\ ]$. By definition and Theorem 7.2.7, since P is of type A we have

$$x_1 : \mathit{qbit}, \ldots, x_n : \mathit{qbit} \rhd M : A, \quad x_1 : \mathit{qbit}, \ldots, x_n : \mathit{qbit} \rhd N[V/x][W/y] : A.$$

The former being valid, its context splits as $(\Delta, \Gamma_1, \Gamma_2)$ where $\Delta, x : B, y : C \rhd N : A$, $\Gamma_1 \rhd V : B$ and $\Gamma_2 \rhd W : C$. Let

$$F = [\![\Delta, x : B, y : C \rhd N : A]\!], \qquad G_1 = [\![\Gamma_1 \rhd V : B]\!], \qquad G_2 = [\![\Gamma_2 \rhd W : C]\!].$$

By definition, $[\![\Gamma_1, \Gamma_2 \rhd \langle V, W\rangle : B \otimes C]\!]$ is the map $G_1 \otimes G_2$. By definition still, the map $[\![\Delta, \Gamma_1, \Gamma_2 \rhd M : A]\!]$ is $((\mathit{id} \otimes G_1 \otimes G_2); F)$. By Lemma 7.3.5, this is the denotation of the latter. Thus, $[\![\vDash P' : A]\!] = 1 \cdot [\![\vDash P : A]\!]$.

($\rightarrow_1^{\top}$). In this case, $M \equiv (\mathit{let}\ * = *\ \mathit{in}\ N)$, and then $P' = [\ Q, L, N\ ]$. Since P is of type A we have

$$x_1 : \mathit{qbit}, \ldots, x_n : \mathit{qbit} \rhd M : A, \qquad x_1 : \mathit{qbit}, \ldots, x_n : \mathit{qbit} \rhd N : A.$$

Since $[\![\rhd * : \top]\!] = \mathit{id}_1$, the denotation of the former is equal to the denotation of the latter: we have as requested $[\![\vDash P' : A]\!] = 1 \cdot [\![\vDash P : A]\!]$.

($\rightarrow_1^{\mathit{if}_0}$) *and* ($\rightarrow_1^{\mathit{if}_1}$). In this case, $M \equiv (\mathit{if}0\ \mathit{then}\ N\ \mathit{else}\ N')$ (respectively $M \equiv (\mathit{if}1\ \mathit{then}\ N\ \mathit{else}\ N')$), and then $P' = [\ Q, L, N'\ ]$. Since P is of type A we have

$$\begin{aligned} &x_1 : \mathit{qbit}, \ldots, x_n : \mathit{qbit} \rhd \mathit{if}0\ \mathit{then}\ N\ \mathit{else}\ N' : A,\\ &x_1 : \mathit{qbit}, \ldots, x_n : \mathit{qbit} \rhd \mathit{if}1\ \mathit{then}\ N\ \mathit{else}\ N' : A,\\ &x_1 : \mathit{qbit}, \ldots, x_n : \mathit{qbit} \rhd N, N' : A, \end{aligned}$$

and $\rhd 0, 1 : \mathit{bit}$. Since $[\![\rhd 0 : \mathit{bit}]\!](a) = (a, 0)$ and $[\![\rhd 1 : \mathit{bit}]\!](a) = (0, a)$, the denotation of M is then equal to the denotation of N' (respectively, of N). Therefore $[\![\vDash P' : A]\!] = 1 \cdot [\![\vDash P : A]\!]$.

($\rightarrow_1^{\omega}$). Since $P = P'$, they have the same denotation.

($\rightarrow_1^{n_0}$) *and* ($\rightarrow_1^{n_1}$). We prove the case ($\rightarrow_1^{n_0}$), the other one is similar. The term M is of the form $\mathit{new}\,0$, and $A = \mathit{qbit}$. The denotation of $\rhd M : \mathit{qbit}$ is the map f such that $f(1) = |0\rangle\langle 0|$. The denotation of P is

$$G = 1 \xrightarrow{g} 2^{\otimes n} \xrightarrow{f \otimes \mathit{id}} 2 \otimes 2^{\otimes n} \xrightarrow{\mathit{id} \otimes \mathbf{tr}} 2,$$

where g is the map $g(1) = QQ^*$. Since Q is a normalized vector, $G(1) = |0\rangle\langle 0|$.

Now, $P' = [\ Q \otimes |0\rangle, |x_1 \ldots x_n w\rangle, w\ ] : \mathit{qbit}$ and its denotation is

$$G' = 1 \xrightarrow{g'} 2 \otimes 2^{\otimes n} \xrightarrow{\mathit{id} \otimes \mathit{id}} 2 \otimes 2^{\otimes n} \xrightarrow{\mathit{id} \otimes \mathbf{tr}} 2,$$

where $g'(1) = |0\rangle\langle 0| \otimes QQ^*$. The map G' is therefore equal to G.

($\rightarrow_1^{U}$). This case is similar: the denotation of P is of the form

$$G = 1 \xrightarrow{g} 2^{\otimes i} \otimes 2^{\otimes (n-i)} \xrightarrow{f \otimes \mathit{id}} 2^{\otimes i} \otimes 2^{\otimes (n-i)} \xrightarrow{\mathit{id} \otimes \mathbf{tr}} 2^{\otimes i},$$

where $F = \Phi^{-1}(U)$, and g is QQ^*. The denotation of P' is

$$G' = 1 \xrightarrow{g'} 2^{\otimes i} \otimes 2^{\otimes (n-i)} \xrightarrow{\mathit{id} \otimes \mathit{id}} 2^{\otimes i} \otimes 2^{\otimes (n-i)} \xrightarrow{\mathit{id} \otimes \mathbf{tr}} 2^{\otimes i},$$

where g' is $g; (\Phi^{-1}(U) \otimes \mathit{id})$. Therefore they are the same map.

The induction cases are trivial.

Finally, suppose that $\kappa = m_0$ or m_1. Then $P \rightarrow_\alpha^{m_0} P_0$ and $P \rightarrow_\beta^{m_1} P_0$ for some P_0 and some P_1. we prove the result again by induction on the derivation of one of the reduction.

Base case. We use the notation in Definition 7.2.2. In both cases, M is of the form $meas\, x_i$, and $A = bit$. By α-equivalence one can assume that $i = 1$: in this case Q is of the form

$$\sum_j \alpha_j |0\rangle \otimes |\tilde{\psi}_j^0\rangle + \sum_j \beta_j |1\rangle \otimes |\tilde{\psi}_j^1\rangle = |0\rangle \otimes \sum_j \alpha_j |\tilde{\psi}_j^0\rangle + |1\rangle \otimes \sum_j \beta_j |\tilde{\psi}_j^1\rangle.$$

Since the denotation $[\![x_1 : qbit \rhd meas\, x_1 : bit]\!]$ is the map p of Definition 7.3.2, the denotation $[\![\vDash [\, Q, L, M\,] : bit]\!]$ is the map

$$G = 1 \xrightarrow{g} 2 \otimes 2^{\otimes(n-1)} \xrightarrow{p \otimes \mathbf{tr}} 1 \oplus 1.$$

The value $g(1)$ is QQ^*, that is

$$\begin{aligned} &|0\rangle\langle 0| \otimes \sum_{j,k} \alpha_j \alpha_k^* |\tilde{\psi}_j^0\rangle\langle\tilde{\psi}_k^0| + |1\rangle\langle 1| \otimes \sum_{j,k} \beta_j \beta_k^* |\tilde{\psi}_j^1\rangle\langle\tilde{\psi}_k^1| \\ &+ |0\rangle\langle 1| \otimes \sum_{j,k} \alpha_j \beta_k^* |\tilde{\psi}_j^0\rangle\langle\tilde{\psi}_k^1| + |1\rangle\langle 0| \otimes \sum_{j,k} \beta_j \alpha_k^* |\tilde{\psi}_j^1\rangle\langle\tilde{\psi}_k^0|. \end{aligned}$$

Since $|\psi_j^0\rangle$, $|\psi_j^1\rangle$, $|\psi_j^0\rangle$ and $|\psi_j^1\rangle$ are basis vectors, applying $id \otimes \mathbf{tr}$ on $g(1)$ yields

$$\alpha|0\rangle\langle 0| + \beta|1\rangle\langle 1| + \delta|0\rangle\langle 1| + \gamma|1\rangle\langle 0|,$$

for some values δ, γ, and where α and β are as in Definition 7.2.2. Thus, $G(1)$ is precisely $(\alpha, \beta) = [\![prob' P : A]\!]$.

The induction cases are trivial. □

Lemma 7.3.7. *Suppose that $P : A$ is a fixed point. Then $[\![P]\!]$ is the zero map.*

Proof. Write P as $[\, Q, L, M\,]$. The proof is done by structural induction on M. □

Theorem 7.3.8 (Soundness)**.** *Given a program $P : A$,*

$$[\![P : A]\!] = [\![prob'_U P : A]\!].$$

Proof. By induction on n, using Lemma 7.3.6 one can prove that $[\![prob'^n P]\!] = [\![prob' P]\!]$. From Theorem 7.2.22, there exists an integer n such that $prob'^n P = prob'_U P + \sum_i P_i$ with P_i being fixed points. Therefore, from Lemma 7.3.7, $[\![P : A]\!] = [\![prob'^n P]\!] = [\![prob'_U P : A]\!]$. □

7.3.2 Fullness of the First-Order Fragment

In this section we will show a result analog as Theorem 4.3.16, namely that any superoperator is the image of a term. For proving this result, we use the same techniques as in (Selinger, 2004b).

Lemma 7.3.9. *Suppose that $n = 2^s$ and $m = 2^t$, and let $F : \mathbb{C}^{n\times n} \to \mathbb{C}^{m\times m}$ be a contraction. Then there exists a valid typing judgement $x_1 : qbit, \ldots, x_s : qbit \rhd M : qbit^{\otimes t}$ whose denotation is the map F.*

Proof. By definition, the map F can be written in the form $F(A) = UAU^*$, where $U \in \mathbb{C}^{m\times n}$ is a contraction. There exists matrices U_1, U_2 and U_3 such that

$$U' = \left(\begin{array}{c|c} U & U_1 \\ \hline U_2 & U_3 \end{array}\right)$$

is unitary of size $\mathbb{C}^{l\times l}$. Without loss of generality, one can assume that $l = 2^p$: one can always extend U' to

$$\left(\begin{array}{c|c} U' & 0 \\ \hline 0 & I_{2^p-l} \end{array}\right)$$

where I_{2^p-l} is the identity matrix. The requested program performs the following:

- Allocation of the variables $y_1 : qbit, \ldots, y_{l-s} : qbit$, and let the list of variables $(z_1, \ldots, z_l)$ be $(x_1, \ldots, x_s, y_1, \ldots, y_{l-s})$;
- application of the unitary U' to $\langle z_1, \ldots, z_l\rangle$;
- measurement of the variables $z_t, \ldots, z_l$: if the outcome is $\langle 0, \ldots, 0\rangle$, the program returns the term $\langle z_1, \ldots, z_t\rangle$. Otherwise it outputs the term $\Omega^{qbit^{\otimes t}}_{x_1:qbit,\ldots,x_s:qbit}$. □

Lemma 7.3.10. *Let $F : \mathbb{C}^{2^s\times 2^s} \to \mathbb{C}^{2^t\times 2^t}$ be a superoperator and $E \circ G$ is decomposition from Theorem 3.1.32. One can construct another decomposition $E' \circ G'$, where $G' : \mathbb{C}^{2^s\times 2^s} \to \mathbb{C}^{2^{t+k}\times 2^{t+k}}$ is a contraction and $E' : \mathbb{C}^{2^{t+k}\times 2^{t+k}} \to \mathbb{C}^{2^t\times 2^t}$ is a partial trace operator, for some integer k.*

Proof. The proof uses Lemma 3.1.29 to adjust the size of the output space of the sub-unitary E' to a power of 2. □

Lemma 7.3.11. *Let $E : \mathbb{C}^{2^{t+k}\times 2^{t+k}} \to \mathbb{C}^{2^t\times 2^t}$ be a partial trace operator. Then there exists a valid typing judgement $x_1 : qbit, \ldots, x_{t+k} : qbit \rhd M : qbit^{\otimes t}$ whose denotation is E.*

Proof. The requested term M measures the variables $x_{t+1}, \ldots, x_{t+k}$, perform a test on the result and in all cases returns $\langle x_1, \ldots, x_t\rangle$. □

Lemma 7.3.12. *Let $\sigma = (1,1)^{\otimes n} \otimes 2^{\otimes m}$, and $\sigma' = 2^{\otimes(m+n)}$. Then the measurement operator $\mu_\sigma : V_{\sigma'} \to V_\sigma$ is the image of some typing judgement $x : qbit^{\otimes(m+n)} \rhd M : bit^{\otimes n} \otimes qbit^{\otimes m}$.*

Proof. The requested term M measures the variables $x_{t+1}, \ldots, x_{t+k}$ and stores the results in $(z_1 : bit, \ldots, z_k : bit)$. It outputs the product $\langle z_1, \ldots, z_k, x_1, \ldots, x_t\rangle$. □

Theorem 7.3.13. *For any types $A = U_1 \otimes \cdots \otimes U_n$ and $B = V_1 \otimes \cdots \otimes V_m$, where each U_i and V_j is either bit or qbit, if $F : [\![A]\!] \to [\![B]\!]$ is any superoperator, then there exists a valid typing judgement $x : A \rhd M : B$ of denotation F.*

Proof. Without loss of generality, one can assume that $A = bit^{\otimes k} \otimes qbit^{\otimes l}$ and that $B = bit^{\otimes s} \otimes qbit^{\otimes t}$. The map $F : [\![A]\!] \otimes [\![B]\!]$ is of the form $(F_1, \ldots F_{2^k})$, where each $F_i : V_{2^l} \to V_{(1,1)^{\otimes s}\otimes 2^t}$ is a superoperator. From Theorem 4.3.15, F_i splits into $M_i \circ E_i \circ G_i$, where G_i is a contraction, E_i is a partial trace operator, and M_i is a measurement operator. Using Lemma 7.3.10, one can assume that all of them goes from images of tensors of *bit*'s and *qbit*'s. Therefore, using Lemma 7.3.12, Lemma 7.3.11 and Lemma 7.3.9, the F_i's are representable by some terms M_i. The function F is represented by a term that does a test on its input *bit* and perform the corresponding M_i. □

Corollary 7.3.14. *For every type $A = U_1 \otimes \cdots \otimes U_n$, where each U_i is either bit or qbit, and for every hermitian positive element $v \in V_{[\![A]\!]}$ of trace at most 1, $v = [\![\rhd M : A]\!](1)$ for some closed valid term M.*

Proof. Consider the function $F : \mathbb{C} \to V_{[\![A]\!]}$ defined by $F(1) = v$. Since $\chi_F = v$ is positive, and since $\mathbf{tr}(v) \leqslant 1$, F is a superoperator. From Theorem 7.3.13 it is the image of some typing judgement $\rhd M : A$. □

7.3.3 Fullness up to Scalar Multiple

Theorem 7.3.13 does not hold for generic types. However, a weaker statement does hold. We explicate it in this section.

Definition 7.3.15. Given a type A, we define the *canonical first-order representation* $\S A$ *of* A by the following: $\S bit = bit$, $\S qbit = qbit$, $\S\top = \top$, $\S(A \otimes B) = \S A \otimes \S B$, $\S(A \multimap B) = \S A \otimes \S B$.

Lemma 7.3.16. *For all types* A, $[\![A]\!] = [\![\S A]\!]$.

Proof. Proof by structural induction on A. □

Definition 7.3.17. We define the $\S$-*size* $s(A)$ of a type A as follows. First, $s(bit) = s(qbit) = s(\top) = 1$. Then, $s(A \multimap B) = 2^{s(A)} + s(B)$. Finally, $s(A \otimes B) = s(A) + s(B)$.

Lemma 7.3.18. *For all types* A, $s(A) \geqslant 1$. *Also,* $s(\S A) \leqslant s(A)$, *with equality if and only if* $A = \S A$.

Proof. Proof by structural induction on A. □

Next, we want to write a correspondence between types A and $\S A$, in the form of typing judgements $x : A \rhd \Upsilon^A_{x\Downarrow} : \S A$ and $x : \S A \rhd \Upsilon^A_{x\Uparrow} : A$ whose denotations are multiples of the identity function. The correspondences are built by induction on the $\S$-size of A:

- knowing the result for A we deduce it for $bit \multimap A$, $qbit \multimap A$ and $\top \multimap A$;
- knowing the result for A and B we deduce it for $A \otimes B$;
- knowing the result for $C \multimap (D \multimap B)$ we deduce it for $(C \otimes D) \multimap B$;
- knowing the result for $C \multimap D$ and $(\S C \otimes \S D) \multimap B$ we deduce it for $(C \multimap D) \multimap B$.

The detailed definition goes as follows:

Definition 7.3.19. We define the terms $\Upsilon^A_{x\Downarrow}$ and $\Upsilon^A_{x\Uparrow}$ by induction on the $\S$-size of A. First,

$$\Upsilon^{bit}_{x\Downarrow} = \Upsilon^{bit}_{x\Uparrow} = x, \qquad \Upsilon^{qbit}_{x\Downarrow} = \Upsilon^{qbit}_{x\Uparrow} = x, \qquad \Upsilon^{\top}_{x\Downarrow} = \Upsilon^{\top}_{x\Uparrow} = x$$

Then,

$$\begin{aligned}
\Upsilon^{A\otimes B}_{x\Downarrow} &= \quad let\ \langle y, z\rangle = x\ in\ \langle \Upsilon^A_{y\Downarrow}, \Upsilon^B_{z\Downarrow}\rangle,\\
\Upsilon^{A\otimes B}_{x\Uparrow} &= \quad let\ \langle y, z\rangle = x\ in\ \langle \Upsilon^A_{y\Uparrow}, \Upsilon^B_{z\Uparrow}\rangle.
\end{aligned}$$

For types of the form $A \multimap B$, we first consider the case where $A = \top$, $A = bit$ and $A = qbit$:

$$\begin{aligned}
\Upsilon^{\top\multimap B}_{x\Downarrow} &= \quad \langle *, let\ t^B = x*\ in\ \Upsilon^B_{t\Downarrow}\rangle\\
\Upsilon^{\top\multimap B}_{x\Uparrow} &= \quad let\ \langle y^\top, z^{\S B}\rangle = x\ in\ \lambda t.\ let\ * = y\ in\ \Upsilon^B_{z\Uparrow}\\
\Upsilon^{bit\multimap B}_{x\Downarrow} &= \quad if\, meas(\mathbf{H}(new\, 0))\ then\ \langle 1, let\ t^B = x1\ in\ \Upsilon^B_{t\Downarrow}\rangle\\
&\qquad\ \ else\ \langle 0, let\ t^B = x0\ in\ \Upsilon^B_{t\Downarrow}\rangle\\
\Upsilon^{bit\multimap B}_{x\Uparrow} &= \quad let\langle y^{bit}, z^{\S B}\rangle = x\ in
\end{aligned}$$

$$\begin{aligned}
& \lambda t^{bit}.\mathit{if}\, x \ \mathit{then}\ \ \mathit{if}\, t \ \mathit{then}\ \Upsilon^{B}_{z\Uparrow} \ \mathit{else}\ \Omega \\
& \qquad \mathit{else}\ \ \mathit{if}\, t \ \mathit{then}\ \Omega \ \mathit{else}\ \Upsilon^{B}_{z\Uparrow} \\
\Upsilon^{qbit\multimap B}_{x\Downarrow} = \quad & \mathit{let}\langle u, v\rangle = \mathbf{N}_C\langle \mathbf{H}(\mathit{new}\, 0), \mathit{new}\, 0\rangle \ \mathit{in}\ \langle u, \mathit{let}\ t^B = xv \ \mathit{in}\ \Upsilon^{B}_{t\Downarrow}\rangle \\
\Upsilon^{qbit\multimap B}_{x\Uparrow} = \quad & \mathit{let}\langle y^{qbit}, z^{\S B}\rangle = x \ \mathit{in} \\
& \lambda t^{qbit}.\ \mathit{let}\ \langle y', t'\rangle = \mathbf{N}_C\langle y, t\rangle \ \mathit{in} \\
& \qquad \mathit{let}\ \langle y'', t''\rangle = \langle \mathbf{H} y', t'\rangle \ \mathit{in} \\
& \qquad\quad \mathit{if}\, \mathit{meas}\, y'' \ \mathit{then}\ \Omega \ \mathit{else}\ \Upsilon^{B}_{z\Uparrow}
\end{aligned}$$

Then, when A is of the form $C \otimes D$, we proceed the following way:

$$\begin{aligned}
\Upsilon^{(C\otimes D)\multimap B}_{x\Downarrow} = \quad & \mathit{let}\ f^{C\multimap(D\multimap B)} = \lambda t^C.\lambda u^D.(x\langle t, u\rangle) \ \mathit{in} \\
& \mathit{let}\ \ \langle u^{\S C}, v^{\S D\otimes \S B}\rangle = \Upsilon^{C\multimap(D\multimap B)}_{f\Downarrow} \ \mathit{in} \\
& \mathit{let}\ \ \langle y^{\S D}, z^{\S B}\rangle = v \ \mathit{in}\ \langle\langle u, y\rangle, z\rangle \\
\Upsilon^{(C\otimes D)\multimap B}_{x\Uparrow} = \quad & \lambda t^{C\otimes D}.\ \mathit{let}\ \langle y^C, z^D\rangle = t \ \mathit{in}\ \ \mathit{let}\ \langle u^{\S C\otimes \S D}, v^{\S B}\rangle = x \ \mathit{in} \\
& \qquad \mathit{let}\ \langle w^{\S C}, s^{\S D}\rangle = x \ \mathit{in} \\
& \qquad \mathit{let}\ a = \langle w, \langle s, v\rangle\rangle \ \mathit{in}\ ((\Upsilon^{C\multimap(D\multimap B)}_{a\Uparrow})y)z
\end{aligned}$$

Finally, when A is of the form $C \multimap D$, we reduce $C \multimap D$ to $\S C \otimes \S D$ to fall back on the previous case. The details are as follows:

$$\begin{aligned}
\Upsilon^{(C\multimap D)\multimap B}_{x\Downarrow} = \quad & \mathit{let}\ f^{(\S C\otimes \S D)\multimap B} = \lambda t^{\S C\otimes \S D}.x(\Upsilon^{C\multimap D}_{t\Uparrow}) \ \mathit{in}\ \Upsilon^{(\S C\otimes \S D)\multimap B}_{f\Downarrow} \\
\Upsilon^{(C\multimap D)\multimap B}_{x\Uparrow} = \quad & \lambda t^{C\multimap D}.\Upsilon^{(\S C\otimes \S D)\multimap B}_{x\Uparrow}\Upsilon^{C\multimap D}_{t\Downarrow}
\end{aligned}$$

Lemma 7.3.20. *For every types A, the typing judgements $x : A \rhd \Upsilon^{A}_{x\Downarrow} : \S A$ and $x : \S A \rhd \Upsilon^{A}_{x\Uparrow} : A$ are valid. Moreover, there exist constants $\lambda, \lambda' > 0$, such that $[\![\Upsilon^{A}_{x\Downarrow}]\!] = \lambda id_{[\![A]\!]}$ and $[\![\Upsilon^{A}_{x\Uparrow}]\!] = \lambda' id_{[\![A]\!]}$.*

Proof. The proof that the typing judgements are valid is an easy induction on the §-size of A.

The proof that their denotation is a multiple of the identity is also done by induction on the §-size of A. We provide the computation for the difficult cases, namely when the type is $bit \multimap B$ and $qbit \multimap B$.

Case $\Upsilon^{bit\multimap B}_{x\Uparrow}$. We are considering

$$f = [\![x : bit \otimes \S B \rhd \Upsilon^{bit\multimap B}_{x\Uparrow} : bit \multimap B]\!] : [\![B]\!] \oplus [\![B]\!] \to [\![B]\!] \oplus [\![B]\!].$$

An element $(a, b) \in [\![\S B]\!] \oplus [\![\S B]\!]$ corresponds to a probability a to get the boolean 0 and a probability b to get the boolean 1 for the *bit*. This means that $f(a, b) = ([\![\Upsilon^{B}_{z\Uparrow}]\!](a), [\![\Upsilon^{B}_{z\Uparrow}]\!](b))$. By induction hypothesis, $[\![\Upsilon^{B}_{z\Uparrow}]\!] = \lambda id$ for some $\lambda > 0$. Thus we have then $f = \lambda id$.

Case $\Upsilon^{bit\multimap B}_{x\Downarrow}$. We are considering

$$f = [\![x : bit \multimap B \rhd \Upsilon^{bit\multimap B}_{x\Downarrow} : qbit \otimes \S B]\!] : [\![B]\!] \oplus [\![B]\!] \to [\![B]\!] \oplus [\![B]\!].$$

A function $g \in V_{[\![bit\multimap B]\!]}$ is sent to $\frac{1}{2}([\![\Upsilon^{B}_{z\Downarrow}]\!](g(1, 0)), [\![\Upsilon^{B}_{z\Downarrow}]\!](g(0, 1)))$, where we identify the function g and its representation. By induction hypothesis, $[\![\Upsilon^{B}_{z\Downarrow}]\!] = \lambda id$ for some $\lambda > 0$. Thus $g \in V_{[\![bit\multimap B]\!]}$, represented by $(g(1, 0), g(0, 1))$, is sent to $\frac{1}{2}\lambda g$. This means that $f = \frac{1}{2}\lambda id$.

Case $\Upsilon^{qbit \multimap B}_{x\Uparrow}$. We are considering

$$f = [\![x : qbit \otimes \S B \rhd \Upsilon^{qbit \multimap B}_{x\Uparrow} : bit \multimap B]\!] : 2 \otimes [\![B]\!] \to 2 \otimes [\![B]\!].$$

We interpret the term as in the definition. Suppose that $x = \langle y, z\rangle$ and t are respectively the matrix of blocks and the quantum bits

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}, \qquad \begin{pmatrix} a & b \\ c & d \end{pmatrix},$$

where the blocks A, B, C and D belongs to $V_{[\![B]\!]}$. Then the products $\langle y, t, z\rangle$ and $\langle y', t', z\rangle$ are respectively

$$\begin{pmatrix} aA & bA & aB & bB \\ cA & dA & cB & dB \\ aC & bC & aD & bD \\ cC & dC & cD & dD \end{pmatrix}, \qquad \begin{pmatrix} aA & bA & bB & aB \\ cA & dA & dB & cB \\ cC & dC & dD & cD \\ aC & bC & bD & aD \end{pmatrix}.$$

Applying $\mathbf{H}$ on y' yields a matrix with first entry

$$\frac{1}{2}\lambda(aA + bB + cC + dD),$$

if $[\![\Upsilon^{B}_{z\Uparrow}]\!] = \lambda id$, corresponding to the probability for y'' to be $|0\rangle$. Thus,

$$f \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} a & b \\ c & d \end{pmatrix} = \frac{1}{2}\lambda(aA + bB + cC + dD),$$

meaning that x maps to a multiple of the function $qbit \to [\![B]\!]$ of same denotation. That is, $f = \frac{1}{2}\lambda id$.

Case $\Upsilon^{qbit \multimap B}_{x\Downarrow}$. We are considering

$$f = [\![x : qbit \multimap B \rhd \Upsilon^{qbit \multimap B}_{x\Downarrow} : bit \otimes \S B]\!] : 2 \otimes [\![B]\!] \to 2 \otimes [\![B]\!].$$

In this case, the function $f(x)$ creates a pair of two quantum bits in state

$$\begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix},$$

and applies (a multiple of) the function x on each 2×2 block: it is precisely (a multiple of) the characteristic function of x: the function f is indeed a multiple of the identity. □

Theorem 7.3.21 (Fullness up to scalar multiple)**.** *For any types A and B, there exists a constant $\lambda > 0$ such that if $F : [\![A]\!] \to [\![B]\!]$ is any superoperator, then there is a valid typing judgement $x : A \rhd M : B$ of denotation λF.*

Proof. For the given types A and B, consider the terms $\Upsilon^{A}_{a\Downarrow}$ and $\Upsilon^{B}_{b\Uparrow}$. From Lemma 7.3.20, they have for denotation respectively λid and $\lambda' id$, for some λ, λ'. We claim that any superoperator $[\![A]\!] \to [\![B]\!]$ can be interpreted modulo a factor of $\lambda\lambda'$.

Indeed, consider a superoperator $F : [\![A]\!] \to [\![B]\!]$. From Theorem 7.3.13, there exists a valid typing judgement $y : \S A \rhd N : \S B$ of denotation F. Let $a : A \rhd M : B$ be the term

$$a : A \rhd let\ y = \Upsilon^{A}_{a\Downarrow}\ in\ let\ b = N\ in\ \Upsilon^{B}_{b\Uparrow} : B.$$

It has denotation $\lambda\lambda' F$. □

(β)	Γ	$\rhd (\lambda x.M)N$	$\approx_{ax}$	$M[N/x] : A$
(η)	Γ	$\rhd \lambda x.Mx$	$\approx_{ax}$	$M : A \multimap B$
$(\beta_\otimes)$	Γ	$\rhd \mathit{let}\ \langle x, y\rangle = \langle N, P\rangle\ \mathit{in}\ M$	$\approx_{ax}$	$M[N/x, P/y] : A$
$(\eta_\otimes)$	Γ	$\rhd \mathit{let}\ \langle x, y\rangle = M\ \mathit{in}\ \langle x, y\rangle$	$\approx_{ax}$	$M : A \otimes B$
(β_*)	Γ	$\rhd \mathit{let}\ * = *\ \mathit{in}\ M$	$\approx_{ax}$	$M : A$
(η_*)	Γ	$\rhd \mathit{let}\ * = M\ \mathit{in}\ *$	$\approx_{ax}$	$M : \top$
(β^1_{if})	Γ	$\rhd \mathit{if}\, 1\ \mathit{then}\ M\ \mathit{else}\ N$	$\approx_{ax}$	$M : A$
(β^0_{if})	Γ	$\rhd \mathit{if}\, 0\ \mathit{then}\ M\ \mathit{else}\ N$	$\approx_{ax}$	$N : A$
(Ω)	Γ	$\rhd M[\Omega/x]$	$\approx_{ax}$	$\Omega : A$
(η_{if})	Γ	$\rhd \mathit{if}\, B\ \mathit{then}\ M[1/x]\ \mathit{else}\ M[0/x]$	$\approx_{ax}$	$M[B/x] : A$
(id)	Γ	$\rhd \mathit{meas}(\mathit{new}\ M)$	$\approx_{ax}$	$M : \mathit{bit}$

Table 7.4: Axiomatic equivalence

7.4 Equivalence Classes of Terms

Being able to build terms, we need some tools to compare them. One can compare them through syntactic manipulations, or one can have a finer approach using the two semantics we have built: in the case of the operational semantics, the *behavior* of the terms is what defines the equivalence, and in the case of the denotational semantics, the equivalence is expressed by the *denotation* of the terms.

7.4.1 Axiomatic Equivalence

A first notion of equality of terms can be defined by a set of syntactic rules. This is known as the axiomatic equivalence.

Definition 7.4.1. We define an equivalence relation $\approx_{ax}$ on typing judgements. We write the relation as $\Gamma \rhd M \approx_{ax} N : A$, and we define it to be the smallest relation satisfying the rules in Table 7.4, the alpha-equivalence and one congruence rule (ξ) per term constructor. This means for example:

$$\frac{\Gamma \rhd M \approx_{ax} M' : A \multimap B \quad \Delta \rhd N \approx_{ax} N' : A}{\Gamma, \Delta \rhd MN \approx_{ax} M'N' : B}\ (\xi_{app})$$

$$\frac{\Gamma, x : A \rhd M \approx_{ax} M' : B}{\Gamma \rhd \lambda x.M \approx_{ax} \lambda x.M' : A \multimap B}\ (\xi_\lambda)$$

We call it *the axiomatic equivalence relation*. We extend it to quantum closures in the following way: if $\Delta \vDash [\ Q, L, M\ ] : A$ and $\Delta \vDash [\ Q, L, M'\ ] : A$ are two well-typed quantum closures where $FV(M) \setminus |\Delta| = FV(M') \setminus |\Delta| = \{x_1, \ldots, x_n\}$, then we write $\Delta \vDash [\ Q, L, M\ ] \approx_{ax} [\ Q, L, M'\ ] : A$ when

$$\Delta, x_1 : \mathit{qbit}, \ldots, x_n : \mathit{qbit} \rhd M \approx_{ax} M' : A.$$

Lemma 7.4.2. *The order of the arguments in an application does not matter. Similarly, one can apply the arguments as a pairing or sequentially. More precisely:*

$$\begin{aligned} [\ Q, L, ((\lambda xy.M)N)P\ ] &\approx_{ax} [\ Q, L, ((\lambda yx.M)P)N\ ] \\ [\ Q, L, \mathit{let}\ \langle x, y\rangle = \langle N, P\rangle\ \mathit{in}\ M\ ] &\approx_{ax} [\ Q, L, ((\lambda xy.M)N)P\ ] \end{aligned}$$

Proof. By application of rule (β) and rule $(\beta_\otimes)$. □

7.4.2 Operational Context

To say that two arbitrary terms have the same behavior, we need a way to *observe* them. The only *observable* types at our disposal are the types *bit* and $\top$. So the fact that two terms M and M' have the same behavior can be understood as the fact that in whichever context $C[-]$ we "use" them, if $C[-] : bit$, then $C[M]$ reduces to 0, respectively, 1, with the same probability as $C[M']$. Such a term $C[-]$ is called an *operational context*.

Definition 7.4.3. We define a *formal operational context* to be a formula defined by the following BNF:

$$\begin{array}{rcl} C[-] & ::= & [-] \mid (C[-]M) \mid (MC[-]) \mid \lambda x.C[-] \mid \langle C[-], M\rangle \mid \langle M, C[-]\rangle \mid \\ & & \mathit{let}\ \langle x, y\rangle = C[-]\ \mathit{in}\ M \mid \mathit{let}\ \langle x, y\rangle = M\ \mathit{in}\ C[-] \mid \\ & & \mathit{let}\ * = C[-]\ \mathit{in}\ M \mid \mathit{let}\ * = M\ \mathit{in}\ C[-] \mid \\ & & \mathit{if}\, C[-]\ \mathit{then}\ M\ \mathit{else}\ N \mid \mathit{if}\, M\ \mathit{then}\ C[-]\ \mathit{else}\ C'[-]. \end{array}$$

We call $[-]$ the *hole* of the context.

The notions of well-typed contexts and free variables in contexts are defined the same ways as for terms. Note that there exists a new notion: the notion of *captured variables*, which are the variables whose scope includes the hole. We can make this more precise by speaking of typed contexts:

Definition 7.4.4. A *typed operational context* is a typing tree with root $\Gamma' \rhd C[-] : B$, considering the additional axiom $\Gamma \rhd [-] : A$, i.e., a typing tree of the form

$$\begin{array}{c} \underline{\Gamma \rhd [-] : A} \\ \vdots \\ \hline \Gamma' \rhd C[-] : B. \end{array}$$

We say that this context is of type B, with free variables Γ', a hole of type A, and captured variables Γ. We also use the notation $\Gamma' \rhd C[\Gamma \rhd - : A] : B$ for a typed operational context.

Lemma 7.4.5. *If*

$$\begin{array}{c} \underline{\Gamma \rhd [-] : A} \\ \vdots \\ \hline \Gamma' \rhd C[-] : B. \end{array}$$

is a valid typing derivation, then so is

$$\begin{array}{c} \underline{\Delta, \Gamma \rhd [-] : A} \\ \vdots \\ \hline \Delta, \Gamma' \rhd C[-] : B, \end{array}$$

provided the variables that occur in Δ are fresh.

Proof. By easy induction on the typing derivation of $\Gamma' \rhd C[-] : B$. □

7.4.3 Operational Equivalence

We define a notion of operational equivalence, based on the reduction rules and observations of type *bit*, as in Danos and Harmer (2002). (Equivalently, it would suffice to consider observations of type $\top$).

Definition 7.4.6. Let $\rhd C[\Gamma \rhd - : A] : bit$ be a closed typed operational context of type bit, and let $R = [Q, L, M]$ be a well-typed quantum closure with typing judgement $\Gamma \vDash [Q, L, M] : A$. In this case we define the substitution $C[R]$ by $[Q, L, C[M]]$, where M is syntactically replacing $[-]$ in $C[-]$. We linearly extend this definition to probabilistic distributions of quantum closures of the form $\Gamma \vDash \sum_i \rho_i R_i : A$ by setting $C[\sum_i \rho_i R_i] = \sum_i \rho_i C[R_i]$.

Lemma 7.4.7. *In Definition 7.4.6, the substitution is well typed and a valid typing judgement for it is* $\vDash [Q, L, C[M]] : bit$.

Proof. This lemma is a direct consequence of Lemma 7.4.5. □

Definition 7.4.8. Given two well-typed quantum closures $\Gamma \vDash R, R' : A$, we say that R is operationally equivalent to R' with respect to Γ if for all closed typed operational contexts $\rhd C[\Gamma \rhd - : A] : bit$, $C[R] \Downarrow 0 = C[R'] \Downarrow 0$ and $C[R] \Downarrow 1 = C[R'] \Downarrow 1$. In this case, we write $\Gamma \vDash R \approx_{op} R' : A$. If M, M' are terms, we say that $\Gamma \rhd M \approx_{op} M' : A$ if $\Gamma \vDash [\ |\rangle, |\rangle, M\ ] \approx_{op} [\ |\rangle, |\rangle, M'\ ] : A$.

7.4.4 Denotational Equivalence

The last equivalence we can define is the *denotational equivalence*. This equivalence relation is simply stated:

Definition 7.4.9. We say that two typing judgements $\Gamma \rhd M, M' : A$ are *denotationally equivalent* if $[\![\Gamma \rhd M : A]\!]$ and $[\![\Gamma \rhd M' : A]\!]$ are the same map in **CPM**. In that case we write $\Gamma \rhd M \approx_{den} M' : A$. We extend this definition to quantum closures: $\Gamma \vDash R \approx_{den} R' : A$ is true if $[\![\Gamma \vDash R : A]\!] = [\![\Gamma \vDash R' : A]\!]$.

7.5 Soundness and Full Abstraction

The three defined equivalence relations we have built have the expected behavior: The axiomatic equivalence is sound with respect to the operational equivalence and the denotational semantic is fully abstract with respect to the operational semantics:

Theorem 7.5.1 (Soundness)**.** *If* $\Gamma \rhd M \approx_{ax} M' : A$ *then* $\Gamma \rhd M \approx_{op} M' : A$.

Remark 7.5.2. An immediate consequence of soundness and Lemma 7.4.2 is that the order of evaluation does not affect the outcome:

$$\Gamma \rhd ((\lambda xy.R)M)N \approx_{op} ((\lambda yx.R)N)M.$$

Theorem 7.5.3 (Full abstraction)**.** *The denotational semantics is fully abstract with respect to the operational equivalence of typing judgements, i.e.*

$$[\![\Gamma \rhd M : A]\!] = [\![\Gamma \rhd M' : A]\!] \text{ if and only if } \Gamma \rhd M \approx_{op} M' : A.$$

Remark 7.5.4. The presence of the non-terminating term Ω is necessary for full abstraction to hold. Without it, every program terminates with probability 1, and there is only one definable map $bit \to \top$. Thus, although $\lambda f.(f0)$ and $\lambda f.(f1)$ of type $(bit \multimap \top) \multimap \top$ have different denotations, no context will distinguish them.

7.5.1 Proof of the Soundness Theorem

Assume Theorem 7.5.3. It suffices to show that if $\Gamma \rhd M \approx_{ax} M' : A$ then $\Gamma \rhd M \approx_{den} M' : A$. We show this by structural induction on the proof of $\Gamma \rhd M \approx_{ax} M' : A$.

The cases (β), $(\beta_\otimes)$, (β_*), (β^1_{if}) and (β^0_{if}) are similar as in the proof of Lemma 7.3.6.

We detail the remaining cases

(η). If $[\![\Gamma \rhd M : A \multimap B]\!] = \Phi(g) : [\![\Delta]\!] \to [\![A]\!] \otimes [\![B]\!]$, then the denotation $[\![\Gamma, x : A \rhd Mx : B]\!]$ is a map $f : [\![\Gamma]\!] \otimes [\![A]\!] \to [\![B]\!]$, with

$$f(x \otimes y) = g(x \otimes id_A(y)) = g(x \otimes y).$$

$(\eta_\otimes)$. If $f = [\![\Gamma \rhd M : A \otimes B]\!]$, if $g = [\![\Gamma \rhd \mathit{let}\ \langle x, y\rangle = M\ \mathit{in}\ \langle x, y\rangle : A \otimes B]\!]$ and if $a \in [\![\Gamma]\!]$, we have $g(a) = (id_A \otimes id_B)(f(a)) = f(a)$.

(η_*). If $f = [\![\Gamma \rhd M : \top]\!]$, if $g = [\![\Gamma \rhd \mathit{let}\ * = M\ \mathit{in}\ * : \top]\!]$, and if $a \in [\![\Gamma]\!]$, then $g(a)$ is equal to $id_\top(1) \cdot f(a)$, that is, $f(a)$.

(η_{if}). In this case, the context Γ splits into (Γ_1, Γ_2), where $\Gamma_1 \rhd B : \mathit{bit}$ and $\Gamma_2, x : \mathit{bit} \rhd M : A$. If (p, q) is the denotation of the former and f the denotation of the latter, if $a \in [\![\Gamma_1]\!]$ and $b \in [\![\Gamma_2]\!]$, $p(a)f(0, b) + q(a)f(b, 0) = f(0, b \cdot p(a)) + f(b \cdot q(a), 0) = f((p(a), q(a)) \otimes b)$.

(id). This last case is done by noticing that $[\![x : \mathit{bit} \rhd \mathit{meas}(\mathit{new}\, x) : \mathit{bit}]\!]$ is the identity. □

7.5.2 Full Abstraction: Preliminary Lemmas

Lemma 7.5.5. *For any two programs P, P' of type bit, they have the same denotation if and only if for all $b \in \{0, 1\}$, $P \Downarrow b = P' \Downarrow b$.*

Proof. Consider two well-typed programs $P, P' : \mathit{bit}$. Suppose they have the same denotation f. Then from Lemma 7.3.8, f is also the denotation of $\mathit{prob}'_U P$ and of $\mathit{prob}'_U P'$. But by definition, $[\![\mathit{prob}'_U P]\!] = (p, q)$, where $p = (\mathit{prob}'_U P) \Downarrow 0 = P \Downarrow 0$ and $q = (\mathit{prob}'_U P) \Downarrow 1 = P \Downarrow 1$, and similarly for P'. Thus $P \Downarrow 1 = P' \Downarrow 1$ and $P \Downarrow 0 = P' \Downarrow 0$. The argument being reversible, we get the other implication. □

Lemma 7.5.6. *If $[\![\Gamma \rhd M{:}A]\!] = [\![\Gamma \rhd M'{:}A]\!]$ and $C[\Gamma \rhd - : A] : \mathit{bit}$ is a valid context, then $[\![C[M] : \mathit{bit}]\!] = [\![C[M'] : \mathit{bit}]\!]$.*

Proof. The proof uses Lemma 7.3.5. □

Lemma 7.5.7. *Let A be a type and let $v \neq v'$ be hermitian positive elements in $V_{[\![A]\!]}$. Then there exists a valid typing judgement $x : A \rhd M : \mathit{bit}$ such that $[\![M]\!](v) \neq [\![M]\!](v')$.*

Proof. By Lemma 7.3.20, it suffices, without loss of generality, to consider the case where $A = \mathit{bit}^{\otimes n} \otimes \mathit{qbit}^{\otimes m}$. Then $[\![A]\!] = (2^m, \dots 2^m)$, $v = (v_1, \dots v_{2^n})$ and $v' = (v'_1, \dots v'_{2^n})$. If v and v' differ, there exists some i for which $v_i \neq v'_i$. By Theorem 3.1.13 there exists orthonormal bases of eigenvectors corresponding to v_i and v'_i. Since they differ, there exists a vector w such that $w^* v_i w \neq w^* v'_i w$ (otherwise, v_i and v'_i would have the same eigenvectors and eigenvalues, making them equal).

Consider the map

$$\begin{array}{rccc} f: & (2^m, \dots, 2^m) & \longrightarrow & 1, 1 \\ & (w_1, \dots, w_n) & \longmapsto & (w^* w_i w, 0) \end{array}$$

From Theorem 4.3.11, this is a superoperator. By definition, $f(v) \neq f(v')$ and by Theorem 7.3.13, it is representable by some valid typing judgement $x : A \rhd M : \mathit{bit}$. □

Lemma 7.5.8. *Given any type A and any hermitian positive $v \in V_{[\![A]\!]}$, there exists a closed term $M : A$ and $\lambda > 0$ such that $[\![M]\!](1) = \lambda v$.*

Proof. If $v = 0$, let $M = \Omega$. Else, from Corollary 7.3.14, there exists a valid typing judgement $\rhd N : \S A$ such that $v/Tr(v) = [\![\rhd N : A]\!](1)$. Then let $M = \mathit{let}\ x = N\ \mathit{in}\ \Upsilon^{A}_{x\Uparrow} : A$. From Lemma 7.3.20, $[\![\rhd M : A]\!](1)$ and v are collinear. □

7.5.3 Proof of the Full Abstraction Theorem

If $[\![\Gamma \rhd M : A]\!] = [\![\Gamma \rhd M' : A]\!]$, take any valid context $C[\Gamma \rhd - : A] : \mathit{bit}$ for those two terms. Then from Lemma 7.5.6, $[\![\rhd C[M] : \mathit{bit}]\!] = [\![\rhd C[M'] : \mathit{bit}]\!]$. From Lemma 7.5.5, $C[M] \Downarrow b = C[M'] \Downarrow b$, for $b \in \{0, 1\}$. Since this holds for arbitrary contexts, M and M' are operationally equivalent.

The opposite implication follows from Lemma 7.5.7 and Lemma 7.5.8. Consider two typing judgements $\Gamma \rhd M, M'{:}A$ with denotations

$$F{=}[\![\Gamma \rhd M : A]\!] \quad \text{and} \quad G = [\![\Gamma \rhd M' : A]\!],$$

such that $F \neq G$.. Since the vector space $V_{[\![\Gamma]\!]}$ is spanned by hermitian positive elements, there exists a hermitian positive $v \in V_{[\![\Gamma]\!]}$ such that $F(v) \neq G(v)$.

If $\Gamma = x_1{:}A_1, \ldots, x_n{:}A_n$, let $B = A_1 \otimes \ldots \otimes A_n$. By Lemma 7.5.8, there exists a closed term $R : B$ such that $[\![R : B]\!](1) = \lambda v$, for some $\lambda > 0$. By Lemma 7.5.7, there exists a term $x : A \rhd S : \mathit{bit}$ such that $[\![x : A \rhd S : \mathit{bit}]\!](Fv) \neq [\![x : A \rhd S : \mathit{bit}]\!](Gv)$. Now consider $C[\Gamma \rhd - : A] : \mathit{bit}$ defined by

$$\mathit{let}\ \langle x_1, \ldots x_n \rangle = R\ \mathit{in}\ \mathit{let}\ x = [-]\ \mathit{in}\ S.$$

Then $[\![C[M]]\!](1) \neq [\![C[M']]\!](1)$, hence by Lemma 7.5.5, $C[M] \Downarrow b \neq C[M'] \Downarrow b$, for some $b \in \{0, 1\}$. It follows that $M \not\approx_{op} M'$, which completes the proof of full abstraction. □

Chapter 8

Structure of the Linear-Non-Linear Fragment

In the previous chapter, we described a semantics for the linear fragment of the quantum lambda calculus of Chapter 6. Doing so, we omitted the issue of duplicability. This chapter and the following ones are concerned with the study of the linear-non-linear fragment of the lambda calculus described in Chapter 6. An extended abstract of the work presented here has been published in (Selinger and Valiron, 2008).

8.1 Computations and Values

In Chapter 7, we developed an interpretation of computations as completely positive linear maps between specific vectors spaces. Due to the strict linearity, there was no need to distinguish between values and computations within the denotation. In the case of the general quantum lambda calculus, the computations are not the whole story. Due to the probabilistic behavior of the measurement, in order to give a categorical description of the quantum lambda calculus, the analysis of the *values* has to be separated from the analysis of the *computations*. For this purpose we will use the semantics described in Section 5.4.3 and describe a computational model to handle the probabilistic side effect.

8.2 Duplicability Versus Non-Duplicability

While the strong monad handles the probabilistic nature of the computation, it does not deal with the question of the non-duplicability.

Coecke and Pavlovic (2007) offer a solution for duplication at first order using the notion of *classical objects*. The idea is that the boolean structure $(1, 1)$ admits a "copy" operator such as

$$\delta_{bit} : \begin{array}{ccc} 1,1 & \longrightarrow & 1,1,1,1 \\ (a,b) & \longmapsto & (a,0,0,b), \end{array}$$

sending the Boolean 0 to the pair $\langle 0, 0\rangle$ and the Boolean 1 to the pair $\langle 1, 1\rangle$. This approach does not generalize well at higher order. Indeed, consider the set of functions $(\top \multimap bit)$ and the set of pairs of functions $(\top \multimap bit) \otimes (\top \multimap bit)$. Using the denotation in Chapter 7, a function $f : \top \multimap bit$ is represented by a pair (a, b) where a is the probability of outputting the Boolean 0 and b the probability of getting the Boolean 1. Thus, in the denotation, it is the same entity as a boolean. If we define $\delta_{\top \multimap bit}$ as the same "copy" operator, $\delta_{\top \multimap bit}(f)$ becomes the pair of functions that outputs

$\langle 0,0\rangle$ with probability a and $\langle 1,1\rangle$ with probability b. This is not how we expect the duplication of f to behave.

8.2.1 Computations as Proofs

Instead of working at the level of the denotational semantics, we will study the language at a syntactic level and work out the required structure from there, using the powerful Curry-Howard isomorphism of Section 5.2.2.

Our type system has a particular type construct, namely "!", attached to the subtyping relation defined in Definition 6.3.3. As was noted in (Valiron, 2004a), it is possible to rewrite this subtyping relation in the following way:

$$\frac{A <: A' \quad B <: B'}{A' \multimap B <: A \multimap B'}\ (\multimap) \qquad \frac{A <: A' \quad B <: B'}{A \otimes B <: A' \otimes B'}\ (\otimes)$$

$$\frac{}{\alpha <: \alpha}\ (ax) \qquad \frac{A <: B}{!A <: B}\ (D) \qquad \frac{!A <: B}{!A <: !B}\ (!)$$

This formulation of the subtyping relation guides us to the theory of linear logic.

8.3 Structure of the Exponential "!"

The model of linear logic offered by Bierman (1993) and described in Section 5.6.2 is almost a perfect fit. However, it is slightly too general for our purpose. In this section we describe the modifications that are needed.

8.3.1 Idempotency

Indeed, the "!" operator of the type system of the quantum lambda calculus is idempotent, in the sense that $x : !A \rhd x : !!A$ and $x : !!A \rhd x : !A$ are inverse one of each other. This is not reflected in the definition of a generic comonad, where the map

$$LLA \xrightarrow{L\epsilon_A} LA \xrightarrow{\delta_A} LLA$$

is not equal to the identity.

Definition 8.3.1 (Taylor (1999))**.** A comonad (L, ϵ, δ) on some category is said to be *idempotent* if $\delta : L \dot{\to} LL$ is an isomorphism.

Lemma 8.3.2. *If (L, ϵ, δ) is an idempotent comonad, then $\delta L = L\delta$, $L\epsilon = \epsilon L$ and $\delta^{-1} = L\epsilon$.*

Proof. Equation (2.6.1) states that $\delta; L\delta = \delta; \delta L$. Multiplying on the left by δ^{-1}, we get $L\delta = \delta L$. Equation (2.6.2) says $\delta; \epsilon L = \delta; L\epsilon$. Multiplying on the left by δ^{-1}, we get $\epsilon L = L\epsilon$. Finally, since $\delta; L\epsilon = id$, again multiplying on the left by δ^{-1} we get the last required equation: $L\epsilon = \delta^{-1}$. □

Lemma 8.3.3. *Suppose that $(L, \epsilon, \delta, d^L, d^L)$ is an idempotent monoidal comonad. Then $\epsilon_\top; d^L_\top = id_{L\top}$.*

Proof. Consider the following diagram:

$$\begin{tikzcd}
L\top \arrow[dr, "Ld^L_\top"] \arrow[ddr, bend left, "Lid_\top = id_{L\top}"] \arrow[dd, "\epsilon_\top"'] & & \\
& L^2\top \arrow[dr, "\epsilon_{L\top} = L\epsilon_\top"] & \\
\top \arrow[rr, "d^L_\top"'] & & L\top.
\end{tikzcd}$$

The equality is obtained from Lemma 8.3.2, the lower triangle by naturality of ϵ and the upper triangle by functoriality of L on the monoidality of ϵ (Equation (2.8.8)). □

8.3.2 Coherence Property for Idempotent Comonads

We want to be able to define a map $[\![A]\!] \to [\![B]\!]$, image of the relation $A <: B$. Since by Lemma 6.3.4 $(<:)$ is an ordering relation, we would like to have a similar notion on the images. The category $\mathcal{C}$ has an idempotent comonad. In the following we will show a coherence property for the comonad, and use it to define the image of $A <: B$.

Definition 8.3.4. Consider a category $\mathcal{C}$ with an idempotent comonad (L, δ, ϵ). Let $(F_1, \dots F_n)$ be a list of functors $F_i : \mathcal{C}^{m_i} \times (\mathcal{C}^{op})^{l_i} \to \mathcal{C}$. Suppose $\mathbb{A}$ is the set of elements of the form

$$A, B ::= a \mid F_i(A_1, \dots, A_{m_i}, B_1, \dots, B_{l_i}) \mid LA,$$

where a spans over a given alphabet $\mathcal{A}$. Given a map $G : \mathcal{A} \to |\mathcal{C}|$, we build the $\mathbb{A}$-*G-category* $\mathcal{C}^G$ a follows:

- The set of objects is $\mathbb{A}$.
- Arrows are arrows of $\mathcal{C}$, defined by induction as follows:
  - for all objects A in $\mathcal{C}^G$, the arrows $id_{G(A)}$, $\epsilon_{G(A)}$ and $\delta_{G(A)}$ are arrows of $\mathcal{C}^G$.
  - If $f : A \to B$ is an arrow in $\mathcal{C}^G$ then so are $Lf : LA \xrightarrow{L(f)} LB$,

$$\begin{aligned}
F_i(X_1, \dots f \dots, X_{l_i}, Y_1, \dots Y_{m_i}) = \\
F_i(G(X_1), \dots f \dots, G(X_{l_i}), G(Y_1), \dots G(Y_{m_i})), \\
F_i(X_1, \dots, X_{l_i}, Y_1, \dots f \dots, Y_{m_i}) = \\
F_i(G(X_1), \dots, G(X_{l_i}), G(Y_1), \dots f \dots, G(Y_{m_i})).
\end{aligned}$$

  Finally, if $f : A \to B$ and $g : B \to C$ are arrows of $\mathcal{C}^G$, $(f; g) : A \to C$ is also an arrow of $\mathcal{C}^G$.

Theorem 8.3.5. *Consider a category $\mathcal{C}$, an alphabet $\mathcal{A}$ and a list of functors $(F_1, \dots, F_n)$ as in definition 8.3.4. If $f : A \to B$ and $g : A \to B$ are two maps of $\mathcal{C}^G$, they are equal.*

Following Kelly's methodology Kelly and Laplaza (1980), we define the free category verifying the conditions, and show that this free category is in fact a poset. Without loss of generality, we consider only one F_i, that we name $\ltimes : \mathcal{C}^{op} \times \mathcal{C} \to \mathcal{C}$.

Definition 8.3.6. Given an alphabet $\mathbb{A}$, we define the graph $\mathcal{G}$ as follows: the vertices of $\mathcal{G}$ are

$$\textit{Vertices } A, B ::= a \mid (A \ltimes B) \mid LA,$$

where a ranges over $\mathbb{A}$, and the arrows are two-fold. First, for each vertex A there exists arrows $\delta_A : LA \to LLA$ and $\epsilon_A : LA \to A$. Then for each arrow $f : A \to B$ all expansions of f with respect to $(L, \ltimes)$ are arrows in the graph.

Then we define $\mathcal{K}$ to be the free category generated by $\mathcal{G}$. We write id_A for the identity on A. We call the arrows id_A, ϵ_A and δ_A *elementary arrows*.

We define a set of relations $\mathcal{R}$. This set contains first (2.6.1) and (2.6.2), the equations of functoriality of L and of naturality of ϵ and δ, so that (L, δ, ϵ) is a comonad. It contains the equations for the bifunctoriality of $\ltimes$. We also request equations $L\epsilon_A; \delta_A = id_{L^2A}$ and $\epsilon_{LA} = L\epsilon_A$. Furthermore, we request all expansions of the above relations with respect to $(L, \ltimes)$ to belong to $\mathcal{R}$.

Finally, we define the category $\mathcal{D}$ to be $\mathcal{K}/\mathcal{R}$.

Lemma 8.3.7. *In $\mathcal{D}$, $\ltimes$ is a bifunctor and (L, δ, ϵ) is an idempotent comonad.*

Proof. All the requested equations are there. □

Definition 8.3.8. We define a map $K : |\mathcal{D}| \to |\mathcal{D}|$ by induction

$$\begin{array}{cc} K(a) = a & K(L^2A) = K(LA) \\ K(La) = La & K(A \ltimes B) = (KA) \ltimes (KB) \end{array}$$
$$K(L(A \ltimes B)) = L((KA) \ltimes (KB))$$

and a set of maps $\zeta_A : A \to KA$, one for each object A:

$$\zeta_a = id_a : a \to a, \quad \zeta_{La} = id_{La} : La \to La, \tag{8.3.1}$$

$$\begin{aligned} \zeta_{L^2A} &= L^2A \xrightarrow{\delta_A^{-1}} LA \xrightarrow{\zeta_{LA}} K(LA), \\ \zeta_{A \ltimes B} &= (KA) \ltimes B \xrightarrow{\zeta_A \ltimes \zeta_B} A \ltimes (KB). \end{aligned} \tag{8.3.2}$$

We define subsets *LSubVert* and *SubVert* of vertices:

$$\begin{array}{llll} \textit{LSubVert} & U & ::= & a \mid A \ltimes B, \\ \textit{SubVert} & A, B & ::= & U \mid LU. \end{array}$$

An element in *SubVert* is called a *subvertex*, and an element of *LSubVert* a *linear subvertex*. We define a partial ordering relation $\leqslant$ on subvertices (where $A, B \in \textit{SubVert}$ and $U, V \in \textit{LSubVert}$):

$$\frac{}{a \leqslant a} \quad \frac{U \leqslant V}{LU \leqslant V} \quad \frac{U \leqslant V}{LU \leqslant LV} \quad \frac{A \leqslant A' \quad B \leqslant B'}{A' \ltimes B \leqslant A \ltimes B'}$$

and a map $m_{A,B}$ for each $A \leqslant B$:

$$\frac{U \xrightarrow{m_{U,V}} V}{m_{LU,V} = LU \xrightarrow{\epsilon_U; m_{U,V}} V} \quad \frac{U \xrightarrow{m_{U,V}} V}{m_{LU,LV} = LU \xrightarrow{Lm_{U,V}} LV}$$

$$\frac{}{m_a = a \xrightarrow{id_a} a} \quad \frac{A \xrightarrow{m_{A,A'}} A' \quad B \xrightarrow{m_{B,B'}} B'}{m_{A' \ltimes B, A \ltimes B'} = A' \ltimes B \xrightarrow{m_{A,A'} \ltimes m_{B,B'}} A \ltimes B'}$$

Lemma 8.3.9. *For all A, ζ_A is an isomorphism.*

Proof. Proof by induction on A, knowing that δ_A is an isomorphism. □

Lemma 8.3.10. *For any vertex A, KA is a subvertex.*

Proof. Proof by induction on A. □

Lemma 8.3.11. *For any vertex A, there exists a unique linear subvertex U and a unique $n \geqslant 0$ such that $A = L^n U$. In that case, $KA = U$ if $n = 0$, $KA = LU$ otherwise.*

Proof. Proof by induction on A. □

Lemma 8.3.12. *If U is a subvertex, then $K(LU) = L(KU)$.*

Proof. Proof by case distinction. □

Lemma 8.3.13. *If U is a subvertex, then*

$$LU \xrightarrow{\zeta_{LU}} K(LU) \qquad = \qquad LU \xrightarrow{L\zeta_U} L(KU)$$

Proof. Proof using Lemma 8.3.12 and by case distinction on the possible ζ_{LU}. □

Lemma 8.3.14. *$\leqslant$ is an partial ordering relation on subvertices.*

Proof. For A, B and C subvertices.

- Reflexivity: $A \leqslant A$: proved by structural induction on A.
- Antisymmetry: $A \leqslant B$ and $B \leqslant A$ implies $A = B$: proved by structural induction on A.
- Transitivity: $A \leqslant B$ and $B \leqslant C$ implies $A \leqslant C$: proved by structural induction on A.

□

Lemma 8.3.15. *If $A \leqslant B$ and $B \leqslant C$ are in SubVert,*

$$A \xrightarrow{m_{A,B}} B \xrightarrow{m_{B,C}} C \quad = \quad A \xrightarrow{m_{A,C}} C. \tag{8.3.3}$$

Proof. We prove by induction on the derivation of $A \leqslant C$ that for all B such that $A \leqslant B$ and $B \leqslant C$, Equation (8.3.3) is valid.

- If it is an axiom, then $A = C$ are a or $\top$ and the only for B is to be the same as A and B: we are done.
- Suppose it is true for sub-derivations of $LU \leqslant LW$, U and W being linear subvertices. By case distinction the only B such that $LU \leqslant B$ and $B \leqslant LW$ is $B = LV$, where V is a linear subvertex. Then by functoriality of L and and by induction hypothesis, the result is true.
- Suppose it is true for sub-derivations of $LU \leqslant W$, U and W being linear subvertices. By case distinction the only possible B such that $LU \leqslant B$ and $B \leqslant W$ are $B = V$ or LV, V a linear subvertex.

– If $B = V$: then $m_{LU,V} = LU \xrightarrow{\epsilon_U} U \xrightarrow{m_{U,V}} V$, and $m_{LU,V}; m_{V,W}$ is

$$\begin{tikzcd}
LU \arrow[rr, "m_{LU,V}"] \arrow[dr, "\epsilon_U"'] \arrow[ddrr, bend right, "m_{LU,W}"'] & (a) & V \arrow[ddd, "m_{V,W}"] \\
& U \arrow[ur, "m_{U,V}"] \arrow[dr, "m_{U,W}"] \arrow[d, phantom, "(b)"] & \\
& (c) & \\
& & W
\end{tikzcd}$$

where the (a) and (c) commute by definition, and (b) commutes by induction hypothesis.

– If $B = LV$: then $m_{LU,LV} = LU \xrightarrow{Lm_{U,V}} LV$, $m_{LV,W} = LV \xrightarrow{\epsilon_V} V \xrightarrow{m_{V,W}} W$, and $m_{LU,LV}; m_{LV,W}$ is

$$\begin{tikzcd}
LU \arrow[rr, "Lm_{U,V}"] \arrow[d, "\epsilon_U"] \arrow[ddrr, bend right, "m_{LU,W}"'] & (a) & LV \arrow[dl, "\epsilon_V"'] \arrow[ddd, "m_{LV,W}"] \\
U \arrow[r, "m_{U,V}"] \arrow[drr, bend right, "m_{U,V}"'] & V \arrow[dr, "m_{V,W}"] \arrow[d, phantom, "(b)"] & \\
& (c) & \\
& (d) & W,
\end{tikzcd}$$

where (a) commutes by naturality of ϵ, (b) and (d) by definition and (c) by induction hypothesis.

- If it is true for sub-derivations of $(D'' \ltimes E) \leqslant (D \ltimes E'')$, then by case distinction the only possible B such that $(D'' \ltimes E) \leqslant B$ and $B \leqslant (D \ltimes E'')$ is $B = (D' \ltimes E')$ for some vertices D' and E'. Then $D \leqslant D' \leqslant D''$ and $E \leqslant E' \leqslant E'''$, and by induction hypothesis we get the result.

□

Lemma 8.3.16. *For any subvertex A, $m_{A,A} = id_A$.*

Proof. Proof by induction on A. □

Lemma 8.3.17. *Given any arrow $f : A \to B$, $KA \leqslant KB$.*

Proof. We prove it by induction on the construction of f: first the result is true for any elementary arrow and for the identity ion objects. Then it is true by induction on any expansion of such arrows, using the definition. Finally, from Lemma 8.3.14, $\leqslant$ is transitive and thus it is true for any composition of expansions. □

Lemma 8.3.18. *For any map $g : A \to B$, the following diagram commutes:*

$$\begin{tikzcd}
A \arrow[r, "g"] \arrow[d, "\zeta_A"'] & B \arrow[d, "\zeta_B"] \\
KA \arrow[r, "m_{A,B}"] & KB.
\end{tikzcd}$$

Proof. First note that $m_{A,B}$ is well-defined using Lemma 8.3.17. Then we proceed by case distinctions.

- First, g can be the identity, then it's done.
- It can be an elementary arrow. In the case δ_A, the following diagram has to commute:

$$\begin{tikzcd} LA \arrow[r, "\delta_A"] \arrow[d, "\zeta_{LA}"'] & L^2A \arrow[d, "\zeta_{L^2A}"] \\ K(LA) \arrow[r, "m_{K(LA),K(L^2A)}"'] & K(L^2A). \end{tikzcd}$$

By definition $K(L^2A) = K(LA)$, and from Lemma 8.3.15 $m_{K(LA),K(LA)} = id_{K(LA)}$. Thus diagram becomes:

$$\begin{tikzcd} LA \arrow[rr, "\delta_A"] \arrow[dr, "id"] \arrow[ddd, "\zeta_{LA}"'] & (a) & L^2A \arrow[dl, "\epsilon_{LA}"'] \arrow[ddd, "\zeta_{L^2A}"] \\ & LA \arrow[dr, "\zeta_{LA}"] & (b) \\ (c) & & \\ K(LA) \arrow[rr, "id"] & & K(LA), \end{tikzcd}$$

where (a) commutes by property of comonads, (b) commutes by definition, (c) by vacuity.

- In the case ϵ_A, from Lemma 8.3.11 A can be either a subvertex U or of the form LB. In the case $A = LB$, using the same equality as above, the following diagram has to commute:

$$\begin{tikzcd} L^2B \arrow[r, "\epsilon_{LB}"] \arrow[d, "\zeta_{L^2B}"'] \arrow[dr, phantom, "(a)"] & LB \arrow[dl, "\zeta_{LB}"'] \arrow[d, "\zeta_{LB}"] \arrow[dl, phantom, "(b)"] \\ K(LA) \arrow[r, "id"] & K(LA), \end{tikzcd}$$

and (a) commutes by definition, and (b) by vacuity. In the case $A = LU$, the diagram that needs to commute, using Lemma 8.3.11 and Lemma 8.3.12 is:

$$\begin{tikzcd} LU \arrow[r, "\epsilon_U"] \arrow[d, "\zeta_{LU}"'] & U \arrow[d, "\zeta_U"] \\ L(KU) \arrow[r, "id"] & L(KU), \end{tikzcd}$$

which commutes by definition of ζ_{LU}.

- Then it can be the expansion f of an elementary arrow: we prove this case by induction on such an expansion. The base case has been done, it remains to show the general case. Suppose then that the result is true for some expansion $f : A \to B$. Then we need to show that the diagram commute for LF, for $f \ltimes X$ and for $X \ltimes f$.
- Finally, it can be a composition f of such expansions. By induction on the number of expansions composed, we show that the diagram commutes for f. What remains to be shown is the

iterative case: Suppose $f : A \to B$ is such a composition, and $g : B \to C$ is an expansion, we want the following diagram to commute:

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[rr, bend left, "f;g"] \arrow[d, "\zeta_A"'] & B \arrow[r, "g"] \arrow[d, "\zeta_B"] & C \arrow[d, "\zeta_C"] \\
KA \arrow[r, "m_{KA,KB}"] \arrow[rr, bend right, "m_{KA,KC}"'] & KB \arrow[r, "m_{KB,KC}"] & C.
\end{tikzcd}$$

The upper triangle commutes by definition, the lower one using Lemma 8.3.15, the left square by induction hypothesis and the right square because we just proved the result for expansions.

□

Corollary 8.3.19. *In $\mathcal{D}$, any two arrows $f, g : A \to B$ are equal.*

Proof. From Lemma 8.3.18, they are both equal to $\zeta_A; m_{KA,KB}; \zeta_B^{-1}$. □

Proof of Theorem 8.3.5. We prove the theorem when $(F_1, \ldots F_N)$ consists only of one functor $\ltimes : \mathcal{C}^{op} \times \mathcal{C} \to \mathcal{C}$.

Consider the free category $\mathcal{D}$ with an idempotent comonad described in Definition 8.3.6. There is an inclusion $\mathcal{I} : \mathcal{D} \to \mathcal{C}^G$, and any arrow in $\mathcal{C}^G$ is the image of an arrow in $\mathcal{D}$. Let $f, g : A \to B$ be arrows of $\mathcal{C}^G$. There exists f' and g' in $\mathcal{D}$ such that $\mathcal{I}(f') = f$ and $\mathcal{I}(g') = g$. From Corollary 8.3.19, $f' = g'$. Thus $f = g$. □

8.3.3 Duplicable Pairs and Pairs of Duplicable Elements

Unlike the work of Benton et al. (1993); Bierman (1993), we want to think of the type $!(A \otimes B)$ as a type of pairs of elements of type A and B: we want to use the same operation to access the components as one would use for a pair of type $A \otimes B$, without having to use a dereliction operation.

This immediately raises a concern: consider a pair of elements $\langle x, y\rangle$ of type $!(A \otimes B)$. Are x and y duplicable? In the usual linear logic interpretation, they are not. Having a infinite supply of pair of shoes does not mean one has an infinite supply of right shoes: we cannot discard the left shoes. On the other hand, in our interpretation of "classical" data as residing in "classical" memory and therefore being duplicable, if the string $\langle x, y\rangle$ is duplicable, then so should be the elements x and y. In other words, we want the duplication to "permeate" the pairing.

The choice of such a "permeable" pairing is more or less forced on us by our desire to have no explicit term syntax for structural rules. Consider the following untyped terms, which can be typed if t is of type $!(A \otimes !(B \otimes C))$:

$$let\ \langle x, u\rangle = t\ in\ let\ \langle y, z\rangle = u\ in\ \langle\langle z, y\rangle, x\rangle, \tag{8.3.4}$$

$$let\ \langle x, u\rangle = t\ in\ \langle let\ \langle y, z\rangle = u\ in\ \langle z, y\rangle, x\rangle. \tag{8.3.5}$$

First, we expect these two terms to be axiomatically equal. Term (8.3.5) should be of type $!(!(C \otimes B) \otimes A)$, regardless of the permeability of the pairing: if $\langle y, z\rangle$ is duplicable, so should be $\langle z, y\rangle$. Now, consider the term (8.3.4) with a non-permeable pairing. In the naive type system, u ends up being of type $B \otimes C$, and the variables y and z in the final recombination end up being respectively of type B and C. It is not possible to make $\langle z, y\rangle$ of the duplicable type $!(C \otimes B)$.

We therefore choose a permeable pairing. Is will be reflected, albeit subtly, in the typing rules $(\otimes.I)$ and $(\otimes.E)$ of Table 9.1. To enforce it in the semantics, we will pick a *strong* monoidal comonad to model the operator "!".

8.4 Linear Category for Duplication

We now have enough background to define a candidate for the categorical model of the quantum lambda-calculus. As it was advertised, the structure of the categorical semantics will closely follow the one proposed by Bierman (1993), but with the added twist of a computational monad à la Moggi (1991).

Definition 8.4.1. A *linear category for duplication* is a category $\mathcal{C}$ with the following structure:

- a symmetric monoidal structure $(\otimes, \top, \alpha, \lambda, \rho, \sigma)$;
- an idempotent, strongly monoidal, linear exponential comonad $(L, \delta, \epsilon, d^L, d^L, \Diamond, \triangle)$;
- a strong monad (T, μ, η, t);
- a Kleisli exponential $\multimap$, that is, a natural bijective map of arrows

$$\Phi : \mathcal{C}(A \otimes B, TC) \to \mathcal{C}(A, B \multimap C)$$

  and a bifunctor $\multimap : \mathcal{C}^{op} \times \mathcal{C} \to \mathcal{C}$.

The *computational linear category* is defined as the Kleisli category $\mathcal{C}_T$.

Categories with a combination of a monad and a comonad have been studied by many authors (Barr, 1990; Hyland and Schalk, 2003; Power and Watanabe, 2002). However, none of them are equivalent to linear categories for duplication: either the monad and the comonad commute or the comonad is not idempotent.

Remark 8.4.2. A linear category for duplication gives rise to a double adjunction

$$\begin{tikzcd} \mathcal{C}_L \arrow[r, bend left, "U^L"] \arrow[r, phantom, "\perp"] & \mathcal{C} \arrow[l, bend left, "F^L"] \arrow[r, bend left, "U^T"] \arrow[r, phantom, "\perp"] & \mathcal{C}_T, . \arrow[l, bend left, "F^T"] \end{tikzcd}$$

Here the left adjunction arises from the co-Kleisli category $\mathcal{C}_L$ of the comonad L. It is as in the linear-non-linear models of Benton (1994), and $\mathcal{C}_L$ is a category of classical (non-quantum) values. The right adjunction arises from the Kleisli category $\mathcal{C}_T$ of the computational monad T, as in Moggi (1991). Here $\mathcal{C}_T$ is a category of (effectful) quantum computations.

Chapter 9

A Computational Lambda Calculus for Duplication

In this chapter, we describe a subset of the lambda calculus for quantum computation of Chapter 6. We focus primarily on the type system and language, and not on the structure of the actual "built-in" quantum operations (such as unitary operators and measurements). The language will be a generic call-by-value linear lambda calculus, which is parametric on some primitive operations that are not further explained. It should be understood, however, that the need to support primitive quantum operations motivates particular features of the type system.

9.1 An Indexed Lambda Calculus

We describe a linear typed lambda calculus with higher-order functions and pairs. The language is designed to manipulate both classical data, which is duplicable, and quantum data, which is non-duplicable. For simplicity, we assume the language is strictly linear, and not affine linear as in Chapter 6. This means duplicable values are both copyable and discardable, whereas non-duplicable values must be used once, and only once.

9.1.1 Type System

We define a type system together with a subtyping relation similar as in Section 6.3. The only difference is that type constants are parametrized.

Definition 9.1.1. The set of types is

$$\textit{Type } A, B ::= \alpha \mid (A \multimap B) \mid (A \otimes B) \mid \top \mid !A,$$

where α ranges over type constants. While the remainder of this thesis does not depend on the choice of type constants, in our main application this is intended to include a type *qbit* of quantum bits, and a type *bit* of classical bits. The meaning of the types is the same as in Section 6.3. The type system comes also with the *subtyping relation* of Definition 6.3.3.

Remark 9.1.2. The type system satisfies Lemmas 6.3.4 and 6.3.5.

9.1.2 Terms

The language is a computational language: it is divided into values on the one hand, and general terms, or computations, on the other.

As in Chapter 7, our goal is to find a semantics for the language. The semantics has to be *compositional*, that is, the semantics of a given term has to be derived from the semantics of its subterm. We are focusing on a language close to the one of Chapter 6. In this language, consider the following two typing judgements, where V and W are values:

$$x : A \rhd V : !B, \qquad\qquad y : B \rhd W : C.$$

The typing judgement $x : A \rhd \mathit{let}\ y = V\ \mathit{in}\ W : C$ has two valid typing derivations:

$$\frac{x : A \rhd V : !B \quad y : !B \rhd W : C}{x : A \rhd \mathit{let}\ y = V\ \mathit{in}\ W : C,} \qquad \frac{x : A \rhd V : B \quad y : B \rhd W : C}{x : A \rhd \mathit{let}\ y = V\ \mathit{in}\ W : C.}$$

In order to be able to define a semantics for judgements like $x : A \rhd \mathit{let}\ y = V\ \mathit{in}\ W : C$, we will use *indexed terms*. The purpose of the indexing will be to make sure that each valid typing judgement has a reasonably unique typing tree.

Definition 9.1.3. We define the notion of *core values*, *extended values* and *terms* as follows:

$$\begin{aligned}
&\mathit{CoreValue}\ U, U' ::= x^A \mid c^A \mid *^n \mid \lambda^n x^A.M \mid \langle U, U'\rangle^n,\\
&\mathit{ExtValue}\ V, W \ ::= U \mid \langle V, W\rangle^n \mid \mathit{let}\ \ x^A = V\ \mathit{in}\ W \mid \mathit{let}\ \ \langle x^A, y^B\rangle^n = V\ \mathit{in}\ W \mid\\
&\qquad\qquad\quad \mathit{let}\ * = V\ \mathit{in}\ W,\\
&\mathit{Term}\ M, N \qquad ::= U \mid \langle M, N\rangle^n \mid (MN) \mid \mathit{let}\ \ \langle x^A, y^B\rangle^n = M\ \mathit{in}\ N \mid \mathit{let}\ * = M\ \mathit{in}\ N,
\end{aligned}$$

where n is an integer, c ranges over a set of constant terms, x over a set of term variables and α over a set of constant types. We abbreviate $(\lambda^0 x^A.M)N$ by $\mathit{let}\ x^A = N\ \mathit{in}\ M$, $\lambda^n x^{!^m\top}.\ \mathit{let}\ * = x^\top\ \mathit{in}\ M$ by $\lambda^n *^m.M$ and we omit numerical indexes when they are null.

Core values and extended values are often simply called *values* when the context is clear. General terms are also called *computations*.

Remark 9.1.4. A technicality we introduced is the distinction between *core values* and *extended values*. The core values have the same structure as the values of Section 6.2.4. The need for extended values comes from the fact that we wish to have a destructor for the unit type $\top$ in the category of values, so that we are allowed to write the judgement

$$x : A, y : \top \rhd \mathit{let}\ * = y^\top\ \mathit{in}\ x^A : A.$$

However, as it will be further developed in Remark 9.1.30, this term will cause problem with substitution in certain situations. We are forced to restrict the substitution to core values.

Convention 9.1.5. We use the notations $\boxdot$, $\boxdot$ and $\boxdot$ as place holders for x^A, $*$ and $\langle x^A, y^B\rangle^n$ (not respectively). We use typed version of them when we want to enforce the underlying type. For example, $\boxdot^A$ can stand for x^A, $*$ if $A = \top$, or $\langle x^B, y^C\rangle^n$ if $A = !^n(B \otimes C)$.

We also define a notion of *untyped terms* as terms with no index:

$$\begin{aligned}
\mathit{PureTerm}\ M, N \ ::=\ & x \mid c \mid * \mid \lambda x.M \mid (MN) \mid \langle M, N\rangle \mid\\
& \mathit{let}\ \langle x, y\rangle = M\ \mathit{in}\ N \mid \mathit{let}\ * = M\ \mathit{in}\ N\,.
\end{aligned}$$

The erasure operation $\mathit{Erase} : \mathit{Term} \to \mathit{PureTerm}$ is defined as the operation of removing the types and integers attached to a given indexed term. If $M = \mathit{Erase}(\bar{M})$, we say that $\bar{M}$ is an *indexation of* M. We define notions of free variables and α-equivalence on terms, denoted by $=_\alpha$, in the usual way (see for example Section 6.1).

$$\frac{A <: B}{!\Delta, x : A \rhd x^B : B}\ (ax_1) \qquad \frac{!A_c <: B}{!\Delta \rhd c^B : B}\ (ax_2) \qquad \frac{}{!\Delta \rhd *^n : !^n\top}\ (\top.I)$$

$$\frac{\Delta, x : A \rhd M : B}{\Delta \rhd \lambda^0 x^A.M : A \multimap B}\ (\lambda_1) \qquad \frac{!\Delta, x : A \rhd M : B}{!\Delta \rhd \lambda^{n+1} x^A.M : !^{n+1}(A \multimap B)}\ (\lambda_2)$$

$$\frac{\Gamma_1, !\Delta \rhd M : A \multimap B \quad \Gamma_2, !\Delta \rhd N : A}{\Gamma_1, \Gamma_2, !\Delta \rhd MN : B}\ (app)$$

$$\frac{!\Delta, \Gamma_1 \rhd M : !^n A_1 \quad !\Delta, \Gamma_2 \rhd N : !^n A_2}{!\Delta, \Gamma_1, \Gamma_2 \rhd \langle M, N\rangle^n : !^n(A_1 \otimes A_2)}\ (\otimes.I)$$

$$\frac{!\Delta, \Gamma_1 \rhd M : \top \quad !\Delta, \Gamma_2 \rhd N : A}{!\Delta, \Gamma_1, \Gamma_2 \rhd \mathit{let} * = M \mathit{ in } N : A}\ (\top.E)$$

$$\frac{!\Delta, \Gamma_1 \rhd M : !^n(A_1 \otimes A_2) \quad !\Delta, \Gamma_2, x_1 : !^n A_1, x_2 : !^n A_2 \rhd N : A}{!\Delta, \Gamma_1, \Gamma_2 \rhd \mathit{let}\ \langle x_1^{A_1}, x_2^{A_2}\rangle^n = M \mathit{ in } N : A}\ (\otimes.E)$$

Table 9.1: Typing rules for the linear-non-linear quantum lambda calculus

Lemma 9.1.6. *Suppose that V is a core value (respectively an extended value). Then if M is a term such that $Erase(M) = Erase(V)$ then M is a core value (respectively an extended value).* □

Lemma 9.1.7. *Suppose that M and M' are two terms such that $Erase(M) = Erase(M')$. Then $FV(M) = FV(M') = FV(Erase(M'))$.* □

Definition 9.1.8. The indexation of terms induces a partial map from terms to types, called the *raw type* of M. We write $M : A$ to say that A is the *raw type* of M, or that M is raw-typed, and we define it inductively as follows:

$$\begin{aligned} x^A &: A, & \{M : A \multimap B\}N &: B,\\ c^B &: B, & \langle\{M : !^n A\}, \{N : !^n B\}\rangle^n &: !^n(A \otimes B),\\ *^n &: !^n\top, & \mathit{let}\ \langle x^A, y^B\rangle^n = M \mathit{ in } \{N : C\} &: C,\\ \lambda^n x^A.\{M : B\} &: !^n(A \multimap B), & \mathit{let} * = M \mathit{ in } \{N : C\} &: C. \end{aligned}$$

We use the notation $\{M : A\}$ to say "the term M when it is of type A".

9.1.3 Typing Judgements

Definition 9.1.9. We define the notion of *typing context* of Definition 6.3.6.

A typing judgement is a tuple $\Delta \rhd M : A$, where M is an indexed term, A is a type, and Δ is a typing context. As before, to each constant term c we assign a type $!A_c$. A *valid typing judgement* is a typing judgement that can be derived from the typing rules in Table 9.1.

Definition 9.1.10. We define a special derived rule called (*let*);

$$\frac{!\Delta, \Gamma_1, x : A \rhd M : B \quad !\Delta, \Gamma_2 \rhd N : A}{!\Delta, \Gamma_1, \Gamma_2 \rhd \mathit{let}\ x^A = N \mathit{ in } M : B}\ (\mathit{let}).$$

It is derived using Definition 9.1.3, stating that *let* $x^A = N$ *in* M is the term $(\lambda^0 x^A.M)N$:

$$\frac{\dfrac{!\Delta, \Gamma_1, x : A \rhd M : B}{!\Delta, \Gamma_1 \rhd \lambda^0 x^A.M : B}\ (\lambda_1) \quad !\Delta, \Gamma_2 \rhd N : A}{!\Delta, \Gamma_1, \Gamma_2 \rhd (\lambda^0 x^A.M)N : B}\ (app).$$

Lemma 9.1.11. *Consider a valid typing judgement $\Delta \rhd M : A$. Then $FV(M) \subseteq |\Delta|$.*

Proof. Proof by induction on the size of M.

Case $M \equiv x^B$. The only applicable rule is (ax_1). Then the typing judgement is of the form $!\Delta, x : A \rhd x^B : B$. We thus have $FV(M) = \{x\} \subseteq |\Delta| \cup \{x\} = |\Delta, x : A|$.

Cases $M \equiv c^B$ and $M \equiv *^n$. In both cases $FV(M) = \emptyset$. The result is true by vacuity.

Case $M \equiv \lambda^n x^A.N$. The only applicable rule is (λ_i), with $i = 1$ if $n = 0$, $i = 2$ otherwise. Thus there exists some B such that the typing judgement is of the form $\Delta \rhd \lambda^n x^A.N : !^n(A \multimap B)$ with $\Delta, x : A \rhd N : B$. By induction hypothesis, we have $FV(N) \subseteq |\Delta, x : A| = |\Delta| \cup \{x\}$. By definition of free variables, $FV(\lambda^n x^A.N) = FV(N) \setminus \{x\} \subseteq |\Delta|$.

Case $M \equiv NP$, $\langle N, P\rangle^n$ and let $* = M$ in N. The applicable rules are respectively (app), $(\otimes.I)$ and $(\top.E)$. The context Δ splits into $!\Delta'$, Γ_1, Γ_2 such that $!\Delta', \Gamma_1 \rhd N : B$ and $!\Delta', \Gamma_2 \rhd P : C$ for some types B and C. By induction hypothesis, $FV(N) \subseteq |!\Delta', \Gamma_1|$ and $FV(P) \subseteq |!\Delta', \Gamma_2|$. Since $FV(M) = FV(N) \cup FV(P)$, we have $FV(M) \subseteq |!\Delta'| \cup |\Gamma_1| \cup |\Gamma_1| = |\Delta|$.

Case $M \equiv$ let $\langle x^{A_1}, y^{A_2}\rangle^n = N$ in P. The only applicable rule is $(\otimes.E)$. There exists some type B for which the typing judgement $\Delta \rhd$ *let* $\langle x^{A_1}, y^{A_2}\rangle^n = N \, in \, P : B$ is valid. Moreover, Δ splits into $!\Delta', \Gamma_1, \Gamma_2$ such that $!\Delta', \Gamma_1 \rhd N : !^n(A_1 \otimes A_n)$ and such that $!\Delta', \Gamma_2, x : !^n A_1, y : !^n A_2 \rhd P : B$. By induction hypothesis, $FV(N) \subseteq |!\Delta', \Gamma_1| = |!\Delta'| \cup |\Gamma_1|$ and $FV(P) \subseteq |!\Delta', \Gamma_2, x : !^n A_1, y : !^n A_2| = |!\Delta'| \cup |\Gamma_2| \cup \{x, y\}$. By definition of free variables, $FV(M) = FV(N) \cup (FV(P) \setminus \{x, y\})$. Thus $FV(M) \subseteq |\Delta|$.

And this closes the proof of Lemma 9.1.11. □

Lemma 9.1.12. *Consider a term M and any valid typing judgement $\Delta \rhd M : B$. Then the raw type A of M exists, and the type B is equal to A.*

Proof. Proof by structural induction on M, looking at a typical typing judgement $\Delta \rhd M : A$.

Case $M \equiv x^B$. The only applicable rule is (ax_1). Then the typing judgement is of the form $!\Delta, x : A \rhd x^B : B$. We thus have $M : B$.

Case $M \equiv c^B$. The only applicable rule is (ax_2). Thus the typing judgement is of the form $!\Delta \rhd c^B : B$. We thus have $M : B$.

Case $M \equiv *^n$. The only applicable rule is $(\top.I)$. The type B is of the form $!^n\top$ and the typing judgement is of the form $!\Delta \rhd *^n : !^n\top$. Thus we have $M : B$.

Case $M \equiv \lambda^0 x^A.N$. The only applicable rule is (λ_1). Thus there exists some B such that the typing judgement is of the form $\Delta \rhd \lambda^0 x^A.N : A \multimap B$ with $\Delta, x : A \rhd N : B$. By induction hypothesis, we have $N : B$. Thus $\lambda^0 x^A.N : A \multimap B$.

Case $M \equiv \lambda^{n+1} x^A.N$. The only applicable rule is (λ_2). Then $\Delta = !\Delta'$, and there exists some B such that the typing judgement is of the form $!\Delta' \rhd \lambda^{n_1} x^A.N : !^{n+1}(A \multimap B)$. The judgement is inferred from $!\Delta', x : A \rhd N : B$. By induction hypothesis, we have $N : B$. This makes $\lambda^{n_1} x^A.N : !^{n+1}(A \multimap B)$.

Case $M \equiv NP$. The only applicable rule is (app). The context Δ splits into $!\Delta'$, Γ_1, Γ_2 such that $!\Delta', \Gamma_1 \rhd N : A \multimap B$ and $!\Delta', \Gamma_2 \rhd P : A$. By induction hypothesis, $N : A \multimap B$ and $M : A$. Thus $NP : B$.

Case $M \equiv \langle N, P\rangle^n$. The only applicable rule is $(\otimes.I)$. This means $\Delta \rhd \langle N, P\rangle^n : !^n(A \otimes B)$, and the context Δ splits into $!\Delta', \Gamma_1, \Gamma_2$ such that $!\Delta', \Gamma_1 \rhd N : !^n A$ and such that $!\Delta', \Gamma_2 \rhd P : !^n B$. By induction hypothesis, $N : !^n A$ and $P : !^n B$. Thus $l\langle N, P\rangle^n : !^n(A \otimes B)$

Case $M \equiv let * = N\ in\ P$. The only applicable rule is $(\top.E)$. For some type A, the typing judgement $\Delta \rhd let * = N\ in\ P : A$ is valid and Δ splits into $!\Delta', \Gamma_1, \Gamma_2$ such that $!\Delta', \Gamma_1 \rhd N : \top$ and such that $!\Delta', \Gamma_2 \rhd P : A$. By induction hypothesis, $P : A$. This means $let * = N\ in\ P : A$.

Case $M \equiv let\ \langle x^{A_1}, y^{A_2}\rangle^n = N\ in\ P$. The only applicable rule is $(\otimes.E)$. We have the valid typing judgement $\Delta \rhd let\ \langle x^{A_1}, y^{A_2}\rangle^n = N\ in\ P : B$ for some type B and Δ splits into $!\Delta', \Gamma_1, \Gamma_2$ such that $!\Delta', \Gamma_1 \rhd N : !^n(A_1 \otimes A_n)$ and such that $!\Delta', \Gamma_2, x : !^n A_1, y : !^n A_2 \rhd P : B$. By induction hypothesis, $P : B$. We deduce that $M : B$.

Therefore, regardless of the chosen typing judgement $\Delta \rhd M : A$, A is always equal to the raw type of M. □

Remark 9.1.13. Although the type A in a valid typing judgement $\Delta \rhd M : A$ is only dictated by M, the context Δ cannot be chosen arbitrarily for the judgement to be valid. Consider the term $f^{A \multimap A}(f^{A \multimap A} x^A) : A$. The context $(f : !(A \multimap A), x : A)$ makes a valid typing judgement, but $(f : A \multimap A, x : A)$ does not.

Lemma 9.1.14. *Consider a valid typing judgement* $\Delta \rhd M : B$ *and the subtyping relation* $\Delta' <: \Delta$. *Then* $\Delta' \rhd M : B$ *is valid.*

Proof. Let π be a derivation of $\Delta \rhd M : B$ The proof is done by induction on the size of π.

Case (ax_1). The typing judgement is of the form $!\dot{\Delta}, x : A \rhd x^B : B$ with $A <: B$. We have then $\Delta = (!\dot{\Delta}, x : A)$. Since $\Delta' <: \Delta$, the context Δ' is of the form $(!\dot{\Delta}', x : !A')$, where $!\dot{\Delta}' <: !\dot{\Delta}$ and $A' <: A$. Note that $!\dot{\Delta}'$ is duplicable from Lemma 6.3.5. Since $A <: B$, we have $A' <: B$. Applying rule (ax_1), we have that $!\dot{\Delta}', x : A' \rhd x^B : B$ is valid.

Cases (ax_2) *and* $(\top.I)$. The typing judgements are respectively of the form $!\dot{\Delta} \rhd c^B : B$ and $!\dot{\Delta} \rhd *^n : !^n\top$, where $\Delta = !\dot{\Delta}$. Since $\Delta' <: \Delta$, from Lemma 6.3.5 $\Delta' = !\dot{\Delta}'$. We can therefore apply (ax_2) (respectively $(\top.I)$) and get that $!\dot{\Delta}' \rhd M : B$ is valid.

Case (λ_1). There exists some B such that the typing judgement is of the form $\Delta \rhd \lambda^0 x^A.N : A \multimap B$ with $\Delta, x : A \rhd N : B$. Since $(\Delta', x : A) <: (\Delta, x : A)$, by induction hypothesis, we have $\Delta', x : A \rhd N : B$. Applying (λ_1), we get that $\Delta \rhd \lambda^0 x^A.N : A \multimap B$ is valid.

Case (λ_2). We have $\Delta = !\dot{\Delta}$, and there exists some B such that the typing judgement is of the form $!\dot{\Delta} \rhd \lambda^{n_1} x^A.N : !^{n+1}(A \multimap B)$. The judgement is inferred from $!\dot{\Delta}, x : A \rhd N : B$. Since $\Delta' <: !\dot{\Delta}$, from Lemma 6.3.5 $\Delta' = !\dot{\Delta}'$. We have therefore $(!\dot{\Delta}', x : A) <: (!\dot{\Delta}, x : A)$. By induction hypothesis, we have $!\dot{\Delta}', x : A \rhd N : B$. Finally, we can apply the typing rule (λ_2) in order to get that $!\dot{\Delta}' \rhd \lambda^{n_1} x^A.N : !^{n+1}(A \multimap B)$ is valid.

Cases (app), $(\otimes_I)$, $(\top.E)$ *and* $(\otimes.E)$. The term M is respectively of the form

$$M \equiv NP, \qquad M \equiv \langle N, P\rangle^n,$$
$$M \equiv let * = N\ in\ P, \qquad M \equiv let\ \langle x^{D_1}, y^{D_2}\rangle^n = N\ in\ P.$$

The typing context Δ splits into $(!\dot{\Delta}, \Gamma_1, \Gamma_2)$ such that $!\dot{\Delta}, \Gamma_1 \rhd N : B_1$ and such that $!\dot{\Delta}, \Gamma_2, \Lambda \rhd P : B_2$, for some types B_1, B_2 and a context Λ that is empty for the first three cases and equal to $(x : D_1, y : D_2)$ in the last case.

Since $\Delta' <: \Delta$, the context Δ' splits into $(!\dot{\Delta}', \Gamma_1', \Gamma_2')$, where $!\dot{\Delta}' <: !\dot{\Delta}$, $\Gamma_1' <: \Gamma_1$ and $\Gamma_2' <: \Gamma_2$. By induction hypothesis, in each of the four cases, $!\dot{\Delta}', \Gamma_1' \rhd N : B_1$ and $!\dot{\Delta}', \Gamma_2', \Lambda \rhd P : B_2$ are valid. Applying the typing rule corresponding to the case considered, we get that $\Delta' \rhd M : B$ is valid.

Thus, by induction the lemma is valid. □

Lemma 9.1.15. *Let $\Gamma \rhd V : !A$ be a valid typing judgement with V a core value. Then $\Gamma = !\Delta$ for some context Δ.*

Proof. The proof is done by induction on the structure of V. Base cases:

Case $V = x^B$. The only typing rule available for it is (ax_1). The type B is therefore of the form $!B'$. The typing judgement is then of the form $!\Delta, x : A \rhd x^{!B'} : !B'$, with $A <: !B$. By Lemma 6.3.5, the type A is of the form $!A'$, and the context Γ is of the requested form.

Case $V = c^B$. Here, the only typing rule available for it is (ax_2). The typing judgement is therefore of the form $!\Delta \rhd c^B : B$: the lemma is true in this case.

Case $V = *^m$. The only typing rule available for it is $(\top.I)$. The typing judgement is therefore of the form $!\Delta \rhd *^{n+1} : !^{n+1}\top$: as in the previous case, the lemma is true.

Case $V = \lambda^m x^B.M$. There are two available typing rules for validating $\Gamma \rhd \lambda^m x^B.M : !A$, but only one, that is (λ_2), for a duplicable lambda abstraction. This means $m = n + 1$, and $!A = !^{n+1}(B \multimap C)$. The fact that Γ is of the form $!\Delta$ is one of the requirement of this typing rule.

The only inductive case is the following.

Case $V = \langle V_1, V_2\rangle^m$. The only available typing rule is $(\otimes.I)$, stating that the type $!A$ is of the form $!^{n+1}(B \otimes C)$, that $m = n + 1$ and that $\Gamma = !\Delta, \Gamma_1, \Gamma_2$, with $!\Delta \rhd V_1 : !^{n+1}B$ and $!\Delta, \Gamma_2 \rhd V_2 : !^{n+1}C$ being valid typing judgements. Since V_1 and V_2 are core values, the induction hypothesis applies, and Γ_1 is of the form $!\Gamma_1'$ and Γ_2 is of the form $!\Gamma_2'$. This makes Γ of the requested form.

Hence, by structural induction on V, Lemma 9.1.15 is valid. □

Lemma 9.1.16. *Consider the following valid typing judgement: $\Delta, x : A \rhd M : B$. Then for every free instance $x^{A'}$ in M, $A <: A'$.*

Proof. Proof by induction on the size of the derivation of $\Delta, x : A \rhd M : B$.

Case (ax_1). The typing judgement is of the form $!\Delta, y : A \rhd y^B : B$, with $A <: B$. If $y = x$, then the result is true since x^B is the only free variable. If $y \neq x$, there is no instance of x, so the result is true by vacuity.

Case (ax_2). The typing judgement is of the form $!\Delta \rhd c^B : B$: there is no free variable, the result is true by vacuity.

Cases (λ_1) *and* (λ_2). The typing judgement is of the form $\Delta, x : A \rhd \lambda^n y^B.M : !^n(B \multimap C)$, with $\Delta, x : A, y : B \rhd M : C$ and $y \neq x$. By induction hypothesis, for every free instance $x^{A'}$ of M one has $A <: A'$. Thus it is also true for every free instance $x^{A'}$ of $\lambda^n y^B.M$.

Cases (app), $(\otimes.I)$, $(\top.E)$ *and* $(\otimes.E)$. The term M is respectively of the form

$$M \equiv NP, \qquad M \equiv \langle N, P\rangle^n,$$
$$M \equiv let\ * = N\ in\ P, \qquad M \equiv let\ \langle y^{D_1}, z^{D_2}\rangle^n = N\ in\ P.$$

The typing context $\Delta, x : A$ splits into $!\dot{\Delta}, \Gamma_1, \Gamma_2$ such that $!\dot{\Delta}, \Gamma_1 \rhd N : B_1$ and such that $!\dot{\Delta}, \Gamma_2, \Lambda \rhd P : B_2$, for some types B_1, B_2 that depend on the rule used and for some context Λ, empty in the three first cases and equal to $(y : D_1, z : D_2)$ in the last case. Using α-equivalence one can assume that x is different from y and z. There are three cases, depending on where $x : A$ is to be found.

Found in Γ_1. Then there is no free instance of x in P, and by induction hypothesis every free instance $x^{A'}$ in N are such that $A <: A'$. Then for all free instances $x^{A'}$ in M, $A <: A'$.

Found in Γ_2. Then there is no free instance of x in N, and by induction hypothesis every free instance $x^{A'}$ in P are such that $A <: A'$. Then for all free instances $x^{A'}$ in M, $A <: A'$.

Found in $!\Delta$. Applying induction hypothesis on both typing judgements, we get that or all free instances $x^{A'}$ in N and in P, thus in M, $A <: A'$.

Thus by induction, the lemma is valid. □

Lemma 9.1.17. *Consider a valid typing judgement $\Delta \rhd M : A$, where M is of the form $\langle N, P\rangle^n$, NP, let $* = N$ in P or let $\langle y^C, z^D\rangle^n = N$ in P. If $x : B$ $(x \neq y, z)$ is in Δ such that B is not of the form $!B'$ then x cannot be free both in N and in P.*

Proof. The proof is done by case distinction on the first typing rule used in the derivation of $\Delta \rhd M : A$. □

Definition 9.1.18. In a typing judgement $\Delta \rhd M : A$, a term variable $x \in |\Delta|$ is called *dummy* if $x \notin FV(M)$.

Lemma 9.1.19. *Let $\Delta, x : A \rhd M : B$ be a valid typing judgement where x is a dummy variable for M. Then*

1. *the type A is of the form $!A'$;*
2. *the typing judgement $\Delta \rhd M : B$ is valid;*
3. *for any fresh variable y and any type C, the typing judgement $\Delta, x : A, y : !C \rhd M : B$ is valid.*

Proof. We prove the result by induction on the size of the derivation of $\Delta, x : A \rhd M : B$.

Case (ax_1). The typing judgement is of the form $!\Delta', z : D \rhd z^E : E$, with $D <: E$. If there is a dummy variable x, it is in $|\Delta'|$ and is therefore duplicable. If we write $!\Delta' = (!\Delta'', x : A)$, by rule (ax_1) the typing judgement $!\Delta'', z : D \rhd z^E : E$ is valid. Now, if for some fresh variable y if we consider the typing judgement $!\Delta', z : D, y : !C \rhd z^E : E$, it is valid by the same rule (ax_1).

Case (ax_2) *and case* $(\top.I)$. The typing judgement has a context of the form $!\Delta'$. If there is a dummy variable x, it is in $|\Delta'|$ and is therefore duplicable. Removing this variable from the context does not change the validity of the judgement. Now, if we extend the context to $!\Delta', y : !C$, the typing judgement is again valid by the same rule (ax_2) or $(\top.I)$.

Case (λ_1) *and case* (λ_2). The typing judgement is $\Delta, x : A \rhd \lambda^n z^D.N : !^n(D \multimap E)$, such that $\Delta, x : A, z : D \rhd N : E$ is valid. If $x \notin FV(\lambda^n z^D.N)$, then x is not in $FV(N)$ (since $x \neq z$), and by induction hypothesis its type is of the form $!A'$.

Now, consider some fresh y and some type C. By induction hypothesis, $\Delta, y : !C, x : A, z : D \rhd N : E$ and $\Delta, z : D \rhd N : E$ are valid. If the rule applied was (λ_1), then the integer n is 0, and one can again apply (λ_1) to get $\Delta, x : A, y : !C \rhd \lambda^n z^D.N : !^n(D \multimap E)$ and $\Delta \rhd \lambda^n z^D.N : !^n(D \multimap E)$. If the rule applied was (λ_2), then the integer n is greater than 0 and the context Δ is of the form $!\Delta'$. Then rule (λ_2) can again be applied to get $!\Delta', x : A, y : !C \rhd \lambda^n z^D.N : !^n(D \multimap E)$ and $!\Delta' \rhd \lambda^n z^D.N : !^n(D \multimap E)$.

Cases (*app*), ($\otimes.I$), ($\top.E$) *and* ($\otimes.E$). The term M is respectively of the form

$$M \equiv NP, \qquad M \equiv \langle N, P\rangle^n,$$
$$M \equiv \mathit{let} * = N \mathit{ in } P, \qquad M \equiv \mathit{let}\ \langle x^{D_1}, y^{D_2}\rangle^n = N \mathit{ in } P.$$

The typing context $(\Delta, x : A)$ splits into $!\Delta', \Gamma_1, \Gamma_2$ such that $!\Delta', \Gamma_1 \rhd N : B_1$ and such that $!\Delta', \Gamma_2 \rhd P : B_2$, for some types B_1, B_2 that depend on the rule used.

First let us prove that $\Delta, x : A, y : !C \rhd M : B$ is valid when y is a fresh variable and C is any type. By induction hypothesis, $!\Delta', \Gamma_1, y : !C \rhd N : B_1$ is valid. So applying back the rule that was used, we get that $\Delta, x : A, y : !C \rhd M : B$ is valid.

Second, let us show that if $x \notin FV(M)$, then its type is of the form $!A'$ and $\Delta \rhd M : B$ is valid. There are three cases, depending on where x is to be found.

Found in Γ_1. Then since $x \in |!\Delta', \Gamma_1|$ but $x \notin FV(N)$, by induction hypothesis the type of x in the context is of the form $!A$, and if $\Gamma_1 = (\Gamma_1', x : A)$, we have $!\Delta, \Gamma_1' \rhd N : B_1$ valid. Applying back the typing rule used, we get $\Delta \rhd M : B$ valid.

Found in Γ_2. Then since $x \in |!\Delta', \Gamma_2|$ but $x \notin FV(P)$, by induction hypothesis the type of x in the context is of the form $!A$, and if $\Gamma_2 = (\Gamma_2', x : A)$, we have $!\Delta, \Gamma_2' \rhd P : B_2$ valid. Applying back the typing rule used, we get $\Delta \rhd M : B$ valid.

Found in $!\Delta'$. Then the type of x is automatically of the form $!A$. By induction hypothesis, if $!\Delta' = (!\Delta'', x : A)$, both $!\Delta'', \Gamma_1 \rhd N : B_1$ and $!\Delta'', \Gamma_2 \rhd P : B_2$ are valid. Applying back the typing rule used, we get $\Delta \rhd M : B$ valid.

And this ends the proof. □

9.1.4 Type Casting and Substitution Lemma

We now wish to define a set of call-by-value equations on typed terms. To be able to define a well-typed β-reduction rule, we first need a substitution lemma. Given a value $x : A \rhd V : B$ and a term $y : B \rhd M : C$, one expects $x : A \rhd M[V/y] : C$ to be well typed. Unfortunately, in the presence of Church-style type indices and subtyping, the naive notion of substitution is not technically well-typed. For example, consider the following valid typing derivations:

$$\frac{A \multimap A <: !A \multimap A}{x : A \multimap A \rhd x^{!A \multimap A} : !A \multimap A,} \qquad \frac{\dfrac{A <: A}{y : A \rhd y^A : A}}{\rhd \lambda^0 y^A.y^A : A \multimap A.}$$

Having two valid typing judgements of the form $x : B \rhd M : C$ and $\rhd V : B$, it is reasonable to expect to be able to substitute V for x in M. Doing so yields the following typing judgement:

$$\rhd \lambda^0 y^A.y^A : !A \multimap A,$$

which is not technically valid due to the superscript on the bound variable. The standard trick to resolve this issue (see e.g. Pierce, 2002) is to use implicit typecasting, or coercion. In this section we give a precise definition of substitution using typecasting.

Definition 9.1.20. Given $M : A$ and $A <: B$, we define the map $\mathit{TypeCast}(M, B)$, written $\{M : A <: B\}$, or simply $\{M <: B\}$, by induction on M as follows:

- Base cases:

$$\{x^A <: B\} = x^B, \qquad \{c^A <: B\} = c^B, \qquad \{*^n <: !^m\top\} = *^m.$$

- Induction cases:

$$\begin{aligned}
\{\lambda^n x^A.M <: !^k(A' \multimap B')\} &= \lambda^k x^{A'}.\{M <: B'\},\\
\{M\{N : A\} <: B\} &= \{M <: A \multimap B\}N,\\
\{\langle M, N\rangle^n <: !^m(A \otimes B)\} &= \langle\{M <: !^m A\}, \{N <: !^m B\}\rangle^m,\\
\{let\ \langle x^A, y^B\rangle^n = M\ in\ N <: C\} &= let\ \langle x^A, y^B\rangle^n = M\ in\ \{N <: C\},\\
\{let\ * = M\ in\ N <: C\} &= let\ * = M\ in\ \{N <: C\}\,.
\end{aligned}$$

Lemma 9.1.21. *Suppose $N : A$ and $A <: A'$. Then*

$$\{let\ x^B = M\ in\ N <: A'\} = (let\ x^B = M\ in\ \{N <: A'\}).$$

Proof. Using Definition 9.1.20:

$$\begin{aligned}
\{let\ x^B = M\ in\ N <: A'\} &= \{(\lambda^0 x^B.N)M <: A'\}\\
&= \{\lambda^0 x^B.N <: B \multimap A'\}M\\
&= (\lambda^0 x^B.\{N <: A'\})M\\
&= (let\ x^B = M\ in\ \{N <: A'\}).
\end{aligned}$$

And this ends the proof. □

Lemma 9.1.22. *Suppose that $M : A$ and that $A <: A' <: A''$. Then $\{\{M <: A'\} <: A''\} = \{M <: A''\}$.*

Proof. Proof by induction on the size of M. □

Lemma 9.1.23. *Suppose that $M : A$. then $\{M <: A\} = M$.*

Proof. Proof by structural induction on M. □

Lemma 9.1.24. *If the raw type A of M exists and if $A <: A'$, then $\{M <: A'\}$ is well-defined and of raw type A'.*

Proof. Proof by structural induction on M. □

Lemma 9.1.25. *Suppose $M : A$ and $A <: A'$. Then $Erase(M) = Erase\{M <: A'\}$.*

Proof. Proof by structural induction on M. □

Lemma 9.1.26. *Consider a valid typing judgement $\Delta \rhd M : A$ and the subtyping relation $A <: A'$. Then $\Delta \rhd \{M <: A'\} : A'$ is valid.*

Proof. Proof by induction on the size of M. □

Lemma 9.1.27. *Suppose $M : A$ and $A <: A'$. If M is a core value (respectively an extended value), so is $M' = \{M <: A'\}$.*

Proof. Lemma 9.1.25 states that $Erase(M) = Erase(M')$ and Lemma 9.1.6 that since M is a (core, extended) value, so is M'. □

Definition 9.1.28. Given two valid typing judgements $!\Delta, \Gamma_1 \rhd V : A$ and $!\Delta, \Gamma_2, x : A \rhd M : B$ where V is a core value, where M is any term, and where $|\Gamma_1| \cap |\Gamma_2| = \emptyset$, we define the *substitution* $M[V/x]$ (capture avoiding) of x by V in M as follows: we replace each free instance $x^{A'}$ (where $A <: A'$ from Lemma 9.1.16) in M by $\{V <: A'\}$. This is formally defined as follows:

$$\begin{aligned}
(x^{A'})[V/x] &= \{V <: A'\},\\
(c^B)[V/x] &= c^B,\\
(*^n)[V/x] &= *^n,\\
(MN)[V/x] &= (M[V/x])(N[V/x]),\\
(\langle M, N\rangle^n)[V/x] &= \langle M[V/x], N[V/x]\rangle^n,\\
(\mathit{let} * = M \ \mathit{in}\ N)[V/x] &= \mathit{let} * = (M[V/x]) \ \mathit{in}\ (N[V/x]),\\
(\lambda^n x^C.M)[V/x] &= \lambda^n x^C.M,\\
(\lambda^n y^C.M)[V/x] &= \lambda^n y^C.(M[V/x]) \qquad (\text{if } y \neq x \text{ and } y \notin FV(V)).
\end{aligned}$$

If $y = x$ or $z = x$:

$$(\mathit{let}\ \langle y^C, z^D\rangle^n = M \ \mathit{in}\ N)[V/x] = (\mathit{let}\ \langle y^C, z^D\rangle^n = (M[V/x]) \ \mathit{in}\ N).$$

If $y \neq x$ and $z \neq x$ (and $y, z \notin FV(V)$):

$$(\mathit{let}\ \langle y^C, z^D\rangle^n = M \ \mathit{in}\ N)[V/x] = (\mathit{let}\ \langle y^C, z^D\rangle^n = (M[V/x]) \ \mathit{in}\ (N[V/x])).$$

Definition 9.1.29. Suppose that $!\Delta, \Gamma_i \rhd V_i : A_i$ are valid typing judgements, for $i = 1 \ldots n$, where V_i is a core value. Suppose that $!\Delta, \Gamma, x_1 : A_1, \ldots, x_n : A_n \rhd M : B$ is valid. By α-equivalence one can assume that the x_i's are not free in any V_j's. Assuming this, we define the multiple substitution $M[V_1/x_1, \ldots V_n/x_n]$ by induction on n:

- If $n = 0$, we define$M[]$ to be M.
- If $n \neq 0$, we define $M[V_1/x_1, \ldots V_n/x_n]$ to be $(M[V_1/x_1, \ldots V_{n-1}/x_{n-1}])[V_n/x_n]$, using Definition 9.1.28.

Remark 9.1.30. Note that we only allowed substitution by core values, and not by extended values nor by general terms. This is due to the strict linearity of the type system. Suppose that we are given the valid typing judgements

$$x : \top \rhd \mathit{let} * = x^\top \ \mathit{in}\ *^1 : !\top, \qquad y : !\top \rhd c : A_c.$$

If we were allowed to substitute term variables by extended values, then one could substitute the former term in place of y in the latter term. However, doing so yield the non-valid typing judgement

$$x : \top \rhd c : A_c.$$

The problem comes from the fact that an object of type $\top$ can be weakened, although it is not explicitly duplicable.

Lemma 9.1.31. *In Definition 9.1.28, if $x \notin FV(M)$, $M[V/x] = M$.*

Proof. One proves by structural induction on M that if $x \notin FV(M)$ then $M[V/x] = M$.

Case $M \equiv y^C$. Since $x \notin FV(M)$, $y \leqslant x$. Then by definition $M[V/x] = M$.

Cases $M \equiv *^n$ *and* $M \equiv c^B$. Again, by definition $M[V/x] = M$.

Case $M \equiv \lambda^n y^C.N$. If $y = x$ then by definition $(\lambda^n y^C.N)[V/x] = \lambda^n y^C.N$ and we are done. If $y \neq x$, the substitution $(\lambda^n y^C.N)[V/x]$ is equal to $\lambda^n y^C.(N[V/x])$. Then by induction hypothesis, $N[V/x] = N$, and thus $M[V/x] = M$.

Cases $M \equiv NP$, $M \equiv \langle N, P\rangle^n$ *and* $M \equiv (\mathit{let} * = N \mathit{ in } P)$. By induction hypothesis, $N[V/x] = N$ and $P[V/x] = P$. Using the definition of substitution, $M[V/x] = M$.

Case $M \equiv (\mathit{let}\ \langle y^C, z^D\rangle^n = N \mathit{ in } P)$. Using α-equivalence one can suppose that y and z are not free variables of V. Then, if $y = x$ or $z = x$, $M[V/x] = (\mathit{let}\ \langle y^C, z^D\rangle^n = N[V/x] \mathit{ in } P)$. By induction hypothesis $N[V/x] = N$ and thus $M[V/x] = M$. If $x \neq y$ and $x \neq z$, $M[V/x] = (\mathit{let}\ \langle y^C, z^D\rangle^n = N[V/x] \mathit{ in } P[V/x])$. By induction hypothesis $N[V/x] = N$ and $P[V/x] = P$, thus making $M[V/x] = M$.

Then Lemma 9.1.31 is valid. □

Lemma 9.1.32. *In Definition 9.1.28, if M is a core value, so is $M[V/x]$, and if M is an extended value, so is $M[V/x]$.*

Proof. By induction on the structure of M we prove that if M is a core value, so is $M[V/x]$. Base cases:

Case $M \equiv y^B$. If $y \neq x$, then $M[V/x] = M$, and it is a core value. If $y = x$, then $M[V/x] = \{V{<:}B\}$, which is a core value from Lemma 9.1.27.

Cases $M \equiv *^n$ *and* $M \equiv c^B$. In both cases $M[V/x] = M$, already in the form of a core value.

Case $M \equiv \lambda^n y^C.N$. In this situation $M[V/x]$ is of the form $\lambda\ldots$, which is a core value.

Induction case:

Case $M \equiv \langle V_1, V_2\rangle^n$, *with* V_1, V_2 *core values.* We have $M[V/x] = \langle V_1[V/x], V_2[V/x]\rangle^n$. By induction hypothesis, $V_1[V/x]$ and $V_2[V/x]$ are core values, making the term $\langle V_1[V/x], V_2[V/x]\rangle^n = M[V/x]$ into a core value.

This exhausts the possibilities. Thus, by structural induction, $M[V/x]$ is a core value is M is. We do the same for extended values, and prove that $M[V/x]$ is an extended value if M is, by structural induction on M. The base cases are the same as the base cases for core values, replacing the word "core" by the word "extended", and remembering that any core value is an extended value. The induction cases go as follows.

Cases $M \equiv \langle V_1, V_2\rangle^n$ *and* $(\mathit{let} * = V_1 \mathit{ in } V_2)$. In this case, V_1 and V_2 are extended values. Write $f(V_1, V_2)$ for M. Then we have $M[V/x] = f(V_1[V/x], V_2[V/x])$. By induction hypothesis, $V_1[V/x]$ and $V_2[V/x]$ are extended values, making the term $M[V/x]$ into an extended value.

Case $M \equiv (\mathit{let}\ y^C = V_1 \mathit{ in } V_2)$ (*respectively* $M \equiv (\mathit{let}\ \langle y^C, z^D\rangle^n = N \mathit{ in } P)$). We write $f(V_1, V_2)$ in place of M. Modulo α-equivalence one can assume that y (resp. y and z) is not a free variable of V. If $y = x$ (resp. $y = x$ or $z = x$) then $M[V/x] = f(V_1[V/x], V_2)$. If $y \neq x$ (resp. $y \neq x$ and $z \neq x$), $M[V/x] = f(V_1[V/x], V_2[V/x])$. By induction hypothesis, $V_1[V/x]$ (and $V_2[V/x]$ in the second case) are extended values. This means that in both cases, $M[V/x]$ is an extended value.

This exhausts the possible cases, and the proof by induction is done: if M is an extended value, so is $M[V/x]$ □

Lemma 9.1.33 (Substitution Lemma)**.** *Given two valid typing judgements* $!\Delta, \Gamma_1 \rhd V : A$ *and* $!\Delta, \Gamma_2, x : A \rhd M : B$ *such that* V *is a core value and such that* $|\Gamma_1| \cap |\Gamma_2| = \emptyset$*, the typing judgement* $!\Delta, \Gamma_1, \Gamma_2 \rhd M[V/x] : A$ *is valid.*

Proof. The proof is done by induction on the size of M.

Case $M \equiv y^B$. Then the derivation of the typing judgement of M starts with the rule (ax_1). This implies that $\Gamma_2 = !\Gamma'_2$ and the typing judgement of M is $!\Delta, !\Gamma'_2, x : A \rhd y^B : B$.

If $y = x$, then $A <: B$ and $M[V/x] = !\Delta, \Gamma_1 \rhd \{V <: B\}$ is well-defined. From Lemma 9.1.26 $!\Delta, \Gamma_1 \rhd M[V/x] : B$ is well-typed. Using Lemma 9.1.19 $!\Delta, \Gamma_1, !\Gamma'_2 \rhd M[V/x] : B$ is valid.

If $y \neq x$, then $M[V/x] = y^B$. Since $!\Delta, \Gamma_2, x : A \rhd y^B : B$ is valid, A has to be of the form $!A'$. Then by Lemma 9.1.15, Γ_1 is of the form $!\Gamma'_1$. From Lemma 9.1.19 $!\Delta, !\Gamma'_1, !\Gamma'_2 \rhd y^B : B$ is valid.

Case $M \equiv c^B$ *(resp.* $M \equiv *^n$*).* A derivation of M starts with rule (ax_2) (resp. rule $(\top.I)$). In both cases the context $(\Gamma_2, x : A)$ is duplicable: $\Gamma_2 = !\Gamma'_2$ and $A = !A'$. The core value V is then in fact of type $!A'$. By Lemma 9.1.15, Γ_1 is of the form $!\Gamma'_1$ for some context Γ'_1.

In both cases, the substitution $M[V/x]$ yields back M. Thus, $!\Delta, !\Gamma'_1, !\Gamma'_2 \rhd M[V/x] : B$ is valid by applying the typing rule (ax_2) (resp. $(\top.I)$).

Case $M \equiv \lambda^n y^C.N$. The typing judgement for the term M is of the form $!\Delta, \Gamma_2, x : A \rhd \lambda^n y^C.N : !^n(C \multimap D)$, and any typing derivation starts with either (λ_1) or (λ_2) depending on the value n. In both cases, the judgement $!\Delta, \Gamma_2, x : A, y : C \rhd N : D$ is valid.

Note that x cannot be equal to y in this situation. Using α-equivalence, one can furthermore choose y such that it does not belong to Γ_1.

The induction hypothesis can be applied on the judgements $!\Delta, \Gamma_1 \rhd V : B$ and $!\Delta, \Gamma_2, y : C, x : A \rhd N : D$, and we get the valid typing judgement

$$!\Delta, \Gamma_1, \Gamma_2, y : C \rhd N[V/x] : D. \tag{9.1.1}$$

Note that since $x \neq y$, the substitution $(\lambda^n y^C.N)[V/x] = \lambda^n y^C.(N[V/x])$. There are two cases, depending on n.

If $n = 0$, applying (λ_1) on Equation (9.1.1) gives that $!\Delta, \Gamma_1, \Gamma_2 \rhd (\lambda^0 y^C.N)[V/x] : C \multimap D$ is valid.

If $n \neq 0$, then (λ_2) is the first rule for the derivation of M, and $(\Gamma_2, x : A)$ is of the form $(!\Gamma'_2, x : !A')$. Since $A = !A'$ is duplicable and since V is a core value, by Lemma 9.1.15 the context Γ_1 is of the form $!\Gamma'_1$. The rule (λ_2) can then be applied on Equation (9.1.1) to get the valid typing judgement $!\Delta, \Gamma_1, \Gamma_2 \rhd (\lambda^0 y^C.N)[V/x] : !^n(C \multimap D)$.

Cases $M \equiv N_1 N_2$, $\langle N_1, N_2 \rangle^n$, *(let* $* = N_1$ *in* N_2*) and (let* $\langle y^C, z^D \rangle^n = N_1$ *in* N_2*).*
A typing derivation of $!\Delta, \Gamma_2, x : A \rhd M : B$ starts with

$$\dfrac{\begin{array}{c}\vdots \\ !\Delta', \Gamma_{21} \rhd N_1 : B_1\end{array} \quad \begin{array}{c}\vdots \\ !\Delta', \Gamma_{22}, \Lambda \rhd N_2 : B_2\end{array}}{!\Delta, \Gamma_2, x : A \rhd f(N_1, N_2) : B}\ (X),$$

where $(!\Delta', \Gamma_{21}, \Gamma_{22}) = (!\Delta, \Gamma_2, x : A)$, where $M = f(N_1, N_2)$, where

if $f(N_1, N_2)$ is...	(X) is...	B is...	B_1 is...	B_2 is...
$N_1 N_2$	(app)	D	$C \multimap D$	C
$\langle N_1, N_2 \rangle^n$	$(\otimes.I)$	$!^n(C \otimes D)$	$!^n C$	$!^n D$
let $* = N_1$ *in* N_2	$(\top.E)$	D	$\top$	D
let $\langle y^C, z^D \rangle^n = N_1$ *in* N_2	$(\otimes.E)$	E	$!^n(C \otimes D)$	E

for some types C, D and E, and where Λ is empty in the three first cases and equal to $(y : !^n C, z : !^n D)$ in the last case.

By α-equivalence one can assume that in the last case, the variables y and z in Λ does not occur anywhere in the other contexts.

From Lemma 9.1.19, one can assume that Γ_{21} and Γ_{22} do not contain any duplicable variables, implying that $|!\Delta| \subseteq |!\Delta'|$. Thus there exists some context $!\Psi$ such that $!\Delta' = (!\Psi, !\Delta)$. In particular, this means that

$$(!\Psi, \Gamma_{21}, \Gamma_{22}) = (\Gamma_2, x : A). \tag{9.1.2}$$

This equation is the main result that will allow us to apply the induction hypothesis, and thus finish the proof.

The variable $x : A$ belongs to either $!\Psi$, Γ_{21} or Γ_{22}. We study each one of these three possibilities.

$x \in |!\Psi|$. In this case, $A = !A'$ and $!\Psi = (!\Psi', x : !A')$. The contexts $(!\Delta', \Gamma_{21})$ and $(!\Delta', \Gamma_{22}, \Lambda)$ becomes respectively $(!\Psi', !\Delta, \Gamma_{21}, x : !A')$ and $(!\Psi', !\Delta, \Gamma_{22}, \Lambda, x : !A')$, meaning that the typing judgements $!\Psi', !\Delta, \Gamma_{21}, x : !A' \rhd N_1 : B_1$ and $!\Psi', !\Delta, \Gamma_{22}, \Lambda, x : !A' \rhd N_2 : B_2$ are valid.

Since $!\Psi = (!\Psi', x : !A')$, Equation (9.1.2) implies that $(!\Psi', \Gamma_{21}, \Gamma_{22}) = \Gamma_2$. Since $|\Gamma_1| \cap |\Gamma_2| = \emptyset$, the set $|\Gamma_1|$ does not intersect $|\Psi'|$, $|\Gamma_{21}|$ nor $|\Gamma_{22}|$. This means that the typing context $(!\Psi', !\Delta, \Gamma_{21}, \Gamma_{22}, \Gamma_1)$ is valid. Together with the fact that Λ does not intersect anything, the requirements on the contexts in the following are satisfied.

By Lemma 9.1.19, the typing judgement $!\Psi', !\Delta, \Gamma_1 \rhd V : !A'$ is valid.

One can apply the induction hypothesis and get that $!\Psi', !\Delta, \Gamma_{21}, \Gamma_1 \rhd N_1[V/x] : B_1$ and $!\Psi', !\Delta, \Gamma_{22}, \Lambda, \Gamma_1 \rhd N_2[V/x] : B_2$ are valid.

Since $A = !A'$, from Lemma 9.1.15 the context Γ_1 is of the form $!\Gamma'_1$.

Using the fact that $f(N_1[V/x], N_2[V/x]) = f(N_1, N_2)[V/x]$ and applying the typing rule (X), one gets that $!\Psi', !\Delta, \Gamma_{21}, \Gamma_{22}, !\Gamma'_1 \rhd f(N_1, N_2)[V/x] : B$ is valid. One can rewrite this typing judgement as $!\Delta, \Gamma_1, \Gamma_2 \rhd M[V/x] : B$.

Case $x \in |\Gamma_{21}|$ and case $x \in |\Gamma_{22}|$. The two cases are symmetrical (modulo the use of Λ, which does not bring any difficulty), by inverting the role of 1 and 2 in the indices. We provide the proof for 2.

The context Γ_{22} is of the form $(\Gamma'_{22}, x : A)$. This means that $!\Psi, !\Delta, \Gamma_{22}, \Lambda, x : A \rhd N_2 : B_2$ is valid.

Equation (9.1.2) implies that $(!\Psi, \Gamma_{21}, \Gamma'_{22}) = \Gamma_2$. Since $|\Gamma_1| \cap |\Gamma_2| = \emptyset$, the set $|\Gamma_1|$ does not intersect $|\Psi'|$, $|\Gamma_{21}|$ nor $|\Gamma_{22}|$. This means that the typing context $(!\Psi', !\Delta, \Gamma_{21}, \Gamma'_{22}, \Gamma_1)$ is valid. Together with the fact that Λ does not intersect anything, the requirements on the contexts in the following are satisfied.

By Lemma 9.1.19 the typing judgement $!\Psi, !\Delta, \Gamma_1 \rhd V : A$ is valid.

One can apply the induction hypothesis: the judgement $!\Psi, !\Delta, \Gamma'_{22}, \Lambda, \Gamma_1 \rhd N_2[V/x] : B_2$ is valid.

Since $(!\Psi, !\Delta) = !\Delta'$, the typing judgement $!\Psi, !\Delta, \Gamma_{21} \rhd N_1 : B_1$ is valid.

Since $x \in |\Gamma_{22}|$, x is not a free variable of N_1. From Lemma 9.1.31, this means that $f(N_1, N_2)[V/x] = f(N_1, N_2[V/x])$. (In the case $x \in |\Gamma_{21}|$, the substitution $f(N_1, N_2)[V/x]$ is equal to $f(N_1[V/x], N_2)$.)

One can then apply the typing rule (X) and get that $!\Psi', !\Delta, \Gamma_{21}, \Gamma'_{22}, \Gamma_1 \rhd f(N_1, N_2)[V/x] : B$ is valid.

By definition, the context of this typing judgement is $(!\Delta, \Gamma_1, \Gamma_2)$. This makes $!\Delta, \Gamma_1, \Gamma_2 \rhd M[V/x] : B$ a valid typing judgement.

Hence, by structural induction on the typing derivation of $!\Delta, \Gamma_2, x : A \rhd M : B$, Lemma 9.1.33 is valid. □

Lemma 9.1.34. *Suppose that* $!\Delta, \Gamma_1 \rhd V : A$ *and* $!\Delta, \Gamma_2, x : A \rhd M : B$ *such that* V *is a core value and such that* $|\Gamma_1| \cap |\Gamma_2| = \emptyset$. *Suppose moreover that* $B <: B'$. *Then* $\{M[V/x] <: B'\} = \{M <: B'\}[V/x]$.

Proof. By induction on the size of M.

Case $M \equiv x^B$. In this case, the typing judgement $!\Delta, \Gamma_2, x : A \rhd M : B$ comes from the rule (ax_1), meaning that $A <: B$.

By definition, $\{M[V/x] <: B'\} = \{\{V <: B\} <: B'\}$ and $\{M <: B'\}[V/x] = \{V <: B'\}$. They are equivalent from Lemma 9.1.22.

Case $M \equiv c^B$, $*^n$ *and* y^B, *with* $y \neq x$. In these three cases, $M[V/x] = M$ and $\{M <: B'\}[V/x] = \{M <: B'\}$, making the result true by vacuity.

Case $M \equiv \lambda^n y^C.N$. By α-equivalence one can assume that $y \neq x$. since M is well-typed of type B, there exists a type D such that $B = !^n(C \multimap D)$. Therefore, $B' = !^m(C' \multimap D')$, with $D <: D'$, $C' <: C$ and $m = 0$ if $n = 0$. One has

$$\{M[V/x] <: B'\} = \{(\lambda^n y^C.N[V/x]) <: !^m(C' \multimap D')\} = \lambda^m y^{C'}.\{N[V/x] <: D'\}$$

which is by induction hypothesis $\lambda^m y^{C'}.\{N <: D'\}[V/x]$. Therefore

$$\begin{aligned}\{M[V/x] <: B'\} &= \lambda^m y^{C'}.\{N <: D'\}[V/x] = (\lambda^m y^{C'}.\{N <: D'\})[V/x] \\ &= \{\lambda^n y^C.N <: !^m(C' \multimap D')\}[V/x] = \{M <: B'\}[V/x].\end{aligned}$$

Case $M \equiv NP$. Since M is of type B, there exists a type C such that $N : C \multimap B$ and such that $P : C$. We have therefore:

$$\begin{aligned}\{M[V/x] <: B'\} &= \{(N[V/x])(P[V/x]) <: B'\} \\ &= (N[V/x])\{P[V/x] <: C \multimap B'\} \\ &= (N[V/x])(\{P <: C \multimap B'\}[V/x]) && \text{(by ind. hyp.)} \\ &= (N\{P <: C \multimap B'\})[V/x] \\ &= \{NP <: B'\}[V/x] \\ &= \{M <: B'\}[V/x].\end{aligned}$$

Case $M \equiv (\mathit{let}\ \langle y^C, z^D\rangle^n = N\ \mathit{in}\ P)$. By α-equivalence one can assume that $x \neq y, z$. Since M is of type B, so in P. We have therefore:

$$\begin{aligned}\{M[V/x] <: B'\} &= \{\mathit{let}\ \langle y^C, z^D\rangle^n = N[V/x]\ \mathit{in}\ P[V/x] <: B'\} \\ &= \mathit{let}\ \langle y^C, z^D\rangle^n = N[V/x]\ \mathit{in}\ \{P[V/x] <: B'\} \\ &= \mathit{let}\ \langle y^C, z^D\rangle^n = N[V/x]\ \mathit{in}\ \{P <: B'\}[V/x] && \text{(by ind. hyp.)} \\ &= (\mathit{let}\ \langle y^C, z^D\rangle^n = N\ \mathit{in}\ \{P <: B'\})[V/x] \\ &= \{\mathit{let}\ \langle y^C, z^D\rangle^n = N\ \mathit{in}\ P <: B'\}[V/x] \\ &= \{M <: B'\}[V/x].\end{aligned}$$

The remaining cases are similar. □

9.2 Equational Logic of Typed Terms

Definition 9.2.1. We define an equivalence relation on (indexed) typing judgements. We write $\Delta \rhd M \approx_{ax} M' : A$, or simply $M \approx_{ax} M'$, to indicate that $\Delta \rhd M : A$ and $\Delta \rhd M' : A$ are equivalent. Axiomatic equivalence is defined as the reflexive, symmetric, transitive, and congruence closure of

- the rules from Tables 9.2, Table 9.3, so long as both sides of the equivalences are well-typed. We recall from Convention 9.1.5 that the symbols $\boxdot$ and $\boxdot$ are place holders for x^A, $*$, or $\langle x^A, y^B \rangle^n$.
- the following two rules:

  Rule ($app_{<:}$). Suppose that $A <: B$. If $\Delta = (!\Delta', \Gamma_1, \Gamma_2)$ such that

  $$!\Delta', \Gamma_1 \rhd N : B \multimap C, \qquad\qquad !\Delta', \Gamma_2 \rhd P : A,$$

  then $\Delta \rhd N\{P <: B\} \approx_{ax} \{N <: A \multimap C\}P : C$.

  Rule ($let^{\otimes}_{<:}$). Suppose that $A <: A'$, $B <: B'$, and $n' = 0$ if $n = 0$.

  If $\Delta = (!\Delta', \Gamma_1, \Gamma_2)$ such that

  $$!\Delta', \Gamma_1 \rhd N : !^n(A \otimes B), \qquad !\Delta', \Gamma_2, x : !^{n'}A', y : !^{n'}B' \rhd P : E,$$

  then

  $$\Delta \rhd \Big(let\ \langle x^{A'}, y^{B'} \rangle^{n'} = \{N <: !^{n'}(A' \otimes B')\}\ in\ P \Big) \approx_{ax} \big(let\ \langle x^A, y^B \rangle^n = N\ in\ P \big) : E.$$

Lemma 9.2.2. *Suppose that $A <: A'$. If $\Delta = (!\dot{\Delta}, \Gamma_1, \Gamma_2)$ such that*

$$!\dot{\Delta}, \Gamma_1 \rhd N : A, \qquad\qquad !\dot{\Delta}, \Gamma_2, x : A' \rhd P : E,$$

then

$$\Delta \rhd \Big(let\ x^{A'} = \{N <: A'\}\ in\ P \Big) \approx_{ax} \big(let\ x^A = N\ in\ P \big) : E.$$

Proof. By definition. □

Convention 9.2.3. The rule stated in Lemma 9.2.2 will be referred to using the symbol ($let^{x}_{<:}$).

Lemma 9.2.4. *The equivalences of Table 9.4 are derivable.*

Proof. Proof by case distinction. □

Lemma 9.2.5. *Suppose that $\Delta \rhd M \approx_{ax} M' : A$ and that $\Delta' <: \Delta$ and $A <: A'$. Then $\Delta' \rhd \{M <: A'\} \approx_{ax} \{M' <: A'\} : A'$.*

Proof. We have $\Delta \rhd M : A$ and $\Delta \rhd M' : A$. From Lemma 9.1.14, $\Delta' \rhd M : A$ and $\Delta' \rhd M' : A$ are valid. From Lemma 9.1.26, $\Delta' \rhd \{M <: A'\} : A'$ and $\Delta' \rhd \{M' <: A'\} : A'$ are valid. It remains to prove that they are axiomatically equivalent. We prove it by induction on the size of the derivation of $\Delta \rhd M \approx_{ax} M' : A$.

Base case: β-η-rules.

$$
\begin{array}{lll}
(\beta_\lambda)\Delta \rhd \mathit{let}\ x = V\ \mathit{in}\ M & \approx_{ax} M[V/x] & : A \\
(\beta_\otimes)\Delta \rhd \mathit{let}\ \langle x, y\rangle^n = \langle V, W\rangle^n\ \mathit{in}\ M & \approx_{ax} M[V/x, W/y] & : A \\
(\beta_*)\Delta \rhd \mathit{let}\ * = *\ \mathit{in}\ M & \approx_{ax} M & : A \\
(\eta_\lambda)\Delta \rhd \lambda^n x^A.\{V <: A \multimap B\}x^A & \approx_{ax} V & : !^n(A \multimap B). \\
(\beta^2_\lambda)\Delta \rhd \mathit{let}\ x^A = N\ \mathit{in}\ x^A & \approx_{ax} N & : A. \\
(\eta_\otimes)\Delta \rhd \mathit{let}\ \langle x^A, y^B\rangle^n = N\ \mathit{in}\ \langle x^{!^n A}, y^{!^n B}\rangle^n & \approx_{ax} N & : !^n(A \otimes B). \\
(\eta_*)\Delta \rhd \mathit{let}\ * = \{N <: \top\}\ \mathit{in}\ *^n & \approx_{ax} N & : !^n\top.
\end{array}
$$

Table 9.2: Axiomatic equivalence axioms: beta-eta-rules

$$
\begin{array}{lll}
(\mathit{let}_1) & \Delta \rhd \mathit{let}\ \boxdot = (\mathit{let}\ \boxdot = M\ \mathit{in}\ N)\ \mathit{in}\ P & \approx_{ax}\ \mathit{let}\ \boxdot = M\ \mathit{in}\ \mathit{let}\ \boxdot = N\ \mathit{in}\ P \quad : A \\
(\mathit{let}_2) & \Delta \rhd \mathit{let}\ \boxdot = V\ \mathit{in}\ \mathit{let}\ \boxdot = W\ \mathit{in}\ M & \approx_{ax}\ \mathit{let}\ \boxdot = W\ \mathit{in}\ \mathit{let}\ \boxdot = V\ \mathit{in}\ M \quad : A \\
(\mathit{let}^{app}) & \Delta \rhd \mathit{let}\ x^{A\multimap B} = M\ \mathit{in}\ \mathit{let}\ y^A = N\ \mathit{in}\ xy & \approx_{ax} MN \quad : B \\
(\mathit{let}^\lambda) & \Delta \rhd \mathit{let}\ x^D = V\ \mathit{in}\ \lambda^n y^A.M & \approx_{ax} \lambda^n y^A.\mathit{let}\ x^D = V\ \mathit{in}\ M : !^n(A \multimap B) \\
(\mathit{let}^\otimes) & \Delta \rhd \mathit{let}\ x^{!^n A} = M\ \mathit{in}\ \mathit{let}\ y^{!^n B} = N\ \mathit{in}\ \langle x^{!^n A}, y^{!^n B}\rangle^n & \approx_{ax} \langle M, N\rangle^n \quad : !^n(A \otimes B)
\end{array}
$$

Table 9.3: Axiomatic equivalence axioms: commutation rules

Case (β_λ). In this case, $M = (\mathit{let}\ x^B = V\ \mathit{in}\ N)$ and $M' = N[V/x]$. From Lemma 9.1.21, $\{M <: A'\} = (\mathit{let}\ x^B = V\ \mathit{in}\ \{N <: A'\})$. From Lemma 9.1.34, $\{M' <: A'\} = \{N <: A'\}[V/x]$. Using equivalence rule (β_λ), $\Delta' \rhd \mathit{let}\ x^B = V\ \mathit{in}\ \{N <: A'\} \approx_{ax} \{N <: A'\}[V/x] : A'$.

Case $(\beta_\otimes)$. We have $M = (\mathit{let}\ \langle x^B, y^C\rangle^n = \langle V, W\rangle^n\ \mathit{in}\ N)$ and $M' = N[V/x, W/y]$. From Definition 9.1.20, $\{M <: A'\} = (\mathit{let}\ \langle x^B, y^C\rangle^n = \langle V, W\rangle^n\ \mathit{in}\ \{N <: A'\})$. From Lemma 9.1.34, $\{M' <: A'\} = \{N <: A'\}[V/x, W/y]$. Using equivalence rule $(\beta_\otimes)$, $\Delta' \rhd \mathit{let}\ \langle x^B, y^C\rangle^n = \langle V, W\rangle^n\ \mathit{in}\ \{N <: A'\} \approx_{ax} \{N <: A'\}[V/x, W/y] : A'$.

Case (β_*). In this case, the term M is $(\mathit{let}\ * = *\ \mathit{in}\ N)$ and $M' = N$. From Definition 9.1.20, $\{M <: A'\} = (\mathit{let}\ * = *\ \mathit{in}\ \{N <: A'\})$. Using equivalence rule (β_*), we obtain the axiomatic equivalence $\Delta' \rhd \mathit{let}\ * = *\ \mathit{in}\ \{N <: A'\} \approx_{ax} \{N <: A'\} : A'$.

Case (η_λ). In this case, $M = \lambda^n x^B.\{V <: B \multimap C\}x^B$, $M' = V$, $A = !^n(B \multimap C)$ and $A' = !^m(B' \multimap C')$ with $m = 0$ if $n = 0$, $C <: C'$ and $B' <: B$.

From Definition 9.1.20, $\{M <: A'\} = \lambda^m x^{B'}.\{\{V <: B \multimap C\}x^B <: C'\} = \lambda^m x^{B'}.\{\{V <: B \multimap C\} <: B \multimap C'\}x^B$. From Lemma 9.1.22, $\{M <: A'\} = \lambda^m x^{B'}.\{V <: B \multimap C'\}x^B$.

We have $\Delta' \rhd V : !^n(B \multimap C)$. Since $x \notin |\Delta'|$ and since $x^B = \{x^{B'} <: B\}$, by equivalence rule $(app_{<:})$ and by Lemma 9.1.22 we have $\Delta', x : B' \rhd \{V <: B \multimap C'\}x^B \approx_{ax} \{V <: B' \multimap C'\}x^{B'} : C'$. Thus by congruence $\Delta' \rhd \{M <: A'\} \approx_{ax} \lambda^m x^{B'}.\{V <: B' \multimap C'\}x^{B'} : !^m(B' \multimap C')$.

From Lemma 9.1.22, $\{V <: B' \multimap C'\} = \{\{V <: !^m(B' \multimap C')\} <: B' \multimap C'\}$. Using rule (η_λ), by transitivity we get that $\Delta' \rhd \{M <: !^m(B' \multimap C')\} \approx_{ax} \{V <: !^m(B' \multimap C')\} : !^m(B' \multimap C')$.

Case (β^2_λ). In this case, $M = (\mathit{let}\ x^A = N\ \mathit{in}\ x^A)$ and $M' = N$.

From Definition 9.1.20, $\{M <: A'\} = (\mathit{let}\ x^A = N\ \mathit{in}\ x^{A'})$. From Lemma 9.2.2, $\Delta' \rhd (\mathit{let}\ x^A = N\ \mathit{in}\ x^{A'}) \approx_{ax} (\mathit{let}\ x^{A'} = \{N <: A'\}\ \mathit{in}\ x^{A'}) : A'$. Applying rule (β^2_λ), we get that this is equivalent to $\Delta' \rhd \{N <: A'\} : A'$.

Case $(\eta_\otimes)$. In this case, $M = (\mathit{let}\ \langle x^B, y^C\rangle^n = N\ \mathit{in}\ \langle x^{!^n B}, y^{!^n C}\rangle^n)$ and $M' = N$. We also have

$$
\begin{array}{rlll}
(\alpha_{let}) & \Delta, y : A \rhd M : B & \approx_{ax} \Delta, x : A \rhd let\ y^A = x^A\ in\ M : B & \\
(let^{!\lambda}) & !\Delta \rhd let\ x^{!C} = V\ in\ \lambda y.M & \approx_{ax} \lambda^{n+1} y.\ let\ \ x^{!C} = V\ in\ M & : !^{n+1}(A \multimap B) \\
(let_1^{\otimes}) & \Delta \rhd \langle V, let\ \boxdot = M\ in\ N\rangle & \approx_{ax} let\ \boxdot = M\ in\ \langle V, N\rangle & : !^n(A \otimes B) \\
(let_2^{\otimes}) & \Delta \rhd \langle let\ \boxdot = M\ in\ N, V\rangle & \approx_{ax} let\ \boxdot = M\ in\ \langle N, V\rangle & : !^n(A \otimes B) \\
(let_1^{app}) & \Delta \rhd V(let\ \boxdot = M\ in\ N) & \approx_{ax} let\ \boxdot = M\ in\ VN & : B \\
(let_2^{app}) & \Delta \rhd (let\ \boxdot = M\ in\ N)V & \approx_{ax} let\ \boxdot = M\ in\ NV & : B
\end{array}
$$

Table 9.4: Axiomatic equivalence: derived rules

$A = !^n(B \otimes C)$, and since $A <: A'$, $A' = !^m(B' \otimes C')$ with $m = 0$ if $n = 0$, with $B <: B'$ and with $C <: C'$.

From Definition 9.1.20, $\{M <: A'\} = (let\ \langle x^B, y^C\rangle^n = N\, in\, \langle x^{!^m B'}, y^{!^m C'}\rangle^n)$. From equivalence rule $(let^{\otimes}_{<:})$,

$$
\begin{aligned}
&\Delta' \rhd let\ \langle x^B, y^C\rangle^n = N\ in\ \langle x^{!^m B'}, y^{!^m C'}\rangle^n \\
&\qquad\qquad \approx_{ax} let\ \langle x^{B'}, y^{C'}\rangle^m = \{N <: !^m(B' \otimes C')\}\ in\ \langle x^{!^m B'}, y^{!^m C'}\rangle^n : A'.
\end{aligned}
$$

Applying rule $(\eta_{\otimes})$, we get that this is equivalent to $\Delta' \rhd \{N <: A'\} : A'$

Case (η_*). In this case, $M = (let * = \{N <: \top\}\ in\ *^n)$, $M' = N$ and $A = !^n\top$. Since $A <: A'$, $A' = !^m\top$ with $m = 0$ if $n = 0$.

From Definition 9.1.20, $\{M <: A'\} = (let * = \{N <: \top\}\ in\ *^m)$.

From Lemma 9.1.22, $\{N <: \top\} = \{\{N <: !^m\top\} <: \top\}$.

From rule (η_*), $\Delta' \rhd let * = \{\{N <: !^m\top\} <: \top\}\ in\ *^m \approx_{ax} \{N <: !^m\top\}$.

And this prove the lemma for the β-η-rules of Table 9.2.

Next base cases: commutation rules.

Case (let_1). Here, $M = (let\ -_1 = (let\ -_2 = N_2\ in\ N_1)\ in\ P)$ and $M' = (let\ -_2 = N_2\ in\ let\ -_1 = N_1\ in\ P)$.

Using Definition 9.1.20, $\{M <: A'\} = (let\ -_1 = (let\ -_2 = N_2\ in\ N_1)\ in\ \{P <: A'\})$ and $\{M' <: A'\} = (let\ -_2 = N_2\ in\ let\ -_1 = N_1\ in\ \{P <: A'\})$. Using equivalence rule (let_1), we find that $\Delta' \rhd \{M <: A'\} \approx_{ax} \{M' <: A'\} : A'$.

Case (let_2). This case is similar to the previous one. Here, $M = (let\ -_1 = V\, in\, let\ -_2 = W\, in\, N)$ and $M' = (let\ -_2 = W\ in\ let\ -_1 = V\ in\ N)$.

Using Definition 9.1.20, $\{M <: A'\} = (let\ -_1 = V\, in\, let\ -_2 = W\, in \{N <: A'\})$ and $\{M' <: A'\} = (let\ -_2 = W\ in\ let\ -_1 = V\ in\ \{N <: A'\})$. Using equivalence rule (let_2), we find that $\Delta' \rhd \{M <: A'\} \approx_{ax} \{M' <: A'\} : A'$.

Case (let^{app}). Here, $M = (let\ x^{B \multimap A} = N\ in\ let\ y^B = P\ in\ (x^{B \multimap A} y^B))$ and $M' = NP$. Using Definition 9.1.20, $\{M <: A'\} = (let\ x^{B \multimap A} = N\ in\ let\ y^B = P\ in\ (x^{B \multimap A'} y^B))$. From Lemma 9.2.2,

$$
\begin{aligned}
&\Delta' \rhd let\ x^{B \multimap A} = N\ in\ let\ y^B = P\ in\ (x^{B \multimap A'} y^B) \\
&\qquad\qquad \approx_{ax} let\ x^{B \multimap A'} = \{N <: B \multimap A'\}\ in\ let\ y^B = P\ in\ (x^{B \multimap A'} y^B) : A'.
\end{aligned}
$$

Rule (let^{app}) says that this is equivalent to $\Delta' \rhd \{N <: B \multimap A'\}P : A'$. The result is obtained by using Definition 9.1.20: $\{M' <: A'\} = \{N <: B \multimap A'\}P$.

Case (let^{λ}). In this case, $M = (let\ x^D = V\ in\ \lambda^n y^B.N)$, $M' = \lambda^n y^B.(let\ x^D = V\ in\ N)$, $A = !^n(B \multimap C)$ and $A' = !^m(B' \multimap C')$ with $m = 0$ if $n = 0$, $C <: C'$ and $B' <: B$.

Using Definition 9.1.20, $\{M <: A'\} = (let\ x^D = V\ in\ \lambda^m y^{B'}.\{N <: C'\})$ and $\{M' <: A'\} = \lambda^m y^{B'}.(let\ x^D = V\ in\ \{N <: C'\})$.

Using equivalence rule (let^{λ}), we have as required that $\Delta' \rhd \{M <: A'\} \approx_{ax} \{M' <: A'\} : A'$.

Case ($let^{\otimes}$). In this situation, $M = (let\ x^{!^n B} = N in let\ y^{!^n C} = P in \langle x^{!^n B}, y^{!^n C}\rangle^n)$, $M' = \langle N, P\rangle^n$, $A = !^n(B \otimes C)$ and $A' = !^m(B' \otimes C')$ with $m = 0$ if $n = 0$, $C <: C'$ and $B <: B'$.

Using Definition 9.1.20, $\{M <: A'\} = (let\ x^{!^n B} = N\ in\ let\ y^{!^n C} = P\ in\ \langle x^{!^m B'}, y^{!^m C'}\rangle^m)$ and $\{M' <: A'\} = \langle\{N <: !^m C'\}, \{P <: !^m D'\}\rangle^m$.

Using twice Lemma 9.2.2, we get that $\Delta' \rhd \{M <: A'\} \approx_{ax} (let\ x^{!^m B'} = \{N <: !^m B'\}\ in\ let\ y^{!^m C'} = \{P <: !^m C'\}\ in\ \langle x^{!^m B'}, y^{!^m C'}\rangle^m) : A'$.

Applying ($let^{\otimes}$) and transitivity gives the required result.

Rules concerning typecasting. For these two rules, we use in each case the notation of Definition 9.2.1. In both cases, we have $\Delta \rhd M \approx_{ax} M' : C$. We want to show that if $\Delta' <: \Delta$ and $C <: C'$ then $\Delta' \rhd \{M <: C'\} \approx_{ax} \{M' <: C'\} : C'$.

The fact that $\Delta' \rhd \{M <: C'\} : C'$ and that $\Delta' \rhd \{M' <: C'\} : C'$ is proved using Lemma 9.1.14 and Lemma 9.1.26.

The fact that they are axiomatically equivalent is proved as follows.

Case ($app_{<:}$). We have $M = N\{P <: B\}$ and $M' = \{N <: A \multimap C\}P$.

Consider $\{M <: C'\}$. From Definition 9.1.20, it is equal to $\{N <: B \multimap C'\}\{P <: B\}$. Using rule ($app_{<:}$), we infer that $\Delta \rhd \{M <: C'\} \approx_{ax} \{\{N <: B \multimap C'\} <: A \multimap C'\}P : C'$. Using Lemma 9.1.22, we deduce that $\Delta' \rhd \{N <: A \multimap C'\}P : C'$. We conclude by noticing that from Definition 9.1.20, $\{M' <: C'\} = \{N <: A \multimap C'\}P$.

Case ($let^{\otimes}_{<:}$). Here, $M = (let\ \langle x^{A'}, y^{B'}\rangle^{n'} = \{N <: !^{n'}(A' \otimes B')\}\, in\, P)$ and $M' = (let\ \langle x^A, y^B\rangle^n = N\ in\ P)$.

The term $\{M <: C'\}$ is equal to $let\ \langle x^{A'}, y^{B'}\rangle^{n'} = \{N <: !^{n'}(A' \otimes B')\}\ in\ \{P <: C'\}$ by Definition 9.1.20. The term $\{M' <: C'\}$ is equal from the same definition to the term $let\ \langle x^A, y^B\rangle^n = N\ in\ \{P <: C'\}$.

They are axiomatically equivalent because of the equivalence rule ($let^{\otimes}_{<:}$).

And this ends the base cases.

Rules specific to equivalence relations. The reflexivity, the symmetry, the transitivity are trivial.

Reflexivity. In this case, $M = M'$. Therefore $\{M <: A'\} = \{M' <: A'\}$. One has that $\Delta' \rhd \{M <: A'\} \approx_{ax} \{M' <: A'\} : A'$ by reflexivity.

Symmetry. Here, $\Delta \rhd M \approx_{ax} M' : A'$ was proved by first proving $\Delta \rhd M' \approx_{ax} M : A'$. By induction hypothesis, we conclude that $\Delta' \rhd \{M' <: A'\} \approx_{ax} \{M <: A'\} : A'$. We then apply again symmetry.

Transitivity. Here, $\Delta \rhd M \approx_{ax} M' : A'$ was proved introducing a third term M'' such that $\Delta \rhd M \approx_{ax} M'' : A$ and $\Delta \rhd M'' \approx_{ax} M' : A$. By induction hypothesis, this means that $\Delta' \rhd \{M <: A'\} \approx_{ax} \{M'' <: A'\} : A'$ and $\Delta \rhd \{M'' <: A'\} \approx_{ax} \{M' <: A'\} : A'$. We can then apply transitivity to get the result.

Congruence rules. There is one congruence rule per term constructor. We list the cases in the following.

Case $M \equiv \lambda^n x^C.N$. Then $M' \equiv \lambda^n x^C.N'$, the type A is of the form $!^n(C \multimap D)$ and the type A' is of the form $!^n(C \multimap D)$ with $D <: D'$. We have $\Delta, x : C \rhd N \approx_{ax} N' : D$, and Δ is of the form $!\dot{\Delta}$ if $n \neq 0$.

By induction hypothesis, $\Delta', x : C \rhd \{N <: D'\} \approx_{ax} \{N' <: D'\} : D'$. It is then possible to apply the congruence rule for (λ_i) to get the result.

Cases $M \equiv N_1 N_2$, $\langle N_1, N_2\rangle^n$, $(\mathit{let} * = N_1 \mathit{ in } N_2)$ *and* $(\mathit{let}\ \langle y^C, z^D\rangle^n = N_1 \mathit{ in } N_2)$.

In each case we use the shortcut notation $f(N_1, N_2)$ in place of the corresponding construct. Then $M' \equiv f(N_1', N_2')$ for some terms N_1' and N_2', and the typing judgements $\Delta \rhd M, M' : A$ comes from

$$!\dot{\Delta}, \Gamma_1 \rhd N_1 : B_1, \qquad !\dot{\Delta}, \Gamma_2, \Lambda \rhd N_2 : B_2,$$
$$!\dot{\Delta}, \Gamma_1 \rhd N_1' : B_1, \qquad !\dot{\Delta}, \Gamma_2, \Lambda \rhd N_2' : B_2,$$

and from the application of a typing rule (X), where $\Delta = (!\dot{\Delta}, \Gamma_1, \Gamma_2)$, and where

if $f(N_1, N_2)$ is...	(X) is...	A is...	B_1 is...	B_2 is...
$N_1 N_2$	(app)	D	$C \multimap D$	C
$\langle N_1, N_2\rangle^n$	$(\otimes.I)$	$!^n(C \otimes D)$	$!^n C$	$!^n D$
$\mathit{let} * = N_1 \mathit{ in } N_2$	$(\top.E)$	D	$\top$	D
$\mathit{let}\ \langle y^C, z^D\rangle^n = N_1 \mathit{ in } N_2$	$(\otimes.E)$	E	$!^n(C \otimes D)$	E

for some types C, D and E, and where Λ is empty in the three first cases and equal to $(y : !^n C, z : !^n D)$ in the last case. By α-equivalence one can assume that in the last case, the variables y and z in Λ does not occur anywhere in the other contexts.

The type A' is such that $A <: A'$. Its structure is then governed by the one of A. Let us construct B_1' and B_2' as follows:

if $f(N_1, N_2)$ is...	A' is...	B_1' is...	B_2' is...
$N_1 N_2$	D'	$C \multimap D'$	C
$\langle N_1, N_2\rangle^n$	$!^m(C' \otimes Dm)$	$!^m C'$	$!^m D'$
$\mathit{let} * = N_1 \mathit{ in } N_2$	D'	$\top$	D'
$\mathit{let}\ \langle y^C, z^D\rangle^n = N_1 \mathit{ in } N_2$	E'	$!^n(C \otimes D)$	E'

for some types C', D' and E' such that $C <: C'$, $D <: D'$ and $E <: E'$.

Note that in each case, $B_1 <: B_1'$ and $B_2 <: B_2'$. One can therefore apply the induction hypothesis, and get that

$$!\dot{\Delta}', \Gamma_1' \rhd \{N_1 <: B_1'\} : B_1', \qquad !\dot{\Delta}', \Gamma_2', \Lambda \rhd \{N_2 <: B_2'\} : B_2',$$
$$!\dot{\Delta}', \Gamma_1' \rhd \{N_1' <: B_1'\} : B_1', \qquad !\dot{\Delta}', \Gamma_2', \Lambda \rhd \{N_2' <: B_2'\} : B_2',$$

are valid, where $\Delta' = (!\dot{\Delta}', \Gamma_1', \Gamma_2')$, where $!\dot{\Delta}' <: !\dot{\Delta}$, $\Gamma_1' <: \Gamma_1$ and $\Gamma_2' <: \Gamma_2$.

One can apply the congruence rule for (X) and get that

$$\Delta' \rhd \{N_1 <: B_1'\}\{N_2 <: B_2'\} \approx_{ax} \{N_1' <: B_1'\}\{N_2' <: B_2'\}' : A'.$$

Using Definition 9.1.20 and Lemma 9.1.23, this means that $\Delta' \rhd \{M <: A'\} \approx_{ax} \{M' <: A'\} : A'$.

and this closes the list of possible cases. □

Lemma 9.2.6. *If* $!\Delta, \Gamma_1, x : A \rhd M : B$ *is a valid typing judgement, and if* $!\Delta, \Gamma_2 \rhd V \approx_{ax} V' : A$ *are axiomatically equivalent core values such that* $|\Gamma_1| \cap |\Gamma_2| = \emptyset$*, then* $!\Delta, \Gamma_1, \Gamma_2 \rhd M[V/x] \approx_{ax} M[V'/x] : B$ *are axiomatically equivalent typing judgements.*

Proof. First note that $!\Delta, \Gamma_1, \Gamma_2 \rhd M[V/x] : B$ and $!\Delta, \Gamma_1, \Gamma_2 \rhd M[V'/x] :$ are valid typing judgements from Lemma 9.1.33. We prove that they are axiomatically equivalent by induction on the size of M.

Cases $M \equiv y^C$, c^C *and* $*^n$. In these cases, either $M[V/x] = M$ and $M[V'/x] = M$ or $M[V/x] = \{V : A <: C\}$ and $M[V'/x] = \{V' : A <: C\}$ In the former case we have axiomatic equivalence be reflexivity of the relation and in the latter case we have it using Lemma 9.2.5.

Case $M \equiv \lambda^n y^C.N$. Note that since $!\Delta, \Gamma_1, x : A \rhd M : B$ is well-typed, we have $x \neq y$ and $B = !^n(C \multimap D)$ for some type D. In particular, $M[V/x] = \lambda^n y^C.(N[V/x])$ and $M[V'/x] = \lambda^n y^C.(N[V'/x])$.

The typing tree of the typing judgement starts with typing rule (λ_i) and has for hypothesis $!\Delta, \Gamma_1, x : A, y : C \rhd N : D$. The integer $i = 1$ if $n = 0$, otherwise $i = 2$. In the latter, $(\Gamma_1, x : A)$ is of the form $(!\dot{\Gamma}_1, x : !\dot{A})$.

By induction hypothesis, $!\Delta, \Gamma_1, y : C \rhd N[V/x] \approx_{ax} N[V'/x] : D$. Applying congruence, we get the result.

The remaining cases are done similarly. □

The following result stipulates that all the indexations of a given erasure live in the same axiomatic class. In other words, the axiomatic equivalence class of a term is independent of its indexation.

Theorem 9.2.7. *If* $Erase(M) = Erase(M')$ *and if* $\Delta \rhd M, M' : A$ *are valid typing judgements, then* $M \approx_{ax} M'$.

The proof of Theorem 9.2.7 requires some machinery that we develop in Chapter 10.

9.3 The Category $\mathcal{C}_\lambda$

The language of Section 9.1 together with its axiomatic equivalence can be made into a syntactic category.

Lemma 9.3.1. *We can define a category as follows: Objects are types, and arrows* $A \to B$ *are axiomatic classes of valid typing judgements of the form* $x : A \rhd V : B$*, where* V *is an extended value. We define the composition of arrows* $x : A \rhd V : B$ *and* $y : B \rhd W : C$ *to be* $x : A \rhd let\ y = V\ in\ W : C$*. The identity on* A *is set to be the arrow* $x : A \rhd x : A$.

Proof. The composition of two arrows yields an arrow axiomatically equivalent to a value due to Axiom (β_λ) and Lemmas 9.1.32 and 9.1.33. Composition is associative due to Axiom (let_1). The arrow $x : A \rhd x : A$ is indeed the identity on A due to axioms (α_{let}) and (β^2_λ). □

Definition 9.3.2. The category described in Lemma 9.3.1 will be referred as $\mathcal{C}_\lambda$.

9.3.1 Monoidal Structure

Lemma 9.3.3. *If we equip $\mathcal{C}_\lambda$ with the arrows*

$$\begin{aligned}
\alpha_{A,B,C} = \quad & x : A \otimes (B \otimes C) \rhd \mathit{let}\ \langle y, z\rangle = x\ \mathit{in} \\
& \qquad \mathit{let}\ \ \langle t, u\rangle = z\ \mathit{in}\ \langle\langle y, t\rangle, u\rangle : (A \otimes B) \otimes C, \\
\lambda_A = \quad & x : \top \otimes A \rhd \mathit{let}\ \ \langle y, z\rangle = x\ \mathit{in}\ \mathit{let} * = y\ \mathit{in}\ z : A, \\
\rho_A = \quad & x : A \otimes \top \rhd \mathit{let}\ \ \langle y, z\rangle = x\ \mathit{in}\ \mathit{let} * = z\ \mathit{in}\ y : A, \\
\sigma_{A,B} = \quad & x : A \otimes B \rhd \mathit{let}\ \ \langle y, z\rangle = x\ \mathit{in}\ \langle z, y\rangle : B \otimes A,
\end{aligned}$$

and with the map of arrows $(x : A \rhd V : B) \otimes (y : C \rhd W : D) = z : A \otimes B \rhd \mathit{let}\ \langle x, y\rangle = z\ \mathit{in}\ \langle V, W\rangle : C \otimes D$, then $(\mathcal{C}_\lambda, \otimes, \top)$ is a symmetric monoidal category.

Proof. We need to show that α, λ, ρ and σ are isomorphic natural transformations, that they satisfy Equations (2.7.1), (2.7.2), (2.7.3), (2.7.4), (2.7.5) and (2.7.6), and that the constructor $\otimes$ is a bifunctor.

Isomorphisms. Let's show that α, λ, ρ and σ are isomorphisms. It is sufficient to exhibit their inverse:

$$\begin{aligned}
\alpha^{-1}_{A,B,C} = \quad & x' : (A \otimes B) \otimes C \rhd \mathit{let}\ \langle y', z'\rangle = x'\ \ \mathit{in}\ \mathit{let}\ \langle t', u'\rangle = y'\mathit{in} \\
& \qquad \langle t', \langle u', z'\rangle\rangle : A \otimes (B \otimes C), \\
\lambda^{-1}_A = \quad & x' : A \rhd \langle *, x'\rangle : \top \otimes A, \\
\rho_A = \quad & x' : A \rhd \langle x', *\rangle : A \otimes \top, \\
\sigma_{A,B} = \quad & x' : B \otimes A \rhd \mathit{let}\ \ \langle y', z'\rangle = x'\ \mathit{in}\ \langle z', y'\rangle : A \otimes B,
\end{aligned}$$

Naturality of α, λ, ρ and σ. Let's consider λ. Let $f = a : A \rhd V : A'$ be any arrow $A \to A'$. We need to show that the diagram

$$\begin{tikzcd}
\top \otimes A \arrow[r, "\lambda_A"] \arrow[d, "\top \otimes f"'] & A \arrow[d, "f"] \\
\top \otimes A \arrow[r, "\lambda_{A'}"] & A'
\end{tikzcd}$$

commutes, i.e. $\lambda_A; f = (\top \otimes f); \lambda_{A'}$. The arrow $\top \otimes f$ is $b : \top \otimes A \rhd \mathit{let}\ \langle t, a\rangle = b\, \mathit{in}\, \langle t, V\rangle : \top \otimes A'$. So the left hand side of the equation is

$$\begin{aligned}
b : \top \otimes A \rhd \mathit{let}\ \ & x = (\mathit{let}\ \langle t, a\rangle = b\ \mathit{in}\ \langle t, V\rangle)\ \mathit{in} \\
& \mathit{let}\ \ \langle y, z\rangle = x\ \mathit{in}\ \mathit{let} * = y\ \mathit{in}\ z \\
\approx_{ax} \mathit{let}\ \ & \langle t, a\rangle = b\ \ \mathit{in}\ \mathit{let}\ x = \langle t, V\rangle\ \mathit{in} \\
& \mathit{let}\ \ \langle y, z\rangle = x\ \mathit{in}\ \mathit{let} * = y\ \mathit{in}\ z && \text{by } (\mathit{let}_1) \\
\approx_{ax} \mathit{let}\ \ & \langle t, a\rangle = b\ \ \mathit{in}\ \mathit{let}\ \langle y, z\rangle = \langle t, V\rangle\ \mathit{in}\ \mathit{let} * = y\ \mathit{in}\ z && \text{by } (\beta_\lambda) \\
\approx_{ax} \mathit{let}\ \ & \langle t, a\rangle = b\ \ \mathit{in}\ \mathit{let} * = t\ \mathit{in}\ V && \text{by } (\beta_\otimes).
\end{aligned}$$

The right hand side of the equation is

$$\begin{aligned}
x : \top \otimes A \rhd \mathit{let}\ \ & a = (\mathit{let}\ \langle y, z\rangle = x\ \mathit{in}\ \mathit{let} * = y\ \mathit{in}\ z)\ \ \mathit{in}\ \ V \\
\approx_{ax} \mathit{let}\ \ & \langle y, z\rangle = x\ \ \mathit{in}\ \mathit{let} * = y\ \ \mathit{in}\ \mathit{let}\ a = z\ \ \mathit{in}\ \ V : A' && \text{by } (\mathit{let}_1) \\
\approx_{ax} \mathit{let}\ \ & \langle y, a\rangle = x\ \ \mathit{in}\ \mathit{let} * = y\ \ \mathit{in}\ \ V : A' && \text{by } (\alpha_{\mathit{let}}).
\end{aligned}$$

Both side are then alpha equivalent, which proves that λ is a natural transformation.

The proof of naturality of ρ is exactly similar.

For $\sigma_{A,B}$, consider two arrows $f = a : A \rhd V : A'$ and $g = b : B \rhd W : B'$. We aim to show that

$$\begin{tikzcd} A \otimes B \arrow[r, "\sigma_{A,B}"] \arrow[d, "f \otimes g"'] & B \otimes A \arrow[d, "g \otimes f"] \\ A' \otimes B' \arrow[r, "\sigma_{A',B'}"'] & B' \otimes A' \end{tikzcd}$$

commutes, i.e. $\sigma_{A,B}; (g \otimes f) = (f \otimes g); \sigma_{A',B'}$. The left hand side is the arrow:

$$\begin{aligned} x : A \otimes B \rhd & \; let \;\; c = (let \; \langle y, z \rangle = x \; in \; \langle z, y \rangle) \; in \\ & \; let \;\; \langle b, a \rangle = c \; in \; \langle W, V \rangle \\ \approx_{ax} & \; let \;\; \langle y, z \rangle = x \;\; in \; let \; c = \langle z, y \rangle \;\; in \\ & \; let \;\; \langle b, a \rangle = c \; in \; \langle W, V \rangle && \text{by } (let_1) \\ \approx_{ax} & \; let \;\; \langle y, z \rangle = x \;\; in \; let \; \langle b, a \rangle = \langle z, y \rangle \; in \; \langle W, V \rangle && \text{by } (\beta_\lambda) \\ \approx_{ax} & \; let \;\; \langle a, b \rangle = x \;\; in \; \langle W, V \rangle : B' \otimes A' && \text{by } (\alpha_{let}). \end{aligned}$$

The right hand side is

$$\begin{aligned} x : A \otimes B \rhd & \; let \;\; c = (let \; \langle a, b \rangle = x \; in \; \langle V, W \rangle) \; in \\ & \; let \;\; \langle y, z \rangle = c \; in \; \langle z, y \rangle \\ \approx_{ax} & \; let \;\; \langle a, b \rangle = x \;\; in \; let \; c = \langle V, W \rangle \;\; in \\ & \; let \;\; \langle y, z \rangle = c \; in \; \langle z, y \rangle && \text{by } (let_1) \\ \approx_{ax} & \; let \;\; \langle a, b \rangle = x \;\; in \; let \; \langle y, z \rangle = \langle V, W \rangle \; in \; \langle z, y \rangle && \text{by } (\beta_\lambda) \\ \approx_{ax} & \; let \;\; \langle a, b \rangle = x \;\; in \;\; \langle W, V \rangle : B' \otimes A' && \text{by } (\beta_\otimes), \end{aligned}$$

which is axiomatically equivalent to the left hand side.

To prove the naturality of α, let's choose three functions $f = a : A \rhd V : A'$, $g = b : B \rhd W : B'$ and $h : c : C \rhd U : C'$. It is enough to show that

$$\begin{tikzcd} A \otimes (B \otimes C) \arrow[r, "\alpha_{A,B,C}"] \arrow[d, "f \otimes (g \otimes h)"'] & (A \otimes B) \otimes C \arrow[d, "(f \otimes g) \otimes h"] \\ A' \otimes (B' \otimes C') \arrow[r, "\alpha_{A',B',C'}"'] & (A' \otimes B') \otimes C' \end{tikzcd}$$

commutes, i.e. that $\alpha_{A,B,C}; ((f \otimes g) \otimes h) = (f \otimes (g \otimes h)); \alpha_{A',B',C'}$.

Let's write down the components of this equation:

$$\begin{aligned} f \otimes g = \quad & d : A \otimes B \rhd let \; \langle a, b \rangle = d \;\; in \; \langle V, W \rangle : A' \otimes B' \\ g \otimes h = \quad & i : B \otimes C \rhd let \; \langle b, c \rangle = i \;\; in \; \langle W, U \rangle : B' \otimes C' \end{aligned}$$

Then

$$(f \otimes g) \otimes h \qquad e : (A \otimes B) \otimes C \rhd$$

$$
\begin{aligned}
& \mathit{let}\ \langle d,c\rangle = e\ \ \mathit{in} \left\langle \begin{array}{c} \mathit{let}\ \langle a,b\rangle = d \\ \mathit{in}\langle V,W\rangle \end{array} ,\ U \right\rangle \\
\approx_{ax}\ & \mathit{let}\ \langle d,c\rangle = e\ \ \mathit{in}\ \mathit{let}\ \langle a,b\rangle = d\ \mathit{in} \\
& \langle\langle V,W\rangle,U\rangle : (A'\otimes B')\otimes C' && \text{by } (\mathit{let}_2^{\otimes})
\end{aligned}
$$

$$
\begin{aligned}
f\otimes(g\otimes h) \qquad & j : A\otimes(B\otimes C) \rhd \\
& \mathit{let}\ \langle a,i\rangle = j\ \ \mathit{in} \left\langle V,\ \begin{array}{c} \mathit{let}\ \langle b,c\rangle = i\ \mathit{in} \\ \langle W,U\rangle \end{array} \right\rangle \\
\approx_{ax}\ & \mathit{let}\ \langle a,i\rangle = j\ \ \mathit{in}\ \mathit{let}\ \langle b,c\rangle = i\ \mathit{in} \\
& \langle V,\langle W,U\rangle\rangle : A'\otimes(B'\otimes C') && \text{by } (\mathit{let}_1^{\otimes}).
\end{aligned}
$$

Finally,

$$
\begin{aligned}
\alpha_{A,B,C} = \qquad & x : A\otimes(B\otimes C) \rhd \mathit{let}\ \langle x_1,x_2\rangle = x\ \mathit{in} \\
& \qquad \mathit{let}\ \langle x_3,x_4\rangle = x_2\ \mathit{in} \\
& \qquad \langle\langle x_1,x_3\rangle,x_4\rangle : (A\otimes B)\otimes C \\
\alpha_{A',B',C'} = \qquad & y : A'\otimes(B'\otimes C') \rhd \mathit{let}\ \langle y_1,y_2\rangle = y\ \mathit{in} \\
& \qquad \mathit{let}\ \langle y_3,y_4\rangle = y_2\ \mathit{in} \\
& \qquad \langle\langle y_1,y_3\rangle,y_4\rangle : (A'\otimes B')\otimes C'
\end{aligned}
$$

Now let's consider the left hand side of the equation:

$$
\begin{aligned}
x : A\otimes(B\otimes C) \rhd \mathit{let}\ \ e = & \left(\begin{array}{l} \mathit{let}\ \langle x_1,x_2\rangle = x\ \mathit{in} \\ \mathit{let}\ \langle x_3,x_4\rangle = x_2\ \mathit{in} \\ \langle\langle x_1,x_3\rangle,x_4\rangle \end{array} \right)\ \mathit{in} \\
& \mathit{let}\ \langle d,c\rangle = e\ \mathit{in} \\
& \mathit{let}\ \langle a,b\rangle = d\ \ \mathit{in}\ \langle\langle V,W\rangle,U\rangle \\
\approx_{ax}\ & \mathit{let}\ \langle x_1,x_2\rangle = x\ \ \mathit{in}\ \mathit{let}\ \langle x_3,x_4\rangle = x_2\ \mathit{in} \\
& \mathit{let}\ \ e = \langle\langle x_1,x_3\rangle,x_4\rangle\ \mathit{in} \\
& \mathit{let}\ \langle d,c\rangle = e\ \mathit{in} \\
& \mathit{let}\ \langle a,b\rangle = d\ \ \mathit{in}\ \langle\langle V,W\rangle,U\rangle && \text{by } (\mathit{let}_1) \\
\approx_{ax}\ & \mathit{let}\ \langle x_1,x_2\rangle = x\ \ \mathit{in}\ \mathit{let}\ \langle x_3,x_4\rangle = x_2\ \mathit{in} \\
& \mathit{let}\ \langle d,c\rangle = \langle\langle x_1,x_3\rangle,x_4\rangle\ \mathit{in} \\
& \mathit{let}\ \langle a,b\rangle = d\ \ \mathit{in}\ \langle\langle V,W\rangle,U\rangle && \text{by } (\beta_\lambda) \\
\approx_{ax}\ & \mathit{let}\ \langle x_1,x_2\rangle = x\ \ \mathit{in}\ \mathit{let}\ \langle x_3,x_4\rangle = x_2\ \mathit{in} \\
& \mathit{let}\ \langle a,b\rangle = \langle x_1,x_3\rangle\ \ \mathit{in}\ \langle\langle V,W\rangle,U\rangle[x_4/c] && \text{by } (\beta_\otimes) \\
\approx_{ax}\ & \mathit{let}\ \langle x_1,x_2\rangle = x\ \ \mathit{in}\ \mathit{let}\ \langle x_3,x_4\rangle = x_2\ \mathit{in} \\
& \langle\langle V,W\rangle,U\rangle[x_1/a,x_3/b,x_4/c] && \text{by } (\beta_\otimes) \\
\approx_{ax}\ & \mathit{let}\ \langle a,x_2\rangle = x\ \ \mathit{in}\ \mathit{let}\ \langle b,c\rangle = x_2\ \mathit{in} \\
& \langle\langle V,W\rangle,U\rangle : (A'\otimes B')\otimes C' && \text{by } (\alpha_{\mathit{let}}).
\end{aligned}
$$

The right hand side is

$$
\begin{aligned}
j : A \otimes (B \otimes C) \rhd \mathit{let}\ \ y = & \begin{pmatrix} \mathit{let}\ \langle a, i\rangle = j\ \mathit{in} \\ \mathit{let}\ \langle b, c\rangle = i\ \mathit{in} \\ \langle V, \langle W, U\rangle\rangle \end{pmatrix}\ \mathit{in} \\
& \mathit{let}\ \ \langle y_1, y_2\rangle = y\ \mathit{in} \\
& \mathit{let}\ \ \langle y_3, y_4\rangle = y_2\ \ \mathit{in}\ \langle\langle y_1, y_3\rangle, y_4\rangle \\
\approx_{ax} & \mathit{let}\ \ \langle a, i\rangle = j\ \ \mathit{in}\ \mathit{let}\ \langle b, c\rangle = i\ \mathit{in} \\
& \mathit{let}\ \ y = \langle V, \langle W, U\rangle\rangle\ \mathit{in} \\
& \mathit{let}\ \ \langle y_1, y_2\rangle = y\ \mathit{in} \\
& \mathit{let}\ \ \langle y_3, y_4\rangle = y_2\ \mathit{in} \\
& \langle\langle y_1, y_3\rangle, y_4\rangle && \text{by } (\mathit{let}_1) \\
\approx_{ax} & \mathit{let}\ \ \langle a, i\rangle = j\ \ \mathit{in}\ \mathit{let}\ \langle b, c\rangle = i\ \mathit{in} \\
& \langle\langle V, W\rangle, U\rangle : (A' \otimes B') \otimes C' && \text{by } (\beta_\lambda),(\beta_\otimes) \text{ twice,}
\end{aligned}
$$

which is axiomatically equivalent to the left hand side. Thus α is a natural transformation.

Bifunctoriality of $\otimes$. First we need to show that $\otimes$ is well defined on equivalence classes. Given $x : A \rhd V \approx_{ax} V' : B$ and $x : C \rhd W \approx_{ax} W' : D$:

$$
\begin{aligned}
z : A \otimes C \rhd \mathit{let}\ \langle x, y\rangle = z\ \mathit{in}\ \langle V, W\rangle & \\
\approx_{ax} \mathit{let}\ \langle x, y\rangle = z\ \mathit{in}\ \langle V', W'\rangle : B \otimes D &
\end{aligned}
$$

since $\langle V, W\rangle \approx_{ax} \langle V', W'\rangle$.

Then we need to show that $(- \otimes B)$ and $(A \otimes -)$ are functors, and that they verify

$$(f \otimes B); (A' \otimes g) = (A \otimes g); (f \otimes B')$$

for every $f : A \to A'$ and every $g : B \to B'$.

$(id_A \otimes id_B)$ is $(x : A \rhd x : A) \otimes (y : B \rhd y : B)$, which is by definition $z : A \otimes B \rhd \mathit{let}\ \langle x, y\rangle = z\ \mathit{in}\ \langle x, y\rangle : A \otimes B$, which is by Rule $(\eta_\otimes)$ the identity $z : A \otimes B \rhd z : A \otimes B$. So both $(- \otimes B)$ and $(A \otimes -)$ preserves identities.

Let f be $x : A \to V : A'$ and f' be $y : A' \to W : A''$. $((f, f') \otimes B)$ is

$$
\begin{aligned}
z : A \otimes B \rhd \mathit{let}\ \langle x, t\rangle = z\ \ \mathit{in}\ \langle \mathit{let}\ y = V\ \ \mathit{in}\ \ W, t\rangle & \\
\approx_{ax} \mathit{let}\ \langle x, t\rangle = z\ \ \mathit{in}\ \mathit{let}\ y = V\ \ \mathit{in}\ \ \langle W, t\rangle & \qquad \text{by } (\mathit{let}_2^{\otimes}).
\end{aligned}
$$

On another hand, $(f \otimes B); (f' \otimes B)$ is

$$
\begin{aligned}
z : A \otimes B \rhd & \mathit{let}\ \ u = (\mathit{let}\ \langle x, t\rangle = z\ \ \mathit{in}\ \langle V, t\rangle)\mathit{in} \\
& \mathit{let}\ \ \langle y, v\rangle = u\ \ \mathit{in}\ \langle W, v\rangle \\
\approx_{ax} & \mathit{let}\ \ \langle x, t\rangle = z\ \ \mathit{in}\ \mathit{let}\ u = \langle V, t\rangle \mathit{in} \\
& \mathit{let}\ \ \langle y, v\rangle = u\ \ \mathit{in}\ \langle W, v\rangle && \text{by } (\mathit{let}_1) \\
\approx_{ax} & \mathit{let}\ \ \langle x, t\rangle = z\ \ \mathit{in}\ \mathit{let}\ \langle y, v\rangle = \langle V, t\rangle\ \ \mathit{in}\ \langle W, v\rangle && \text{by } (\beta_\lambda) \\
\approx_{ax} & \mathit{let}\ \ \langle x, t\rangle = z\ \ \mathit{in}\ \mathit{let}\ y = V\ \ \mathit{in}
\end{aligned}
$$

$$\begin{aligned}
& \mathit{let}\ \ v = t\ \ \mathit{in}\ \langle W, v\rangle && \text{by } (\beta_\otimes),(\beta_\lambda)\\
\approx_{ax}\ & \mathit{let}\ \ \langle x, t\rangle = z\ \ \mathit{in}\ \mathit{let}\ y = V\ \ \mathit{in}\ \langle W, t\rangle && \text{by } (\beta_\lambda),
\end{aligned}$$

and thus the two morphisms are equals.

Using $(let_1^\otimes)$ in place of $(let_2^\otimes)$, we can show that $(A \otimes (g; g')) = (A \otimes g); (A \otimes g')$, for $g : B \to B'$ and $g' : B' \to B''$.

Finally, we must show that if f is $x : A \rhd V : A'$ and g is $y : B \rhd U : B'$, then $(f \otimes B); (A' \otimes g) = (A \otimes g); (f \otimes B')$. The left hand side is

$$\begin{aligned}
z : A \otimes B \rhd\ & \mathit{let}\ \ z' = (\mathit{let}\langle x, t\rangle = z\ \ \mathit{in}\ \langle V, t\rangle)\mathit{in}\\
& \mathit{let}\ \langle u, y\rangle = z'\ \ \mathit{in}\ \langle u, W\rangle\\
\approx_{ax}\ & \mathit{let}\ \langle x, t\rangle = z\ \ \mathit{in}\ \mathit{let}\ z' = \langle V, t\rangle\ \mathit{in}\\
& \mathit{let}\ \langle u, y\rangle = z'\ \ \mathit{in}\ \langle u, W\rangle && \text{by } (\mathit{let}_1)\\
\approx_{ax}\ & \mathit{let}\ \langle x, t\rangle = z\ \ \mathit{in}\ \mathit{let}\langle u, y\rangle = \langle V, t\rangle\ \ \mathit{in}\ \langle u, W\rangle && \text{by } (\beta_\lambda)\\
\approx_{ax}\ & \mathit{let}\ \langle x, t\rangle = z\ \ \mathit{in}\ \mathit{let}\ u = V\ \ \mathit{in}\\
& \mathit{let}\ \ y = t\ \ \mathit{in}\ \langle u, W\rangle && \text{by } (\beta_\otimes), (\beta_\lambda)\\
\approx_{ax}\ & \mathit{let}\ \langle x, y\rangle = z\ \ \mathit{in}\ \langle V, W\rangle && \text{by } (\beta_\lambda), (\alpha_{let}).
\end{aligned}$$

The right hand side is

$$\begin{aligned}
z : A \otimes B \rhd\ & \mathit{let}\ \ z' = (\mathit{let}\langle u, y\rangle = z\ \ \mathit{in}\ \langle u, W\rangle)\mathit{in}\\
& \mathit{let}\ \langle x, t\rangle = z'\ \ \mathit{in}\ \langle V, t\rangle
\end{aligned}$$

and by (let_1), (β_λ), $(\beta_\otimes), (\beta_\lambda)$, (β_λ) and (α_{let}), this is axiomatically equivalent to $\mathit{let}\langle x, y\rangle = z\ \ \mathit{in}\ \langle V, W\rangle$. Thus the required equality hold, and $\otimes$ is bifunctorial.

Commutative diagrams. We need to verify Equations (2.7.1), (2.7.2), (2.7.3), (2.7.4), (2.7.5) and Equation (2.7.6).

Equations (2.7.1). We want to prove that

$$\alpha_{A,B,C\otimes D}; \alpha_{A\otimes B,C,D} = (id_A \otimes \alpha_{B,C\otimes D}); \alpha_{A,B\otimes C,D}; \alpha_{A,B\otimes C} \otimes id_D.$$

For the left hand side, the components are as follows:

$$\begin{aligned}
\alpha_{A,B,C\otimes D} = x : A \otimes (B \otimes (C \otimes D)) \rhd\ & \mathit{let}\ \ \langle x_1, x_2\rangle = x\ \mathit{in}\\
& \mathit{let}\ \ \langle x_3, x_4\rangle = x_2\ \mathit{in}\\
& \langle\langle x_1, x_3\rangle, x_4\rangle : (A \otimes B) \otimes (C \otimes D)),\\
\alpha_{A\otimes B,C,D} = y : (A \otimes B) \otimes (C \otimes D)) \rhd\ & \mathit{let}\ \ \langle y_1, y_2\rangle = y\ \mathit{in}\\
& \mathit{let}\ \ \langle y_3, y_4\rangle = y_2\ \mathit{in}\\
& \langle\langle y_1, y_3\rangle, y_4\rangle : ((A \otimes B) \otimes C) \otimes D)).
\end{aligned}$$

Then

$$\alpha_{A,B,C\otimes D}; \alpha_{A\otimes B,C,D} = x : A \otimes (B \otimes (C \otimes D)) \rhd$$
$$\mathit{let}\ y = \begin{pmatrix} \mathit{let}\ \langle x_1, x_2\rangle = x\ \mathit{in}\\ \mathit{let}\ \langle x_3, x_4\rangle = x_2\ \mathit{in}\\ \langle\langle x_1, x_3\rangle, x_4\rangle \end{pmatrix}\ \mathit{in} \begin{pmatrix} \mathit{let}\ \langle y_1, y_2\rangle = y\ \mathit{in}\\ \mathit{let}\ \langle y_3, y_4\rangle = y_2\ \mathit{in}\\ \langle\langle y_1, y_3\rangle, y_4\rangle \end{pmatrix}$$

$$
\begin{aligned}
\approx_{ax}\ & let\ \ \langle x_1, x_2\rangle = x\ \ in\ let\ \langle x_3, x_4\rangle = x_2\ \ in \\
& let\ \ y = \langle\langle x_1, x_3\rangle, x_4\rangle\ \ in\ let\ \langle y_1, y_2\rangle = y\ \ in \qquad \text{by } (let_1) \\
& let\ \ \langle y_3, y_4\rangle = y_2\ \ in\ \ \langle\langle y_1, y_3\rangle, y_4\rangle \\
\approx_{ax}\ & let\ \ \langle x_1, x_2\rangle = x\ \ in\ let\ \langle x_3, x_4\rangle = x_2\ \ in \\
& let\ \ \langle y_1, y_2\rangle = \langle\langle x_1, x_3\rangle, x_4\rangle\ \ in \qquad \text{by } (let_1) \\
& let\ \ \langle y_3, y_4\rangle = y_2\ \ in\ \ \langle\langle y_1, y_3\rangle, y_4\rangle \\
\approx_{ax}\ & let\ \ \langle x_1, x_2\rangle = x\ \ in\ let\ \langle x_3, x_4\rangle = x_2\ \ in \qquad \text{by } (\beta_\otimes) \\
& let\ \ \langle y_3, y_4\rangle = x_4\ \ in\ \langle\langle\langle x_1, x_3\rangle, y_3\rangle, y_4\rangle
\end{aligned}
$$

For the right hand side, the components are as follows:

$$
\begin{aligned}
\alpha_{B,C\otimes D} = \quad & x : B \otimes (C \otimes D) \rhd let\ \ \langle x_1, x_2\rangle = x\ \ in \\
& \qquad let\ \ \langle x_3, x_4\rangle = x_2\ \ in \\
& \qquad \langle\langle x_1, x_3\rangle, x_4\rangle : (B \otimes C) \otimes D, \\
\alpha_{A,B\otimes C} = \quad & y : A \otimes (B \otimes C) \rhd let\ \ \langle y_1, y_2\rangle = y\ \ in \\
& \qquad let\ \ \langle y_3, y_4\rangle = y_2\ \ in \\
& \qquad \langle\langle y_1, y_3\rangle, y_4\rangle : (A \otimes B) \otimes C, \\
\alpha_{A,B\otimes C,D} = \quad & z : A \otimes ((B \otimes C) \otimes D) \rhd let\ \ \langle z_1, z_2\rangle = z\ \ in \\
& \qquad let\ \ \langle z_3, z_4\rangle = z_2\ \ in \\
& \qquad \langle\langle z_1, z_3\rangle, z_4\rangle : (A \otimes (B \otimes C)) \otimes D,
\end{aligned}
$$

and $id_A = t : A \rhd t : A$, $id_D = u : D \rhd u : D$. Then we compute $id_A \otimes \alpha_{B,C\otimes D}$ and $\alpha_{A,B\otimes C} \otimes id_D$:

$$
\begin{aligned}
& id_A \otimes \alpha_{B,C\otimes D} = a : A \otimes (B \otimes (C \otimes D)) \rhd \\
& \qquad let\ \ \langle t, x\rangle = a\ \ in\ \left\langle t,\ \begin{array}{l} let\ \langle x_1, x_2\rangle = x\ in \\ let\ \langle x_3, x_4\rangle = x_2\ in \\ \langle\langle x_1, x_3\rangle, x_4\rangle \end{array} \right\rangle \\
& \approx_{ax} let\ \ \langle t, x\rangle = a\ \ in\ let\ \langle x_1, x_2\rangle = x\ \ in \\
& \qquad let\ \ \langle x_3, x_4\rangle = x_2\ \ in\ \langle t, \langle\langle x_1, x_3\rangle, x_4\rangle\rangle \qquad \text{by } (let_1^\otimes)
\end{aligned}
$$

$$
\begin{aligned}
& \alpha_{A,B\otimes C} \otimes id_D = b : (A \otimes (B \otimes C)) \otimes D \rhd \\
& \qquad let\ \ \langle y, u\rangle = b\ \ in\ \left\langle \begin{array}{l} let\ \langle y_1, y_2\rangle = y\ in \\ let\ \langle y_3, y_4\rangle = y_2\ in \\ \langle\langle y_1, y_3\rangle, y_4\rangle \qquad\qquad , u \end{array} \right\rangle \\
& \approx_{ax} let\ \ \langle y, u\rangle = b\ \ in\ let\ \langle y_1, y_2\rangle = y\ \ in \\
& \qquad let\ \ \langle y_3, y_4\rangle = y_2\ \ in\ \langle\langle\langle y_1, y_3\rangle, y_4\rangle, u\rangle \qquad \text{by } (let_2^\otimes).
\end{aligned}
$$

Let's compose $id_A \otimes \alpha_{B,C\otimes D}$ and $\alpha_{A,B\otimes C,D}$:

$$
\begin{aligned}
& (id_A \otimes \alpha_{B,C\otimes D}); \alpha_{A,B\otimes C,D} = \\
& a : A \otimes (B \otimes (C \otimes D)) \rhd let\ \ z = \left(\begin{array}{l} let\ \langle t, x\rangle = a\ \ in\ let\ \langle x_1, x_2\rangle = x\ \ in \\ let\ \langle x_3, x_4\rangle = x_2\ \ in\ \langle t, \langle\langle x_1, x_3\rangle, x_4\rangle\rangle \end{array} \right)\ \ in \\
& \qquad\qquad let\ \ \langle z_1, z_2\rangle = z\ \ in\ let\ \langle z_3, z_4\rangle = z_2\ \ in\ \langle\langle z_1, z_3\rangle, z_4\rangle
\end{aligned}
$$

$$
\begin{aligned}
\approx_{ax}\ & let\ \ \langle t,x\rangle = a\ \ in\ let\ \langle x_1,x_2\rangle = x\ in \\
& let\ \ \langle x_3,x_4\rangle = x_2\ \ in\ let\ z = \langle t,\langle\langle x_1,x_3\rangle,x_4\rangle\rangle\ in \\
& let\ \ \langle z_1,z_2\rangle = z\ \ in\ let\ \langle z_3,z_4\rangle = z_2\ \ in\ \langle\langle z_1,z_3\rangle,z_4\rangle && \text{by } (let_1) \\
\approx_{ax}\ & let\ \ \langle t,x\rangle = a\ \ in\ let\ \langle x_1,x_2\rangle = x\ \ in\ let\ \langle x_3,x_4\rangle = x_2\ in \\
& let\ \ \langle z_1,z_2\rangle = \langle t,\langle\langle x_1,x_3\rangle,x_4\rangle\rangle\ in \\
& let\ \ \langle z_3,z_4\rangle = z_2\ \ in\ \langle\langle z_1,z_3\rangle,z_4\rangle && \text{by } (\beta_\lambda) \\
\approx_{ax}\ & let\ \ \langle t,x\rangle = a\ \ in\ let\ \langle x_1,x_2\rangle = x\ \ in\ let\ \langle x_3,x_4\rangle = x_2\ in \\
& let\ \ \langle z_3,z_4\rangle = \langle\langle x_1,x_3\rangle,x_4\rangle\ \ in\ \langle\langle t,z_3\rangle,z_4\rangle && \text{by } (\beta_\otimes) \\
\approx_{ax}\ & let\ \ \langle t,x\rangle = a\ \ in\ let\ \langle x_1,x_2\rangle = x\ in \\
& let\ \ \langle x_3,x_4\rangle = x_2\ \ in\ \langle\langle t,\langle x_1,x_3\rangle\rangle,x_4\rangle : (A\otimes(B\otimes C))\otimes D && \text{by } (\beta_\otimes)
\end{aligned}
$$

Now we can compute the right hand of the equation:

$$
\begin{aligned}
& (id_A\otimes\alpha_{B,C\otimes D});\alpha_{A,B\otimes C,D};\alpha_{A,B\otimes C}\otimes id_D = \\
& a : A\otimes(B\otimes(C\otimes D)) \rhd let\ \ b = \left(\begin{array}{l} let\ \langle t,x\rangle = a\ \ in\ let\ \langle x_1,x_2\rangle = x\ in \\ let\ \langle x_3,x_4\rangle = x_2\ \ in\ \langle\langle t,\langle x_1,x_3\rangle\rangle,x_4\rangle \end{array}\right) in \\
& \qquad\qquad let\ \ \langle y,u\rangle = b\ \ in\ let\ \langle y_1,y_2\rangle = y\ in \\
& \qquad\qquad let\ \ \langle y_3,y_4\rangle = y_2\ \ in\ \langle\langle\langle y_1,y_3\rangle,y_4\rangle,u\rangle \\
\approx_{ax}\ & let\ \ \langle t,x\rangle = a\ \ in\ let\ \langle x_1,x_2\rangle = x\ \ in\ let\ \langle x_3,x_4\rangle = x_2\ in \\
& let\ \ b = \langle\langle t,\langle x_1,x_3\rangle\rangle,x_4\rangle\ \ in\ let\ \langle y,u\rangle = b\ in \\
& let\ \ \langle y_1,y_2\rangle = y\ \ in\ let\ \langle y_3,y_4\rangle = y_2\ \ in\ \langle\langle\langle y_1,y_3\rangle,y_4\rangle,u\rangle && \text{by } (let_1) \\
\approx_{ax}\ & let\ \ \langle t,x\rangle = a\ \ in\ let\ \langle x_1,x_2\rangle = x\ \ in\ let\ \langle x_3,x_4\rangle = x_2\ in \\
& let\ \ \langle y,u\rangle = \langle\langle t,\langle x_1,x_3\rangle\rangle,x_4\rangle\ \ in\ let\ \langle y_1,y_2\rangle = y\ in \\
& let\ \ \langle y_3,y_4\rangle = y_2\ \ in\ \langle\langle\langle y_1,y_3\rangle,y_4\rangle,u\rangle && \text{by } (\beta_\lambda) \\
\approx_{ax}\ & let\ \ \langle t,x\rangle = a\ \ in\ let\ \langle x_1,x_2\rangle = x\ \ in\ let\ \langle x_3,x_4\rangle = x_2\ in \\
& let\ \ \langle y_1,y_2\rangle = \langle t,\langle x_1,x_3\rangle\rangle\ in \\
& let\ \ \langle y_3,y_4\rangle = y_2\ \ in\ \langle\langle\langle y_1,y_3\rangle,y_4\rangle,x_4\rangle && \text{by } (\beta_\otimes) \\
\approx_{ax}\ & let\ \ \langle t,x\rangle = a\ \ in\ let\ \langle x_1,x_2\rangle = x\ \ in\ let\ \langle x_3,x_4\rangle = x_2\ in \\
& let\ \ \langle y_3,y_4\rangle = \langle x_1,x_3\rangle\ \ in\ \langle\langle\langle t,y_3\rangle,y_4\rangle,x_4\rangle && \text{by } (\beta_\otimes) \\
\approx_{ax}\ & let\ \ \langle t,x\rangle = a\ \ in\ let\ \langle x_1,x_2\rangle = x\ in \\
& let\ \ \langle x_3,x_4\rangle = x_2\ \ in\ \langle\langle\langle t,x_1\rangle,x_3\rangle,x_4\rangle : ((A\otimes B)\otimes C)\otimes D && \text{by } (\beta_\otimes),
\end{aligned}
$$

and this is alpha-equivalent to $\alpha_{A,B,C\otimes D};\alpha_{A\otimes B,C,D}$. Thus Equation (2.7.1) is verified.

Equation (2.7.2). We want to show that $\alpha_{A,\top,C};(\rho_A\otimes id_C) = id_A\otimes\lambda_C$. We first write down the components of the equation:

$$
\begin{aligned}
\alpha_{A,\top,C} = \quad & a : A\otimes(\top\otimes C) \rhd let\ \ \langle a_1,a_2\rangle = a\ in \\
& \qquad\qquad let\ \ \langle a_3,a_4\rangle = a_2\ in\ \langle\langle a_1,a_3\rangle,a_4\rangle : (A\otimes\top)\otimes C, \\
\lambda_C = \quad & b : \top\otimes C \rhd let\ \ \langle b_1,b_2\rangle = b\ \ in\ let\ * = b_1\ in\ b_2 : C, \\
\rho_A = \quad & c : A\otimes\top \rhd let\ \ \langle c_1,c_2\rangle = c\ \ in\ let\ * = c_2\ in\ c_1 : A,
\end{aligned}
$$

and $id_A = u : A \rhd u : A$ and $id_C = v : C \rhd v : C$. Then we construct $\rho_A\otimes id_C$ and $id_A\otimes\lambda_C$:

$$
\rho_A\otimes id_C = \qquad x : (A\otimes\top)\otimes C \rhd let\ \ \langle c,v\rangle = x\ \ in \left\langle \begin{array}{l} let\ \langle c_1,c_2\rangle = c\ in \\ let\ * = c_2\ in\ c_1 \end{array}, v \right\rangle
$$

$$
\begin{aligned}
& \approx_{ax} \mathit{let}\ \langle c, v\rangle = x\ \ \mathit{in}\ \mathit{let}\ \langle c_1, c_2\rangle = c\ \mathit{in} \\
& \qquad \mathit{let}\ *= c_2\ \ \mathit{in}\ \langle c_1, v\rangle : A \otimes C \quad \text{by } (\mathit{let}_2^{\otimes}), \\
id_A \otimes \lambda_C = \quad & y : A \otimes (\top \otimes C) \rhd \mathit{let}\ \langle u, b\rangle = y\ \ \mathit{in} \left\langle u, \begin{array}{l} \mathit{let}\ \langle b_1, b_2\rangle = b\ \mathit{in} \\ \mathit{let}\ *= b_1\ \mathit{in}\ b_2 \end{array} \right\rangle \\
& \approx_{ax} \mathit{let}\ \langle u, b\rangle = y\ \ \mathit{in}\ \mathit{let}\ \langle b_1, b_2\rangle = b\ \mathit{in} \\
& \qquad \mathit{let}\ *= b_1\ \ \mathit{in}\ \langle u, b_2\rangle : A \otimes C \quad \text{by } (\mathit{let}_1^{\otimes}).
\end{aligned}
$$

We then compose $\alpha_{A,\top,C}$ and $\rho_A \otimes id_C$:

$$
\begin{aligned}
a : A \otimes (\top \otimes C) \rhd \mathit{let}\ x = & \left(\begin{array}{l} \mathit{let}\ \langle a_1, a_2\rangle = a\ \mathit{in} \\ \mathit{let}\ \langle a_3, a_4\rangle = a_2\ \mathit{in}\ \langle\langle a_1, a_3\rangle, a_4\rangle \end{array} \right)\ \mathit{in} \\
& \mathit{let}\ \langle c, v\rangle = x\ \ \mathit{in}\ \mathit{let}\ \langle c_1, c_2\rangle = c\ \mathit{in} \\
& \mathit{let}\ *= c_2\ \ \mathit{in}\ \langle c_1, v\rangle \\
\approx_{ax} & \mathit{let}\ \langle a_1, a_2\rangle = a\ \ \mathit{in}\ \mathit{let}\ \langle a_3, a_4\rangle = a_2\ \mathit{in} \\
& \mathit{let}\ x = \langle\langle a_1, a_3\rangle, a_4\rangle\ \mathit{in} \\
& \mathit{let}\ \langle c, v\rangle = x\ \ \mathit{in}\ \mathit{let}\ \langle c_1, c_2\rangle = c\ \mathit{in} \\
& \mathit{let}\ *= c_2\ \ \mathit{in}\ \langle c_1, v\rangle \qquad \text{by } (\mathit{let}_1) \\
\approx_{ax} & \mathit{let}\ \langle a_1, a_2\rangle = a\ \ \mathit{in}\ \mathit{let}\ \langle a_3, a_4\rangle = a_2\ \mathit{in} \\
& \mathit{let}\ \langle c, v\rangle = \langle\langle a_1, a_3\rangle, a_4\rangle\ \ \mathit{in}\ \mathit{let}\ \langle c_1, c_2\rangle = c\ \mathit{in} \\
& \mathit{let}\ *= c_2\ \ \mathit{in}\ \langle c_1, v\rangle \qquad \text{by } (\beta_\lambda) \\
\approx_{ax} & \mathit{let}\ \langle a_1, a_2\rangle = a\ \ \mathit{in}\ \mathit{let}\ \langle a_3, a_4\rangle = a_2\ \mathit{in} \\
& \mathit{let}\ \langle c_1, c_2\rangle = \langle a_1, a_3\rangle\ \ \mathit{in}\ \mathit{let}\ *= c_2\ \ \mathit{in}\ \langle c_1, a_4\rangle \qquad \text{by } (\beta_\otimes) \\
\approx_{ax} & \mathit{let}\ \langle a_1, a_2\rangle = a\ \ \mathit{in}\ \mathit{let}\ \langle a_3, a_4\rangle = a_2\ \mathit{in} \\
& \mathit{let}\ *= a_3\ \ \mathit{in}\ \langle a_1, a_4\rangle : A \otimes C \qquad \text{by } (\beta_\otimes),
\end{aligned}
$$

which is alpha-equivalent to $id_A \otimes \lambda_C$: Diagram 2.7.2 commutes.

Equation (2.7.3). We want to show $\lambda_\top = \rho_\top$.

$$
\begin{aligned}
\lambda_\top = \quad & x : \top \otimes \top \rhd \mathit{let}\ \langle y, z\rangle = x\ \mathit{in}\ \mathit{let}\ *= y\ \mathit{in}\ z \\
& \approx_{ax} \mathit{let}\ \langle y, z\rangle = x\ \mathit{in} \\
& \qquad \mathit{let}\ *= y\ \ \mathit{in}\ \mathit{let}\ *= z\ \ \mathit{in}\ \ *: \top \qquad \text{by } (\eta_*) \\
\rho_\top = \quad & x : \top \otimes \top \rhd \mathit{let}\ \langle y, z\rangle = x\ \mathit{in}\ \mathit{let}\ *= z\ \mathit{in}\ y \\
& \approx_{ax} \mathit{let}\ \langle y, z\rangle = x\ \mathit{in} \\
& \qquad \mathit{let}\ *= z\ \ \mathit{in}\ \mathit{let}\ *= y\ \ \mathit{in}\ \ * \qquad \text{by } (\eta_*) \\
& \approx_{ax} \mathit{let}\ \langle y, z\rangle = x\ \mathit{in} \\
& \qquad \mathit{let}\ *= y\ \ \mathit{in}\ \mathit{let}\ *= z\ \ \mathit{in}\ \ *: \top \qquad \text{by } (\mathit{let}_2).
\end{aligned}
$$

Thus the equation holds.

Equation (2.7.4). We want $\sigma_{A,B}; \sigma_{B,A} = id_{A\otimes B}$.

$$
\begin{aligned}
\sigma_{A,B} &= x : A \otimes B \rhd \mathit{let}\ \langle x_1, x_2\rangle = x\ \mathit{in}\ \langle x_2, x_1\rangle : B \otimes A \\
\sigma_{B,A} &= y : B \otimes A \rhd \mathit{let}\ \langle y_1, y_2\rangle = y\ \mathit{in}\ \langle y_2, y_1\rangle : A \otimes B \\
\sigma_{A,B}; \sigma_{B,A} &= x : A \otimes B \rhd \mathit{let}\ y = (\mathit{let}\ \langle x_1, x_2\rangle = x\ \mathit{in}\ \langle x_2, x_1\rangle)\ \mathit{in} \\
& \qquad \mathit{let}\ \langle y_1, y_2\rangle = y\ \mathit{in}\ \langle y_2, y_1\rangle
\end{aligned}
$$

$$
\begin{aligned}
&\approx_{ax} \mathit{let}\ \ \langle x_1, x_2\rangle = x\ \mathit{in} \\
&\qquad \mathit{let}\ \ y = \langle x_2, x_1\rangle\ \mathit{in} \\
&\qquad \mathit{let}\ \ \langle y_1, y_2\rangle = y\ \ \mathit{in}\ \langle y_2, y_1\rangle && \text{by } (\mathit{let}_1) \\
&\approx_{ax} \mathit{let}\ \ \langle x_1, x_2\rangle = x\ \mathit{in} \\
&\qquad \mathit{let}\ \ \langle y_1, y_2\rangle = \langle x_2, x_1\rangle\ \ \mathit{in}\ \langle y_2, y_1\rangle && \text{by } (\beta_\lambda) \\
&\approx_{ax} \mathit{let}\ \ \langle x_1, x_2\rangle = x\ \ \mathit{in}\ \langle x_1, x_2\rangle && \text{by } (\beta_\otimes) \\
&\approx_{ax} x : A \otimes B && \text{by } (\eta_\otimes),
\end{aligned}
$$

and thus (2.7.4) holds.

Equation (2.7.5). We want $\sigma_{B,\top}; \lambda_B = \rho_B$. The components are:

$$
\begin{aligned}
\sigma_{B,\top} &= x : B \otimes \top \rhd \mathit{let}\ \ \langle x_1, x_2\rangle = x\ \mathit{in}\ \langle x_2, x_1\rangle : B \otimes \top, \\
\lambda_B &= y : \top \otimes B \rhd \mathit{let}\ \ \langle y_1, y_2\rangle = y\ \mathit{in}\ \mathit{let}\ * = y_1\ \mathit{in}\ y_2 : B, \\
\rho_B &= x : B \otimes \top \rhd \mathit{let}\ \ \langle x_1, x_2\rangle = x\ \mathit{in}\ \mathit{let}\ * = x_2\ \mathit{in}\ x_1 : B.
\end{aligned}
$$

Let's compute the left hand side:

$$
\begin{aligned}
\sigma_{B,\top}; \lambda_B = x : B \otimes \top \rhd\ & \mathit{let}\ \ y = (\mathit{let}\ \langle x_1, x_2\rangle = x\ \mathit{in}\ \langle x_2, x_1\rangle)\ \mathit{in} \\
& \mathit{let}\ \ \langle y_1, y_2\rangle = y\ \mathit{in}\ \mathit{let}\ * = y_1\ \mathit{in}\ y_2 \\
\approx_{ax}\ & \mathit{let}\ \ \langle x_1, x_2\rangle = x\ \ \mathit{in}\ \mathit{let}\ y = \langle x_2, x_1\rangle\ \mathit{in} \\
& \mathit{let}\ \ \langle y_1, y_2\rangle = y\ \ \mathit{in}\ \mathit{let}\ * = y_1\ \mathit{in}\ y_2 && \text{by } (\mathit{let}_1) \\
\approx_{ax}\ & \mathit{let}\ \ \langle x_1, x_2\rangle = x\ \mathit{in} \\
& \mathit{let}\ \ \langle y_1, y_2\rangle = \langle x_2, x_1\rangle\ \ \mathit{in}\ \mathit{let}\ * = y_1\ \mathit{in}\ y_2 && \text{by } (\beta_\lambda) \\
\approx_{ax}\ & \mathit{let}\ \ \langle x_1, x_2\rangle = x\ \ \mathit{in}\ \mathit{let}\ * = x_2\ \mathit{in}\ x_1 : B && \text{by } (\beta_\otimes),
\end{aligned}
$$

which is exactly ρ_B.

Equation (2.7.6). We want

$$
\alpha_{A,B,C}; \sigma_{A\otimes B,C}; \alpha_{C,A,B} = (id_A \otimes \sigma_{B,C}); \alpha_{A,C,B}; (\sigma_{A,C} \otimes id_B).
$$

Left hand side. Let's write down the components:

$$
\begin{aligned}
\alpha_{A,B,C} = x : A \otimes (B \otimes C) \rhd\ & \mathit{let}\ \langle x_1, x_2\rangle = x\ \mathit{in} \\
& \mathit{let}\ \ \langle x_3, x_4\rangle = x_2\ \mathit{in}\ \langle\langle x_1, x_3\rangle, x_4\rangle : (A \otimes B) \otimes C, \\
\sigma_{A\otimes B,C} = y : (A \otimes B) \otimes C \rhd\ & \mathit{let}\ \ \langle y_1, y_2\rangle = y\ \mathit{in}\ \langle y_2, y_1\rangle : C \otimes (A \otimes B), \\
\alpha_{C,A,B} = z : C \otimes (A \otimes B) \rhd\ & \mathit{let}\ \langle z_1, z_2\rangle = z\ \mathit{in} \\
& \mathit{let}\ \ \langle z_3, z_4\rangle = z_2\ \mathit{in}\ \langle\langle z_1, z_3\rangle, z_4\rangle : (C \otimes A) \otimes B.
\end{aligned}
$$

Let's compose $\alpha_{A,B,C}$ and $\sigma_{A\otimes B,C}$:

$$
\alpha_{A,B,C}; \sigma_{A\otimes B,C} = x : A \otimes (B \otimes C) \rhd
$$

$$
\begin{aligned}
& let\ \ y = \left(\begin{array}{l} let\langle x_1, x_2\rangle = x\ in \\ let\ \langle x_3, x_4\rangle = x_2\ in\ \langle\langle x_1, x_3\rangle, x_4\rangle \end{array}\right)\ in \\
& let\ \ \langle y_1, y_2\rangle = y\ in\ \langle y_2, y_1\rangle \\
\approx_{ax}\ & let\ \langle x_1, x_2\rangle = x\ \ in\ let\langle x_3, x_4\rangle = x_2\ in \\
& let\ \ y = \langle\langle x_1, x_3\rangle, x_4\rangle\ in \\
& let\ \ \langle y_1, y_2\rangle = y\ in\ \langle y_2, y_1\rangle && \text{by } (let_1) \\
\approx_{ax}\ & let\ \langle x_1, x_2\rangle = x\ \ in\ let\langle x_3, x_4\rangle = x_2\ in \\
& let\ \ \langle y_1, y_2\rangle = \langle\langle x_1, x_3\rangle, x_4\rangle\ in\ \langle y_2, y_1\rangle && \text{by } (\beta_\lambda) \\
\approx_{ax}\ & let\ \langle x_1, x_2\rangle = x\ in\ let\langle x_3, x_4\rangle = x_2\ in \\
& \langle x_4, \langle x_1, x_3\rangle\rangle : C \otimes (A \otimes B) && \text{by } (\beta_\otimes).
\end{aligned}
$$

Let's compute the complete left hand side:

$$
\begin{aligned}
& \alpha_{A,B,C}; \sigma_{A\otimes B,C}; \alpha_{C,A,B} = x : A \otimes (B \otimes C) \rhd \\
& \qquad let\ \ z = \left(\begin{array}{l} let\langle x_1, x_2\rangle = x\ in \\ let\langle x_3, x_4\rangle = x_2\ in\ \langle x_4, \langle x_1, x_3\rangle\rangle \end{array}\right)\ in \\
& \qquad let\ \langle z_1, z_2\rangle = z\ in\ let\ \langle z_3, z_4\rangle = z_2\ in\ \langle\langle z_1, z_3\rangle, z_4\rangle \\
\approx_{ax}\ & let\ \langle x_1, x_2\rangle = x\ \ in\ let\langle x_3, x_4\rangle = x_2\ in \\
& let\ \ z = \langle x_4, \langle x_1, x_3\rangle\rangle\ \ in\ let\langle z_1, z_2\rangle = z\ in \\
& let\ \ \langle z_3, z_4\rangle = z_2\ in\ \langle\langle z_1, z_3\rangle, z_4\rangle && \text{by } (let_1) \\
\approx_{ax}\ & let\ \langle x_1, x_2\rangle = x\ \ in\ let\langle x_3, x_4\rangle = x_2\ in \\
& let\ \langle z_1, z_2\rangle = \langle x_4, \langle x_1, x_3\rangle\rangle\ in \\
& let\ \ \langle z_3, z_4\rangle = z_2\ in\ \langle\langle z_1, z_3\rangle, z_4\rangle && \text{by } (\beta_\lambda) \\
\approx_{ax}\ & let\ \langle x_1, x_2\rangle = x\ \ in\ let\langle x_3, x_4\rangle = x_2\ in \\
& let\ \ \langle z_3, z_4\rangle = \langle x_1, x_3\rangle\ in\ \langle\langle x_4, z_3\rangle, z_4\rangle && \text{by } (\beta_\otimes) \\
\approx_{ax}\ & let\ \langle x_1, x_2\rangle = x\ in \\
& let\ \langle x_3, x_4\rangle = x_2\ \ in\ \langle\langle x_4, x_1\rangle, x_3\rangle : (C \otimes A) \otimes B && \text{by } (\beta_\otimes).
\end{aligned}
$$

Right hand side: let's right down the components:

$$
\begin{aligned}
id_A = a : A \rhd a : A, \\
\sigma_{B,C} = b : B \otimes C \rhd let\ \ \langle b_1, b_2\rangle = b\ \ in\ \langle b_2, b_1\rangle : C \otimes B,
\end{aligned}
$$

thus

$$
\begin{aligned}
id_A \otimes \sigma_{B,C} = x : A \otimes (B \otimes C) \rhd\ & let\ \ \langle a, b\rangle = x\ in \\
& \langle a, let\ \langle b_1, b_2\rangle = b\ \ in\ \langle b_2, b_1\rangle\rangle \\
\approx_{ax}\ & let\ \ \langle a, b\rangle = x\ in \\
& let\ \ \langle b_1, b_2\rangle = b\ \ in\ \langle a, \langle b_2, b_1\rangle\rangle && \text{by } (let_1^\otimes).
\end{aligned}
$$

$$
\begin{aligned}
id_B = c : B \rhd c : B, \\
\sigma_{A,C} = d : A \otimes C \rhd let\ \ \langle d_1, d_2\rangle = d\ \ in\ \langle d_2, d_1\rangle : C \otimes A,
\end{aligned}
$$

thus

$$
\sigma_{A,C} \otimes id_B = y : (A \otimes C) \otimes B \rhd let\ \ \langle d, c\rangle = y\ in
$$

$$
\begin{aligned}
& \langle let\ \langle d_1, d_2\rangle = d\ \ in\ \langle d_2, d_1\rangle, c\rangle \\
\approx_{ax}\ & let\ \ \langle d, c\rangle = y\ in \\
& let\ \ \langle d_1, d_2\rangle = d\ in \\
& \langle\langle d_2, d_1\rangle, c\rangle : (B \otimes A) \otimes B && \text{by } (let_2^{\otimes}).
\end{aligned}
$$

Finally,

$$
\begin{aligned}
\alpha_{A,C,B} = z : A \otimes (C \otimes B) \rhd\ & let\ \langle z_1, z_2\rangle = z\ in \\
& let\ \ \langle z_3, z_4\rangle = z_2\ in\ \langle\langle z_1, z_3\rangle, z_4\rangle : (A \otimes C) \otimes B.
\end{aligned}
$$

Let's compose $id_A \otimes \sigma_{B,C}$ and $\alpha_{A,C,B}$:

$$
\begin{aligned}
& (id_A \otimes \sigma_{B,C}); \alpha_{A,C,B} = x : A \otimes (B \otimes C) \rhd \\
& \quad let\ \ z = \left(\begin{array}{l} let\ \langle a, b\rangle = x\ in \\ let\ \langle b_1, b_2\rangle = b\ \ in\ \langle a, \langle b_2, b_1\rangle\rangle \end{array}\right)\ in \\
& \quad let\ \langle z_1, z_2\rangle = z\ in \\
& \quad let\ \ \langle z_3, z_4\rangle = z_2\ in\ \langle\langle z_1, z_3\rangle, z_4\rangle \\
\approx_{ax}\ & \quad let\ \ \langle a, b\rangle = x\ \ in\ let\ \langle b_1, b_2\rangle = b\ in \\
& \quad let\ \ z = \langle a, \langle b_2, b_1\rangle\rangle\ \ in\ let\langle z_1, z_2\rangle = z\ in \\
& \quad let\ \ \langle z_3, z_4\rangle = z_2\ in\ \langle\langle z_1, z_3\rangle, z_4\rangle && \text{by } (let_1) \\
\approx_{ax}\ & \quad let\ \ \langle a, b\rangle = x\ \ in\ let\ \langle b_1, b_2\rangle = b\ in \\
& \quad let\ \langle z_1, z_2\rangle = \langle a, \langle b_2, b_1\rangle\rangle\ in \\
& \quad let\ \ \langle z_3, z_4\rangle = z_2\ in\ \langle\langle z_1, z_3\rangle, z_4\rangle && \text{by } (\beta_\lambda) \\
\approx_{ax}\ & \quad let\ \ \langle a, b\rangle = x\ \ in\ let\ \langle b_1, b_2\rangle = b\ in \\
& \quad let\ \ \langle z_3, z_4\rangle = \langle b_2, b_1\rangle\ in\ \langle\langle a, z_3\rangle, z_4\rangle && \text{by } (\beta_\otimes) \\
\approx_{ax}\ & \quad let\ \ \langle a, b\rangle = x\ \ in\ let\ \langle b_1, b_2\rangle = b\ in \\
& \quad \langle\langle a, b_2\rangle, b_1\rangle : (A \otimes C) \otimes B && \text{by } (\beta_\otimes).
\end{aligned}
$$

Let's compute the complete right hand side:

$$
\begin{aligned}
& (id_A \otimes \sigma_{B,C}); \alpha_{A,C,B}; (\sigma_{A,C} \otimes id_B) = x : A \otimes (B \otimes C) \rhd \\
& \quad let\ \ y = \left(\begin{array}{l} let\ \langle a, b\rangle = x\ \ in\ let\ \langle b_1, b_2\rangle = b\ in \\ \langle\langle a, b_2\rangle, b_1\rangle \end{array}\right)\ in \\
& \quad let\ \ \langle d, c\rangle = y\ \ in\ let\ \langle d_1, d_2\rangle = d\ \ in\ \langle\langle d_2, d_1\rangle, c\rangle \\
\approx_{ax}\ & \quad let\ \ \langle a, b\rangle = x\ \ in\ let\ \langle b_1, b_2\rangle = b\ in \\
& \quad let\ \ y = \langle\langle a, b_2\rangle, b_1\rangle\ \ in\ let\ \langle d, c\rangle = y\ in \\
& \quad let\ \ \langle d_1, d_2\rangle = d\ \ in\ \langle\langle d_2, d_1\rangle, c\rangle && \text{by } (let_1) \\
\approx_{ax}\ & \quad let\ \ \langle a, b\rangle = x\ \ in\ let\ \langle b_1, b_2\rangle = b\ in \\
& \quad \langle\langle b_2, a\rangle, b_1\rangle : (C \otimes A) \otimes B && \text{by } (\beta_\lambda) \text{ and } (\beta_\otimes) \text{ twice.}
\end{aligned}
$$

And this is alpha equivalent to the left hand side. So the equation holds.

□

9.3.2 Monadic Structure

Definition 9.3.4. We define the term $\lambda *.M$ to be $\lambda y.(let\ * = y\ in\ M)$, for M a fresh variable.

Lemma 9.3.5. *The following axiomatic relations are satisfied:*

$$\Delta \ \rhd \ (\lambda *.M)* \approx_{ax} M \ : A \qquad (\beta_*)$$
$$\Delta \ \rhd \ (\lambda *.V*) \approx_{ax} V \ : \top \multimap A \qquad (\eta_*)$$

Proof. Proof by simple calculation. □

Lemma 9.3.6. *Let η_A be the arrow $x : A \rhd \lambda *.x : \top \multimap A$ and*

$$(x : A \rhd V : \top \multimap B)^* = y : \top \multimap A \rhd \lambda *.\ let\ \ x = (y*)\ in\ (V*) : \top \multimap B.$$

Then $(\top \multimap -, \eta, -^*)$ is a Kleisli triple in $\mathcal{C}_\lambda$.

Proof. We need to check Equations (2.5.3), (2.5.4) and (2.5.5).

Equation (2.5.3). We want $\eta_A^* = id_{\top \multimap A}$:

$$\begin{aligned}
\eta_A &= x : A \rhd \lambda *.x : \top \multimap A && \text{thus}\\
\eta_A^* &= y : \top \multimap A \rhd \lambda *.\ let\ \ x = (y*)\ in\ ((\lambda *.x)*) : \top \multimap A \\
&= y : \top \multimap A \rhd \lambda *.\ let\ \ x = (y*)\ in\ x \top \multimap A && \text{by } (\beta_*), (\beta_\lambda)\\
&= y : \top \multimap A \rhd \lambda *.y* : \top \multimap A && \text{by } (\beta_\lambda^2)\\
&= y : \top \multimap A \rhd y : \top \multimap A && \text{by } (\eta_*), (\eta_\lambda)\\
&= id_{\top \multimap A}.
\end{aligned}$$

Equation (2.5.4). We want $\eta_A; f^* = f$:

$$\eta_A = x : A \rhd \lambda *.x : \top \multimap A, \qquad f = y : A \rhd V : \top \multimap B,$$

thus

$$\begin{aligned}
f^* &= z : \top \multimap A \rhd \lambda *.\ let\ \ y = (z*)\ in\ (V*) : \top \multimap B, && \text{and}\\
\eta_A; f^* &= x : A \rhd letz = \lambda *.x\ in \\
&\qquad \lambda *.\ let\ \ y = (z*)\ in\ (V*) : \top \multimap B \\
&= x : A \rhd \lambda *.\ let\ \ y = ((\lambda *.x)*)\ in\ (V*) : \top \multimap B && \text{by } (\beta_\lambda)\\
&= x : A \rhd \lambda *.\ let\ \ y = x\ in\ (V*) : \top \multimap B && \text{by } (\beta_*)\\
&= y : A \rhd \lambda *.V* : \top \multimap B && \text{by } (\alpha_{let})\\
&= y : A \rhd V : \top \multimap B && \text{by } (\eta_*), (\eta_\lambda)\\
&= f
\end{aligned}$$

Equation (2.5.5). We want $f^*; g^* = (f; g^*)^*$. If

$$f = x : A \rhd V : \top \multimap B, \qquad g = y : B \rhd W : \top \multimap C,$$

then

$$\begin{aligned}
f^* &= z : \top \multimap A \rhd \lambda *.\ let\ \ x = (z*)\ in\ (V*) : \top \multimap B,\\
g^* &= t : \top \multimap B \rhd \lambda *.\ let\ \ y = (t*)\ in\ (W*) : \top \multimap C.
\end{aligned}$$

Let's compute

$$f^*; g^* = z : \top \multimap A \rhd let\ t = (\lambda *.\ let\ \ x = (z*)\ in\ (V*))$$

$$
\begin{aligned}
& \quad in\ \lambda *.\ let\ \ y = (t*)\ in\ (W*) : \top \multimap C, \\
&= z : \top \multimap A \rhd \lambda *.\ let\ \ y = \\
&\quad (\lambda *.\ let\ \ x = (z*)\ in\ (V*))*\ in\ \ (W*) : \top \multimap C && \text{by } (\beta_\lambda) \\
&= z : \top \multimap A \rhd \lambda *.\ let\ \ y = (let\ x = (z*)\ in\ (V*)) \\
&\quad in\ \ (W*) : \top \multimap C && \text{by } (\beta_*), (\beta_\lambda) \\
&= z : \top \multimap A \rhd \lambda *.\ let\ \ x = (z*)\ in \\
&\quad let\ \ y = (V*)\ in\ (W*) : \top \multimap C && \text{by } (let_1).
\end{aligned}
$$

Next we need

$$
\begin{aligned}
f; g^* &= x : A \rhd let\ t = V \\
&\quad in\ \lambda *.\ let\ \ y = (t*)\ in\ (W*) : \top \multimap C, \\
&= x : A \rhd \lambda *.\ let\ \ t = V\ \ in \\
&\quad let\ \ y = (t*)\ in\ (W*) : \top \multimap C && \text{by } (let^\lambda),
\end{aligned}
$$

and thus

$$
\begin{aligned}
(f; g^*)^* &= z : \top \multimap A \rhd \lambda *.\ let\ \ x = (z*)\ \ in \\
&\quad \lambda *.(let\ t = V\ in\ let\ y = (t*)\ in\ (W*))* : \top \multimap C \\
&= z : \top \multimap A \rhd \lambda *.\ let\ \ x = (z*)\ \ in \\
&\quad let\ \ t = V\ in\ let\ y = (t*)\ in\ (W*) : \top \multimap C && \text{by } (\beta_*), (\beta_\lambda) \\
&= z : \top \multimap A \rhd \lambda *.\ let\ \ x = (z*)\ \ in \\
&\quad let\ \ y = (V*)\ in\ (W*) : \top \multimap C && \text{by } (\beta_\lambda) \\
&= f^*; g^*
\end{aligned}
$$

This ends the proof. □

Lemma 9.3.7. *Define the following maps:*

$$
\begin{aligned}
\eta_A &= x : A \rhd \lambda *.x : \top \multimap A, \\
\mu_A &= x : \top \multimap (\top \multimap A) \rhd \lambda *.(x*)* : \top \multimap A, \\
t_{A,B} &= z : A \otimes (\top \multimap B) \rhd let\ \langle x, y\rangle = z\ in\ \lambda *.\langle x, y*\rangle : \top \multimap (A \otimes B),
\end{aligned}
$$

and define $\top \multimap (x : A \rhd V : B)$ *as the arrow*

$$y : \top \multimap A \rhd \lambda *.\ let\ \ x = (y*)\ in\ V : \top \multimap B.$$

Then $(\top \multimap -, \eta, \mu, t)$ *is a strong monad.*

Proof. The fact that $(\top \multimap -, \eta, \mu)$ is a monad is a corollary of Lemmas 9.3.6 and 2.5.3. It remains to show that $t_{A,B}$ is natural in A and B, and that it satisfies Equations (5.4.1), (5.4.5) and (5.4.4).

Naturality of $t_{A,B}$ *in* A. We must show that given a map $f : A \to A'$, the diagram

$$
\begin{tikzcd}
A \otimes (\top \multimap B) \arrow[r, "t_{A,B}"] \arrow[d, "f \otimes (\top \multimap B)"'] & \top \multimap (A \otimes B) \arrow[d, "\top \multimap (f \otimes B)"] \\
A' \otimes (\top \multimap B) \arrow[r, "t_{A',B}"'] & \top \multimap (A' \otimes B)
\end{tikzcd}
$$

commutes. If f is the value $t : A \rhd V : A'$, then

$$f \otimes (\top \multimap B) : u : A \otimes (\top \multimap B) \rhd let\ \langle t, u' \rangle = u\ in\ \langle V, u' \rangle : A' \otimes (\top \multimap B),$$

$$\top \multimap (f \otimes B) : v : \top \multimap (A \otimes B) \rhd \lambda *. \left(\begin{array}{l} let\ v' = v*\ in \\ let \langle t, v'' \rangle = v'\ in \\ \langle V, v'' \rangle \end{array} \right) : \top \multimap (A' \otimes B).$$

One needs to show that $f \otimes (\top \multimap B); t_{A',B} = t_{A,B}; \top \multimap (f \otimes B)$. The right hand side is

$$\begin{array}{rl}
z : A \otimes (\top \multimap B) \rhd let\ v = \left(\begin{array}{l} let\ \langle x, y \rangle = z\ in \\ \lambda *. \langle x, y* \rangle \end{array} \right) & \\
in\ \ \lambda *. \left(\begin{array}{l} let\ v' = v*\ in \\ let\ \langle t, v'' \rangle = v'\ in \\ \langle V, v'' \rangle \end{array} \right) & \\
\approx_{ax} \lambda *. \left(\begin{array}{l} let\ \langle x, y \rangle = z\ in \\ let\ v = \lambda *. \langle x, y* \rangle\ in \\ let\ v' = v*\ in \\ let\ \langle t, v'' \rangle = v'\ \ in\ \langle V, v'' \rangle \end{array} \right) & \text{by } (let^\lambda) \text{ and } (let_1) \\
\approx_{ax} \lambda *. \left(\begin{array}{l} let\ \langle x, y \rangle = z\ in \\ let\ v' = \langle x, y* \rangle\ in \\ let\ \langle t, v'' \rangle = v'\ \ in\ \ \langle V, v'' \rangle \end{array} \right) & \text{by } (\beta_\lambda) \text{ and } (\beta_*) \\
\approx_{ax} \lambda *. \left(\begin{array}{l} let\ \langle x, y \rangle = z\ in \\ let\ v' = \left(\begin{array}{l} let\ b = y*\ in \\ let\ a = x\ \ in\ \ \langle a, b \rangle \end{array} \right)\ in \\ let\ \langle t, v'' \rangle = v'\ \ in\ \ \langle V, v'' \rangle \end{array} \right) & \text{by } (let^\otimes) \\
\approx_{ax} \lambda *. \left(\begin{array}{l} let\ \langle x, y \rangle = z\ \ in\ let\ b = y*\ in \\ let\ a = x\ \ in\ let\ v' = \langle a, b \rangle\ in \\ let\ \langle t, v'' \rangle = v'\ \ in\ \ \langle V, v'' \rangle \end{array} \right) & \text{by } (let_1) \\
\approx_{ax} \lambda *. \left(\begin{array}{l} let\ \langle x, y \rangle = z\ \ in\ let\ b = y*\ in \\ let\ a = x\ \ in\ let\ \langle t, v'' \rangle = \langle a, b \rangle\ \ in\ \ \langle V, v'' \rangle \end{array} \right) & \text{by } (\beta_\lambda) \\
\approx_{ax} \lambda *. \left(\ let\ \langle x, y \rangle = z\ \ in\ let\ b = y*\ \ in\ \ \langle V[x/t], b \rangle\ \right) & \text{by } (\beta_\otimes) \text{ and } (\beta_\lambda) \\
\approx_{ax} \lambda *. \left(\begin{array}{l} let\ \langle x, y \rangle = z\ \ in\ let\ b = y*\ in \\ let\ b' = b\ \ in\ \ \ let\ \ a' = V[x/t]\ \ in\ \langle a', b' \rangle \end{array} \right) & \text{by } (let^\otimes) \\
\approx_{ax} \lambda *. \left(\begin{array}{l} let\ \langle x, y \rangle = z\ in \\ let\ b' = (let\ b = y*\ \ in\ \ b)\ in \\ let\ a' = V[x/t]\ \ in\ \langle a', b' \rangle \end{array} \right) & \text{by } (let_1) \\
\approx_{ax} \lambda *. \left(\begin{array}{l} let\ \langle x, y \rangle = z\ \ in\ let\ b' = y*\ in \\ let\ a' = V[x/t]\ \ in\ \langle a', b' \rangle \end{array} \right) & \text{by } (\beta_\lambda^2) \\
\approx_{ax} \lambda *. \left(\ let\ \langle x, y \rangle = z\ \ in\ \langle V[x/t], y* \rangle\ \right) & \text{by } (let^\otimes).
\end{array}$$

The left hand side is

$$\begin{array}{rl}
u : A \otimes (\top \multimap B) \rhd & let\ \ z = (let\ \langle t, u' \rangle = u\ in\ \langle V, u' \rangle)\ in \\
& let\ \ \langle x, y \rangle = z\ in\ \lambda *. \langle x, y* \rangle \\
\approx_{ax} & let\ \ \langle t, u' \rangle = u\ in \\
& let\ \ z = \langle V, u' \rangle\ in
\end{array}$$

$$\begin{array}{rll} & let\ \ \langle x,y\rangle = z\ in\ \lambda *.\langle x,y*\rangle & \text{by } (let_1) \\ \approx_{ax} & let\ \ \langle t,u'\rangle = u\ in & \\ & let\ \ \langle x,y\rangle = \langle V,u'\rangle\ in\ \lambda *.\langle x,y*\rangle & \text{by } (\beta_\lambda) \\ \approx_{ax} & let\ \ \langle t,u'\rangle = u\ \ in\ \lambda *.\langle V,u'*\rangle. & \text{by } (\beta_\otimes) \end{array}$$

Since they are α-equivalent, they are axiomatically equivalent, and the diagram commutes.

Naturality of $t_{A,B}$ in B. One needs to show that given a map $f : B \to B'$, the diagram

$$\begin{array}{ccc} A\otimes(\top\multimap B) & \xrightarrow{t_{A,B}} & \top\multimap(A\otimes B) \\ {\scriptstyle A\otimes(\top\multimap f)}\downarrow & & \downarrow{\scriptstyle \top\multimap(A\otimes f)} \\ A'\otimes(\top\multimap B') & \xrightarrow[t_{A,B'}]{} & \top\multimap(A\otimes B') \end{array}$$

commutes. That is to say, $t_{A,B};\top\multimap(A\otimes f) = A\otimes(\top\multimap f);t_{A,B'}$. If f is the value $t : B \rhd V : B'$, one has:

$$\begin{array}{rl} A\otimes(\top\multimap f): & u : A\otimes(\top\multimap B) \rhd let\ \langle t',u'\rangle = u\ in \\ & \qquad \langle t',\lambda *.\ let\ \ t = u'*\ in\ V\rangle : A'\otimes(\top\multimap B') \\ \top\multimap(A\otimes f): & v : \top\multimap(A\otimes B) \rhd \lambda *.\left(\begin{array}{l} let\ v' = v*\ in \\ let\ \langle t',t\rangle = v'\ in\ t',V \end{array}\right) : \top\multimap(A\otimes B'). \end{array}$$

This ends the proof. □

Lemma 9.3.8. *Consider the functor $\multimap : {\mathcal{C}_\lambda}^{op}\times\mathcal{C}_\lambda \to \mathcal{C}_\lambda$, defined on arrows by*

$$\begin{array}{rcl} A\multimap(x : B\rhd V : C) & = & y : A\multimap B \rhd \lambda z.\ let\ \ x = yz\ in\ V : A\multimap C, \\ (x : A\rhd V : B)\multimap C & = & y : B\multimap C \rhd \lambda x.yV : A\multimap C, \end{array}$$

and the map $\Phi_{A,B,C} : \mathcal{C}_\lambda(A, B\multimap C) \to \mathcal{C}_\lambda(A\otimes B, \top\multimap C)$:

$$(x : A\rhd V : B\multimap C) \quad\longrightarrow\quad (t : A\otimes B \rhd \lambda *.\ let\ \ \langle x,y\rangle = t\ in\ Vy : \top\multimap C)\,.$$

With these structures, $\mathcal{C}_\lambda$ has $(\top\multimap -)$-exponentials, as in Definition 5.4.4.

Proof. We define $\Phi^{-1}_{A,B,C}$:

$$(t : A\otimes B\rhd V : \top\multimap C) \quad\longrightarrow\quad (x : A\rhd \lambda y.\ let\ \ t = \langle x,y\rangle\ in\ V* : B\multimap C)\,,$$

and we show that they are inverse of each other. Given $x : A\rhd V : B\multimap C$, let's apply $\Phi^{-1}_{A,B,C}\circ\Phi_{A,B,C}$. We get

$$\begin{array}{ll} x : A\rhd \lambda y.\,(let\ t = \langle x,y\rangle\ in\ (\lambda *.\ let\ \ \langle x,y\rangle = t\ in\ Vy)*) & \\ \quad\approx_{ax} \lambda y.\,(let\ t = \langle x,y\rangle\ in\ let\ \langle x,y\rangle = t\ in\ Vy) & \text{by } (\beta_*) \\ \quad\approx_{ax} \lambda y.\,(let\ \langle x,y\rangle = \langle x,y\rangle\ in\ Vy) & \text{by } (\beta_\lambda) \\ \quad\approx_{ax} \lambda y.(Vy) & \text{by } (\beta_\otimes) \\ \quad\approx_{ax} V : B\multimap C & \text{by } (\eta_\lambda). \end{array}$$

Similarly, if we apply $\Phi_{A,B,C}\circ\Phi^{-1}_{A,B,C}$ to $t : A\otimes B\rhd V : \top\multimap C$, we get

$$t : A\otimes B\rhd \lambda *.\ let\ \ \langle x,y\rangle = t\ in\ (\lambda y.\ let\ \ t = \langle x,y\rangle\ in\ V*)y$$

$$\begin{aligned}
&\approx_{ax} \lambda *.\ let\ \ \langle x,y\rangle = t\ in\ let\ t = \langle x,y\rangle\ in\ V* && \text{by } (\beta_\lambda)\\
&\approx_{ax} \lambda *.\ let\ \ t = \langle x,y\rangle\ in\ let\ \langle x,y\rangle = t\ in\ V* && \text{by } (let_2)\\
&\approx_{ax} \lambda *.\ let\ \ \langle x,y\rangle = \langle x,y\rangle\ in\ V* && \text{by } (\beta_\lambda)\\
&\approx_{ax} \lambda *.V* && \text{by } (\beta_\otimes)\\
&\approx_{ax} V : \top \multimap C
\end{aligned}$$

and thus $\Phi^{-1}_{A,B,C}$ is the inverse of $\Phi_{A,B,C}$. To finish the proof it is enough to show that $\multimap$ is bifunctorial. We need to show that $(- \multimap B)$ and $(A \multimap -)$ are functors, and that $(f \multimap B); (A \multimap g) = (A' \multimap g); (f \multimap B)$, if $f : A \to A'$ and $g : B \to B'$.

First consider $id_A \multimap id_B$: If $id_A = a : A \rhd a : A$ and $id_B = b : B \rhd b : B$, it is the arrow

$$\begin{aligned}
x : A \multimap B \rhd &\lambda a.\ let\ \ b = xa\ in\ b\\
&\approx_{ax} \lambda a.xa && \text{by } (\beta^2_\lambda)\\
&\approx_{ax} x && \text{by } (\eta_\lambda)
\end{aligned}$$

Then consider $f = x : A \rhd V : A'$ and $f' = y : A' \rhd W : A''$. We compute $(f \multimap B)$ and $(f' \multimap B)$:

$$\begin{aligned}
t : A'' \multimap B \rhd &\lambda y.\ let\ \ b = tW\ in\ b\\
&\approx_{ax} \lambda y.tW : A' \multimap B && \text{by } (\beta^2_\lambda),\\
z : A' \multimap B \rhd &\lambda x.\ let\ \ b = zV\ in\ b\\
&\approx_{ax} \lambda x.zV : A \multimap B && \text{by } (\beta^2_\lambda).
\end{aligned}$$

The composition of these two functions is

$$\begin{aligned}
t : A'' \multimap B \rhd &\ let\ z = \lambda y.tW\ \ in\ \lambda x.zV\\
&\approx_{ax} \lambda x.(\lambda y.tW)V && \text{by } (\beta_\lambda)\\
&\approx_{ax} \lambda x.\ let\ \ y = V\ in\ tW : A \multimap B && \text{by definition.}
\end{aligned}$$

Let's compute $(f; f') \multimap B$:

$$\begin{aligned}
t : A'' \multimap B \rhd &\lambda x.\ let\ \ b = t(let\ y = V\ in\ W)\ in\ b\\
&\approx_{ax} \lambda x.t(let\ y = V\ in\ W) && \text{by } (\beta^2_\lambda)\\
&\approx_{ax} \lambda x.\ let\ \ y = V\ in\ tW : A \multimap B && \text{by } (let^{app}_1),
\end{aligned}$$

which is precisely $(f' \multimap B); (f \multimap B)$.

Now consider $g = x : B \rhd V : B'$ and $g' = y : B' \rhd W : B''$. We compute $(A \multimap g)$ and $(A \multimap g')$:

$$\begin{aligned}
t : A \multimap B \rhd &\lambda a.\ let\ \ x = ta\ in\ V : A \multimap B',\\
z : A \multimap B' \rhd &\lambda a'.\ let\ \ y = za'\ in\ W : A \multimap B''.
\end{aligned}$$

The composition of these two functions is

$$\begin{aligned}
t : A \multimap B \rhd &\ let\ \ z = (\lambda a.\ let\ \ x = ta\ in\ V)\ in\\
&\qquad \lambda a'.\ let\ \ y = za'\ in\ W\\
&\approx_{ax} \lambda a'.\ let\ \ y = (\lambda a.\ let\ \ x = ta\ in\ V)a'\ in\ W && \text{by } (\beta_\lambda)\\
&\approx_{ax} \lambda a'.\ let\ \ y = (let\ x = ta'\ in\ V)\ in\ W && \text{by } (\beta_\lambda)
\end{aligned}$$

$$\approx_{ax} \lambda a'.\ let\ \ x = ta'\ in\ let\ y = V\ in\ W \qquad \text{by } (let_1),$$

which is exactly $A \multimap (g; g')$.

Remains to show that if $f = x : A \rhd U : A'$ and $g = y : B \rhd V : B'$ then $(A' \multimap g); (f \multimap B') = (f \multimap B); (A \multimap g)$.

Let us write down the components of the equations:

$$\begin{aligned}
A' \multimap g &= && x_1 : A' \multimap B \rhd \lambda x_2.\ let\ y = x_1 x_2\ in\ \ V : A' \multimap B' \\
A \multimap g &= && y_1 : A \multimap B \rhd \lambda y_2.\ let\ y = y_1 y_2\ in\ \ V : A \multimap B' \\
f \multimap B' &= && z : A' \multimap B' \rhd \lambda x.zU : A \multimap B' \\
f \multimap B &= && t : A' \multimap B \rhd \lambda x.tU : A \multimap B
\end{aligned}$$

Computing $(A' \multimap g); (f \multimap B')$, we get

$$\begin{aligned}
x_1 : A' \multimap B \rhd\ & let\ z = (\lambda x_2.\ let\ y = x_1 x_2\ in\ \ V)\ in\ \lambda x.zU \\
&\approx_{ax} \lambda x.(\lambda x_2.\ let\ y = x_1 x_2\ in\ \ V)U && \text{by } (\beta_\lambda) \\
&\approx_{ax} \lambda x.\ let\ \ y = x_1 U\ in\ \ V : A \multimap B' && \text{by } (\beta_\lambda)
\end{aligned}$$

The other hand of the equation, $(f \multimap B); (A \multimap g)$, is

$$\begin{aligned}
t : A' \multimap B \rhd\ & let\ y_1 = \lambda x.tU\ in\ \lambda y_2.\ let\ y = y_1 y_2\ in\ \ V \\
&\approx_{ax} \lambda y_2.\ let\ y = (\lambda x.tU) y_2\ in\ \ V && \text{by } (\beta_\lambda) \\
&\approx_{ax} \lambda x.\ let\ y = tU\ in\ \ V && \text{by } (\alpha_{let}),
\end{aligned}$$

and thus the equation holds. □

9.3.3 Comonadic Structure

The goal is to have the type constructor ! to be a comonad. Ideally, we would like the following to be a derived typing rule:

$$\frac{!\Delta \rhd V : A}{!\Delta \rhd V : !A.}$$

However, in the context of indexed terms, the correct statement of this property is slightly more technical. It means there should be another indexation of the original value, well-typed of duplicable type.

Definition 9.3.9. We define a map ! on extended values in the following way:

$$\begin{aligned}
!(x^B) &= x^{!B} & !(\langle V, W\rangle^n), &= \langle !V, !W\rangle^{n+1}, \\
!(c^B) &= c^{!B,} & !(let\ * = V\ in\ W) &= let\ * = V\ in\ !W, \\
!(*^n) &= *^{n+1}, & !(let\ \langle x^A, y^B\rangle^n = V\ in\ W) &= let\ \langle x^A, y^B\rangle^{n+1} = !V\ in\ !W, \\
!(\lambda^n x^A.M) &= \lambda^{n+1} x^A.M, & !(let\ x^A = V\ in\ W) &= let\ x^{!A} = !V\ in\ !W.
\end{aligned}$$

We call $!V$ the *promotion* of V.

Lemma 9.3.10. *Given a valid typing judgement $!\Delta \rhd V : A$, the typing judgement $!\Delta \rhd !V : !A$ is valid.*

Proof. Proof by structural induction on the derivation of $!\Delta \rhd V : A$ □

Lemma 9.3.11. *Consider an extended value V. Then $Erase(!V) = Erase(V)$.*

Proof. Proof by structural induction on V. □

Lemma 9.3.12. *If $\Delta \rhd V \approx_{ax} W : A$, then $!\Delta \rhd !V \approx_{ax} !W : !A$*

Proof. By structural induction on the derivation of $\Delta \rhd V \approx_{ax} W : A$. We fully develop the case (β_λ).

Suppose that $\Delta \rhd let\ x^A = V\ in\ W : B$ is valid. Then so is $\Delta \rhd W[V/x] : B$. We have $!(let\ x^A = V\ in\ W) = (let\ x^{!A} = !V\ in\ !W)$. From Lemma 9.3.10,

$$!\Delta \rhd let\ x^{!A} = !V\ in\ !W : !B$$

is valid. We therefore have $!\Delta \rhd !W[!V/x] : !B$, and by rule (β_λ), they are axiomatically equivalent.

Now, using 9.3.10, Lemma 9.3.11 and Theorem 9.2.7, we have $!\Delta \rhd !(W[V/x]) \approx_{ax} !W[!V/x] : !B$. Therefore,

$$!\Delta \rhd !(let\ x^A = V\ in\ W) \approx_{ax} !(W[V/x]) : !B.$$

The other cases are handled similarly. □

Lemma 9.3.13. *Given the map ! sending A to $!A$ and sending $x : A \rhd V : B$ to $x : !A \rhd !V : !B$, and given the arrows*

$$\epsilon_A = x : !A \rhd x^A : A, \qquad \delta_A = x : !A \rhd x^{!^2 A} : !^2 A,$$

$(!, \epsilon, \delta)$ is an idempotent comonad in $\mathcal{C}_\lambda$.

Proof. We show the functoriality of !.

First, from Lemma 9.3.12, ! is a well-defined function on equivalence class of arrows.

Then, given $x : A \rhd V : B$ and $y : B \rhd W : C$ we need to show that

$$!(x : A \rhd V : B); !(y : B \rhd W : C) \approx_{ax} !(x : A \rhd let\ y = V\ in\ W : C).$$

Both hands ends up being by definition of composition equal to

$$x : !A \rhd let\ y^{!B} = !V\ in\ !W : !C.$$

Finally, the image $!id_A$ of the unit on A is $!(x : A \rhd x : A)$, which is by definition $x : !A \rhd x : !A$, namely the unit on $!A$.

Using Lemmas 9.1.25, 9.3.11 and Theorem 9.2.7, the equations

$$x : !A \rhd \{!V <: B\} \approx_{ax} V : B,$$
$$x : !A \rhd \{!V <: !!B\} \approx_{ax} \{!V :> !!B\} : !!B$$

are valid, making ϵ and δ natural transformations.

The facts that δ_A is an isomorphism and that δ and ϵ verify Equation (2.6.1) and (2.6.2) use similar techniques. □

$$
\begin{array}{lcrll}
\alpha_{A,B,C} & = & x : A\otimes(B\otimes C) & \rhd \mathit{let}\langle y,z\rangle = x \mathit{in} & \\
 & & & \quad \mathit{let}\langle t,u\rangle = z \mathit{ in } \langle\langle y,t\rangle,u\rangle & : (A\otimes B)\otimes C \\
\lambda_A & = & x : \top\otimes A & \rhd \mathit{let}\ \langle y,z\rangle = x \mathit{ in let } * = y \mathit{ in } z & : A \\
\rho_A & = & x : A\otimes\top & \rhd \mathit{let}\ \langle y,z\rangle = x \mathit{ in let } * = z \mathit{ in } y & : A \\
\sigma_{A,B} & = & x : A\otimes B & \rhd \mathit{let}\ \langle y,z\rangle = x \mathit{ in } \langle z,y\rangle & : B\otimes A \\
\eta_A & = & x : A & \rhd \lambda *.x & : \top\multimap A \\
\mu_A & = & x : \top\multimap(\top\multimap A) & \rhd \lambda *.(x*)* & : \top\multimap A \\
t_{A,B} & = & z : A\otimes(\top\multimap B) & \rhd \mathit{let}\ \langle x,y\rangle = z \mathit{ in } \lambda *.\langle x,y*\rangle & : \top\multimap(A\otimes B) \\
\epsilon_A & = & x : !A & \rhd x^A & : A \\
\delta_A & = & x : !A & \rhd x^{!^2 A} & : !^2 A \\
d^!_{A,B} & = & z : !A\otimes !B & \rhd \mathit{let}\ \langle x,y\rangle = z \mathit{ in } \langle x,y\rangle & : !(A\otimes B) \\
d^!_\top & = & z : \top & \rhd \mathit{let} * = z \mathit{ in } * & : !\top \\
\triangle_A & = & x : !A & \rhd \langle x,x\rangle & : !A\otimes !A \\
\Diamond_A & = & x : !A & \rhd * & : \top
\end{array}
$$

$$
\begin{array}{l}
(x : A \rhd V : B)\otimes(y : C \rhd W : D) \quad = \\
\qquad z : A\otimes B \rhd \mathit{let}\ \langle x,y\rangle = z \mathit{ in } \langle V,W\rangle : C\otimes D
\end{array}
$$

$$
\begin{array}{l}
(x : A \rhd V : B)\multimap(y : C \rhd W : D) \quad = \\
\qquad z : B\multimap C \rhd \lambda x.(\mathit{let}\ y = zV \mathit{ in } W) : A\multimap D
\end{array}
$$

$$
\begin{array}{l}
(x : A \rhd V : \top\multimap B)^* \quad = \\
\qquad y : \top\multimap A \rhd \lambda *.\ \mathit{let}\ \ x = (y*) \mathit{ in } (V*) : \top\multimap B
\end{array}
$$

$$
\begin{array}{l}
\Phi_{A,B,C}\,(x : A \rhd V : B\multimap C) \quad = \\
\qquad t : A\otimes B \rhd \lambda *.\ \mathit{let}\ \ \langle x,y\rangle = t \mathit{ in } Vy : \top\multimap C
\end{array}
$$

Table 9.5: Definitions of maps and operations on maps in $\mathcal{C}_\lambda$

9.3.4 The Category $\mathcal{C}_\lambda$ is a Linear Category for Duplication

Lemma 9.3.14. *Given the maps*

$$
\begin{array}{rl}
d^!_{A,B} = & z : !A\otimes !B \rhd \mathit{let}\ \langle x^{!A}, y^{!B}\rangle^1 = z \mathit{ in } \langle x^{!A}, y^{!B}\rangle^1 : !(A\otimes B), \\
d^!_\top = & z : \top \rhd \mathit{let} * = z^\top \mathit{ in } * : !\top,
\end{array}
$$

$(!, \delta, \epsilon, d, d)$ is a strong symmetric monoidal comonad in $\mathcal{C}_\lambda$.

Proof. The proof makes heavy use of Lemmas 9.1.25, 9.3.11 and Theorem 9.2.7. □

Lemma 9.3.15. *If we define the maps*

$$
\triangle_A = \quad x : !A \rhd \langle x^{!A}, x^{!A}\rangle^0 : !A\otimes !A, \qquad\qquad \Diamond_A = \quad x : !A \rhd *^0 : \top,
$$

the functor "!" is a linear exponential comonad.

Proof. Again, the proof makes heavy use of Lemmas 9.1.25, 9.3.11 and Theorem 9.2.7. □

Theorem 9.3.16. *Together with the maps and the operations on maps defined in Table 9.5, $\mathcal{C}_\lambda$ is a linear category for duplication.*

Proof. This is a corollary of Lemmas 9.3.1, 9.3.3, 9.3.6, 9.3.7, 9.3.8 and 9.3.15. □

Chapter 10

Proof of Theorem 9.2.7

This chapter is uniquely devoted to the proof of Theorem 9.2.7. Recall that in Chapter 9, we defined a typed lambda-calculus with explicit indexing. The type system is based on linear logic: a special type operator "!" is used to distinguish duplicable and non-duplicable terms. We recall Definitions 9.1.1 and 9.1.3 giving the set of terms and types:

$$
\begin{array}{lcl}
\mathit{Type}\ A, B & ::= & \alpha \mid (A \multimap B) \mid (A \otimes B) \mid \top \mid !A, \\
\mathit{CoreValue}\ U, U' & ::= & x^A \mid c^A \mid *^n \mid \lambda^n x^A.M \mid \langle U, U' \rangle^n, \\
\mathit{ExtValue}\ V, W & ::= & U \mid \langle V, W \rangle^n \mid \ \mathit{let}\ \ x^A = V\ \mathit{in}\ W \mid \ \mathit{let}\ \ \langle x^A, y^B \rangle^n = V\ \mathit{in}\ W \mid \\
 & & \mathit{let}\ * = V\ \mathit{in}\ W, \\
\mathit{Term}\ M, N & ::= & U \mid \langle M, N \rangle^n \mid (MN) \mid \ \mathit{let}\ \ \langle x^A, y^B \rangle^n = M\ \mathit{in}\ N \mid \ \mathit{let}\ * = M\ \mathit{in}\ N\,.
\end{array}
$$

Terms are separated into core values, extended values and computations. We introduced the notion of valid typing judgement in Section 9.1.2, and we provided an axiomatic equivalence relation on valid typing judgement in Section 9.2. For the axiomatic equivalence relation a notion of substitution is required. This does not come for free due to the subtyping relation: an explicit typecasting is needed and is introduced in Definition 9.1.20: If a term M is valid with type A and if $A <: B$, we inductively define a valid term $\{M <: B\}$ of type B.

One of the desired properties derived from the axiomatic equivalence relation is Theorem 9.2.7, stating that if any two terms are equal without their indexation, they are axiomatically equivalent. In other words, the meaning of a term is not related to the indexation of the term. The main technical obstacle to a direct proof of Theorem 9.2.7 is the possible "wild" use of the application inside any given term. In order to restrict this, we use the notion of "normalization by evaluation" of (Filinski, 2001) and (Ohori, 1999): we work with a subclass of terms that we call neutral terms on which the result can be proved directly. We then define two rewriting systems yielding in two steps neutral terms from any given terms. A weak normalization result is shown, adapting the techniques described in (Girard et al., 1990, Ch. 4): we show that terms admit a well-founded measure of convergence, which decreases with each reduction of the rewriting system.

10.1 A Handy Tool: Neutral Terms

In this section we define the notion of *neutral terms*. We prove some useful results and then that any term is axiomatically equivalent to a neutral term.

Definition 10.1.1. Following Filinski (2001) and Ohori (1999), the notions of *neutral value* and *neutral term* are defined as follows (omitting indexes for readability when possible):

$$\begin{array}{llll} NValue & V, W & ::= & x \mid c \mid * \mid \lambda x.M \mid \langle V, W\rangle, \\ NTerm & M, N & ::= & V \mid xV \mid cV \mid \ let\ \boxdot = xV\ in\ M \mid \ let\ \boxdot = cV\ in\ M \mid \\ & & & let\ \langle y, z\rangle = x\ in\ M \mid \ let\ \langle y, z\rangle = c\ in\ M \mid \\ & & & let\ * = x\ in\ M \ \mid \ let\ * = c\ in\ M, \end{array}$$

where $\boxdot$ means any of x, $\langle x, y\rangle$ or $*$. A *neutral typing judgement* is a typing judgement $\Delta \rhd M : A$ where M is a neutral term.

Lemma 10.1.2. *Suppose that M is a neutral term and that $Erase(M) = Erase(N)$. Then N is a neutral term.*

Proof. Proof by structural induction on M. □

Lemma 10.1.3. *For valid neutral terms $\Delta \rhd M, M' : B$, if $Erase(M) = Erase(M')$ then $\Delta \rhd M \approx_{ax} M' : B$.*

Proof. We prove it by induction on $\dot{M} = Erase(M)$. Note that we also have $\dot{M} = Erase(M')$. Base cases:

Cases $\dot{M} \equiv x$, c or $*$. In each case, there is only one typing rule available for $\Delta \rhd M : B$ and $\Delta \rhd M' : B$, namely (ax_1) in the fist case, (ax_2) in the second case and $(\top.I)$ in the last case. The terms M and M' are determined by the type B, so $M = M'$ and the result is correct by reflexivity.

Induction cases:

Case $\dot{M} \equiv \lambda x.\dot{N}$. In this case, M is of the form $\lambda^n x^A.N$ and M' is of the form $\lambda^{n'} x^{A'}.N'$, where $\dot{N} = Erase(N) = Erase(N')$. Since they are well-typed, from Lemma 9.1.12 the type B is the raw type of M and M' (as in Definition 9.1.8). This implies that $A = A'$, $n = n'$ and that B is equal to $!^n(A \multimap C)$ for some type C. Typing trees for $\Delta \rhd M : B$ and $\Delta \rhd M' : B$ start with a typing rule of the form (λ_i), with $i = 1$ if $n = 0$, $i = 2$ otherwise, and we have the following valid typing judgements:

$$\Delta, x : A \rhd N : C, \qquad\qquad \Delta, x : A \rhd N' : C.$$

By induction hypothesis, $\Delta, x : A \rhd N \approx_{ax} N' : C$. By congruence, we have $\Delta \rhd M \approx_{ax} M' : B$.

Case $\dot{M} \equiv \langle \dot{U}, \dot{V}\rangle$. M is of the form $\langle N, P\rangle^n$ and M' is of the form $\langle N', P'\rangle^{n'}$, where $\dot{U} = Erase(N) = Erase(N')$ and $\dot{V} = Erase(P) = Erase(P')$. Since M and M' are well-typed, from Lemma 9.1.12 the type B is the raw type of M and M'. This implies that $n = n'$ and that B is equal to $!^n(A \otimes C)$ for some types A and C. Typing trees for $\Delta \rhd M : B$ and $\Delta \rhd M' : B$ start with a typing rule of the form $(\otimes.I)$, and we have the following valid typing judgements:

$$!\dot{\Delta}, \Gamma_1 \rhd N : !^n A, \qquad !\dot{\Delta}', \Gamma_1' \rhd N' : !^n A, \qquad !\dot{\Delta}, \Gamma_2 \rhd P : !^n C, \qquad !\dot{\Delta}', \Gamma_2' \rhd P' : !^n C,$$

where $\Delta = (!\dot{\Delta}, \Gamma_1, \Gamma_2) = (!\dot{\Delta}', \Gamma_1', \Gamma_2')$.

From Lemma 9.1.19, it is possible to have Γ_1, Γ_2, Γ_1' and Γ_2' containing only non-duplicable variables. From Lemma 9.1.17, the sets $|\Gamma_1| \cap |\Gamma_2|$ and $|\Gamma_1'| \cap |\Gamma_2'|$ are empty. From Lemma 9.1.7, $FV(N) = FV(N')$ and $FV(P) = FV(P')$.

This means that $\Gamma_1 = \Gamma'_1$, $\Gamma_2 = \Gamma'_2$ and $!\dot{\Delta} = !\dot{\Delta}'$. We can then apply the induction hypothesis twice and deduce

$$!\dot{\Delta}, \Gamma_1 \rhd N \approx_{ax} N' : !^n A, \qquad !\dot{\Delta}, \Gamma_2 \rhd P \approx_{ax} P' : !^n C.$$

Finally, using the congruence rule of the axiomatic equivalence relation, we get that $\Delta \rhd M \approx_{ax} M' : B$.

Case $\dot{M} \equiv x\dot{V}$. Since M and M' are well-typed, from Lemma 9.1.12 the type B is the raw type of M and M'. This implies that M is of the form $x^{A_1 \multimap B} V_1$ and M' is of the form $x^{A_2 \multimap B} V_2$, for some neutral values V_1 and V_2 and some types A_1 and A_2. The typing judgements $\Delta \rhd M, M' : B$ comes from the typing rule (*app*): the variable x occurs in Δ. The context Δ can thus be split into $(\dot{\Delta}, x : !^n(A' \multimap B'))$ for some integer n where, from Lemma 9.1.16, $A_1 <: A'$, $A_2 <: A'$ and $B' <: B$.

Using Lemma 9.1.19, one can construct the typing derivations π and π' of the form

$$\dfrac{\dfrac{}{x : !^n(A' \multimap B') \rhd x^{A_i \multimap B} : A_i \multimap B}\,(ax_1) \qquad \begin{array}{c}\vdots\\ \dot{\Delta}_i \rhd V_i : A_i\end{array}}{\dot{\Delta}, x : !^n(A' \multimap B') \rhd x^{A_i \multimap B} V_i : B}\,(app)$$

where

$$\dot{\Delta}_i = \begin{cases} \dot{\Delta}, x : !^n(A'' \multimap B') & \text{if } x \in FV(V_i), \\ \dot{\Delta} & \text{otherwise,} \end{cases}$$

for $i = 1, 2$. From Lemma 9.1.7, $FV(V_1) = FV(V_2)$. This makes $\dot{\Delta}_1 = \dot{\Delta}_2$. Call it $\dot{\Delta}'$. From Lemma 9.1.26, $\dot{\Delta}' \rhd \{V_1 <: A'\} : A'$ and $\dot{\Delta}' \rhd \{V_2 <: A'\} : A'$ are valid. Applying induction hypothesis, they are axiomatically equivalent.

Finally, applying axiomatic equivalence rule ($app_{<:}$), the axiomatic equivalence

$$\Delta \rhd x^{A_i \multimap B} V_i \approx_{ax} x^{A' \multimap B}\{V_i <: A'\} : B,$$

is valid for $i = 1, 2$. This makes $\Delta \rhd M \approx_{ax} M' : B$ valid by congruence and by transitivity.

Case $\dot{M} \equiv cV$. This case is similar to the previous case.

Since M and M' are well-typed, from Lemma 9.1.12 the type B is the raw type of M and M'. This implies that M is of the form $c^{A_1 \multimap B} V_1$ and M' is of the form $c^{A_2 \multimap B} V_2$, for some neutral values V_1 and V_2 and some types A_1 and A_2 such that $!A_c <: A_1 \multimap B$ and $!A_c <: A_2 \multimap B$. The type A_c is therefore of the form $A' \multimap B'$, with $A_1 <: A'$, $A_2 <: A'$ and $B' <: B$.

The typing judgements $\Delta \rhd M, M' : B$ comes from the typing rule (*app*). Using Lemma 9.1.19, one can construct the typing derivation

$$\dfrac{\dfrac{}{\rhd c^{A_i \multimap B} : A_i \multimap B}\,(ax_1) \qquad \begin{array}{c}\vdots\\ \Delta \rhd V_i : A_i\end{array}}{\Delta \rhd c^{A_i \multimap B} V_i : B}\,(app)$$

for $i = 1, 2$. From Lemma 9.1.26, $\Delta \rhd \{V_1 <: A'\} : A'$ and $\Delta \rhd \{V_2 <: A'\} : A'$ are valid. Applying induction hypothesis, they are axiomatically equivalent.

Finally, applying axiomatic equivalence rule ($app_{<:}$), the axiomatic equivalence

$$\Delta \rhd c^{A_i \multimap B} V_i \approx_{ax} c^{A' \multimap B}\{V_i <: A'\} : B,$$

is valid for $i = 1, 2$. This makes $\Delta \rhd M \approx_{ax} M' : B$ valid by congruence and by transitivity.

Case $\dot{M} \equiv (\mathit{let}\ \boxdot = \boxdot\dot{V}\ \mathit{in}\ \dot{P})$. The placeholder $\boxdot$ stands for $*$, y or $\langle y, z\rangle$, and the placeholder $\boxdot$ for x or c. By α-equivalence one can assume that when relevant, the variables y, z are different from x and are not free in $\dot{V}$.

The terms M and M' are well-typed. From Lemma 9.1.12 the type B is the raw type of M and M'. One can conclude that $M = (\mathit{let}\ \boxdot_1 = \boxdot^{A_1 \multimap B_1} V_1\ \mathit{in}\ P_1)$ and $M' = (\mathit{let}\ \boxdot_2 = \boxdot^{A_2 \multimap B_2} V_2\ \mathit{in}\ P_2)$, where $\dot{V} = \mathit{Erase}(V_1) = \mathit{Erase}(V_2)$, $\dot{P} = \mathit{Erase}(P_1) = \mathit{Erase}(P_2)$, and where ($i$ being $1, 2$)

if $\boxdot$ is...	$\boxdot_i$ is...	B_i is...
$*$	$*$	$\top$
y	y^{C_i}	C_i
$\langle y, z\rangle$	$\langle y^{C_i}, z^{D_i}\rangle^{n_i}$	$!^{n_i}(C_i \otimes D_i)$

for some types C_i and D_i and for some integers n_i.

In the case $\boxdot = x$, the variable x is free in M and in M'. From Lemma 9.1.11 it is in Δ. From Lemma 9.1.16, the type of x in Δ is of the form $!^n(A' \multimap B')$, with $A_1 <: A'$, $A_2 <: A'$, $B' <: B_1$ and $B' <: B_2$.

In the case $\boxdot = c$, we have $!A_c <: (A_1 \multimap B_1)$ and $!A_c <: (A_2 \multimap B_2)$. Let $!^n(A' \multimap B')$ be A_c. We then have $A_1 <: A'$, $A_2 <: A'$, $B' <: B_1$ and $B' <: B_2$.

In both cases, this means that if $\boxdot$ is $\begin{cases} * \\ y \\ \langle y, z\rangle \end{cases}$ then B' is $\begin{cases} \top \\ C' \\ !^{n'}(C' \otimes D'), \end{cases}$ where $C' <: C_i$, $D' <: D_i$, and $n' \geqslant n_i$. Using respectively the identity, the axiomatic equivalence rule $(\mathit{let}^{x}_{<:})$ and the rule $(\mathit{let}^{\otimes}_{<:})$, we get that

$$\Delta \rhd \mathit{let}\ \boxdot_i = \boxdot^{A_i \multimap B_i} V_i\ \mathit{in}\ P_i \approx_{ax} \mathit{let}\ \boxdot' = \boxdot^{A_i \multimap B'} V_i\ \mathit{in}\ P_i : B, \tag{10.1.1}$$

such that if $\boxdot$ is $\begin{cases} * \\ y \\ \langle y, z\rangle \end{cases}$ then $\boxdot'$ is $\begin{cases} * \\ y^{C'} \\ \langle y^{C'}, z^{D'}\rangle^{n'}. \end{cases}$

One can split Δ as $(!\dot{\Delta}, \Gamma_1, \Gamma_2)$ such that Γ_1 contains the free variables of $\dot{V}$ that are not duplicable and Γ_2 contains the free variables of $\dot{P}$ that are not duplicable: from Lemma 9.1.17, $|\Gamma_1|$ and $|\Gamma_2|$ do not intersect. Invoking Lemma 9.1.7 and Lemma 9.1.19, for $i = 1, 2$ one can write a typing derivation

$$\frac{\begin{array}{c}\vdots \\ !\dot{\Delta}, \Gamma_1 \rhd \boxdot^{A_i \multimap B'} V_i : B'\end{array} \quad \begin{array}{c}\vdots \\ !\dot{\Delta}, \Gamma_2, \Lambda \rhd P_i : B\end{array}}{!\dot{\Delta}, \Gamma_1, \Gamma_2 \rhd \mathit{let}\ \boxdot' = \boxdot^{A_i \multimap B'} V_i\ \mathit{in}\ P_i : B}\ (X)$$

where if $\boxdot$ is $\begin{cases} * \\ y \\ \langle y, z\rangle \end{cases}$ then (X) is $\begin{cases} (\top.E) \\ (\mathit{let}) \\ (\otimes.E) \end{cases}$ and Λ is $\begin{cases} () \\ (y : C') \\ (y : !^{n'}C', z : !^{n'}D'). \end{cases}$

One can apply the induction hypothesis on each branch of the derivation. Using the congruence rules of the axiomatic relation and Equation (10.1.1) we obtain finally that $\Delta \rhd M \approx_{ax} M' : B$.

Case $\dot{M} \equiv (\mathit{let}\ \boxdot = \boxdot\ \mathit{in}\ \dot{P})$. The placeholder $\boxdot$ stands for $*$ or $\langle y, z\rangle$, and the placeholder $\boxdot$ for x or c. By α-equivalence one can assume that when relevant, the variables y, z are different from x.

The terms M and M' are well-typed. From Lemma 9.1.12 the type B is the raw type of M and M'. One can conclude that $M = (\mathit{let}\ \boxdot_1 = \boxdot^{B_1}\ \mathit{in}\ P_1)$ and $M' = (\mathit{let}\ \boxdot_2 = \boxdot^{B_2}\ \mathit{in}\ P_2)$, where $\dot{P} = \mathit{Erase}(P_1) = \mathit{Erase}(P_2)$ and where (i being $1, 2$)

if $\boxdot$ is...	$\boxdot_i$ is...	B_i is...
$*$	$*$	$\top$
y	y^{C_i}	C_i
$\langle y, z\rangle$	$\langle y^{C_i}, z^{D_i}\rangle^{n_i}$	$!^{n_i}(C_i \otimes D_i)$

for some types C_i and D_i and for some integers n_i.

In the case $\boxdot = x$, the variable x is free in M and in M'. From Lemma 9.1.11 it is in Δ. From Lemma 9.1.16, the type A' of x in Δ is such that $A' <: B_1$ and $A' <: B_2$.

In the case $\boxdot = c$, we have $!A_c <: B_1$ and $!A_c <: B_2$. Let A' be $!A_c$.

In both cases, this means that if $\boxdot$ is $\begin{cases} * \\ y \\ \langle y, z\rangle \end{cases}$ then A' is $\begin{cases} \top \\ C' \\ !^{n'}(C' \otimes D'), \end{cases}$ where $C' <: C_i$, $D' <: D_i$, and $n' \geqslant n_i$. Using respectively the identity, the axiomatic equivalence rule ($\mathit{let}^x_{<:}$) and the rule ($\mathit{let}^{\otimes}_{<:}$), we get that

$$\Delta \rhd \mathit{let}\ \boxdot_i = \boxdot^{B_i}\ \mathit{in}\ P_i \approx_{ax} \mathit{let}\ \boxdot' = \boxdot^{A'}\ \mathit{in}\ P_i : B, \tag{10.1.2}$$

such that if $\boxdot$ is $\begin{cases} * \\ y \\ \langle y, z\rangle \end{cases}$ then $\boxdot'$ is $\begin{cases} * \\ y^{C'} \\ \langle y^{C'}, z^{D'}\rangle^{n'}. \end{cases}$

Invoking Lemma 9.1.19, for $i = 1, 2$ one can write a typing derivation

$$\frac{\begin{array}{c}\vdots \\ \Gamma \rhd \boxdot^{A'} : A'\end{array} \quad \begin{array}{c}\vdots \\ \Delta, \Lambda \rhd P_i : B\end{array}}{\Delta \rhd \mathit{let}\ \boxdot' = \boxdot^{A'}\ \mathit{in}\ P_i : B}\ (X)$$

where, if $\boxdot$ is $\begin{cases} * \\ y \\ \langle y, z\rangle \end{cases}$ then (X) is $\begin{cases} (\top.E) \\ (\mathit{let}) \\ (\otimes.E) \end{cases}$ and Λ is $\begin{cases} () \\ (y : C') \\ (y : !^{n'}C', z : !^{n'}D'), \end{cases}$ and where, if $\boxdot = c$, $\Gamma = ()$ and if $\boxdot = x$, $\Gamma = (x : A')$.

One can apply the induction hypothesis on each branch of the derivation. Using the congruence rules of the axiomatic relation and Equation (10.1.2) we obtain finally that $\Delta \rhd M \approx_{ax} M' : B$.

This ends the list of cases and proves Lemma 10.1.3. □

10.2 Term Rewriting System Number One

We define a rewriting system of valid terms yielding neutral terms. We want the denotation to stay constant through the rewriting. The rewriting will be two-fold. First we get rid of "complex computations", and then we get rid of beta-reductions. Note that we are not interested in confluence: the only thing we want is normalization.

Definition 10.2.1. We define the notion of *intermediate neutral value* and of *intermediate neutral term* as follows:

$$\begin{array}{llll} \mathit{INValue} & V, W & ::= & x^A \mid c^A \mid *^n \mid \lambda^n x^A.M \mid \langle V, W\rangle^n, \\ \mathit{INTerm} & M, N & ::= & V \mid x^{A\multimap B}V \mid c^{A\multimap B}V \mid \ \mathit{let}\ \boxdot = M\ \mathit{in}\ N, \end{array}$$

where $\boxdot$ is a placeholder for any of $*$, x^A or $\langle x^A, y^B\rangle^n$.

Lemma 10.2.2. *A neutral term is in intermediate neutral form.*

Proof. The proof is done by inspection of the cases. □

Remark 10.2.3. A term in intermediate neutral form is not automatically in neutral form. The two main examples of terms that are intermediate neutral but not neutral are

$$\mathit{let}\ x^A = (\mathit{let}\ y^B = M\ \mathit{in}\ N)\ \mathit{in}\ P, \qquad\qquad \mathit{let}\ x^A = V\ \mathit{in}\ M,$$

where V is a core value.

Lemma 10.2.4. *Suppose that $\Delta \rhd M, M' : B$ are valid typing judgements. Suppose moreover that $\mathit{Erase}(M) = \mathit{Erase}(M')$ and that M is an intermediate neutral term (respectively value). Then M' is also an intermediate neutral form (respectively value).*

Proof. Proof by induction on the size of M. □

Lemma 10.2.5. *Suppose that $\Delta = (!\Delta', \Gamma_1, \Gamma_2)$, that $!\Delta' \rhd V : A$ and $!\Delta', \Gamma_2, x : A \rhd M : B$. If V is an intermediate neutral value and if M is an intermediate neutral term (respectively intermediate neutral value), then $M[V/x]$ is an intermediate neutral term (respectively an intermediate neutral value).*

Proof. Proof by induction on the size of M. The interesting case is when M is of the form $x^{A\multimap B}W$: in this situation, the value V is of type $A\multimap B$, that is, of the form $\lambda^0 y^A.N$ where N is an intermediate neutral term of type B. Then $M[V/x] = (\lambda^0 y^A.N)W$ which is by definition ($\mathit{let}\ y^A = V\ \mathit{in}\ N$), an intermediate neutral term. Note that in this case, M cannot be a value. □

Definition 10.2.6. We define the *rewrite system number one* of terms as the reflexive closure of the relation defined as follows. Given x, y fresh variables, provided that $MN : B$, that $M : A \multimap B$ is not a lambda-abstraction, and that M and N are not both term variables,

$$MN \to_1 \mathit{let}\ x^{A\multimap B} = M\ \mathit{in}\ \mathit{let}\ y^A = N\ \mathit{in}\ x^{A\multimap B}y^A. \tag{rw1.c_1}$$

Provided that $\langle M, N\rangle^n : !^n(A \otimes B)$, and that M and N are not both term variables,

$$\langle M, N\rangle^n \to_1 \mathit{let}\ x^{!^n A} = M\ \mathit{in}\ \mathit{let}\ y^{!^n B} = N\ \mathit{in}\ \langle x^{!^n A}, y^{!^n B}\rangle^n. \tag{rw1.c_2}$$

Finally, we add one congruence rule for each term constructor in the following way: for each rule, M and N are terms such that $M \to_1 M'$ and $N \to_1 N'$.

$$\lambda^n x^A.M \to_1 \lambda^n x^A.M', \tag{rw1.ξ_1}$$

$$MN \to_1 M'N', \tag{rw1.ξ_2}$$

$$\langle M, N\rangle^n \to_1 \langle M', N'\rangle^n, \tag{rw1.ξ_3}$$

$$\mathit{let}\ \langle x^A, y^B\rangle^n = M\ \mathit{in}\ N \to_1 \mathit{let}\ \langle x^A, y^B\rangle^n = M'\ \mathit{in}\ N', \tag{rw1.ξ_4}$$

$$\mathit{let}\ * = M\ \mathit{in}\ N \to_1 \mathit{let}\ * = M'\ \mathit{in}\ N'. \tag{rw1.ξ_5}$$

Lemma 10.2.7. *Suppose that $\Delta \rhd M : C$ is a valid typing judgement, where $M \to_1 M'$. Then $\Delta \rhd M' : C$ is valid and $\Delta \rhd M \approx_{ax} M' : C$.*

Proof. The proof is done by induction of the reduction $M \to_1 M'$.

If reflexivity was used. Then $M = M'$ and the result is immediate.

Case (rw1.c_1). Here, $M \equiv NP$. From Lemma 9.1.12, $B = C$. Suppose that $\Delta \rhd NP : B$, where $N : A \multimap B$. Then one can split Δ into $(!\dot{\Delta}, \Gamma_1, \Gamma_2)$ where

$$\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ !\dot{\Delta}, \Gamma_1 \rhd N : A \multimap B\end{array} \quad \begin{array}{c}\pi_2\\ \vdots\\ !\dot{\Delta}, \Gamma_2 \rhd P : A\end{array}}{!\dot{\Delta}, \Gamma_1, \Gamma_2 \rhd NP : B}\ (app)$$

is a valid typing derivation. Then if x and y are fresh variables, the derivation

$$\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ !\dot{\Delta}, \Gamma_1 \rhd N : A \multimap B\end{array} \quad \dfrac{\begin{array}{c}\pi_2\\ \vdots\\ !\dot{\Delta}, \Gamma_2 \rhd P : A\end{array} \quad \dfrac{\dfrac{}{x : A \multimap B \rhd x^{A \multimap B} : A \multimap B} \quad \dfrac{}{y : A \rhd y^A : A}}{x : A \multimap B, y : A \rhd x^{A \multimap B} y^A : B}\ (app)}{!\dot{\Delta}, \Gamma_2, x : A \multimap B \rhd \mathit{let}\ y^A = P\ \mathit{in}\ x^{A \multimap B} y^A : B}\ (let)}{!\dot{\Delta}, \Gamma_1, \Gamma_2 \rhd \mathit{let}\ x^{A \multimap B} = N\ \mathit{in}\ \mathit{let}\ y^A = P\ \mathit{in}\ x^{A \multimap B} y^A : B}\ (let)$$

is valid. The typing judgements $\Delta \rhd NP : B$ and $\Delta \rhd \mathit{let}\ x^{A \multimap B} = N\, \mathit{in} \mathit{let}\ y^A = P\ \mathit{in}\ x^{A \multimap B} y^A : B$ are axiomatically equivalent by rule (let^{app}).

Case (rw1.c_2). Here, $M \equiv \langle N, P \rangle^n$. From Lemma 9.1.12, $C = !^n(A \otimes B)$. We have therefore $\Delta \rhd \langle N, P \rangle^n : !^n(A \otimes B)$. One can split Δ into $(!\dot{\Delta}, \Gamma_1, \Gamma_2)$ where

$$\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ !\dot{\Delta}, \Gamma_1 \rhd N : !^n A\end{array} \quad \begin{array}{c}\pi_2\\ \vdots\\ !\dot{\Delta}, \Gamma_2 \rhd P : !^n B\end{array}}{!\dot{\Delta}, \Gamma_1, \Gamma_2 \rhd \langle N, P \rangle^n : !^n(A \otimes B)}\ (app)$$

is a valid typing derivation. Then if x and y are fresh variables, the derivation

$$\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ !\dot{\Delta}, \Gamma_1 \rhd N : !^n A\end{array} \quad \dfrac{\begin{array}{c}\pi_2\\ \vdots\\ !\dot{\Delta}, \Gamma_2 \rhd P : !^n B\end{array} \quad \dfrac{\dfrac{}{x : !^n A \rhd x^{!^n A} : !^n A} \quad \dfrac{}{y : !^n B \rhd y^{!^n B} : !^n B}}{x : !^n A, y : !^n B \rhd \langle x^{!^n A} y^{!^n B} \rangle^n : !^n(A \otimes B)}\ (\otimes.I)}{!\dot{\Delta}, \Gamma_2, x : !^n A \rhd \mathit{let}\ y^{!^n B} = P\ \mathit{in}\ \langle x^{!^n A} y^{!^n B} \rangle^n : !^n(A \otimes B)}\ (let)}{!\dot{\Delta}, \Gamma_1, \Gamma_2 \rhd \mathit{let}\ x^{!^n A} = N\ \mathit{in}\ \mathit{let}\ y^{!^n B} = P\ \mathit{in}\ \langle x^{!^n A} y^{!^n B} \rangle^n : !^n(A \otimes B)}\ (let)$$

is valid. The two typing judgements are axiomatically equivalent by rule $(\mathit{let}^{\otimes})$.

Case (rw1.ξ_1). We have $M \equiv \lambda^n x^A.N$. Suppose that $\Delta \rhd \lambda^n x^A.N : C$. Then $C = !^n(A \multimap B)$ for some type B, the judgement comes from $\Delta, x : A \rhd N : B$ using rule (λ_i), where $i = 1$ if $n = 0$, $i = 2$ otherwise. By induction hypothesis, if $N \to_1 N'$ then $\Delta, x : A \rhd N' : B$ is valid. Then, if n is non null, Δ is duplicable, and one can apply (λ_2) to get $\Delta \rhd \lambda^n x^A.N' : C$. Otherwise, $n = 0$ and we apply (λ_1) to get $\Delta \rhd \lambda^n x^A.N' : C$.

By induction hypothesis, $\Delta, x : A \rhd N \approx_{ax} N' : B$. Therefore, by congruence $\Delta \rhd \lambda^n x^A.N \approx_{ax} \lambda^n x^A.N' : C$.

Cases (rw1.ξ_2), (rw1.ξ_3), (rw1.ξ_4) *and* (rw1.ξ_5). We have $M = f(N, P)$, where $f(N, P)$ is respectively NP, $\langle N, P\rangle^n$, $(let * = N\ in\ P)$ and $(let\ \langle x^A, y^B\rangle^n = N\ in\ P)$. Suppose that $\Delta \rhd f(N, P) : C$. Then Δ split into $(!\dot{\Delta}, \Gamma_1, \Gamma_2)$ and the typing judgement is originated from the typing rule (X) and the judgements $!\dot{\Delta}, \Gamma_1 \rhd N : C_1$ and $!\dot{\Delta}, \Gamma_2, \Lambda \rhd P : C_2$, where

if $f(N, P)$ is...	(X) is...	C_1 is...	C_2 is...	C is...
NP	(app)	$A \multimap B$	A	B
$\langle N, P\rangle^n$	$(\otimes.I)$	$!^n A$	$!^n B$	$!^n(A \otimes B)$
$let * = N\ in\ P$	$(\top.E)$	$\top$	B	B
$let\ \langle x^A, y^B\rangle^n = N\ in\ P$	$(\otimes.E)$	$!^n(A \otimes B)$	D	D,

with some types A, B and D, and such that Λ is empty in the three first cases and equal to $(x : !^n A, y : !^n B)$ in the last case.

By induction hypothesis, if $N \to_1 N'$ and $P \to_1 P'$, then $!\dot{\Delta}, \Gamma_1 \rhd N' : C_1$ and $!\dot{\Delta}, \Gamma_2, \Lambda \rhd P' : C_2$ are well-typed. Then, applying (X) we get that $\Delta \rhd f(N', P') : C$ is valid.

By induction hypothesis again, $!\dot{\Delta}, \Gamma_1 \rhd N \approx_{ax} N' : C_1$ and $!\dot{\Delta}, \Gamma_2, \Lambda \rhd P \approx_{ax} P' : C_2$. Therefore, by congruence $\Delta \rhd f(N, P) \approx_{ax} f(N', P') : C$.

This closes the proof of Lemma 10.2.7. □

Lemma 10.2.8. *Suppose that $M, N : C$ and that $Erase(M) = Erase(N)$. If $M \to_1 M'$, then there exists N' such that $Erase(M') = Erase(N')$ and such that $N \to_1 N'$.*

Proof. The proof is done by induction on the reduction $M \to_1 M'$.

If reflexivity was used. Then $M = M'$ and the result is immediate by setting $N = N'$.

Case (rw1.c_1). The term M is of the form $M_1 M_2$, where $M_1 : A \multimap B$ and where $M_1 M_2 : B$. In this case, $M' = (let\ x^{A \multimap B} = M_1\ in\ let\ y^A = M_2\ in\ x^{A \multimap B} y^A)$. If N is such that $Erase(M) = Erase(N)$, then $N = N_1 N_2$, where $Erase(N_1) = Erase(M_1)$ and $Erase(N_2) = Erase(M_2)$.

Since the raw type of N is B, there exists a term A' such that $N_1 : A' \multimap B$. If we define $N' = (let\ x^{A' \multimap B} = N_1\ in\ let\ y^{A'} = N_2\ in\ x^{A' \multimap B} y^{A'})$, then by reduction rule (rw1.c_1), we have $N \to_1 N'$, and by definition of the map *Erase*, the terms M' and N' have the same erasure.

Case (rw1.c_2). The term M is of the form $\langle M_1, M_2\rangle^n$, where $\langle M_1, M_2\rangle^n : !^n(A \otimes B)$. In this case, $M' = (let\ x^{!^n A} = M_1\ in\ let\ y^{!^n B} = M_2\ in\ \langle x^{!^n A}, y^{!^n B}\rangle^n)$. If N is such that $Erase(M) = Erase(N)$, then $N = \langle N_1, N_2\rangle^m$, where $Erase(N_1) = Erase(M_1)$ and $Erase(N_2) = Erase(M_2)$.

Since $N : !^n(A \otimes B)$, the raw type of N is $!^n(A \otimes B)$ If we define $N' = (let\ x^{!^m A} = N_1\ in\ let\ y^{!^m B} = N_2\ in\ \langle x^{!^m A}, y^{!^m B}\rangle^m)$, then by reduction rule (rw1.c_2), we have $N \to_1 N'$, and by definition of the map *Erase*, the terms M' and N' have the same erasure.

Case (rw1.ξ_1). The term M is of the form $\lambda^n x^A.\dot{M}$. In this case, $M' = \lambda^n x^A.\dot{M}'$, where $\dot{M} \to_1 \dot{M}'$.

Since $Erase(M) = Erase(N)$, the term N is also of the form $\lambda^m x^{A'}.\dot{N}$, with $Erase(\dot{M}) = Erase(\dot{N})$. By induction hypothesis there exists $\dot{N}'$ such that $\dot{N} \to_1 \dot{N}'$. Let $N' = \lambda^m x^{A'}.\dot{N}'$. One has $Erase(\dot{N}) = Erase(\dot{N}')$. Using congruence rule (rw1.ξ_1), we deduce that $N \to_1 N'$.

Cases (rw1.ξ_2), (rw1.ξ_3), (rw1.ξ_4) *and* (rw1.ξ_5). In these cases, the term M is equal to $f(M_1, M_2)$, where $f(M_1, M_2)$ is respectively of the form $M_1 M_2$, $\langle M_1, M_2\rangle^n$, $(let * = M_1\ in\ M_2)$ and of the form $(let\ \langle x^A, y^B\rangle^n = M_1\ in\ M_2)$, with $M_1 \to_1 M_1'$ and $M_2 \to_1 M_2'$. We have that $M \to_1 M'$ where $M' = f(M_1', M_2')$.

Since $Erase(M) = Erase(N)$, the term $N = g(N_1, N_2)$, where $g(N_1, N_2)$ is respectively of the form N_1N_2, $\langle N_1, N_2\rangle^m$, $(let * = N_1 \ in \ N_2)$ and $(let \ \langle x^{A'}, y^{B'}\rangle^m = N_1 \ in \ N_2)$.

By induction hypothesis, $N_1 \to_1 N_1'$ and $N_2 \to_1 N_2'$ such that $Erase(N_1') = Erase(M_1')$ and $Erase(N_2') = Erase(M_2')$. This means, using respectively rules (rw1.ξ_2), (rw1.ξ_3), (rw1.ξ_4) and (rw1.ξ_5), that $g(N_1, N_2) \to_1 g(N_1', N_2')$.

Since $Erase(g(N_1', N_2')) = Erase(f(M_1', M_2'))$, we can define $N' = g(N_1', N_2')$ and we have that $Erase(M') = Erase(N')$ and $N \to_1 N'$.

This closes the proof of Lemma 10.2.8. □

Definition 10.2.9. We define the non-negative integer measure $m(M)$ of a typed term M by induction on M as follows (we omit indices for legibility):

$$
\begin{aligned}
m(x)&=0, & m(\lambda x.N)&=m(N),\\
m(c)&=0, & m(NP)&=1+m(N)+m(P),\\
m(*)&=0, & m(\langle N,P\rangle)&=1+m(N)+m(P),\\
m(xy)&=0, & m(let * = N \ in \ P)&=m(N)+m(P),\\
m(\langle x,y\rangle)&=0, & m(let \ x = N \ in \ P)&=m(N)+m(P),\\
& & m(let \ \langle x,y\rangle = N \ in \ P)&=m(N)+m(P).
\end{aligned}
$$

In the case $m(NP)$, we assume that N is not a lambda-abstraction. In the cases $m(NP)$ and $m(\langle N, P\rangle)$, we assume that N and P are not both term variables.

Lemma 10.2.10. *Suppose that $M \to_1 M'$. Then M is a lambda abstraction if and only if M' is.*

Proof. The proof is done by case inspection: there are only two possible reductions of the form $\lambda^n x^A.N \to_1 M'$, namely (rw1.ξ_1), and the identity. In both cases, M' is of the form $\lambda^n x^A.N'$.

Similarly, the only rules of the form $M \to_1 \lambda^n x^A.N'$ are (rw1.ξ_1), and the trivial one. Again, M is of the form $\lambda^n x^A.N$ in both cases. □

Lemma 10.2.11. *Suppose that $M \to_1 M'$ where $M \neq M'$. Then $m(M') < m(M)$.*

Proof. Proof by induction on the reduction. Note that reflexivity is not a possibility since we want $M \neq M'$.

Case (rw1.c_1). In this case, $M' = let \ x^{A\multimap B} = N \ in \ let \ y^A = P \ in \ x^{A\multimap B}y^A$ and $M = NP$ for some types A and B, where N is not a lambda-abstraction and N and P are not both term variables. Since $m(M) = 1 + m(N) + m(P)$ and $m(M') = m(N) + m(P)$, the strict inequality $m(M') < m(M)$ is valid.

Case (rw1.c_2). In this case, $M' = let \ x^{!^n A} = N in let \ y^{!^n B} = P \ in \ \langle x^{!^n A}, y^{!^n B}\rangle^n$ and $M = \langle N, P\rangle^n$ for some types A and B, where N and P are not both term variables. Since $m(M) = 1+m(N)+m(P)$ and $m(M') = m(N) + m(P)$, the inequality $m(M') < m(M)$ is valid.

Case (rw1.ξ_1). In this case, $M = \lambda^n x^A.N$ and $M' = \lambda^n x^A.N'$ where $N \to_1 N'$. By induction hypothesis, $m(N') < m(N)$. By definition, we then have that $m(M') = m(N') < m(N) = m(M)$.

Case (rw1.ξ_2). We have $M = NP$, where $N \to N'$ and $P \to_1 P'$. The terms N and P therefore cannot be both a term variable. By induction hypothesis, $m(N') < m(N)$ or $N' = N$, and $m(P') < m(P)$ or $P' = P$, but equality do not hold for both.

If N is a lambda abstraction, $m(NP) = m(N) + m(P)$. Using Lemma 10.2.10, the term N' is a lambda abstraction: we have $m(N'P') = m(N') + m(P')$. One can conclude that $m(M') = m(N') + m(P') < m(N) + m(P) = m(M)$.

If N is not a lambda abstraction, then we have $m(NP) = 1 + m(N) + m(P)$. By Lemma 10.2.10 N' is not a lambda-abstraction: $m(N'P') = 1 + m(N') + m(P')$. Therefore $m(M') = 1 + m(N') + m(P') < 1 + m(N) + m(P) = m(M)$.

Case (rw1.ξ_3). We have $M = \langle N, P\rangle^n$, where $N \to N'$ and $P \to_1 P'$. The terms N and P therefore cannot be both a term variable. By induction hypothesis, $m(N') < m(N)$ or $N' = N$, and $m(P') < m(P)$ or $P' = P$ (but equality do not hold for both).

We have $m(\langle N, P\rangle^n) = 1 + m(N) + m(P)$ and $m(\langle N', P'\rangle^n) = 1 + m(N') + m(P')$. This is enough to say that $m(M') = 1 + m(N') + m(P') < 1 + m(N) + m(P) = m(M)$.

Cases (rw1.ξ_4) *and* (rw1.ξ_5). We have M respectively of the form $(\mathit{let}\ * = N\ \mathit{in}\ P)$ and of the form $(\mathit{let}\ \langle x^A, y^B\rangle^n = N\ \mathit{in}\ P)$, with $N \to_1 N'$ and $P \to_1 P'$. We have that $M \to M'$ where M' is respectively. $(\mathit{let}\ * = N'\ \mathit{in}\ P')$ and $(\mathit{let}\ \langle x^A, y^B\rangle^n = N'\ \mathit{in}\ P')$

By induction hypothesis, $m(N') < m(N)$ or $N' = N$, and $m(P') < m(P)$ or $P' = P$ (but not $N' = N$ and $P' = P$). Since in both cases, $m(M) = m(N) + m(P)$ and $m(M') = m(N') + m(P')$, we have $m(M') < m(M)$.

This ends the induction, and proves the lemma. □

Lemma 10.2.12. *Rewrite system number one is normalizing: For any term M and any sequence $(M_i)_i$ such that $M = M_0$ and such that for all $i \geqslant 0$, $M_i \to_1 M_{i+1}$, there exists a number n such that for all $i \geqslant n$, $M_i = M_{i+1}$.*

Proof. We prove the lemma by contradiction. Suppose there exists a term M and a sequence $(M_i)_i$ such that $M = M_0$ and such that for all $i \geqslant 0$, $M_i \to_1 M_{i+1}$ and such that for all n there exists a number $n' \geqslant n$ such that $M_{n'} \neq M_n$.

Define a sequence $(n(i))_i$ as follows. First, $n(0) = 0$. Then, for all $i \geqslant 0$, define $n(i+1)$ as the first number $n' > n(i)$ such that $M_{n'} \neq M_{n(i)}$. Note that in this case, for all i we have $M_{n(i)} = M_{n(i+1)-1}$ and $M_{n(i+1)-1} \neq M_{n(i+1)}$. By Lemma 10.2.11, this means that $m(M_{n(i+1)}) \leqslant m(M_{n(i)}) - 1$. We prove by induction on i that for all i, $m(M_{n(i)}) \leqslant m(M) - i$:

Case $i = 0$. In this case, $M_{n(0)} = M_0 = M$, therefore $m(M_{n(i)}) = m(M) - 0$.

Case $i > 0$. Suppose that $m(M_{n(i)}) \leqslant m(M) - i$. Since $m(M_{n(i+1)}) < m(M_{n(i+1)-1})$, we have $m(M_{n(i+1)}) \leqslant m(M_{n(i)}) - 1 \leqslant m(M) - i - 1 \leqslant m(M) - (i+1)$.

Therefore, by induction on i, $m(M_{n(i)}) \leqslant m(M) - i$. But for $i > m(M)$, the value $m(M_i)$ is negative, contradicting the non-negativity of the measure. Therefore there exists a number n such that for all $i \geqslant n$, $M_i = M_{i+1}$. □

Lemma 10.2.13. *Suppose that $\Delta \rhd M : B$ is valid and that for all M' such that $M \to_1 M'$, $M = M'$. Then M is in intermediate neutral form.*

Proof. We prove the lemma by induction on the size of M.

Case $M \equiv x^A$, c^A *and* $*^n$. Then M is in intermediate neutral form.

Case $M \equiv \lambda^n x^A.N$. The only applicable rule is (rw1.ξ_1), meaning that the set of possible M' is the set of terms of the form $\lambda^n x^A.N'$, where $N \to_1 N'$. Since $M = M'$, we have $N = N'$. By induction hypothesis, N is in intermediate neutral form. This makes M in intermediate neutral form.

Case $M \equiv NP$. There are three possible subcases:

Case N *and* P *are both term variables.* Then M is in intermediate neutral form.

Case $N \equiv \lambda^n x^A.\dot{N}$. The only applicable rewriting rule is (rw1.ξ_2). Since $\Delta \rhd M : B$ is valid, $n = 0$ and $M \equiv \mathit{let}\ x^A = P\ \mathit{in}\ \dot{N}$.

In this case, the set of M' such that $M \to_1 M'$ is spanned by all the terms of the form $\mathit{let}\ x^A = P'\ \mathit{in}\ \dot{N}'$, when $\dot{N} \to_1 \dot{N}'$ and $P \to_1 P'$. Since $M' = M$, one has $\dot{N}' = \dot{N}$ and $P' = P$. By induction hypothesis, this means that $\dot{N}$ and P are in intermediate neutral form.

The term M is thus in intermediate neutral form.

Otherwise. It is always possible to apply rule (rw1.c_1), in which case M' is not equal to M. This case is not to be considered.

Case $M \equiv \langle N, P\rangle^n$. There are two possible subcases:

Case N *and* P *are both term variables.* Then M is in intermediate neutral form.

Otherwise. It is always possible to apply rule (rw1.c_2), in which case M' is not equal to M. This case is not to be considered.

Cases $M \equiv (\mathit{let}\ * = N\ \mathit{in}\ P)$ *and* $(\mathit{let}\ \langle x^A, y^B\rangle^n = N\ \mathit{in}\ P)$. For these two cases, the only applicable rules are respectively (rw1.ξ_5) and (rw1.ξ_4), which means that M' is respectively equal to $(\mathit{let}\ * = N'\ \mathit{in}\ P')$ and $(\mathit{let}\ \langle x^A, y^B\rangle^n = N'\ \mathit{in}\ P')$, with $N \to_1 N'$ and $P \to_1 P'$. Since $M = M'$, we have $N = N'$ and $P = P'$. By induction hypothesis, N and P are in intermediate neutral form. This makes M in intermediate neutral form.

By exhaustion of the cases, the lemma is verified. □

10.3 Term Rewriting System Number Two

We now define a rewriting system that takes intermediate neutral terms and returns neutral terms. The proof of the convergence of the rewriting system requires several notions of measures on terms.

10.3.1 The Rewriting System

Definition 10.3.1. We define the *rewrite system number two* of valid typed intermediate neutral terms. We write $M \to_2 N$ if either $M = N$, or $M \to_{2\beta} N$ or $M \to_{2c} N$, where $(\to_{2\beta})$ and $(\to_{2c})$ are defined as follows. First,

$$\mathit{let}\ x^A = V\ \mathit{in}\ M \to_{2\beta} M[V/x], \tag{rw2.β_1}$$

$$\mathit{let}\ \langle x^A, y^B\rangle^n = \langle V, W\rangle^n\ \mathit{in}\ M \to_{2\beta} M[V/x, W/y], \tag{rw2.β_2}$$

$$\mathit{let}\ * = *\ \mathit{in}\ M \to_{2\beta} M, \tag{rw2.β_3}$$

(provided that x, y are fresh variables). Then,

$$\begin{aligned} &\mathit{let}\ \boxdot = (\mathit{let}\ \boxdot = M\ \mathit{in}\ N)\ \mathit{in}\ P \\ &\quad \to_{2c} \mathit{let}\ \boxdot = M\ \mathit{in}\ \mathit{let}\ \boxdot = N\ \mathit{in}\ P, \end{aligned} \tag{rw2.c_1}$$

$$
\begin{aligned}
&\mathit{let}\ \ x^{!^n(D\multimap C)} = \lambda^n y^D.R\ \mathit{in}\ \mathit{let}\ \boxdot^{C'} = x^{D'\multimap C'} S\ \mathit{in}\ P \\
&\quad \to_{2c} \mathit{let}\ x^{!^n(D\multimap C)} = \lambda^n y^D.R\ \ \mathit{in} \\
&\qquad \mathit{let}\ \ y^{D'} = S\ \mathit{in}\ \mathit{let}\ \boxdot^{C'} = \{R <: C'\}\ \mathit{in}\ P, && \text{(rw2.}c_2\text{)} \\
&\mathit{let}\ \ x^{!^n(C\otimes D)} = \langle V, W\rangle^n\ \mathit{in}\ \mathit{let}\ \langle y^{C'}, z^{D'}\rangle^m = x^{!^m(C'\otimes D')}\ \mathit{in}\ P \\
&\quad \to_{2c} \mathit{let}\ x^{!^n(C\otimes D)}\ = \langle V, W\rangle^n\ \ \mathit{in} \\
&\qquad \mathit{let}\ \ y^{!^m C'} = \{V <: !^m C'\}\ \mathit{in}\ \mathit{let}\ z^{!^m D'} = \{W <: !^m D'\}\ \mathit{in}\ P, && \text{(rw2.}c_3\text{)} \\
&\mathit{let}\ \ x^{!^n\top} = *^n\ \mathit{in}\ \mathit{let}\ * = x^\top\ \mathit{in}\ P \\
&\quad \to_{2c} \mathit{let}\ x^{!^n\top} = *^n\ \mathit{in}\ P, && \text{(rw2.}c_4\text{)}
\end{aligned}
$$

where $\boxdot$ ranges over $\{*, z^A, \langle z^A, t^B\rangle^n\}$ and $\boxdot$ over $\{*, u^C, \langle u^C, v^D\rangle^n\}$, where z, t, u, v are fresh variables. Then we add one congruence rule for each term constructor in the following way: for each rule, M and N are terms such that $M \to_{2i} M'$ and $N \to_{2i} N'$, where i is β or c.

$$
\begin{aligned}
\lambda^n x^A.M &\to_{2i} \lambda^n x^A.M', && \text{(rw2.}\xi_1\text{)} \\
MN &\to_{2i} M'N', && \text{(rw2.}\xi_2\text{)} \\
\langle M, N\rangle^n &\to_{2i} \langle M', N'\rangle^n, && \text{(rw2.}\xi_3\text{)} \\
\mathit{let}\ \langle x^A, y^B\rangle^n = M\ \mathit{in}\ N &\to_{2i} \mathit{let}\ \langle x^A, y^B\rangle^n = M'\ \mathit{in}\ N', && \text{(rw2.}\xi_4\text{)} \\
\mathit{let}\ * = M\ \mathit{in}\ N &\to_{2i} \mathit{let}\ * = M'\ \mathit{in}\ N'. && \text{(rw2.}\xi_5\text{)}
\end{aligned}
$$

If $M \to_{2i} N$ if and only if $M = N$, we say that M is in $\to_{2i}$*-normal form.*

Lemma 10.3.2. *Let M be an intermediate neutral term (respectively intermediate neutral value). If $M \to_2 N$, then N is an intermediate neutral term (respectively intermediate neutral value).*

Proof. Proof by structural induction on M. □

Lemma 10.3.3. *If $Erase(M) = Erase(N)$ and if $M \to_2 M'$ there exists N' such that $Erase(M') = Erase(N')$ and such that $N \to_2 N'$.*

Proof. Proof by induction on the reduction $M \to_2 M'$. □

Lemma 10.3.4. *Suppose that $\Delta \rhd M : A$ is a valid typing judgements such that $M \to_2 N$. Then $\Delta \rhd N : A$ is valid and $\Delta \rhd M \approx_{ax} N : A$.*

Proof. Proof by induction on the reduction $M \to_2 N$. □

Lemma 10.3.5. *Let M be an intermediate neutral term. If M is in $\to_{2\beta}$-normal form and in $\to_{2c}$-normal form then it is a neutral term.*

Proof. Proof by structural induction on M. □

Lemma 10.3.6. *Suppose that V is an intermediate neutral value such that $V \to_2 V'$. Suppose that $M \to_2 M'$. Then $M[V/x] \to_2^* M[V'/x]$ and $M[V/x] \to_2^* M'[V/x]$.*

Proof. Proof by structural induction on M. □

Lemma 10.3.7. *Let V and W be intermediate neutral values and let M be a neutral term.*

- *If $(\mathit{let}\ x^A = V\ \mathit{in}\ N) \to_2^* M$ then $N[V/x] \to_2^* M$;*
- *If $(\mathit{let}\ \langle x^A, y^B\rangle^n = \langle V, W\rangle^n\ \mathit{in}\ N) \to_2^* M$ then $N[V/x, W/y] \to_2^* M$;*

- *If* $(\mathit{let} * = * \mathit{in}\ N) \to_2^* M$ *then* $N \to_2^* M$*;*

Proof. We prove the result for all the cases in parallel. We consider a term P of the form $(\mathit{let}\ x^A = V\ \mathit{in}\ N)$, $(\mathit{let}\ \langle x^A, y^B\rangle^n = \langle V, W\rangle^n\ \mathit{in}\ N)$ or $(\mathit{let} * = * \mathit{in}\ N)$ reducing to a neutral term M in n step and proceed be induction on n.

In each case, there is at least one reduction possible (using respectively (rw2.β_1), (rw2.β_2) and (rw2.β_3)), and then the reduction $P \to_2^* M$ can be decomposed into $P \to_2 M' \to_*^2 M$. The first reduction can be one of the following cases:

(rw2.β_1), (rw2.β_2) *and* (rw2.β_3). Then M' is respectively of the form $N[V/x]$, $N[V/x, W/y]$ and N, and we are done.

(rw2.c_2). Then M' is of the form $\mathit{let}\ x^A = V\ \mathit{in}\ N'$. By induction hypothesis, $N'[V/x]$ reduces to M. Since $N[V/x] = N'[V/x]$, we have the result.

(rw2.c_3). This case is similar to the previous one. M' is of the form

$$\mathit{let}\ x_1^{A_1} = V_1\ \mathit{in}\ \mathit{let}\ x_2^{A_2} = V_2\ \mathit{in}\ N',$$

and $V = \langle V_1, V_2\rangle^n$. Applying the induction hypothesis twice and noticing that $N[V/x] \to_2 N'[V_1/x_1, V_2/x_2]$, we obtain that $N[V/x] \to_2^* M$

(rw2.c_4). In this case, $V = *^n$ and M' is of the form $\mathit{let}\ x^A = *^n\ \mathit{in}\ N'$. Applying induction hypothesis, we get that $N'[V/x] \to_2^* M$. Since $N[V/x] \to_2 N'[V/x]$, we get the result.

Congruence cases. These cases are taken care of by using the induction hypothesis, Lemma 10.3.6 and Lemma 10.3.2. □

10.3.2 Height of Terms

Definition 10.3.8. Let $h(M)$ be the *height* of the term M to be the positive integer defined as follows:

$$\begin{aligned}
h(x^A) = h(c^A) = h(*^n) &= 1,\\
h(\lambda^n x^A.M) &= h(M) + 1\\
h(\langle V, W\rangle^n) &= h(V) + h(W) + 1,\\
h(c^{A\multimap B} V) &= h(V) + 1,\\
h(x^{A\multimap B} V) &= h(V) + 1,\\
h(\mathit{let}\ \boxdot = M\ \mathit{in}\ N) &= 2^{h(M)} + h(N).
\end{aligned}$$

Lemma 10.3.9. *Suppose that* $M : A$ *and that* $A <: B$*. Then* $h\{M <: B\} = h(M)$*.*

Proof. Proof by structural induction on M. □

Lemma 10.3.10. *For all intermediate neutral terms* P*,* Q *and* R*, the following equality is valid:*

$$h(\mathit{let}\ \boxdot = P\ \mathit{in}\ \mathit{let}\ \boxdot = Q\ \mathit{in}\ R) < h(\mathit{let}\ \boxdot = (\mathit{let}\ \boxdot = P\ \mathit{in}\ Q)\ \mathit{in}\ R).$$

Proof. We have

$$\begin{aligned}
h(\mathit{let}\ \boxdot = (\mathit{let}\ \boxdot = P\ \mathit{in}\ Q)\ \mathit{in}\ R) &= 2^{2^{h(P)} + h(Q)} + h(R)\\
&= 2^{2^{h(P)}} \cdot 2^{h(Q)} + h(R)
\end{aligned}$$

and

$$h(\mathit{let}\ \boxdot = (\mathit{let}\ \boxdot = P\ \mathit{in}\ Q)\ \mathit{in}\ R) = 2^{h(P)} + 2^{h(Q)} + h(R).$$

Since for all $x \geqslant 1$ and for all $y, z \geqslant 2$, the inequalities $2^x < 2^{2^x}$ and $y + z \leqslant y \cdot z$ hold, the lemma is valid. □

10.3.3 Degree of Terms

Definition 10.3.11. We define the *degree* $\delta(A)$ *of a type* A as follows:

$$\delta(\top) = 1, \qquad \delta(\alpha) = 1, \qquad \delta(!A) = \delta(A),$$
$$\delta(A \otimes B) = \delta(A \multimap B) = \max(\delta(A), \delta(B)) + 1.$$

The degree of an intermediate neutral term is defined as follows:

$$\begin{aligned} d(x^A) &= 0, & d(x^{A \multimap B} M) &= d(M), \\ d(c^A) &= 0, & d(c^{A \multimap B} M) &= d(M), \\ d(*^n) &= 0, & d(\lambda^n x^A.M) &= d(M). \end{aligned}$$

$$\begin{aligned} d(\langle V, W\rangle^n) &= \max(d(V), d(W)) \\ d(\mathit{let}\ x^A = V\ \mathit{in}\ N^C) &= \max(d(V), d(N), \delta(A \multimap C)) \\ d(\mathit{let}\ \langle x^A, y^B\rangle^n = \langle V, W\rangle^n\ \mathit{in}\ N^C) &= \max(d(V), d(W), d(N), \delta(A \multimap C), \delta(B \multimap C)) \\ d(\mathit{let}\ * = *\ \mathit{in}\ N^C) &= \max(d(N), d(\top \multimap C)) \\ d(\mathit{let}\ \boxdot = (\mathit{let}\ \boxdot = M\ \mathit{in}\ N)\ \mathit{in}\ P) &= d(\mathit{let}\ \boxdot = M\ \mathit{in}\ \mathit{let}\ \boxdot = N\ \mathit{in}\ P) \end{aligned}$$

and in all other cases,

$$d(\mathit{let}\ \boxdot = M\ \mathit{in}\ N) = \max(d(M), d(N))$$

Lemma 10.3.12. *For all types* A*,* $\delta(A) \geqslant 0$*.*

Proof. Proof by structural induction on A. □

Lemma 10.3.13. *For all terms* M*,* $d(M) \geqslant 0$*.*

Proof. Proof by structural induction on M, using Lemma 10.3.12. □

Lemma 10.3.14. *If* $A <: B$*, then* $\delta(A) = \delta(B)$*.*

Proof. By structural induction on a derivation of $A <: B$. □

Lemma 10.3.15. *Suppose that* $M : A$ *and that* $A <: B$*. Then* $d(M) = d\{M <: B\}$*.*

Proof. By structural induction on M. □

Lemma 10.3.16. *Suppose that* M *is a neutral term. Then* $d(M) = 0$*.*

Proof. Proof by structural induction on M. □

Lemma 10.3.17. *The following inequalities are valid:*

$$\begin{aligned} d(\mathit{let}\ \langle x^A, y^B\rangle^n = M\ \mathit{in}\ N^C) &\leqslant \max(\delta(A \multimap C), \delta(B \multimap C), d(M), d(N)),\\ d(\mathit{let}\ x^A = M\ \mathit{in}\ N^C) &\leqslant \max(\delta(A \multimap C), d(M), d(N)),\\ d(\mathit{let}\ * = M\ \mathit{in}\ N^C) &\leqslant \max(\delta(\top \multimap C), d(M), d(N)),\\ d(\mathit{let}\ \boxdot^A = M\ \mathit{in}\ N) &\geqslant \max(d(M), d(N)). \end{aligned}$$

Proof. The proof is done by induction on the height of $\mathit{let}\ \boxdot^A = M\ \mathit{in}\ N^C$. We proceed by case distinction on $(\boxdot^A = M)$.

Cases $(x^A = V)$,$(\langle x^A, y^B\rangle^n = \langle V, W\rangle^n)$ *and* $(* = *)$. In these three cases, the result is true by the definition of the degree.

Case $(\boxdot^A = (\mathit{let}\ \boxdot^B = P\ \mathit{in}\ Q))$. By definition:

$$d(\mathit{let}\ \boxdot^A = M\ \mathit{in}\ N) = d(\mathit{let}\ \boxdot^B = P\ \mathit{in}\ \mathit{let}\ \boxdot^A = Q\ \mathit{in}\ N), \tag{10.3.1}$$

$$\max(\delta(A \multimap C), d(M), d(N)) = \max(\delta(A \multimap C), d(\mathit{let}\ \boxdot^B = P\ \mathit{in}\ Q), d(N)), \tag{10.3.2}$$

$$\max(d(M), d(N)) = \max(d(\mathit{let}\ \boxdot^B = P\ \mathit{in}\ Q), d(N)), \tag{10.3.3}$$

We prove the equality by case distinction on $(\boxdot^B = P)$:

Subcase $(x^B = V)$. The terms in Equations (10.3.1)-(10.3.3) become

$$\begin{aligned} d(\mathit{let}\ \boxdot^A = M\ \mathit{in}\ N) &= \max(d(P), d(\mathit{let}\ \boxdot^A = Q\ \mathit{in}\ N), \delta(B \multimap C)),\\ \max(\delta(A \multimap C), d(M), d(N)) &= \max(d(P), d(Q), d(N), \delta(A \multimap C), \delta(B \multimap C)),\\ \max(d(M), d(N)) &= \max(d(P), d(Q), d(N), \delta(B \multimap C)). \end{aligned}$$

By induction hypothesis,

$$\max(d(Q), d(N)) \leqslant d(\mathit{let}\ \boxdot^A = Q\ \mathit{in}\ N) \leqslant \max(d(Q), d(N), \delta(A \multimap C)).$$

Therefore,

$$\begin{aligned} &\max(d(P), d(Q), d(N), \delta(B \multimap C))\\ &\leqslant \max(d(P), d(\mathit{let}\ \boxdot^A = Q\ \mathit{in}\ N), \delta(B \multimap C))\\ &\leqslant \max(d(P), d(Q), d(N), \delta(A \multimap C), \delta(B \multimap C)). \end{aligned}$$

Subcases $(\langle x^B, y^{B'}\rangle^n = \langle V, W\rangle^n)$ *and* $(* = *)$. These cases are similar to the previous one.

Subcase $(\boxdot^B = (\mathit{let}\ \boxdot^D = R\ \mathit{in}\ S))$. Then

$$\begin{aligned} &\max(\delta(A \multimap C), d(M), d(N))\\ &\quad = \max(\delta(A \multimap C), d(\mathit{let}\ \boxdot^D = R\ \mathit{in}\ \mathit{let}\ \boxdot^B = S\ \mathit{in}\ Q), d(N)), \end{aligned} \tag{10.3.4}$$

$$\max(d(M), d(N)) = \max(d(\mathit{let}\ \boxdot^D = R\ \mathit{in}\ \mathit{let}\ \boxdot^B = S\ \mathit{in}\ Q), d(N)). \tag{10.3.5}$$

By definition,

$$d(\mathit{let}\ \boxdot^A = (\mathit{let}\ \boxdot^B = (\mathit{let}\ \boxdot^D = R\ \mathit{in}\ S)\ \mathit{in}\ Q)\ \mathit{in}\ N)$$

$$
\begin{aligned}
&= d(\mathit{let}\ \boxdot^B = (\mathit{let}\ \boxdot^D = R\ \mathit{in}\ S)\ \mathit{in}\ \mathit{let}\ \boxdot^A = Q\ \mathit{in}\ N)\\
&= d(\mathit{let}\ \boxdot^D = R\ \mathit{in}\ \mathit{let}\ \boxdot^B = S\ \mathit{in}\ \mathit{let}\ \boxdot^A = Q\ \mathit{in}\ N)\\
&= d(\mathit{let}\ \boxdot^A = (\mathit{let}\ \boxdot^D = R\ \mathit{in}\ \mathit{let}\ \boxdot^B = S\ \mathit{in}\ Q)\ \mathit{in}\ N).
\end{aligned}
$$

Since the height of the right hand side is smaller than the height of the left hand side, one can apply the induction hypothesis and use Equations (10.3.4) and (10.3.4) to get that

$$
\begin{aligned}
&\max(d(M), d(N))\\
&= \max(d(\mathit{let}\ \boxdot^D = R\ \mathit{in}\ \mathit{let}\ \boxdot^B = S\ \mathit{in}\ Q), d(N))\\
&\leqslant d(\mathit{let}\ \boxdot^A = (\mathit{let}\ \boxdot^B = (\mathit{let}\ \boxdot^D = R\ \mathit{in}\ S)\ \mathit{in}\ Q)\ \mathit{in}\ N)\\
&\leqslant \max(\delta(A \multimap C), d(\mathit{let}\ \boxdot^D = R\ \mathit{in}\ \mathit{let}\ \boxdot^B = S\ \mathit{in}\ Q), d(N))\\
&= \max(\delta(A \multimap C), d(M), d(N)).
\end{aligned}
$$

Subcase: catch-all case. The terms in Equations (10.3.1)-(10.3.3) become

$$
\begin{aligned}
d(\mathit{let}\ \boxdot^A = M\ \mathit{in}\ N) &= \max(d(P), d(\mathit{let}\ \boxdot^A = Q\ \mathit{in}\ N)),\\
\max(\delta(A \multimap C), d(M), d(N)) &= \max(d(P), d(Q), d(N), \delta(A \multimap C)),\\
\max(d(M), d(N)) &= \max(d(P), d(Q), d(N)).
\end{aligned}
$$

We conclude by using the induction hypothesis:

$$\max(d(Q), d(M)) \leqslant d(\mathit{let}\ \boxdot^A = Q\ \mathit{in}\ N) \leqslant \max(\delta(A \multimap C), d(Q), d(M)).$$

Last case: catch-all case. Here, $d(\mathit{let}\ \boxdot^A = M\ \mathit{in}\ N) = \max(d(M), d(N))$, which makes the result immediate.

This closes the proof of Lemma 10.3.17. □

10.3.4 Last Notion of Measure: Number of a Term

Definition 10.3.18. Let $e^i_{A_1,\ldots,A_n}$ be 1 when $i = \max(\delta(A_1), \ldots, \delta(A_n))$ and 0 otherwise. We define the *i-th number* $n^i(M)$ *of* M to be a non-negative integer as follows:

$$
\begin{aligned}
n^i(x^A) &= 0, & n^i(x^{A\multimap B}M) &= n^i(M),\\
n^i(c^A) &= 0, & n^i(c^{A\multimap B}M) &= n^i(M),\\
n^i(*^n) &= 0, & n^i(\lambda^n x^A.M) &= n^i(M).
\end{aligned}
$$

$$
\begin{aligned}
n^i(\langle V, W\rangle^n) &= n^i(V) + n^i(W)\\
n^i(\mathit{let}\ x^A = V\ \mathit{in}\ N^C) &= n^i(V) + n^i(N) + e^i_{A\multimap C}\\
n^i(\mathit{let}\ \langle x^A, y^B\rangle^n = \langle V, W\rangle^n\ \mathit{in}\ N^C) &= n^i(\langle V, W\rangle^n) + n^i(N) + e^i_{A\multimap C, B\multimap C}\\
n^i(\mathit{let}\ * = *\ \mathit{in}\ N^C) &= n^i(N) + e^i_{\top\multimap C}\\
n^i(\mathit{let}\ \boxdot^A = (\mathit{let}\ \boxdot^B = M\ \mathit{in}\ N)\ \mathit{in}\ P) &= n^i(\mathit{let}\ \boxdot^B = M\ \mathit{in}\ \mathit{let}\ \boxdot^A = N\ \mathit{in}\ P)
\end{aligned}
$$

and in all other cases,

$$n^i(\mathit{let}\ \boxdot = M\ \mathit{in}\ N) = n^i(M) + n^i(N)$$

We define $n(M) = n^{d(M)}(M)$, and call it the *maximal number of* M.

Lemma 10.3.19. *Suppose that $M : A$ and that $A <: B$. Then $n^i(M) = n^i\{M <: B\}$.*

Proof. Proof by structural induction on M. □

Lemma 10.3.20. *For all i we have $n^i(let\ \boxdot = M\ in\ N) \geqslant \max(n^i(M), n^i(N))$.*

Proof. Proof by induction on M. □

Lemma 10.3.21. *Suppose that $M : C$ and that $i > d(M)$. Then $n^i(M) = 0$.*

Proof. The proof is done by double induction on the size and on the height of M.

Case $M \equiv \langle V, W\rangle^n$. If $i > d(M)$, then $i > d(V)$ and $i > d(W)$. By induction hypothesis (the size of V and the size of W being strictly smaller than the size of M), we have $n^i(V) = n^i(W) = 0$. Since $n^i(M) = n^i(V) + n^i(W)$, it is also null.

Case $M \equiv (let\ \langle x^A, y^B\rangle^n = \langle V, W\rangle^n\ in\ N)$. Here, $d(M) = max(d(V), d(W), d(N), \delta(A \multimap C), \delta(B \multimap C))$, and $n^i(M) = n^i(V) + n^i(W) + n^i(N) + e^i_{A\multimap C, B\multimap C}$. If $i > d(M)$, by induction hypothesis, $n^i(V) = n^i(W) = n^i(N) = 0$. Also, by definition $e^i_{A\multimap C, B\multimap C} = 0$. Thus $n^i(M) = 0$

Case $M \equiv (let\ \boxdot = (let\ \boxdot = N\ in\ P)\ in\ Q)$. In this case, $M \to_{2c} M'$ where $M' = let\ \boxdot = N\ in\ let\ \boxdot = P\ in\ Q$. By Lemma 10.3.10, $h(M') < h(M)$. Since $d(M) = d(M')$ and $n^i(M) = n^i(M')$, by induction hypothesis $n^i(M) = 0$.

The remaining cases are similar. □

10.3.5 Putting Everything Together

The degree, the number and the height of a term are related: if M is not neutral, there exists N such that $M \to_2 N$ and such that the triple $(d(N), n(N), h(N))$ is smaller with the lexicographic order to the triple $(d(M), n(M), h(M))$.

Definition 10.3.22. We define a function μ from intermediate neutral terms to $\mathbb{N}^3$ as follows:

$$\mu(M) = (d(M), n(M), h(M)).$$

We consider the lexicographic ordering on $\mathbb{N}^3$: $(a, b, c) < (a', b', c')$ if and only if

$$\Big(a < a'\Big) \vee \Big(\big(a = a'\big) \wedge \big(b < b'\big)\Big) \vee \Big(\big(a = a'\big) \wedge \big(b = b'\big) \wedge \big(c < c'\big)\Big).$$

Lemma 10.3.23. *Suppose that $(a_n, b_n, c_n)_n$ is a sequence in $\mathbb{N}^3$ such that for all n, the inequality $(a_{n+1}, b_{n+1}, c_{n+1}) \leqslant (a_n, b_n, c_n)$ holds. Then there exists an integer n_0 such that for all $n \geqslant n_0$, $(a_n, b_n, c_n) = (a_{n_0}, b_{n_0}, c_{n_0})$.*

Proof. Consider the sequence $(a_n)_n$. By definition, $a_{n+1} \leqslant a_n$. Since this is a decreasing sequence of natural numbers: there exists n_a such that $a_n = a_{n_a}$ for all $n \geqslant n_a$. Consider the sequence $(b_n)_{n\geqslant n_a}$. By definition, it is a decreasing sequence of natural numbers. Again, there exists some $n_b \geqslant n_a$ such that for all $n \geqslant n_b$, $b_n = b_{n_b}$. Finally, consider the sequence $(c_n)_{n\geqslant n_b}$. Again by definition, it is a decreasing sequence of natural numbers: there exists some $n_0 \geqslant n_b$ such that or all $n \geqslant n_0$, $c_n = c_{n_0}$. Thus, for all $n \geqslant n_0$, $(a_n, b_n, c_n) = (a_{n_0}, b_{n_0}, c_{n_0})$. □

Lemma 10.3.24. *Suppose that let $x^A = V$ in $M : B$, where V is an intermediate neutral value and M an intermediate neutral term. Suppose moreover that let $x^A = V$ in M is in normal form with respect to $\to_{2c}$. Then*

1. *$d(M[V/x]) \leqslant \max(d(M), d(V), \delta(A))$;*
2. *for all $i \geqslant \max(d(M), d(V), \delta(A)+1)$, $n^i(M[V/x]) = n^i(M) + \xi_M \cdot n^i(V)$, where ξ_M is the number of free occurrences of x in M.*

Proof. First note that is *let $x^A = V$ in M* is in normal form with respect to $\rightarrow_{2c}$, then any subterm of V and M are also in normal form with respect to $\rightarrow_{2c}$. The proof is done by induction on the size of M plus the size of V. Cases where M is an intermediate neutral value:

Case $M \equiv x^B$. Then $A <: B$, and $M[V/x] = \{V <: B\}$.

1. From Lemma 10.3.15, $d\{V <: B\} = d(V) \leqslant \max(d(M), d(V), \delta(A)+1)$.
2. From Lemma 10.3.19, for all integers i, $n^i\{V <: B\} = n^i(V) = n^i(M) + 1 \cdot n^i(V)$.

Cases $M \equiv y^B$, $M \equiv c^B$ and $M \equiv *^n$. In these three cases, $M[V/x] = M$.

1. $d(M[V/x]) = d(M) \leqslant \max(d(M), d(V), \delta(A)+1)$.
2. Since $\xi_M = 0$, for all integers i, $n^i(M[V/x]) = n^i(M) + \xi_M \cdot n^i(V)$.

Case $M \equiv \lambda^n y^C.N$. The type B is of the form $!^n(C \multimap D)$, with $N : D$.

Using α-equivalence one can assume that $y \neq x$. Therefore, the term $M[V/x]$ is equal to $\lambda^n y^C.(N[V/x])$. By induction hypothesis,

$$\begin{aligned}
&d(N[V/x]) \leqslant \max(d(N), d(V), \delta(A)),\\
&\forall i \geqslant \max(d(N), d(V), \delta(A)+1),\ n^i(N[V/x]) = n^i(N) + \xi_N \cdot n^i(V).
\end{aligned}$$

Note that $M[V/x] = \lambda^n y^C.N[V/x]$. Therefore:

1. Since $d(M) = d(N)$ and $d(M[V/x]) = d(N[V/x])$,
$$d(M[V/x]) \leqslant \max(d(N), d(V), \delta(A)+1) = \max(d(M), d(V), \delta(A)+1).$$
2. Since $x \neq y$, we have $\xi_M = \xi_N$. Finally, since $n^i(M) = n^i(N)$ and since $d(M) = d(N)$, for all $i \geqslant \max(d(M), d(V), \delta(A)+1)$,
$$n^i(M[V/x]) = n^i(N[V/x]) = n^i(N) + \xi_N \cdot n^i(V) = n^i(M) + \xi_M \cdot n^i(V).$$

Case $M \equiv \langle W_1, W_2\rangle^n$. The substitution $M[V/x]$ is equal to $\langle W_1[V/x], W_2[V/x]\rangle^n$, and the type B is of the form $!^n(B_1 \otimes B_2)$ with $W_1 : !^n B_1$ and $W_2 : !^n B_2$.

By induction hypothesis, for $j = 1, 2$

$$\begin{aligned}
&d(W_j[V/x]) \leqslant \max(d(W_j), d(V), \delta(A)),\\
&\forall i \geqslant \max(d(W_j), d(V), \delta(A)+1),\ n^i(W_j[V/x]) = n^i(W_j) + \xi_{W_j} \cdot n^i(V).
\end{aligned}$$

Note that $M[V/x] = \langle W_1[V/x], W_2[V/x]\rangle^n$.

1. The degree of $M[V/x]$ satisfies the following:
$$\begin{aligned}
d(M[V/x]) &= \max(d(W_1[V/x]), d(W_2[V/x]))\\
&\leqslant \max(d(W_1), d(W_2), d(V), \delta(A))\\
&= \max(d(M), d(V), \delta(A)).
\end{aligned}$$

2. Since $\xi_M = \xi_{W_1} + \xi_{W_2}$ and since for $j = 1, 2$ we have $d(M) \geqslant d(W_j)$, then for all i greater or equal to $\max(d(M), d(V), \delta(A) + 1)$,

$$\begin{aligned} n^i(M[V/x]) &= n^i(W_1[V/x]) + n^i(W_2[V/x]) \\ &= n^i(W_1) + \xi_{W_1} \cdot n^i(V) + n^i(W_2) + \xi_{W_2} \cdot n^i(V) \\ &= n^i(M) + \xi_M \cdot n^i(V). \end{aligned}$$

The cases where M is not a value are the following.

Case $M \equiv x^{C \multimap B} W$. Here, W is an intermediate neutral value with $W : C$. From Lemma 9.1.16, $A <: C \multimap B$. This means that $A = !^n(C' \multimap B')$, where $B' <: B$ and $C <: C'$.

Since V is of type $!^n(C' \multimap B')$, it is of the form $\lambda^n y^{C'}.N$, and then

$$\begin{aligned} M[V/x] &= \{\lambda^n y^{C'}.N <: C \multimap B\}(W[V/x]) \\ &= (\lambda^0 y^C.\{N <: B\})(W[V/x]) \\ &= (\mathit{let}\ y^C = W[V/x]\ \mathit{in}\ \{N <: B\}). \end{aligned}$$

By induction hypothesis,

$$\begin{aligned} &d(W[V/x]) \leqslant \max(d(W), d(V), \delta(A)), \\ &\forall i \geqslant \max(d(W), d(V), \delta(A) + 1),\ n^i(W[V/x]) = n^i(W) + \xi_W \cdot n^i(V). \end{aligned}$$

From Lemma 10.2.5, $W[V/x]$ is an intermediate neutral value. Note that $M[V/x] = (\mathit{let}\ y^C = W[V/x]\ \mathit{in}\ \{N <: B\})$.

1. Using Lemma 10.3.14, $\delta(A) = \delta(C' \multimap B') = \delta(C \multimap B)$. From Lemma 10.3.15, we have $d\{N <: B\} = d(N)$. Finally, $d(N) = d(V)$ and $d(M) = d(W)$. Thus,

$$\begin{aligned} d(M[V/x]) &= \max(d(W[V/x]), d\{N <: B\}, \delta(C \multimap B)) \\ &\leqslant \max(d(W), d(V), \delta(A), d\{N <: B\}, \delta(C \multimap B)) \\ &\leqslant \max(d(M), d(V), \delta(A)). \end{aligned}$$

2. From Lemma 10.3.19, $n^i\{N <: B\} = n^i(N)$. By definition, $n^i(N) = n^i(V)$ and $n^i(W) = n^i(M)$. Finally, $\xi_M = 1 + \xi_W$. Since $d(M) = d(W)$, for all $i \geqslant \max(d(M), d(V), \delta(A) + 1)$, we have $e^i_{C \multimap B} = 0$ and thus

$$\begin{aligned} n^i(M[V/x]) &= n^i(W[V/x) + n^i\{N <: B\} = n^i(W) + \xi_W \cdot n^i(V) + n^i\{N <: B\} \\ &= n^i(M) + \xi_W \cdot n^i(V) + n^i(V) = n^i(M) + \xi_M \cdot n^i(V). \end{aligned}$$

Cases $M \equiv c^{C \multimap B} W$ *and* $M \equiv y^{C \multimap B} W$*, with* $y \neq x$. The term W is an intermediate neutral value such that $W : C$ and the term $M[V/x]$ is respectively $c^{C \multimap B}(W[V/x])$, $y^{C \multimap B}(W[V/x])$. From Lemma 10.2.5, $W[V/x]$ is an intermediate neutral value. By induction hypothesis,

$$\begin{aligned} &d(W[V/x]) \leqslant \max(d(W), d(V), \delta(A)), \\ &\forall i \geqslant \max(d(W), d(V), \delta(A) + 1),\ n^i(W[V/x]) = n^i(W) + \xi_W \cdot n^i(V). \end{aligned}$$

Note that $M[V/x] = c^{C \multimap B}(W[V/x])$.

1. The degree of $M[V/x]$ is equal to $d(W[V/x])$. Thus $d(M[V/x]) \leqslant \max(d(W), d(V), \delta(A)) = \max(d(M), d(V), \delta(A))$.

2. Since $d(M) = d(W)$ and $\xi_M = \xi_W$, for all $i \geqslant \max(d(M), d(V), \delta(A) + 1)$,

$$n^i(M[V/x]) = n^i(W[V/x]) = n^i(W) + \xi_W \cdot n^i(V) = n^i(M) + \xi_M \cdot n^i(V).$$

Case $M \equiv (\mathit{let}\ y^C = W\ \mathit{in}\ N)$. Here, the term W is an intermediate neutral value, and $W : C$ and $N : B$. Using α-equivalence, one can assume that $y \neq x$. We have $M[V/x] = (\mathit{let}\ y^C = W[V/x]\ \mathit{in}\ N[V/x])$.

By induction hypothesis,

$$\begin{aligned}
&d(W[V/x]) \leqslant \max(d(W), d(V), \delta(A)),\\
&d(N[V/x]) \leqslant \max(d(N), d(V), \delta(A)),\\
&\forall i \geqslant \max(d(W), d(V), \delta(A) + 1),\ n^i(W[V/x]) = n^i(W) + \xi_W \cdot n^i(V),\\
&\forall i \geqslant \max(d(N), d(V), \delta(A) + 1),\ n^i(N[V/x]) = n^i(N) + \xi_N \cdot n^i(V).
\end{aligned}$$

From Lemma 10.2.5, $W[V/x]$ is an intermediate neutral value. Note that the degree $d(M)$ is equal to $\max(d(W), d(N), \delta(C \multimap B))$:

1. The degree of $M[V/x]$ satisfies

$$\begin{aligned}
d(M[V/x]) &= \max(d(W[V/x]), d(N[V/x]), \delta(C \multimap B))\\
&\leqslant \max(d(W), d(V), \delta(A), d(N), d(V), \delta(A), \delta(C \multimap B))\\
&= \max(d(M), d(V), \delta(A)).
\end{aligned}$$

2. Note that $\xi_M = \xi_W + \xi_N$, that $d(M) \geqslant d(W)$ and $d(M) \geqslant d(N)$, and that $n^i(M) = n^i(W) + n^i(N) + e^i_{C \multimap B}$. For all $i \geqslant \max(d(M), d(V), \delta(A) + 1)$, we have

$$\begin{aligned}
n^i(M[V/x]) &= n^i(W[V/x]) + n^i(N[V/x]) + e^i_{C \multimap B}\\
&= n^i(W) + \xi_W \cdot n^i(V) + n^i(N) + \xi_N \cdot n^i(V) + e^i_{C \multimap B}\\
&= n^i(M) + \xi_M \cdot n^i(V).
\end{aligned}$$

Case $M \equiv \mathit{let}\ \langle y^C, z^D \rangle^n = \langle W_1, W_2 \rangle^n\ \mathit{in}\ N$.
Idem.

Case $M \equiv \mathit{let}\ * = *\ \mathit{in}\ N$.
Idem.

Case $M \equiv \mathit{let}\ \boxdot = (\mathit{let}\ \boxdot = N\ \mathit{in}\ P)\ \mathit{in}\ Q$. In this case,

$$\dot{M} = (\mathit{let}\ \boxdot = N\ \mathit{in}\ \mathit{let}\ \boxdot = P\ \mathit{in}\ Q)$$

is such that $M \to_{2c} \dot{M}$ but $M \neq \dot{M}$. Therefore M is not in $\to_{2c}$-normal form: We do not consider this case.

Catch-all case: $M \equiv \mathit{let}\ \boxdot = N\ \mathit{in}\ P$. Here, $\boxdot$ is respectively $*$, y^C and $\langle y^C, z^D \rangle^n$. By α-equivalence, one can assume that $x \neq y$ and $x \neq z$. Then by definition, the term $M[V/x]$ is equal to $(\mathit{let}\ \boxdot = N[V/x]\ \mathit{in}\ P[V/x])$. We claim that if we are indeed in the catch-all case, then so is $(\mathit{let}\ \boxdot = N[V/x]\ \mathit{in}\ P[V/x])$.

We study the four other possible cases for $(\boxdot = N[V/x])$:

Case $(y^C = W)$. Here, W is a value. For $W = N[V/x]$ to be a value, N must be a value: we are not in the catch-all case.

Case ($\langle y^C, z^D\rangle^n = \langle V_1, V_2\rangle^n$). This situation occurs either when N is a core value of the form $\langle W_1, W_2\rangle^n$, not possible since we are in the catch-all case, or when N is the variable $x^{!^n(C\otimes D)}$ and $V = \langle V_1, V_2\rangle^n$. We have

$$\begin{aligned}
&(let\ x^A = V\ in\ M)\\
&\qquad = (let\ x^{!^n(C\otimes D)} = \langle W_1, W_2\rangle^n\ in\ let\ \langle y^C, z^D\rangle^n = x^{!^n(C\otimes D)}\ in\ P)\\
&\qquad \to_{2c} let\ x^{!^n(C\otimes D)} = \langle W_1, W_2\rangle^n\ in\ let\ y^{!^n C} = W_1\ in\ let\ z^{!^n D} = W_2\ in\ P.
\end{aligned}$$

which contradicts the hypothesis of the lemma. Therefore this case is not to be considered.

Case ($* = *$). This case is obtained when either $N = *$, not possible since we are in the catch-all case, or when $N = x^\top$ and $V = *^n$. In this situation, $A = !^n\top$. We have

$$\begin{aligned}
(let\ x^A = V\ in\ M) &= (let\ x^{!^n\top} = *^n\ in\ let\ * = x^\top\ in\ P\\
&\to_{2c} let\ x^{!^n\top} = *^n\ in\ P
\end{aligned}$$

which contradicts the hypothesis of the lemma. Therefore this case is not to be considered.

Case ($\boxdot^C = (let\ \boxdot^D = Q\ in\ R)$). There are two subcases. Either $N \equiv let\ \boxdot^D = S\ in\ T$, but then we are not in the catch-all case, or we have $N \equiv x^{D\multimap C}S$ with S a value and $V \equiv \lambda^n y^{D'}.R$. In this case,

$$\begin{aligned}
(let\ x^A = V\ in\ M) &= (let\ x = \lambda y.R\ in\ let\ \boxdot^C = (let\ y^D = S\ in\ R)\ in\ P)\\
&\to_{2c} let\ x = \lambda y.R\ in\ let\ y^D = S\ in\ let\ \boxdot^C = R\ in\ P,
\end{aligned}$$

which contradicts the hypothesis of the lemma. Therefore this case is not to be considered.

Thus the term $M[V/x] = (let\ \boxdot^C = N[V/x]\ in\ P[V/x])$ is in the catch-all form. By induction hypothesis,

$$\begin{aligned}
&d(N[V/x]) \leqslant \max(d(N), d(V), \delta(A)),\\
&d(P[V/x]) \leqslant \max(d(P), d(V), \delta(A)),\\
&\forall i \geqslant \max(d(N), d(V), \delta(A)+1),\ n^i(N[V/x]) = n^i(N) + \xi_N \cdot n^i(V),\\
&\forall i \geqslant \max(d(P), d(V), \delta(A)+1),\ n^i(P[V/x]) = n^i(P) + \xi_P \cdot n^i(V).
\end{aligned}$$

Now:

1. Since the degrees $d(M)$ and $d(M[V/x])$ are respectively equal to $\max(d(P), d(N))$ and equal to $\max(d(P[V/x]), d(N[V/x]))$, we have $d(M[V/x]) \leqslant \max(d(M), d(V), \delta(A))$.
2. We have $\xi_M = \xi_N + \xi_P$, $d(M) \geqslant d(N)$ and $d(M) \geqslant d(P)$. Therefore, for all integer $i \geqslant \max(d(M), d(V), \delta(A)+1)$,

$$\begin{aligned}
n^i(M[V/x]) &= n^i(N[V/x]) + n^i(P[V/x])\\
&= n^i(N) + \xi_N \cdot n^i(V) + n^i(P) + \xi_P \cdot n^i(V)\\
&= n^i(M) + \xi_M \cdot n^i(V).
\end{aligned}$$

Thus, by induction the degree of $M[V/x]$ is smaller or equal to $\max(d(M), d(V), \delta(A))$, and for all integers i bigger or equal to $\max(d(M), d(V), \delta(A)+1)$, $n^i(M[V/x]) = n^i(M) + \xi_M \cdot n^i(V)$. □

Lemma 10.3.25. *Suppose that $M : B$ is in $\to_{2c}$-normal form. Then one of the following three statements is valid:*

1. *M is in $\rightarrow_{2\beta}$-normal form.*
2. *$n(M) = 0$ and there exists $N \neq M$ such that $M \rightarrow_2 N$ with $d(N) < d(N)$.*
3. *$n(M) \neq 0$ and there exists $N \neq M$ such that $M \rightarrow_2 N$ with $d(M) < d(N)$ or $d(M) = d(N)$ and $n(N) < n(M)$.*

Proof. Suppose that M is not in $\rightarrow_{2\beta}$-normal form. We prove the result by induction on M.

Case $M \equiv (let\ \boxdot = (let\ \boxdot = P\ in\ Q)\ in\ R)$. In this case, M is not in $\rightarrow_{2c}$-normal form. Therefore this case is not to be considered.

Case $M \equiv (let\ x^A = V\ in\ P)$. The degree of M is equal to $\max(d(V), d(P), \delta(A \multimap B))$, and $n(M) = n^{d(M)}(P) + n^{d(M)}(V) + e^{d(M)}_{A\multimap B}$. Note that since M is in $\rightarrow_{2c}$-normal form, so are V and P.

Subcase $n(M) = 0$. We construct a term N such that $d(N) < d(M)$ and $M \rightarrow_{2\beta} N$.

From Lemma 10.3.13, $n^{d(M)}(P) = n^{d(M)}(V) = e^{d(M)}_{A\multimap B} = 0$. Since $d(M) \geqslant \delta(A \multimap B)$, this means $d(M) > \delta(A \multimap B)$. Note that $d(V), d(P) \leqslant d(M)$.

Suppose that $d(V) = d(M)$. Then $n(V) = n^{d(M)}(V) = 0$. Since M is in $\rightarrow_{2c}$-normal form, so is V. Then by induction hypothesis, there exists V' such that $V \rightarrow_{2\beta} V'$ with $d(V') < d(V)$. From Lemma 10.3.2 the term V' is an intermediate neutral value. If $d(V) < d(M)$, define V' to be V. Therefore, in both cases we obtain an intermediate neutral value V' such that $V \rightarrow_{2\beta} V'$ with $d(V') < d(M)$.

Suppose that $d(P) = d(M)$. Then $n(P) = n^{d(M)}(P) = 0$. By induction hypothesis, there exists P' such that $P \rightarrow_{2\beta} P'$ with $d(P') < d(P)$. If $d(P) < d(M)$, define P' to be P. Therefore, in both cases we obtain a term P' such that $P \rightarrow_{2\beta} P'$ with $d(P') < d(M)$.

Define N to be $(let\ x^A = V'\ in\ P')$. Since V' is an intermediate neutral value, $d(N) = \max(d(V'), d(P'), \delta(A \multimap B))$.

Since $d(V'), d(P'), \delta(A \multimap B) < d(M)$, we also have $d(N) < d(M)$.

Subcase $n(M) \neq 0$. We construct a term N such that $M \rightarrow_{2\beta} N$ with $d(N) < d(M)$ or with $d(M) = d(N)$ and $n(N) < n(M)$.

Suppose that $n^{d(M)}(P) \neq 0$. From Lemma 10.3.21 and since $d(P) \leqslant d(M)$, the degree of P is equal to the degree of M. Therefore, $n(P) \neq 0$ and we can apply the induction hypothesis: there exists a term P' such that $P \rightarrow_{2\beta} P'$, $d(P') = d(P)$ and $n(P') < n(P)$. Let N be $(let\ x^A = V\ in\ P')$. We have $M \rightarrow_{2\beta} N$ with $d(N) = \max(d(V), d(P'), \delta(A \multimap B)) = d(M)$ and $n(N) = n^{d(M)}(P') + n^{d(M)}(V) + e^{d(M)}_{A\multimap B} < n(M)$

Now, suppose that $n^{d(M)}(V) \neq 0$. Similarly, from Lemma 10.3.21 and since $d(V) \leqslant d(M)$, the degree $d(V) = d(M)$. This means $n(V) \neq 0$ and we can apply the induction hypothesis: there exists a term V' such that $V \rightarrow_{2\beta} V'$, $d(V') = d(V)$ and $n(V') < n(V)$. Let N be $(let\ x^A = V'\ in\ P)$. We have $M \rightarrow_{2\beta} N$ with $d(N) = \max(d(V'), d(P), \delta(A \multimap B)) = d(M)$ and $n(N) = n^{d(M)}(P) + n^{d(M)}(V') + e^{d(M)}_{A\multimap B} < n(M)$

Finally, suppose that both $n^{d(M)}(V)$ and $n^{d(M)}(P)$ are null. Then $n(M) = e^{i}_{A\multimap B} = 1$. Let N be $P[V/x]$. Since V is an intermediate neutral value, $M \rightarrow_{2\beta} N$. By Lemma 10.3.24, $d(N) \leqslant \max(d(P), d(V), \delta(A)) \leqslant d(M)$ and for all $i \geqslant \max(d(P), d(V), \delta(A)+1)$, $n^i(N) = n^i(P) + \xi_P \cdot n^i(V)$. If $d(N) < d(M)$, we are done. If $d(N) = d(M)$, then $d(M) \geqslant \max(d(P), d(V), \delta(A) + 1)$, thus $n(N) = n^{d(M)}(N) = 0$. Hence $d(N) < d(M)$, and we are also done.

The two cases $M \equiv (let\ \langle x^A, y^B\rangle^n = \langle V, W\rangle^n\ in\ P)$ and $N \equiv (let\ * = *\ in\ M)$ are done similarly. Finally, the congruence cases are trivial. □

Definition 10.3.26. We say that an intermediate term M satisfies *reduction* if the following is valid:

" If $M \to_{2c} N$, then there exists N' such that $M \to_2^* N'$ and $\mu(N') < \mu(M)$. "

Lemma 10.3.27. *Let M be an intermediate neutral term satisfying reduction, then either M is a neutral term or there exists N such that $\mu(N) < \mu(M)$ and $M \to_2^* N$.*

Proof. Consider an intermediate neutral term M satisfying reduction. If it is already a neutral term, we are done. If it is not, from Lemma 10.3.5 then it is not in $\to_2$-normal form. There are two cases:

M is not in $\to_{2c}$-normal form. By the reduction property of M there is N such that $M \to_2^* N$ and $\mu(N) < \mu(M)$.

M is in $\to_{2c}$-normal form. It is not in $\to_{2\beta}$-normal form, and from Lemma 10.3.25 there exists N such that $M \to_{2\beta} N$ with $\mu(N) < \mu(M)$.

In both cases, we have as requested the existence of some intermediate neutral term N such that $M \to_2^* N$ with $\mu(N) < \mu(M)$. □

Corollary 10.3.28. *Let M be an intermediate neutral term satisfying reduction. Then there exists a neutral term N such that $M \to_2^* N$.*

Proof. Let S be the set of terms M such that $M \to_2^* M'$. Let $I = \mu(S)$. From Lemma 10.3.23, this set has a minimum element. Let $N \in S$ be a term such that $\mu(N)$ is this minimum. Lemma 10.3.27 states that N is neutral. □

Lemma 10.3.29. *Every intermediate term satisfies reduction, as stated in Definition 10.3.26.*

Proof. Suppose that $M \to_{2c} N$. We prove the result by induction on the size M. Case distinction on the last rule used in the derivation of $M \to_{2c} N$:

Case (rw2.c_1). In this case, by definition, the two sides of the relation have the same degree and the same maximal number, but a smaller height by Lemma 10.3.10. Choose N' to be equal to N and this satisfies the lemma.

Case (rw2.c_2). Suppose that $P : B$. The left hand side is

$$M = (\mathit{let}\ x^{!^n(D \multimap C)} = \lambda^n y^D.R\ \mathit{in}\ \mathit{let}\ \boxdot^{C'} = x^{D' \multimap C'} S\ \mathit{in}\ P.)$$

Its degree is $d(M) = \max(d(R), \delta((D \multimap C) \multimap B), d(S), d(P))$. Note that S is an intermediate value.

The right hand side is

$$N = (\mathit{let}\ x^{!^n(D \multimap C)} = \lambda^n y^D.R\ \ \mathit{in}\ \mathit{let}\ y^{D'} = S\ \mathit{in}\ \mathit{let}\ \boxdot^{C'} = \{R <: C'\}\ \mathit{in}\ P).$$

Its degree is

$$d(N) = \max(d(R), \delta((D \multimap C) \multimap B), d(S), \delta(D' \multimap B), d(\mathit{let}\ \boxdot^{C'} = \{R <: C'\}\ \mathit{in}\ P)).$$

From Lemma 10.3.17, we get $d(N) = d(M)$.

We proceed by case distinction on $d(M)$ to find the term N'.

Suppose that $d(M) > \delta((D \multimap C) \multimap B)$. Consider X being the terms among R, S and P such that $d(X) = d(M)$. Whether X are or not in $\to_{2c}$-normal form, one of Lemma 10.3.25 or the induction hypothesis applies: Either X is in β-normal form, and thus by Lemma 10.3.16 $d(X) = 0$, or X reduces to X' with $d(X') < d(X)$ or with $d(X') = d(X) = d(M)$ and $n(X') < n(M)$. In this last case, since $d(X') = d(M)$, this means that $n(N') < n(M)$ and $d(N') = d(M)$ if N' is M with X' in place of X.

Otherwise, we have a term N' with $M \to_2 N'$ and $d(N') < d(M)$.

Suppose that $d(M) = 1 + \delta(D \multimap C)$ *and* $d(M) > 1 + \delta(B)$. The terms R and S are smaller than M: By induction hypothesis they satisfy the lemma. Then by Lemma 10.3.28, there exist neutral terms R' and S' such that $R \to_2^* R'$ and $S \to_2^* S'$. By Lemma 10.3.16, we have $d(R') = 0$ and $d(S') = 0$.

Since $\mathit{let}\ x^{!^n(D \multimap C)} = \lambda^n y^D.R\ \mathit{in}\ P$ is smaller than M, by the induction hypothesis one can apply Lemma 10.3.28, and it reduces to a neutral term P'. From Lemma 10.3.7, $P[V/x]$ reduces to the same neutral term P'. Therefore M reduces to

$$\begin{aligned} M &\to_2 \mathit{let}\ \ y^D = S'[\lambda^n y^D.R'/x]\ \mathit{in}\ \mathit{let}\ \boxdot^{C'} = \{R' <: C'\}\ \mathit{in}\ P[\lambda^n y^D.R/x] \\ &\to_2 \mathit{let}\ \ y^D = S'[\lambda^n y^D.R'/x]\ \mathit{in}\ \mathit{let}\ \boxdot^{C'} = \{R' <: C'\}\ \mathit{in}\ P'. \end{aligned}$$

Let N' be this last term. Since from Lemma 9.1.32, $S'[\lambda^n y^D.R'/x]$ is a value, the term $\mathit{let}\ x^{D \multimap C} = \lambda^n y^D.R'\ \mathit{in}\ S'$ is in $2c$-normal form. Thus Lemma 10.3.24 applies:

$$\begin{aligned} d(N') &= \max(\delta(D \multimap B), d(S'[\lambda^n y^D.R'/x]), d(\mathit{let}\ \boxdot^{C'} = \{R' <: C'\}\ \mathit{in}\ P')) \\ &\leqslant \max(\delta(D \multimap B), d(S'), d(R'), \delta(D \multimap C), d(\mathit{let}\ \boxdot^{C'} = \{R' <: C'\}\ \mathit{in}\ P')). \\ &\leqslant \max(\delta(D \multimap B), d(S'), d(R'), \delta(D \multimap C), d(\{R' <: C'\}), d(P'), \delta(C' \multimap B)). \end{aligned}$$

Since $d(S') = d(R') = d(P') = 0$ (they are neutral), and since $d(M) > \delta(D) + 1$, $d(M) > \delta(C) + 1$ and $d(M) > \delta(B) + 1$, we have $d(N') < d(M)$.

Suppose that $d(M) = 1 + \delta(B)$. Let $Q = (\mathit{let}\ y^{D'} = S\ \mathit{in}\ \mathit{let}\ \boxdot^{C'} = \{R <: C'\}\ \mathit{in}\ P)$. The term Q is smaller than M, so induction hypothesis applies.

Suppose that Q is not in $\to_{2c}$-normal form. Then there exists Q' such that $Q \to_2^* Q'$ with $\mu(Q') < \mu(Q)$. Let $N' = (\mathit{let}\ x^{!^n(D \multimap C)} = \lambda^n y^D.R\ \ \mathit{in}\ Q')$. Note that $M \to_2^* N'$. We have $d(N') = \max(\delta((D \multimap C) \multimap B), d(R), d(Q')) = 1 + \delta(B) = d(M)$, and $n(N') \leqslant n(M)$. Finally, since $h(Q') < h(Q)$, we have $h(N') < h(M)$. Therefore $\mu(N') < \mu(M)$.

Now, suppose that Q is in $\to_{2c}$ normal form. Then so is N, and we have the result using Lemma 10.3.25.

Case (rw2.c_3). Suppose that $P : B$. The left hand side is

$$M = (\mathit{let}\ x^{!^n(C \otimes D)} = \langle V, W\rangle^n\ \mathit{in}\ \mathit{let}\ \langle y^{C'}, z^{D'}\rangle^m = x^{!^m(C' \otimes D')}\ \mathit{in}\ P)$$

Its degree is $d(M) = \max(d(V), d(W), \delta((D \otimes C) \multimap B), d(P))$. The right hand side is

$$\begin{aligned} N = (\mathit{let}\ x^{!^n(C \otimes D)}\ &= \langle V, W\rangle^n\ \mathit{in} \\ &\mathit{let}\ \ y^{!^m C'} = \{V <: !^m C'\}\ \mathit{in}\ \mathit{let}\ z^{!^m D'} = \{W <: !^m D'\}\ \mathit{in}\ P), \end{aligned}$$

and its degree is the same as the one of M.

We proceed by case distinction on $d(M)$ to find the term N'. The cases are treated similarly as in the previous case.

Suppose that $d(M) > \delta((D \otimes C) \multimap B)$. Using Lemma 10.3.16, one can find V', W' and P' such that $V \to_2^* V'$, $W \to_2^* W$ and $P \to_2^* P'$ such that

$$d(\mathit{let}\ x^{!^n(C\otimes D)} = \langle V', W'\rangle^n\ \mathit{in}\ \mathit{let}\ \langle y^{C'}, z^{D'}\rangle^m = x^{!^m(C'\otimes D')}\ \mathit{in}\ P') < d(M).$$

Suppose that $d(M) = 1 + \delta(D \otimes C)$ *and* $d(M) > 1 + \delta(B)$. Since V, W, and $\mathit{let}\ x = \langle V, W\rangle^n\ \mathit{in} P$ are strictly smaller than M, the induction hypothesis applies and they satisfy the lemma. Thus Lemma 10.3.28 applies and they reduces to some neutral terms V', W' and P'. From Lemma 10.3.7, $P[\langle V, W\rangle^n/x]$ also reduces to P'. Therefore M reduces to

$$N' = (\mathit{let}\ \langle y^{C'}, z^{D'}\rangle^m = \langle V', W'\rangle^n\ \mathit{in}\ P').$$

We have

$$d(N') = \max(d(V'), d(W'), d(P'), \delta(C' \multimap B), \delta(D' \multimap B)).$$

From Lemma 10.3.16, $d(N') < d(M)$.

Suppose that $d(M) = 1 + \delta(B)$. This time, we choose

$$Q = (\mathit{let}\ y^{!^m C'} = \{V <: !^m C'\}\ \mathit{in}\ \mathit{let}\ z^{!^m D'} = \{W <: !^m D'\}\ \mathit{in}\ P)$$

The term Q being smaller than N, the induction hypothesis applies.

Suppose that it is not in $\to_{2c}$-normal form. Then there exists Q' such that $Q \to_2^* Q'$ with $\mu(Q') < \mu(Q)$. Let $N' = (\mathit{let}\ x^{!^n(C\otimes D)} = \langle V', W'\rangle^n\ \mathit{in}\ Q')$. We have $M \to_2^* N'$, with $d(N') = d(M)$, $n(N') \leqslant n(M)$, and $h(N') < h(M)$. Thus $\mu(N') < \mu(M)$.

Now, suppose that it is in $\to_{2c}$-normal form. Then so is N, and we have the result by Lemma 10.3.25.

Case (rw2.c_4). In this case,

$$M = (\mathit{let}\ x^{!^n\top} = *^n\ \mathit{in}\ \mathit{let}\ * = x^\top\ \mathit{in}\ P), \qquad N = (\mathit{let}\ x^{!^n\top} = *^n\ \mathit{in}\ P).$$

Note that $d(M) = d(N)$, that $n(N) = n(P) \leqslant n(M)$ and that $h(N) < h(M)$. Therefore $\mu(N) < \mu(M)$. Choose $N' = N$.

Cases (rw2.ξ_1)-(rw2.ξ_5). One can assume that M is not in one of the form requested by the previous cases. The induction hypothesis can therefore be applied and the result is obtained directly.

By case exhaustion, the lemma is then true. □

Lemma 10.3.30. *Suppose that M is an intermediate neutral term. Then either M is a neutral term or there exists N such that $\mu(N) < \mu(M)$ and $M \to_2^* N$.*

Proof. From Lemma 10.3.29, every intermediate term satisfies reduction. Then from Lemma 10.3.27, for all intermediate neutral term M, either M is a neutral term or there exists N such that $\mu(N) < \mu(M)$ and $M \to_2^* N$. □

10.4 Proof of Theorem 9.2.7

The proof of Theorem 9.2.7 is done in two steps, by reducing first to intermediate neutral terms and then to neutral terms. We make these steps explicit in Lemma 10.4.1 and Lemma 10.4.2.

Lemma 10.4.1. *Suppose that $\Delta \rhd M, N : A$ are two valid typing judgements such that $Erase(M) = Erase(N)$. Then there exists two intermediate neutral terms M' and N' such that $\Delta \rhd M' : A$ and $\Delta \rhd N' : A$ are valid, such that $Erase(M') = Erase(N')$ and such that $\Delta \rhd M \approx_{ax} M' : A$ and $\Delta \rhd N \approx_{ax} N' : A$.*

Proof. Let S be the set of all possible sequences of terms $(M_i)_i$ such that $M_0 = 0$ and $M_i \to_1 M_{i+1}$.

From Lemma 10.2.12, one can define the function f from S to the integers picking out the smallest n such that for all $i \geqslant n$, $M_i = M_n$.

Consider the function g mapping a sequence $(M_i)_i$ in S to the value $m(M_{f((M_i)_i)})$. Since the measure yields non-negative integers, the set $g(S)$ is a subset of the non-negative integers. It contains then an attainable minimal element m_0. Let $(M_i)_i$ be a sequence of S such that $g((M_i)_i) = m_0$.

Let $n_0 = f((M_i)_i)$. We claim that M_{n_0} is an intermediate neutral term. Indeed, suppose otherwise. Then by Lemma 10.2.13, there exists a term M' such that $M_{n_0} \to_1 M'$ and $M' \neq M_{n_0}$. We can now construct a sequence of terms $(\dot{M}_i)_i$ as follows: $\dot{M}_i = M_i$ for all $i = 0 \ldots n_0$, $\dot{M}_i = M'$ for all $i > n_0$. Note that $(\dot{M}_i)_i$ belongs in S. However, using Lemma 10.2.11 we deduce that $g((\dot{M}_i)_i) < m_0$, contradicting the minimality of m_0.

Now, using Lemma 10.2.8, one can construct a sequence of terms $(N_i)_i$ such that $N_0 = N$ and such that for all i, $Erase(N_i) = Erase(M_i)$ and $N_i \to_1 N_{i+1}$. Using Lemma 10.2.7, for all i the typing judgements $\Delta \rhd M_{n_0}, N_{n_0} : B$ are valid, and $\Delta \rhd M \approx_{ax} M_{n_0} : B$ and $\Delta \rhd N \approx_{ax} N_{n_0} : B$. Using Lemma 10.2.4, the term N_{n_0} is in intermediate neutral form. This makes the terms $M' = M_{n_0}$ and $N' = N_{n_0}$ fulfill the requirements of the lemma. □

Lemma 10.4.2. *Suppose that $\Delta \rhd M, N : A$ are two valid typing judgements, where M and N are intermediate neutral terms such that $Erase(M) = Erase(N)$. Then there exists two neutral terms M' and N' such that $Erase(M') = Erase(N')$ and such that $\Delta \rhd M \approx_{ax} M' : A$ and $\Delta \rhd N \approx_{ax} N' : A$.*

Proof. Consider two valid typing judgements $\Delta \rhd M, N : A$ such that M and N are intermediate neutral terms with $Erase(M) = Erase(N)$.

Using Lemma 10.3.30, it is possible to construct sequences of terms $(M_i)_i$ such that $M_0 = M$, and such that for all i, either M_i is a neutral value or $M_i \to_2^* M_{i+1}$ with $\mu(M_{i+1}) < \mu(M_i)$. From Lemma 10.3.23, there exists n_0 such that for all $n \geqslant n_0$, $\mu(M_n) = \mu(M_{n_0})$. Using the definition of the sequence, this means that for all $n \geqslant n_0$, $M_n = M_{n_0}$ is a neutral term. Using Lemma 10.3.3, one can build a sequence $(N_n)_n$ such that for all n, $Erase(N_n) = Erase(M_n)$ and such that $N_n \to_2^* N_{n+1}$. From Lemma 10.3.4, the typing judgements $\Delta \rhd M_{n_0}, N_{n_0} : A$ are valid, and $\Delta \rhd M \approx_{ax} M_{n_0} : A$ and $\Delta \rhd N \approx_{ax} N_{n_0} : A$.

The requested terms M' and N' can be respectively taken as M_{n_0} and N_{n_0}. □

We are now ready to prove Theorem 9.2.7.

Proof of Theorem 9.2.7. Suppose that M and N are two terms such that $Erase(M) = Erase(N)$ and such that $\Delta \rhd M, N : A$ are valid typing judgements.

Using Lemma 10.4.1, there exists two intermediate neutral terms M' and N' such that $\Delta \rhd M', N' : A$ and such that $\Delta \rhd M \approx_{ax} M' : A$ and $\Delta \rhd N \approx_{ax} N' : A$ and such that $Erase(M') = Erase(N')$. Using Lemma 10.4.2, there exists two neutral terms M'' and N'' such that $Erase(M'') = Erase(N'')$ and such that $\Delta \rhd M' \approx_{ax} M'' : A$ and $\Delta \rhd N' \approx_{ax} N'' : A$. Using Lemma 10.1.3, we have $\Delta \rhd M'' \approx_{ax} N'' : A$.

We obtain $\Delta \rhd M \approx_{ax} N : A$ using the transitivity of the axiomatic relation. □

Chapter 11

Categorical Semantics

In Chapter 8, we described the categorical structure believed to be required for interpreting the language described in Chapter 9. In this chapter, we show that one can indeed interpret the language in a linear category for duplication, and we prove a soundness and completeness result.

11.1 Denotational Semantics

The ambient category this section is concerned with is a linear category for duplication $\mathcal{C}$, with the notation of Definition 8.4.1.

Convention 11.1.1. For notation purposes, we assume the category to be strict monoidal. This can be safely assumed, thanks to Theorem 2.8.2.

11.1.1 Interpretation of the Type System

Definition 11.1.2. We define an *interpretation of the type system* to be a map Θ that assigns to each constant type α an object $\Theta(\alpha)$. Each type A is interpreted as an object of $\mathcal{C}$:

$$[\![\alpha]\!]_\Theta = \Theta(\alpha), \quad [\![\top]\!]_\Theta = \top,$$
$$[\![!A]\!]_\Theta = L[\![A]\!]_\Theta, \quad [\![A \otimes B]\!]_\Theta = [\![A]\!]_\Theta \otimes [\![B]\!]_\Theta, \quad [\![A \multimap B]\!]_\Theta = [\![A]\!]_\Theta \multimap [\![B]\!]_\Theta.$$

Consider the alphabet $\{\alpha \mid \alpha \text{ is a constant type}\}$, and the set $\mathbb{A}$ of types. If $G(A) = [\![A]\!]_\Theta$, we are in the context of Theorem 8.3.5, with the list of F_i's being $(\otimes, \multimap)$: any two maps $A \to B$ in $\mathcal{C}^G$ are equal.

Definition 11.1.3. Given a valid subtyping $A <: B$, there exists a unique arrow $[\![A]\!]_\Theta \to [\![B]\!]_\Theta$ in $\mathcal{C}^G$, as defined in Theorem 8.3.5. We extend the map Θ to interpret $A <: B$ as this unique arrow and we denote it by $I_{A,B}$.

11.1.2 Interpretation of the Language

Convention 11.1.4. Typing contexts were defined as sets. In the case we are interested in, we need them to be lists. We will therefore consider them to be lists in this section, the typing rules of Table 9.1 being physically unchanged. In order to still be able to reorder typing contexts, we add a set of new rules (X_s), one for each permutation s on $\{1, \ldots, n\}$:

$$\frac{x_1 : A_1, \ldots, x_n : A_n \rhd M : A}{x_{s(1)} : A_{s(1)}, \ldots, x_{s(n)} : A_{s(n)} \rhd M : A} \ (X_s)$$

Definition 11.1.5. We define the denotation of a typing context $\Delta = (x_1 : A_1, \ldots, x_n : A_n)$ by the following:

$$[\![\Delta]\!] = [\![A_1]\!] \otimes \cdots \otimes [\![A_n]\!].$$

Definition 11.1.6. We use the following shortcut definitions, where $A, B, A_1, \ldots, A_n$ are types and where Δ, Γ_1 and Γ_2 are typing contexts:

- Given a natural transformation $n : F \dot{\to} G$, we write n_A in place of $n_{[\![A]\!]}$. If $\Delta = \{x_1 : A_1 \ldots x_n : A_n\}$ we define $n_\Delta = n_{A_1} \otimes \cdots \otimes n_{A_n}$.
- For the coherence map $d^L_{A,B} : LA \otimes LB \to L(A \otimes B)$, if $\Delta = \{x_1 : A_1 \ldots x_n : A_n\}$, we define $d^L_{!\Delta}$ to be the map

$$L[\![A_1]\!] \otimes \cdots \otimes L[\![A_n]\!] \xrightarrow{id \otimes d^L_{[\![A_{n-1}]\!],[\![A_n]\!]}} L[\![A_1]\!] \otimes \cdots \otimes L[\![A_{n-1} \otimes A_n]\!] \xrightarrow{\cdots} L[\![A_1 \otimes \cdots \otimes A_n]\!].$$

- We define $Split_{!\Delta,\Gamma_1,\Gamma_2}$ to be the canonical map from the monoidal structure and using the map $\triangle_A : LA \to LA \otimes LA$ from Definition 2.7.3:

$$Split_{!\Delta,\Gamma_1,\Gamma_2} : [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \otimes [\![\Gamma_2]\!] \to [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \otimes [\![!\Delta]\!] \otimes [\![\Gamma_2]\!].$$

- Given $f : [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \to [\![A]\!]$ and $g : [\![!\Delta]\!] \otimes [\![\Gamma_2]\!] \to [\![B]\!]$, we define the map $f \otimes_{!\Delta} g : [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \otimes [\![\Gamma_2]\!] \to [\![A]\!] \otimes [\![B]\!]$ as follows:

$$[\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \otimes [\![\Gamma_2]\!] \xrightarrow{Split_{!\Delta,\Gamma_1,\Gamma_2}} [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \otimes [\![!\Delta]\!] \otimes [\![\Gamma_2]\!] \xrightarrow{f \otimes g} [\![A]\!] \otimes [\![B]\!]$$

Lemma 11.1.7. *Suppose that $f : [\![!\Delta, \Gamma_2]\!] \to [\![A]\!]$. Suppose that $!\Gamma_1$ is a context of fresh variables. Then*

$$Split_{!\Delta,!\Gamma_1,\Gamma_2}; (\diamondsuit_{!\Delta,!\Gamma_1} \otimes f) = (\sigma_{[\![!\Delta]\!],[\![!\Gamma_1]\!]} \otimes id_{\Gamma_2}); (\diamondsuit_{!\Gamma_1} \otimes f) : [\![!\Delta, !\Gamma_1, \Gamma_2]\!] \to [\![A]\!].$$

Proof. Using the equations of comonoids. □

Lemma 11.1.8. *Suppose that $f : [\![!\Delta, \Gamma_1]\!] \to [\![A]\!]$ and that $g : [\![!\Delta, \Gamma_2]\!] \to [\![B]\!]$. Suppose moreover that $!\Lambda$ is a context of fresh variables. Then*

$$(\diamondsuit_{!\Lambda} \otimes f) \otimes_{!\Lambda,!\Delta} (\diamondsuit_{!\Lambda} \otimes g) = \diamondsuit_{!\Lambda} \otimes (f \otimes_{!\Delta} g) : [\![!\Lambda, !\Delta, \Gamma_1, \Gamma_2]\!] \to [\![A]\!] \otimes [\![B]\!].$$

Proof. It is enough to show that the maps

$$[\![!\Lambda, !\Delta, \Gamma_1, \Gamma_2]\!] \xrightarrow{Split_{(!\Lambda,!\Delta),\Gamma_1,\Gamma_2}} [\![!\Lambda, !\Delta, \Gamma_1, !\Lambda, !\Delta, \Gamma_2]\!] \xrightarrow{\diamondsuit_{!\Lambda} \otimes f \otimes \diamondsuit_{!\Lambda} \otimes g} [\![A_1]\!] \otimes [\![A_2]\!]$$

and

$$[\![!\Lambda, !\Delta, \Gamma_1, \Gamma_2]\!] \xrightarrow{\diamondsuit_{!\Lambda} \otimes id} [\![!\Delta, \Gamma_1, \Gamma_2]\!] \xrightarrow{Split_{!\Delta,\Gamma_1,\Gamma_2}} [\![!\Delta, \Gamma_1, !\Delta, \Gamma_2]\!] \xrightarrow{f \otimes g} [\![A_1]\!] \otimes [\![A_2]\!]$$

are equal. This is done using the comonoid structure of $([\![!\Lambda]\!], \triangle_{!\Lambda}, \diamondsuit_{!\Lambda})$. □

Lemma 11.1.9. *Suppose that $f : [\![!\Lambda, !\Delta, \Gamma_1]\!] \to [\![A]\!]$ and that $g : [\![!\Delta, \Gamma_2]\!] \to [\![B]\!]$. Then,*

$$f \otimes_{!\Lambda,!\Delta} (\diamondsuit_{!\Lambda} \otimes g) = (id_{!\Lambda} \otimes Split_{!\Delta,\Gamma_1,\Gamma_2}); (f \otimes g) : [\![!\Lambda, !\Delta, \Gamma_1, \Gamma_2]\!] \to [\![A]\!] \otimes [\![B]\!].$$

Proof. Proof using the comonoid structure of $([\![!\Lambda]\!], \triangle_{!\Lambda}, \diamondsuit_{!\Lambda})$. □

Lemma 11.1.10. *Suppose that* $f : [\![\Gamma_1]\!] \to [\![A]\!]$ *and that* $g : [\![A]\!] \to [\![B]\!]$. *Then, if* $!\Delta$ *is a context of fresh variables,*

$$(\diamondsuit_\Delta \otimes f); g = \diamondsuit_\Delta \otimes (f; g) : [\![!\Delta, \Gamma_1]\!] \to [\![B]\!].$$

Proof. In our strict monoidal category, the map $(\diamondsuit_\Delta \otimes f); g$ is

$$\begin{aligned}(\diamondsuit_\Delta \otimes f); g &= (\diamondsuit_\Delta \otimes f); \lambda_A; g \\ &= (\diamondsuit_\Delta \otimes f); (id_\top \otimes g); \lambda_B && \text{by naturality of } \lambda, \\ &= (\diamondsuit_\Delta \otimes (f; g)); \lambda_B && \text{by bifunctoriality of } \otimes.\end{aligned}$$

And this map is precisely $\diamondsuit_\Delta \otimes (f; g)$. □

Lemma 11.1.11. *Suppose that* $f : [\![!\Delta, \Gamma_1]\!] \to [\![A]\!]$, $g : [\![!\Delta, \Gamma_2]\!] \to [\![B]\!]$, $h : [\![A]\!] \to [\![C]\!]$ *and* $k : [\![B]\!] \to [\![D]\!]$. *Then*

$$(f; h) \otimes_{!\Delta, \Gamma_1, \Gamma_2} (g; k) = (f \otimes_{!\Delta, \Gamma_1, \Gamma_2} g); (h \otimes k) : [\![!\Delta, \Gamma_1, \Gamma_2]\!] \to [\![C]\!] \otimes [\![D]\!].$$

Proof. The proof uses the bifunctoriality of $\otimes$. □

Lemma 11.1.12. *Suppose that* $f : [\![!\Delta, \Gamma_1]\!] \to [\![A]\!]$, $g : [\![!\Delta, \Gamma_2]\!] \to [\![B]\!]$. *Then*

$$(f \otimes_{!\Delta} g); \sigma_{A,B} = (id_{!\Delta} \otimes \sigma_{[\![\Gamma_1]\!], [\![\Gamma_2]\!]}); (g \otimes_{!\Delta} f) : [\![!\Delta, \Gamma_1, \Gamma_2]\!] \to [\![B \otimes A]\!].$$

Proof. The proof is done using the correspondence between the symmetry and the comonoid multiplication. □

Definition 11.1.13. The map Θ as in Definition 11.1.2 is said to be an *interpretation of the language* if moreover it assigns to each constant term $c : !A_c$ an arrow $\Theta(c) : \top \to [\![!A_c]\!]$ in $\mathcal{C}$.

Given a linear category for duplication $\mathcal{C}$, it is possible to interpret the typing derivation of a well-typed value as a map in $\mathcal{C}$ and the typing derivation of a valid computation as a map in the Kleisli category $\mathcal{C}_T$.

- If $x_1 : A_1, \ldots x_n : A_n \rhd V : B$ is a value with typing derivation π, its *value interpretation* $[\![\pi]\!]^v_\Theta$ is an arrow $[\![A_1]\!] \otimes \ldots \otimes [\![A_n]\!] \to_{\mathcal{C}} [\![B]\!]$;
- if $x_1 : A_1, \ldots x_n : A_n \rhd M : A$ is a term with typing derivation π, its *computational interpretation* $[\![\pi]\!]^c_\Theta$ is an arrow $[\![A_1]\!] \otimes \ldots \otimes [\![A_n]\!] \to_{\mathcal{C}} T([\![B]\!])$.

We define them inductively in Tables 11.1, 11.2 and 11.3. Moreover, considering Convention 11.1.4, if s is a permutation on $\{1, \ldots, n\}$:

$$\begin{aligned}&[\![x_{s(1)} : A_{s(1)}, \ldots, x_{s(n)} : A_{s(n)} \rhd M : A]\!]^c = \\ &\qquad \sigma_{[\![A_1, \ldots, A_n]\!], [\![A_{s(1)}, \ldots, A_{s(n)}]\!]}; [\![x_1 : A_1, \ldots, x_n : A_n \rhd M : A]\!]^c, \\ &[\![x_{s(1)} : A_{s(1)}, \ldots, x_{s(n)} : A_{s(n)} \rhd M : A]\!]^v = \\ &\qquad \sigma_{[\![A_1, \ldots, A]\!], [\![A_{s(1)}, \ldots, A_{s(n)}]\!]}; [\![x_1 : A_1, \ldots, x_n : A_n \rhd M : A]\!]^v.\end{aligned}$$

Convention 11.1.14. We say that two morphisms

$$f[\![x_1 : A_1, \ldots, x_n : A_n]\!] \to [\![A]\!], \qquad g : [\![x_{s(1)} : A_{s(1)}, \ldots, x_{s(n)} : A_{s(n)}]\!] \to [\![A]\!]$$

are equal whenever $f = \sigma_{[\![A_1, \ldots, A_n]\!], [\![A_{s(1)}, \ldots, A_{s(n)}]\!]}; g$.

$$
\begin{aligned}
[\![!\Delta, x : A \rhd x : B]\!]^v_\Theta &= [\![!\Delta]\!] \otimes [\![A]\!] \xrightarrow{\Diamond_\Delta \otimes I_{A,B}} [\![B]\!] \\
[\![!\Delta \rhd c : B]\!]^v_\Theta &= [\![!\Delta]\!] \xrightarrow{\Diamond_\Delta} \top \xrightarrow{\Theta(c)} [\![A_c]\!] \xrightarrow{I_{A_c,B}} [\![B]\!] \\
[\![!\Delta \rhd * : !^n\top]\!]^v_\Theta &= [\![!\Delta]\!] \xrightarrow{\Diamond_\Delta} \top \xrightarrow{d^L_\top} L\top \xrightarrow{I_{!\top,!^n\top}} L^n\top
\end{aligned}
$$

$$
\frac{[\![\Delta, x : A \rhd M : B]\!]^c_\Theta = [\![\Delta]\!] \otimes [\![A]\!] \xrightarrow{f} T([\![B]\!])}{[\![\Delta \rhd \lambda^0 x^A.M : A \multimap B]\!]^v_\Theta = [\![\Delta]\!] \xrightarrow{\Phi^{-1}(f)} [\![A]\!] \multimap [\![B]\!]}
$$

$$
\frac{[\![!\Delta, x : A \rhd M : B]\!]^c_\Theta = [\![!\Delta]\!] \otimes [\![A]\!] \xrightarrow{f} T([\![B]\!])}{\begin{aligned}[\![!\Delta \rhd \lambda^{n+1} x^A.M : !^{n+1}(A \multimap B)]\!]^v_\Theta = [\![!\Delta]\!] &\xrightarrow{\delta_{!\Delta}; d^L_{!\Delta}} L[\![!\Delta]\!] \\ &\xrightarrow{L(\Phi^{-1}f); I_{!(A\multimap B), !^{n+1}(A\multimap B)}} L^{n+1}([\![A]\!] \multimap [\![B]\!])\end{aligned}}
$$

Table 11.1: Interpretation of core values.

Lemma 11.1.15. *Suppose that*

$$f = [\![!\Delta, \Gamma_1 \rhd N : A]\!]^c, \quad g = [\![!\Delta, \Gamma_2, x : A \rhd P : B]\!]^c.$$

Then

$$[\![!\Delta, \Gamma_1, \Gamma_2 \rhd \mathit{let}\ x^A = N\ \mathit{in}\ P : B]\!]^c = (f \otimes_{!\Delta} id); \sigma; t; g^*.$$

Proof. The proof is done by computing $[\![!\Delta, \Gamma_1, \Gamma_2 \rhd (\lambda^0 x^A.P)N : B]\!]^c$. □

Lemma 11.1.16. *Given a valid typing judgement $\Delta \rhd M : B$ with no dummy variables, there exists $f_c : [\![\Delta]\!] \to T[\![B]\!]$ such that for any dummy context $!\Lambda$ and for any typing derivation π of $!\Lambda, \Delta \rhd M : B$,*

$$[\![\pi]\!]^c = [\![!\Lambda]\!] \otimes [\![\Delta]\!] \xrightarrow{\Diamond_{!\Lambda} \otimes f_c} T[\![B]\!].$$

Moreover, if $M = V$ is a value, there exists some $f_v : [\![\Delta]\!] \to [\![B]\!]$ such that for any dummy context $!\Lambda$ and for any typing derivation π of $!\Lambda, \Delta \rhd V : B$,

$$[\![\pi]\!]^v = [\![!\Lambda]\!] \otimes [\![\Delta]\!] \xrightarrow{\Diamond_{!\Lambda} \otimes f_v} [\![B]\!].$$

Proof. Consider a typing judgement $\Delta \rhd M : A$ where $|\Delta| = FV(M)$. We prove the lemma by structural induction on M.

Case $M \equiv \lambda^0 x^B.N$. In this case, $A = B \multimap C$, and if $\Delta \rhd \lambda^0 x^B.N : B \multimap C$ is valid, then so is $\Delta, x : B \rhd N : C$. By hypothesis, $|\Delta| = FV(M)$. Thus $|\Delta| = FV(N) \setminus \{x\}$.

Let $\dot{\Delta}$ be the subcontext of $(\Delta, x : B)$ consisting of the free variables of N.

The induction hypothesis states that there exists a function $f'_c : [\![\dot{\Delta}]\!] \to T[\![C]\!]$ such that for all dummy typing contexts $!\dot{\Lambda}$ and for all typing derivations π' the denotation $[\![\pi']\!]^c$ is $\Diamond_{!\dot{\Lambda}} \otimes f_c$.

There are two cases.

If $x \in FV(N)$. Then $|\dot{\Delta}| = |\Delta, x : B|$. We define the requested f_v and f_c respectively by $\Phi^{-1}(f'_c)$ and $\Phi^{-1}(f'_c); \eta_{B \multimap C}$.

If $x \notin FV(N)$. In this case, from Lemma 9.1.19 the type B is of the form $!\dot{B}$. The context $|\dot{\Delta}| = |\Delta|$, and x is a dummy variable. We define the requested f_v and f_c respectively by $\Phi^{-1}(\Diamond_{!\dot{B}} \otimes f'_c)$ and $\Phi^{-1}(\Diamond_{!\dot{B}} \otimes f'_c); \eta_{B \multimap C}$.

$$\frac{\begin{array}{ll}[\![!\Delta,\Gamma_1 \rhd V : A]\!]^v_\Theta & = [\![!\Delta]\!]\otimes[\![\Gamma_1]\!] \xrightarrow{f} [\![A]\!] \\ [\![!\Delta,\Gamma_2, x : A \rhd W : B]\!]^v_\Theta & = [\![!\Delta]\!]\otimes[\![\Gamma_2]\!]\otimes[\![A]\!] \xrightarrow{g} [\![B]\!]\end{array}}{\begin{array}{l}[\![!\Delta,\Gamma_1,\Gamma_2 \rhd \mathit{let}\ x = V\ \mathit{in}\ W : B]\!]^v_\Theta = \\ \quad [\![!\Delta]\!]\otimes[\![\Gamma_1]\!]\otimes[\![\Gamma_2]\!] \xrightarrow{(f\otimes_{!\Delta} id);\sigma} [\![!\Delta]\!]\otimes[\![\Gamma_2]\!]\otimes[\![A]\!] \xrightarrow{g} [\![B]\!]\end{array}}$$

$$\frac{\begin{array}{ll}[\![!\Delta,\Gamma_1, \rhd V : !^n(A_1 \otimes A_2)]\!]^v_\Theta & = [\![!\Delta]\!]\otimes[\![\Gamma_1]\!] \xrightarrow{f} L^n([\![A_1]\!] \otimes [\![A_2]\!]) \\ [\![!\Delta,\Gamma_2, x : !^n A_1, y : !^n A_2 \rhd W : C]\!]^v_\Theta & = [\![!\Delta]\!]\otimes[\![\Gamma_2]\!]\otimes L^n[\![A_1]\!] \otimes L^n[\![A_2]\!] \xrightarrow{g} [\![C]\!]\end{array}}{\begin{array}{l}[\![!\Delta,\Gamma_1,\Gamma_2 \rhd \mathit{let}\ \langle x, y\rangle^n = V\ \mathit{in}\ W : C]\!]^v_\Theta = \\ \quad [\![!\Delta]\!]\otimes[\![\Gamma_1]\!]\otimes[\![\Gamma_2]\!] \xrightarrow{f\otimes_{!\Delta} id} L^n([\![A_1]\!]\otimes[\![A_2]\!])\otimes[\![!\Delta]\!]\otimes[\![\Gamma_2]\!] \\ \quad \xrightarrow{\left(\left(d^{L^n}_{[\![A_1]\!],[\![A_2]\!]}\right)^{-1}\otimes id\right);\sigma} [\![!\Delta]\!]\otimes[\![\Gamma_2]\!]\otimes L^n[\![A_1]\!]\otimes L^n[\![A_2]\!] \xrightarrow{g} [\![C]\!]\end{array}}$$

$$\frac{\begin{array}{ll}[\![!\Delta,\Gamma_1 \rhd V : \top]\!]^v_\Theta & = [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \xrightarrow{f} \top \\ [\![!\Delta,\Gamma_2 \rhd W : C]\!]^v_\Theta & = [\![!\Delta]\!] \otimes [\![\Gamma_2]\!] \xrightarrow{g} [\![C]\!]\end{array}}{[\![!\Delta,\Gamma_1,\Gamma_2 \rhd \mathit{let}\ * = V\ \mathit{in}\ W : C]\!]^v_\Theta = [\![!\Delta]\!]\otimes[\![\Gamma_1]\!]\otimes[\![\Gamma_2]\!] \xrightarrow{f\otimes_{!\Delta} g} [\![C]\!]}$$

$$\frac{\begin{array}{ll}[\![!\Delta,\Gamma_1 \rhd V : !^n A]\!]^v_\Theta & = [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \xrightarrow{f} L^n[\![A]\!] \\ [\![!\Delta,\Gamma_2 \rhd W : !^n B]\!]^v_\Theta & = [\![!\Delta]\!] \otimes [\![\Gamma_2]\!] \xrightarrow{g} L^n[\![B]\!]\end{array}}{\begin{array}{l}[\![!\Delta,\Gamma_1,\Gamma_2 \rhd \langle V, W\rangle^n : !^n(A \otimes B)]\!]^v_\Theta = \\ \quad [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \otimes [\![\Gamma_2]\!] \xrightarrow{f\otimes_{!\Delta} g} L^n[\![A]\!] \otimes L^n[\![B]\!] \xrightarrow{d^{L^n}_{A,B}} L^n([\![A]\!] \otimes [\![B]\!])\end{array}}$$

Table 11.2: Interpretation of extended values.

Consider a dummy context $!\Lambda$ for M and a typing derivation π of $!\Lambda, \Delta \rhd M : A$. It starts with

$$\frac{\begin{array}{c}\pi' \\ \vdots \\ !\Lambda, \Delta, x : B \rhd N : C\end{array}}{!\Lambda, \Delta \rhd \lambda^0 x^B.N : B \multimap C}\ (\lambda_1)$$

for some derivation π'. By definition,

$$[\![\pi]\!]^v = \Phi^{-1}([\![\pi']\!]^c) \qquad [\![\pi]\!]^c = \Phi^{-1}([\![\pi']\!]^c); \eta_{B\multimap C}.$$

We aim at showing that $[\![\pi]\!]^v = \Diamond_{!\Lambda} \otimes f_v$ and that $[\![\pi]\!]^c = \Diamond_{!\Lambda} \otimes f_c$.
There are two cases.

If $x \in FV(N)$. By induction hypothesis, $[\![\pi']\!]^c = \Diamond_{!\Lambda} \otimes f'_c$. By naturality of Φ,

$$\begin{array}{ll}[\![\pi]\!]^c = \Phi^{-1}(\Diamond_{!\Lambda} \otimes f'_c); \eta_{B\multimap C} & = \Diamond_{!\Lambda} \otimes \Phi^{-1}(f'_c); \eta_{B\multimap C}, \\ [\![\pi]\!]^v = \Phi^{-1}(\Diamond_{!\Lambda} \otimes f'_c) & = \Diamond_{!\Lambda} \otimes \Phi^{-1}(f'_c).\end{array}$$

If $x \notin FV(N)$. By induction hypothesis, $[\![\pi']\!]^c = \Diamond_{!\Lambda}\otimes\Diamond_{!\dot{B}}\otimes f'_c$. By naturality of Φ,

$$\begin{array}{ll}[\![\pi]\!]^c = \Phi^{-1}(\Diamond_{!\Lambda} \otimes \Diamond_{!\dot{B}} \otimes f'_c); \eta_{B\multimap C} & = \Diamond_{!\Lambda} \otimes \Phi^{-1}(\Diamond_{!\dot{B}} \otimes f'_c); \eta_{B\multimap C}, \\ [\![\pi]\!]^v = \Phi^{-1}(\Diamond_{!\Lambda} \otimes \Diamond_{!\dot{B}} \otimes f'_c) & = \Diamond_{!\Lambda} \otimes \Phi^{-1}(\Diamond_{!\dot{B}} \otimes f'_c).\end{array}$$

First, if U is a core value, $[\![\Delta \rhd U : A]\!]^c_\Theta = [\![\Delta \rhd U : A]\!]^v_\Theta ; \eta_A$.

$$\frac{\begin{array}{ll}[\![!\Delta, \Gamma_1 \rhd M : A \multimap B]\!]^c_\Theta & = [\![!\Delta, \Gamma_1]\!] \xrightarrow{f} T([\![A \multimap B]\!]) \\ [\![!\Delta, \Gamma_2 \rhd N : A]\!]^c_\Theta & = [\![!\Delta, \Gamma_2]\!] \xrightarrow{g} T([\![A]\!])\end{array}}{\begin{array}{l}[\![!\Delta, \Gamma_1, \Gamma_2 \rhd MN : B]\!]^c_\Theta = [\![!\Delta, \Gamma_1, \Gamma_2]\!] \xrightarrow{f \otimes_{!\Delta} g} T[\![A \multimap B]\!] \otimes T[\![A]\!] \\ \xrightarrow{\Psi_1} T[\![(A \multimap B) \otimes A]\!] \xrightarrow{\varepsilon^*_{A,B}} T[\![B]\!]\end{array}}$$

$$\frac{\begin{array}{ll}[\![!\Delta, \Gamma_1 \rhd M : !^n(A_1 \otimes A_2)]\!]^c_\Theta & = [\![!\Delta, \Gamma_1]\!] \xrightarrow{f} TL^n[\![A_1 \otimes A_2]\!] \\ [\![!\Delta, \Gamma_2, x : !^n A_1, y : !^n A_2 \rhd N : C]\!]^v_\Theta & = [\![!\Delta, \Gamma_2]\!] \otimes L^n[\![A_1]\!] \otimes L^n[\![A_2]\!] \xrightarrow{g} T[\![C]\!]\end{array}}{\begin{array}{l}[\![!\Delta, \Gamma_1, \Gamma_2 \rhd \mathit{let}\ \langle x, y\rangle^n = M\ \mathit{in}\ N : !^n C]\!]^c_\Theta = \\ \quad [\![!\Delta, \Gamma_1, \Gamma_2]\!] \xrightarrow{f \otimes_{!\Delta} id} TL^n[\![A_1 \otimes A_2]\!] \otimes [\![!\Delta, \Gamma_1]\!] \\ \quad \xrightarrow{\left(T(d^{L^n})^{-1} \otimes id\right); \sigma; t} T([\![!\Delta, \Gamma_1]\!] \otimes L^n[\![A_1]\!] \otimes L^n[\![A_2]\!]) \xrightarrow{g^*} T[\![C]\!]\end{array}}$$

$$\frac{\begin{array}{ll}[\![!\Delta, \Gamma_1 \rhd M : \top]\!]^c_\Theta & = [\![!\Delta, \Gamma_1]\!] \xrightarrow{f} T(\top) \\ [\![!\Delta, \Gamma_2 \rhd N : C]\!]^c_\Theta & = [\![!\Delta, \Gamma_2]\!] \xrightarrow{g} T[\![C]\!]\end{array}}{\begin{array}{l}[\![!\Delta, \Gamma_1, \Gamma_2 \rhd \mathit{let}\ * = M\ \mathit{in}\ N : C]\!]^c_\Theta = \\ \quad [\![!\Delta, \Gamma_1, \Gamma_2]\!] \xrightarrow{f \otimes_{!\Delta} g} T(\top) \otimes T[\![C]\!] \xrightarrow{\Psi_1} T[\![C]\!]\end{array}}$$

$$\frac{\begin{array}{ll}[\![!\Delta, \Gamma_1 \rhd M : !^n A]\!]^c_\Theta & = [\![!\Delta, \Gamma_1]\!] \xrightarrow{f} TL^n[\![A]\!] \\ [\![!\Delta, \Gamma_2 \rhd N : !^n B]\!]^c_\Theta & = [\![!\Delta, \Gamma_2]\!] \xrightarrow{g} TL^n[\![B]\!]\end{array}}{\begin{array}{l}[\![!\Delta, \Gamma_1, \Gamma_2 \rhd \langle M, N\rangle^n : !^n(A \otimes B)]\!]^c_\Theta = \\ \quad [\![!\Delta, \Gamma_1, \Gamma_2]\!] \xrightarrow{f \otimes_{!\Delta} g} TL^n[\![A]\!] \otimes TL^n[\![B]\!] \xrightarrow{\Psi_1; Td^{L^n}_{A,B}} TL^n[\![A \otimes B]\!]\end{array}}$$

Table 11.3: Interpretation of computations.

Case $M \equiv \lambda^{n+1} x^B.N$. In this case, $A = !^{n+1}(B \multimap C)$, and if $\Delta \rhd \lambda^{n+1} x^B.N : B \multimap C$ is valid, then so is $\Delta, x : B \rhd N : C$. By hypothesis, $|\Delta| = FV(M)$. Thus $|\Delta| = FV(N) \setminus \{x\}$.

Let $\dot{\Delta}$ be the subcontext of $(x : B, \Delta)$ consisting of the free variables of N.

The induction hypothesis states that there exists a function $f'_c : [\![\dot{\Delta}]\!] \to T[\![C]\!]$ such that for all dummy typing contexts $!\dot{\Lambda}$ and for all typing derivations π' the denotation $[\![\pi']\!]^c$ is $\Diamond_{!\dot{\Lambda}} \otimes f_c$.

Note that Δ is of the form $!\ddot{\Delta}$. There are two cases.

If $x \in FV(N)$. Then $|\dot{\Delta}| = |x : B, \Delta|$. We define the requested f_v and f_c respectively by

$$f_v = \delta_\Delta ; d^L_\Delta ; L\Phi^{-1}(f'_c) ; I_{!(B \multimap C), !^{n+1}(B \multimap C)},$$
$$f_c = \delta_\Delta ; d^L_\Delta ; L\Phi^{-1}(f'_c) ; I_{!(B \multimap C), !^{n+1}(B \multimap C)} ; \eta_{!^{n+1}(B \multimap C)}.$$

If $x \notin FV(N)$. In this case, from Lemma 9.1.19 the type B is of the form $!\dot{B}$. The context $|\dot{\Delta}| = |\Delta|$, and x is a dummy variable. We define the requested f_v and f_c respectively by

$$f_v = \delta_\Delta ; d^L_\Delta ; L\Phi^{-1}(\Diamond_{!\dot{B}} \otimes f'_c) ; I_{!(B \multimap C), !^{n+1}(B \multimap C)},$$
$$f_c = \delta_\Delta ; d^L_\Delta ; L\Phi^{-1}(\Diamond_{!\dot{B}} \otimes f'_c) ; I_{!(B \multimap C), !^{n+1}(B \multimap C)} ; \eta_{!^{n+1}(B \multimap C)}.$$

Consider a dummy context $!\Lambda$ for M and a typing derivation π of $!\Lambda, \Delta \rhd M : A$. It starts with

$$\dfrac{\begin{array}{c}\pi' \\ \vdots \\ !\Lambda, \Delta, x : B \rhd N : C\end{array}}{!\Lambda, \Delta \rhd \lambda^{n+1} x^B.N : B \multimap C}\ (\lambda_2)$$

for some derivation π'. By definition,

$$\begin{aligned}
[\![\pi]\!]^v &= \delta_{!\Lambda,\Delta}; d^L_{!\Lambda,\Delta}; L\Phi^{-1}([\![\pi']\!]^c); I_{!(B\multimap C),!^{n+1}(B\multimap C)},\\
[\![\pi]\!]^c &= \delta_{!\Lambda,\Delta}; d^L_{!\Lambda,\Delta}; L\Phi^{-1}([\![\pi']\!]^c); I_{!(B\multimap C),!^{n+1}(B\multimap C)}; \eta_{!^{n+1}(B\multimap C)}.
\end{aligned}$$

We aim at showing that $[\![\pi]\!]^v = \Diamond_{!\Lambda} \otimes f_v$ and that $[\![\pi]\!]^c = \Diamond_{!\Lambda} \otimes f_c$.

One can show that for all $g : [\![\Delta]\!] \to [\![D]\!]$,

$$\delta_{!\Lambda,\Delta}; d^L_{!\Lambda,\Delta}; L(\Diamond_{!\Lambda} \otimes id_\Delta); L(id_\top \otimes g) = \Diamond_{!\Lambda} \otimes (\delta_\Delta; d^L_\Delta; Lid_\Delta; Lg) \tag{11.1.1}$$

by using the monoidality of d^L and δ, and by using the fact that the category is strict. There are two cases.

If $x \in FV(N)$. By induction hypothesis, $[\![\pi']\!]^c = \Diamond_{!\Lambda} \otimes f'_c$. By naturality of Φ, $\Phi^{-1}(\Diamond_{!\Lambda} \otimes f'_c) = \Diamond_{!\Lambda} \otimes \Phi^{-1}(f'_c)$. One can then conclude, by using the functoriality of L and $\otimes$ and replacing g with $\Phi^{-1}(f'_c)$ in Equation 11.1.1.

If $x \notin FV(N)$. By induction hypothesis, $[\![\pi']\!]^c = \Diamond_{!\Lambda} \otimes \Diamond_{!\dot{B}} \otimes f'_c$.
By naturality of Φ, $\Phi^{-1}(\Diamond_{!\Lambda} \otimes \Diamond_{!\dot{B}} \otimes f'_c) = \Diamond_{!\Lambda} \otimes \Phi^{-1}(\Diamond_{!\dot{B}} \otimes f'_c)$ One can then conclude, by using the functoriality of L and $\otimes$ and replacing g with $\Phi^{-1}(\Diamond_{!\dot{B}} \otimes f'_c)$ in Equation 11.1.1.

Case $M \equiv NP$. In this case, one can split Δ into $(!\dot{\Delta}, \Gamma_1, \Gamma_2)$, where Γ_1 consists of all the free variables in N but not in P, Γ_2 of all the free variables in P but not in N, and $!\dot{\Delta}$ of all the free variables both in N and in P. Note that these are the only possibilities, since Δ does not contain any dummy variables.

Using Lemma 9.1.19, If $\Delta \rhd NP : A$, then there exists some B such that $!\dot{\Delta}, \Gamma_1 \rhd N : B \multimap A$ and $!\dot{\Delta}, \Gamma_2 \rhd P : B$.

By induction hypothesis, there exists $f'_c : [\![!\dot{\Delta}]\!] \otimes [\![\Gamma_1]\!] \to T[\![B \multimap A]\!]$ such that for all dummy contexts $!\Lambda$ for N and for all typing derivations π_1 for $!\Lambda, !\dot{\Delta}, \Gamma_1 \rhd N : B \multimap A$ the denotation $[\![\pi_1]\!]^c = \Diamond_{!\Lambda} \otimes f'_c$.

Similarly, by induction hypothesis, there exists $g'_c : [\![!\dot{\Delta}]\!] \otimes [\![\Gamma_2]\!] \to T[\![B]\!]$ such that for all dummy contexts $!\Lambda$ for P and for all typing derivations π_2 for $!\Lambda, !\dot{\Delta}, \Gamma_2 \rhd P : B$ the denotation $[\![\pi_2]\!]^c = \Diamond_{!\Lambda} \otimes g'_c$.

Let f_c be the map

$$[\![!\dot{\Delta}]\!] \otimes [\![\Gamma_1]\!] \otimes [\![\Gamma_2]\!] \xrightarrow{f'_c \otimes_{!\dot{\Delta}} g'_c} T[\![B \multimap A]\!] \otimes T[\![B]\!] \xrightarrow{\Psi_1} T([\![B \multimap A]\!] \otimes [\![B]\!]) \xrightarrow{\varepsilon^*_{A,B}} T[\![A]\!].$$

Consider a dummy context $!\Lambda$ for NP and a valid derivation π for $!\Lambda, \Delta \rhd NP : A$. This derivation starts with

$$\dfrac{\begin{array}{c}\pi_1 \\ \vdots \\ !\ddot{\Delta}, \ddot{\Gamma}_1 \rhd N : B' \multimap A\end{array} \quad \begin{array}{c}\pi_2 \\ \vdots \\ !\ddot{\Delta}, \ddot{\Gamma}_2 \rhd P : B'\end{array}}{!\Lambda, \Delta \rhd NP : A.}\ (app)$$

The context $(!\Lambda, \Delta)$ is equal to $(!\ddot{\Delta}, \ddot{\Gamma}_1, \ddot{\Gamma}_2)$. From Lemma 9.1.12, $B' = B$. From Lemma 9.1.11, we have $|!\dot{\Delta}| \subseteq |!\ddot{\Delta}|$, $|\Gamma_1| \subseteq |!\ddot{\Delta}, \ddot{\Gamma}_1|$ and $|\Gamma_2| \subseteq |!\ddot{\Delta}, \ddot{\Gamma}_2|$. Let us split $!\ddot{\Delta}$, $\ddot{\Gamma}_1$ and $\ddot{\Gamma}_2$ into

$$\begin{aligned}
!\ddot{\Delta} &= (!\ddot{\Delta}^{dnNP}, !\ddot{\Delta}^{dNPn}, !\ddot{\Delta}^{dNnP}, !\ddot{\Delta}^{dPnN}),\\
\ddot{\Gamma}_1 &= (\ddot{\Gamma}_1^{dNPn},\ \ddot{\Gamma}_1^{dPnN}),\\
\ddot{\Gamma}_2 &= (\ddot{\Gamma}_2^{dNPn},\ \ddot{\Gamma}_2^{dNnP}),
\end{aligned}$$

where for each context of the form $X^{dY_1 \ldots Y_k n Z_1 \ldots Z_l}$, if $x \in |X^{dY_1 \ldots Y_k n Z_1 \ldots Z_l}|$, then $x \in FV(Z_1) \cap \cdots \cap FV(Z_l)$ but $x \notin FV(Y_1) \cup \cdots \cup FV(Y_k)$. We did not include the four contexts $\ddot{\Gamma}_1^{dnNP}$, $\ddot{\Gamma}_2^{dnNP}$, $\ddot{\Gamma}_1^{dNnP}$ and $\ddot{\Gamma}_2^{dPnN}$, which are empty by definition: otherwise the context $(\ddot{\Delta}, \ddot{\Gamma}_1, \ddot{\Gamma}_2)$ would have duplicates. We can rewrite the following contexts:

$$\begin{aligned}
!\Lambda &= (!\ddot{\Delta}^{dNPn}, \ddot{\Gamma}_1^{dNPn}, \ddot{\Gamma}_2^{dNPn}), & \Gamma_1 &= (!\ddot{\Delta}^{dPnN}, \ddot{\Gamma}_1^{dPnN}),\\
!\dot{\Delta} &= \ !\ddot{\Delta}^{dnNP}, & \Gamma_2 &= (!\ddot{\Delta}^{dNnP}, \ddot{\Gamma}_2^{dNnP}).
\end{aligned}$$

Note that $\ddot{\Gamma}_1^{dNPn}$, and $\ddot{\Gamma}_2^{dNPn}$, are duplicable.

By definition, $[\![\pi]\!]^c = ([\![\pi_1]\!]^c \otimes_{!\ddot{\Delta}} [\![\pi_2]\!]^c)\,; \Psi_1; \varepsilon^*_{A,B}$. We want to show that $[\![\pi]\!]^c = \Diamond_{!\Lambda} \otimes f_c$. We have from the induction hypothesis that if $f'_\Diamond$ is the morphism $\Diamond_{!\ddot{\Delta}^{dNPn}, !\ddot{\Delta}^{dNnP}, \ddot{\Gamma}_1^{dNPn}}$ and if $g'_\Diamond$ is the morphism $\Diamond_{!\ddot{\Delta}^{dNPn}, !\ddot{\Delta}^{dPnN}, \ddot{\Gamma}_2^{dNPn}}$, then $[\![\pi_1]\!]^c$ and $[\![\pi_2]\!]^c$ are respectively $f'_\Diamond \otimes f'_c$ and $g'_\Diamond \otimes g'_c$, and

$$\begin{aligned}
f'_\Diamond &: [\![!\ddot{\Delta}^{dNPn}, !\ddot{\Delta}^{dNnP}, \ddot{\Gamma}_1^{dNPn}]\!] \to \top\\
g'_\Diamond &: [\![!\ddot{\Delta}^{dNPn}, !\ddot{\Delta}^{dPnN}, \ddot{\Gamma}_2^{dNPn}]\!] \to \top\\
f'_c &: [\![!\ddot{\Delta}^{dnNP}, !\ddot{\Delta}^{dPnN}, \ddot{\Gamma}_1^{dPnN}]\!] \to T[\![B \multimap A]\!],\\
g'_c &: [\![!\ddot{\Delta}^{dnNP}, !\ddot{\Delta}^{dNnP}, \ddot{\Gamma}_2^{dNnP}]\!] \to T[\![B]\!].
\end{aligned}$$

Now, since from Lemma 11.1.8 and Lemma 11.1.9,

$$\begin{aligned}
&(\Diamond_{!\ddot{\Delta}^{dNPn}} \otimes \Diamond_{!\ddot{\Delta}^{dNnP}} \otimes \Diamond_{\ddot{\Gamma}_1^{dNPn}} \otimes f'_c)\\
&\qquad \otimes_{!\ddot{\Delta}^{dNPn}, !\ddot{\Delta}^{dnNP}, !\ddot{\Delta}^{dPnN}, !\ddot{\Delta}^{dNnP}} (\Diamond_{!\ddot{\Delta}^{dNPn}} \otimes \Diamond_{!\ddot{\Delta}^{dPnN}} \otimes \Diamond_{\ddot{\Gamma}_2^{dNPn}} \otimes g'_c)\\
&\qquad\qquad = \Diamond_{!\ddot{\Delta}^{dNPn}} \otimes \Diamond_{\ddot{\Gamma}_1^{dNPn}} \otimes \Diamond_{\ddot{\Gamma}_2^{dNPn}} \otimes (f'_c \otimes_{!\ddot{\Delta}^{dnNP}} g'_c),
\end{aligned}$$

the denotation $[\![\pi]\!]^c$ is then equal to

$$\begin{aligned}
[\![\pi]\!]^c &= \Big(\Diamond_{!\ddot{\Delta}^{dNPn}} \otimes \Diamond_{\ddot{\Gamma}_1^{dNPn}} \otimes \Diamond_{\ddot{\Gamma}_2^{dNPn}} \otimes (f'_c \otimes_{!\dot{\Delta}} g'_c)\Big); \Psi_1; \varepsilon^*_{A,B}\\
&= \Big(\Diamond_{!\Lambda} \otimes (f'_c \otimes_{!\dot{\Delta}} g'_c)\Big); \Psi_1; \varepsilon^*_{A,B}.
\end{aligned}$$

One concludes using Lemma 11.1.10.

The remaining cases are handled similarly. □

Theorem 11.1.17. *Given a valid typing judgement $\Delta \rhd M : B$ with two typing derivations π and π', for any interpretation Θ we have $[\![\pi]\!]^c_\Theta = [\![\pi']\!]^c_\Theta$ (and $[\![\pi]\!]^v_\Theta = [\![\pi']\!]^v_\Theta$ if M is a value).*

Proof. We consider denotations with respect to Θ. The context Δ splits into $(!\Lambda, \dot{\Delta})$, where $!\Lambda$ is a dummy context and where $|\dot{\Delta}| = FV(M)$.

From Lemma 9.1.19, $\dot{\Delta} \rhd M : A$ is valid. We are in the context of Lemma 11.1.16: there exists a function $f_c : [\![\dot{\Delta}]\!] \to T[\![A]\!]$ such that $[\![\pi]\!]^c$ and $[\![\pi']\!]^c$ are both equal to $\Diamond_{!\Lambda} \otimes f_c$. If M is a value, then there exists a function $f_v : [\![\dot{\Delta}]\!] \to [\![A]\!]$ such that $[\![\pi]\!]^v$ and $[\![\pi']\!]^v$ are both equal to $\Diamond_{!\Lambda} \otimes f_v$. □

Definition 11.1.18. Given a interpretation Θ of the language in a category $\mathcal{C}$, we define the denotation of a valid typing judgement $\Delta \rhd M : A$ with typing derivation π to be $[\![\Delta \rhd M : A]\!]^c_\Theta = [\![\pi]\!]^c_\Theta$ and $[\![\Delta \rhd M : A]\!]^v_\Theta = [\![\pi]\!]^v_\Theta$ if M is a value.

Remark 11.1.19. From Theorem 11.1.17, Definition 11.1.18 is valid.

11.2 Soundness of the Denotation

The axiomatic equivalence and the categorical semantics are two faces of the same coin. Indeed, as we will prove in this section, two terms in the same axiomatic equivalence class have the same denotation. A corollary is that the indexation of terms does not influence the denotation. This proves semantically the fact that it is safe to work with untyped terms. An alternate justification of this fact is of course the operational semantics, which was given in Section 6.

Lemma 11.2.1. *Suppose that $\Delta' <: \Delta$ and that $\Delta \rhd M : A$. Then*

$$[\![\Delta' \rhd M : A]\!]^c = I_{\Delta',\Delta}; [\![\Delta \rhd M : A']\!]^c.$$

If $M = V$ is a value, we have moreover

$$[\![\Delta' \rhd V : A]\!]^v = I_{\Delta',\Delta}; [\![\Delta \rhd V : A']\!]^v.$$

Proof. Proof by induction on the size of M. □

Lemma 11.2.2. *Suppose that $\Delta \rhd M : A$ and that $A <: A'$. Then*

$$[\![\Delta \rhd \{M <: A'\} : A']\!]^c = [\![\Delta \rhd M : A]\!]^c; T(I_{A,A'}).$$

If $M = V$ is a value, from Lemma 9.1.27, $\{V <: A'\}$ is a value. Then

$$[\![\Delta \rhd \{V <: A'\} : A']\!]^v = [\![\Delta \rhd V : A]\!]^v; I_{A,A'}.$$

Proof. Proof by induction on the size of M. □

Lemma 11.2.3. *Suppose that $\Delta \rhd M : A$ is a valid typing judgement and that $!\Lambda$ is a context of dummy variables for M. Then*

$$[\![!\Lambda, \Delta \rhd M : A]\!]^c = \Diamond_\Lambda \otimes [\![\Delta \rhd M : A]\!]^c.$$

If $M = V$ is a value,

$$[\![!\Lambda, \Delta \rhd V : A]\!]^v = \Diamond_\Lambda \otimes [\![\Delta \rhd V : A]\!]^v.$$

Proof. Proof by induction on the size of M. □

Lemma 11.2.4 (Substitution)**.** *Given two valid typing judgements $!\Delta, \Gamma_1, x : A \rhd M : B$ and $!\Delta, \Gamma_2 \rhd V : A$, the typing judgement $!\Delta, \Gamma_1, \Gamma_2 \rhd M[V/x] : B$ is valid. Let h be the denotation $[\![!\Delta, \Gamma_1, \Gamma_2 \rhd M[V/x] : B]\!]^c$ and h' be $[\![!\Delta, \Gamma_1, \Gamma_2 \rhd W[V/x] : B]\!]^v$, in the case where $M = W$ is a value. Then they are defined by*

$$\begin{tikzcd}
[\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \otimes [\![\Gamma_2]\!] \arrow[r, dotted, "h"] \arrow[d, "Split_{!\Delta,\Gamma_1,\Gamma_2}"] & T([\![B]\!]) \\
[\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \otimes [\![!\Delta]\!] \otimes [\![\Gamma_2]\!] \arrow[r, "id \otimes [\![!\Delta,\Gamma_2 \rhd V:A]\!]^v"] & [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \otimes [\![A]\!], \arrow[u, "[\![!\Delta,\Gamma_1,x:A \rhd M:B]\!]^c"']
\end{tikzcd} \tag{11.2.1}$$

$$\begin{tikzcd}
[\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \otimes [\![\Gamma_2]\!] \arrow[r, dotted, "h'"] \arrow[d, "Split_{!\Delta,\Gamma_1,\Gamma_2}"] & [\![B]\!] \\
[\![!\Delta]\!]\otimes[\![\Gamma_1]\!]\otimes[\![!\Delta]\!]\otimes[\![\Gamma_2]\!] \arrow[r, "id\otimes[\![!\Delta,\Gamma_2\rhd V:A]\!]^v"] & [\![!\Delta]\!]\otimes[\![\Gamma_1]\!]\otimes[\![A]\!]. \arrow[u, "{[\![!\Delta,\Gamma_1,x:A\rhd W:B]\!]^v}"']
\end{tikzcd} \tag{11.2.2}$$

Proof. Let v be the map $[\![!\Delta, \Gamma_2 \rhd V : A]\!]^v$. We do the proof by induction on the size of M.

Cases where M is a core value. Case $M \equiv x^B$. The term M is a core value: we have to show Equation (11.2.1) and Equation (11.2.2). We show the former, the latter being exactly similar.

The typing derivation $!\Delta, \Gamma_1, x : A \rhd x^B : B$ comes from typing rule (ax_1). This means that Γ_1 is of the form $!\Gamma'_1$, and that $A <: B$. Using Lemma 11.2.2, the map h is

$$h = [\![!\Delta, !\Gamma'_1, \Gamma_2 \rhd \{V <: B\} : B]\!]^c = [\![!\Delta, !\Gamma'_1, \Gamma_2 \rhd V : A]\!]^c; T(I_{A,B}).$$

Since V is a core value and using Lemma 11.2.3 and 11.1.7,

$$\begin{aligned} h &= [\![!\Delta, !\Gamma'_1, \Gamma_2 \rhd V : A]\!]^v; \eta_A; T(I_{A,B}) \\ &= Split_{!\Delta,!\Gamma'_1,\Gamma_2}; (\Diamond_{\Delta,\Gamma'_1} \otimes [\![!\Delta, \Gamma_2 \rhd V : A]\!]^v); \eta_A; T(I_{A,B}). \end{aligned} \tag{11.2.3}$$

By definition

$$[\![!\Delta, !\Gamma'_1, x : A \rhd x^B : B]\!]^c = \Diamond_{\Delta,\Gamma'_1} \otimes I_{A,B}; \eta_B.$$

Thus

$$\begin{aligned} & Split_{!\Delta,!\Gamma'_1,\Gamma_2}; (id \otimes [\![!\Delta, \Gamma_2 \rhd V : A]\!]^v); [\![!\Delta, !\Gamma'_1, x : A \rhd x^B : B]\!]^c \\ = {} & Split_{!\Delta,!\Gamma'_1,\Gamma_2}; (id \otimes [\![!\Delta, \Gamma_2 \rhd V : A]\!]^v); (\Diamond_{\Delta,\Gamma'_1} \otimes I_{A,B}; \eta_B) \\ = {} & Split_{!\Delta,!\Gamma'_1,\Gamma_2}; (\Diamond_{\Delta,\Gamma'_1} \otimes ([\![!\Delta, \Gamma_2 \rhd V : A]\!]^v; I_{A,B}; \eta_B) \qquad \text{(bifunc. of } \otimes) \end{aligned}$$

which is equal to Formula (11.2.3), the naturality of η.

Case $M \equiv *^n$, c^B and y^B, where $y \neq x$. Again in these cases, the term M is a core value: we have to show Equation (11.2.1) and Equation (11.2.2). We show the former, the latter being exactly similar.

The typing judgement $!\Delta, \Gamma_1, x : A \rhd M : B$ comes from respectively rules $(\top.I)$, (ax_2) and (ax_1). In all of these cases, Γ_1 is of the form $!\Gamma'_1$ and A of the form $!A'$. From Lemma 9.1.15, the context Γ_2 is of the form $!\Gamma'_2$. Since the term $M[V/x]$ is equal to M is all three cases, the maps h and h' are of the form $\Diamond_{!\Delta,!\Gamma'_1,!\Gamma'_2}; g$, for some map g. The map $[\![!\Delta, !\Gamma'_1, x : !A' \rhd M : B]\!]$ is of the form $\Diamond_{\Delta,\Gamma'_1,A'}; g$ for the same map g. The bottom part of Diagram (11.2.1) is

$$Split_{!\Delta,!\Gamma'_1,!\Gamma'_2}; (id \otimes [\![!\Delta, !\Gamma'_2 \rhd V : A]\!]^v); \Diamond_{\Delta,\Gamma'_1,A'}; g = \Diamond_{\Delta,\Gamma'_1,\Gamma'_2}; g$$

using Equation (2.7.11). This is precisely the map $[\![!\Delta, !\Gamma'_1, x : !A' \rhd M : B]\!]$, ending the proof for this case.

Case $M \equiv \lambda^0 y^C.N$. In this case, $B = C \multimap D$ for some type D, and the typing judgement $!\Delta, \Gamma_1, x : A \rhd M : B$ comes from typing rule (λ_1). That is, $!\Delta, \Gamma_1, y : C, x : A \rhd N : C$ is valid.

We have $y \neq x$: this means that $M[V/x] = \lambda^0 y^C.N[V/x]$. By induction hypothesis, the morphism $k = [\![!\Delta, \Gamma_1, y : C, \Gamma_2 \rhd N[V/x] : D]\!]^c$ satisfies the following commutative diagram:

$$\begin{tikzcd}
[\![!\Delta, \Gamma_1, y : C, \Gamma_2]\!] \arrow[r, dotted, "k"] \arrow[d, "Split_{!\Delta,(\Gamma_1,y:C),\Gamma_2}"] & T([\![D]\!]) \\
[\![!\Delta, \Gamma_1, y : C]\!]\otimes[\![!\Delta, \Gamma_2]\!] \arrow[r, "id\otimes[\![!\Delta,\Gamma_2\rhd V:A]\!]^v"] & [\![!\Delta, \Gamma_1, y : C]\!]\otimes[\![A]\!], \arrow[u, "{[\![!\Delta,\Gamma_1,y:C,x:A\rhd N:D]\!]^c}"']
\end{tikzcd}$$

Now, by definition,

$$\begin{aligned}
[\![!\Delta, \Gamma_1, \Gamma_2 \rhd M[V/x] : C \multimap D]\!]^v &= \Phi^{-1}(k), \\
[\![!\Delta, \Gamma_1, \Gamma_2 \rhd M[V/x] : C \multimap D]\!]^c &= \Phi^{-1}(k); \mu_{C \multimap D}, \\
[\![!\Delta, \Gamma_1 \rhd M : C \multimap D]\!]^v &= \Phi^{-1}[\![!\Delta, \Gamma_1, y : C \rhd N : D]\!]^c, \\
[\![!\Delta, \Gamma_1 \rhd M : C \multimap D]\!]^c &= (\Phi^{-1}[\![!\Delta, \Gamma_1, y : C \rhd N : D]\!]^c); \mu_{C \multimap D}.
\end{aligned}$$

Equations (11.2.1) and (11.2.2) are derived using the naturality of Φ.

Case $M \equiv \lambda^{n+1} x^C.N$. This case is similar to the previous one. The only difference is in that the equations under consideration are encapsulated into

$$f \mapsto \delta_{!\Delta}; d^L_{!\Delta}; L(f); I_{!(A \multimap B), !^{n+1}(A \multimap B)}.$$

Cases $M \equiv (\mathit{let}\ \boxdot^C = N\ \mathit{in}\ P)$. In all these cases, the typing judgement $!\Delta, \Gamma_1, x : A \rhd M : B$ comes from a typing rule (X) with hypotheses

$$!\Delta', \Gamma_{11} \rhd N : C, \qquad !\Delta', \Gamma_{12}, \Lambda \rhd P : B,$$

for some context Λ, where $(!\Delta', \Gamma_{11}, \Gamma_{12}) = (!\Delta, \Gamma_1, x : A)$. If we call f and g the following maps:

$$f = [\![!\Delta', \Gamma_{11} \rhd N : C]\!]^c, \qquad g = [\![!\Delta', \Gamma_{12}, \Lambda \rhd P : B]\!]^c,$$

we have

$$[\![!\Delta, \Gamma_1, x : A \rhd M : B]\!]^c = (f \otimes_{!\Delta} id); (T(F) \otimes id); \sigma; t; g^*$$

for a given map F, depending on $\boxdot^C$. If M is an extended value, so are N and P. If we call f' and g' the following maps:

$$f' = [\![!\Delta', \Gamma_{11} \rhd N : C]\!]^v, \qquad g' = [\![!\Delta', \Gamma_{12}, \Lambda \rhd P : B]\!]^v,$$

we have

$$[\![!\Delta, \Gamma_1, x : A \rhd M : B]\!]^v = (f' \otimes_{!\Delta} id); (F \otimes id); \sigma; g'.$$

Note that we might need to add a permutation in front of f, g, f' or g', depending on the order of their respective input.

By α-equivalence one can assume that the variables occurring in Λ does not occur anywhere in the other contexts. From Lemma 9.1.19, one can assume that Γ_{11} and Γ_{12} do not contain any duplicable variables, implying that $|!\Delta| \subseteq |!\Delta'|$. Thus there exists some context $!\Psi$ such that $!\Delta' = (!\Psi, !\Delta)$. In particular, this means that

$$(!\Psi, \Gamma_{11}, \Gamma_{12}) = (\Gamma_1, x : A). \tag{11.2.4}$$

This equation is the main point that will allow us to apply the induction hypothesis, and thus finish the proof. The variable $x : A$ belongs to either $!\Psi$, Γ_{11} or Γ_{12}. We study each one of these three possibilities.

If $x \in |!\Psi|$. In this case, $A = !A'$ and $!\Psi = (!\Psi', x : !A')$. The contexts $(!\Delta', \Gamma_{11})$ and $(!\Delta', \Gamma_{12}, \Lambda)$ becomes respectively $(!\Delta, !\Psi', \Gamma_{11}, x : !A')$ and $(!\Delta, !\Psi', \Gamma_{12}, \Lambda, x : !A')$, meaning that the typing judgements $!\Delta, !\Psi', \Gamma_{11}, x : !A' \rhd N : C$ and $!\Delta, !\Psi', \Gamma_{12}, \Lambda, x : !A' \rhd P : B$ are valid.

Since $A = !A'$ and since V is a value, by Lemma 9.1.15 the context Γ_2 is of the form $!\Gamma'_2$. We have $M[V/x] = (\mathit{let}\ \boxdot^C = N[V/x]\ \mathit{in}\ P[V/x])$, and

$$!\Delta, !\Psi', \Gamma_{11}, \Gamma_2 \rhd N[V/x] : C, \qquad !\Delta, !\Psi', \Gamma_{12}, \Gamma_2, \Lambda \rhd P[V/x] : B$$

are valid typing judgements, such that by induction hypotheses their respective computational denotations $\dot{f}$ and $\dot{g}$ are

$$\begin{tikzcd}
[\![!\Delta, !\Psi', \Gamma_{11}, \Gamma_2]\!] \arrow[r, dotted, "\dot{f}"] \arrow[d, "Split_{!\Delta,(!\Psi',\Gamma_{11}),\Gamma_2}"] & T[\![C]\!] \\
[\![!\Delta, !\Psi', \Gamma_{11}]\!] \otimes [\![!\Delta, \Gamma_2]\!] \arrow[r, "id \otimes [\![!\Delta, \Gamma_2 \rhd V : A]\!]^v"] & [\![!\Delta, !\Psi', \Gamma_{11}]\!] \otimes [\![A]\!], \arrow[u, "f"']
\end{tikzcd} \tag{11.2.5}$$

$$\begin{tikzcd}
[\![!\Delta, !\Psi', \Gamma_{12}, \Lambda, \Gamma_2]\!] \arrow[r, dotted, "\dot{g}"] \arrow[d, "Split_{!\Delta,(!\Psi',\Gamma_{12},\Lambda),\Gamma_2}"] & T[\![B]\!] \\
[\![!\Delta, !\Psi', \Gamma_{12}, \Lambda]\!] \otimes [\![!\Delta, \Gamma_2]\!] \arrow[r, "id \otimes [\![!\Delta, \Gamma_2 \rhd V : A]\!]^v"] & [\![!\Delta, !\Psi', \Gamma_{12}, \Lambda]\!] \otimes [\![A]\!], \arrow[u, "g"']
\end{tikzcd} \tag{11.2.6}$$

By definition, the denotation $h = [\![!\Delta, \Gamma_1, !\Gamma'_2 \rhd M[V/x] : B]\!]^c$ is equal to

$$\begin{aligned}
[\![!\Delta, !\Psi', \Gamma_{11}, \Gamma_{12}, !\Gamma'_2]\!] &\xrightarrow{sw_1;Split;sw_2} [\![!\Delta, !\Psi', \Gamma_{11}, !\Gamma'_2, !\Delta, !\Psi', \Gamma_{12}, !\Gamma'_2]\!] \\
&\xrightarrow{\dot{f} \otimes id} T[\![C]\!] \otimes [\![!\Delta, !\Psi', \Gamma_{12}, !\Gamma'_2]\!] \\
&\xrightarrow{T(F) \otimes id} T[\![\Lambda]\!] \otimes [\![!\Delta, !\Psi', \Gamma_{12}, !\Gamma'_2]\!] \\
&\xrightarrow{\sigma;t} T([\![!\Delta, !\Psi', \Gamma_{12}, !\Gamma'_2]\!] \otimes [\![\Lambda]\!]) \\
&\xrightarrow{T(id \otimes \sigma)} T([\![!\Delta, !\Psi', \Gamma_{12}, \Lambda, !\Gamma'_2]\!]) \\
&\xrightarrow{\dot{g}^*} T[\![B]\!].
\end{aligned}$$

Using Equations (11.2.5) and (11.2.6), one can rewrite h as follows:

$$\begin{aligned}
[\![!\Delta, !\Psi', \Gamma_{11}, \Gamma_{12}, !\Gamma'_2]\!] &\xrightarrow{sw_1;Split;sw_2} [\![!\Delta, !\Psi', \Gamma_{11}, !\Gamma'_2, !\Delta, !\Psi', \Gamma_{12}, !\Gamma'_2]\!] \\
&\xrightarrow{(id \otimes_{!\Delta} v) \otimes id} [\![!\Delta, !\Psi', \Gamma_{11}]\!] \otimes [\![A]\!] \otimes [\![!\Delta, !\Psi', \Gamma_{12}, !\Gamma'_2]\!] \\
&\xrightarrow{f \otimes id} T[\![C]\!] \otimes [\![!\Delta, !\Psi', \Gamma_{12}, !\Gamma'_2]\!] \\
&\xrightarrow{T(F) \otimes id} T[\![\Lambda]\!] \otimes [\![!\Delta, !\Psi', \Gamma_{12}, !\Gamma'_2]\!] \\
&\xrightarrow{\sigma;t} T([\![!\Delta, !\Psi', \Gamma_{12}, !\Gamma'_2]\!] \otimes [\![\Lambda]\!]) \\
&\xrightarrow{T(id \otimes \sigma)} T([\![!\Delta, !\Psi', \Gamma_{12}, \Lambda, !\Gamma'_2]\!]) \\
&\xrightarrow{T Split_{!\Delta,(!\Psi',\Gamma_{12},\Lambda),!\Gamma'_2}} T([\![!\Delta, !\Psi', \Gamma_{12}, \Lambda, !\Delta, !\Gamma'_2]\!]) \\
&\xrightarrow{T(id \otimes v)} T([\![!\Delta, !\Psi', \Gamma_{12}, \Lambda]\!] \otimes [\![A]\!]) \\
&\xrightarrow{g^*} T[\![B]\!],
\end{aligned}$$

where sw_1 and sw_2 are permutations. Note that the map $Split_{!\Delta,(!\Psi',\Gamma_{12},\Lambda),!\Gamma'_2}$ is $\triangle_\Delta$ followed by a permutation. It is then possible to lift the second occurrence of v to the top, thus getting h in the required format.

The computation for $h' = [\![!\Delta, \Gamma_1, \Gamma_2 \rhd M[V/x] : B]\!]^v$ is similar.

If $x \in |\Gamma_{11}|$. This time, the context Γ_{11} is of the form $(!\Gamma'_{11}, x : A)$, and Γ_1 can be split into $(!\Delta, !\Psi, \Gamma'_{11}, \Gamma_{12})$. We have $M[V/x] = (\mathit{let}\ \boxdot^C = N[V/x]\ \mathit{in}\ P)$, and

$$!\Delta, !\Psi, \Gamma'_{11}, \Gamma_2 \rhd N[V/x] : C, \qquad !\Delta, !\Psi, \Gamma_{12}, \Lambda \rhd P : B$$

are valid typing judgements. By induction hypothesis, the computational denotation of the former is

$$\begin{tikzcd} \llbracket !\Delta, !\Psi, \Gamma'_{11}, \Gamma_2 \rrbracket \arrow[r, dotted, "\dot{f}"] \arrow[d, "\mathit{Split}_{!\Delta,(!\Psi,\Gamma'_{11}),\Gamma_2}"] & T\llbracket C \rrbracket \\ \llbracket !\Delta, !\Psi, \Gamma'_{11} \rrbracket \otimes \llbracket !\Delta, \Gamma_2 \rrbracket \arrow[r, "id \otimes \llbracket !\Delta, \Gamma_2 \rhd V : A \rrbracket^v"] & \llbracket !\Delta, !\Psi, \Gamma'_{11} \rrbracket \otimes \llbracket A \rrbracket, \arrow[u, "f"'] \end{tikzcd} \tag{11.2.7}$$

By definition, the denotation $h = \llbracket !\Delta, \Gamma_1, \Gamma_2 \rhd M[V/x] : B \rrbracket^c$ is equal to

$$\begin{aligned} \llbracket !\Delta, !\Psi, \Gamma'_{11}, \Gamma_{12}, \Gamma_2 \rrbracket &\xrightarrow{perm} \llbracket !\Delta, !\Psi, \Gamma'_{11}, \Gamma_2, \Gamma_{12} \rrbracket \\ &\xrightarrow{\mathit{Split}_{(!\Delta,!\Psi),(\Gamma'_{11},\Gamma_2),\Gamma_{12}}} \llbracket !\Delta, !\Psi, \Gamma'_{11}, \Gamma_2 \rrbracket \otimes \llbracket !\Delta, !\Psi, \Gamma_{12} \rrbracket \\ &\xrightarrow{\dot{f} \otimes id} T\llbracket C \rrbracket \otimes \llbracket !\Delta, !\Psi, \Gamma_{12} \rrbracket \\ &\xrightarrow{(T(F) \otimes id); \sigma; t} T\llbracket !\Delta, !\Psi, \Gamma_{12}, \Lambda \rrbracket \\ &\xrightarrow{g^*} T\llbracket B \rrbracket. \end{aligned}$$

Using the Equation (11.2.7), this is equal to

$$\begin{aligned} \llbracket !\Delta, !\Psi, \Gamma'_{11}, \Gamma_{12}, \Gamma_2 \rrbracket &\xrightarrow{perm} \llbracket !\Delta, \Gamma'_{11}, \Gamma_2, \Gamma_{12} \rrbracket \\ &\xrightarrow{\mathit{Split}_{(!\Delta,!\Psi),(\Gamma'_{11},\Gamma_2),\Gamma_{12}}} \llbracket !\Delta, !\Psi, \Gamma'_{11}, \Gamma_2 \rrbracket \otimes \llbracket !\Delta, !\Psi, \Gamma_{12} \rrbracket \\ &\xrightarrow{\mathit{Split}_{!\Delta,(!\Psi,\Gamma'_{11}),\Gamma_2} \otimes id} \llbracket !\Delta, !\Psi, \Gamma'_{11} \rrbracket \otimes \llbracket !\Delta, \Gamma_2 \rrbracket \otimes \llbracket !\Delta, !\Psi, \Gamma_{12} \rrbracket \\ &\xrightarrow{id \otimes v \otimes id} \llbracket !\Delta, !\Psi, \Gamma'_{11} \rrbracket \otimes \llbracket A \rrbracket \otimes \llbracket !\Delta, !\Psi, \Gamma_{12} \rrbracket \\ &\xrightarrow{f \otimes id} T\llbracket C \rrbracket \otimes \llbracket !\Delta, !\Psi, \Gamma_{12} \rrbracket \\ &\xrightarrow{(T(F) \otimes id); \sigma; t} T\llbracket !\Delta, !\Psi, \Gamma_{12}, \Lambda \rrbracket \\ &\xrightarrow{g^*} T\llbracket B \rrbracket, \end{aligned}$$

where *perm* is a permutation. Using the coherence properties of the symmetry and the comonoid multiplication, the map

$$\begin{aligned} &\mathit{perm}; \mathit{Split}_{(!\Delta,!\Psi),(\Gamma'_{11},\Gamma_2),\Gamma_{12}}; (\mathit{Split}_{!\Delta,(!\Psi,\Gamma'_{11}),\Gamma_2} \otimes id); (id \otimes v \otimes id) \\ &\qquad = \llbracket !\Delta, !\Psi, \Gamma'_{11}, \Gamma_{12}, \Gamma_2 \rrbracket \xrightarrow{\mathit{Split}_{!\Delta,(!\Psi,\Gamma'_{11},\Gamma_{12}),\Gamma_2}} \llbracket !\Delta, !\Psi, \Gamma'_{11}, \Gamma_{12} \rrbracket \otimes \llbracket !\Delta, \Gamma_2 \rrbracket \\ &\qquad\qquad \xrightarrow{(id \otimes v)} \llbracket !\Delta, !\Psi, \Gamma'_{11}, \Gamma_{12} \rrbracket \otimes \llbracket A \rrbracket, \end{aligned}$$

that is, $(id \otimes_{!\Delta} v)$: the map h is in the requested form.

Again, the computation for $h' = \llbracket !\Delta, \Gamma_1, \Gamma_2 \rhd M[V/x] : B \rrbracket^v$ is similar.

Case $x \in |\Gamma_{12}|$. The context Γ_{12} is of the form $(!\Gamma'_{12}, x : A)$, and Γ_1 is equal to $(!\Delta, !\Psi, \Gamma_{11}, \Gamma'_{12})$. We have $M[V/x] = (\mathit{let}\ \boxdot^C = N\ \mathit{in}\ P[V/x])$, and

$$!\Delta, !\Psi, \Gamma_{11} \rhd N : C, \qquad !\Delta, !\Psi, \Gamma'_{12}, \Lambda, \Gamma_2 \rhd P[V/x] : B$$

are valid typing judgements. By induction hypothesis, the computational denotation of the latter is

$$\begin{tikzcd}
[\![!\Delta, !\Psi, \Gamma'_{12}, \Lambda, \Gamma_2]\!] \arrow[rr, dotted, "\dot{g}"] \arrow[d, "Split_{!\Delta,(!\Psi,\Gamma'_{12},\Lambda),\Gamma_2}"] & & T[\![B]\!] \\
[\![!\Delta, !\Psi, \Gamma'_{12}, \Lambda]\!] \otimes [\![!\Delta, \Gamma_2]\!] \arrow[rr, "id \otimes [\![!\Delta, \Gamma_2 \rhd V : A]\!]^v"] & & [\![!\Delta, !\Psi, \Gamma'_{12}, \Lambda]\!] \otimes [\![A]\!], \arrow[u, "g"]
\end{tikzcd} \tag{11.2.8}$$

By definition, the denotation $h = [\![!\Delta, \Gamma_1, \Gamma_2 \rhd M[V/x] : B]\!]^c$ is equal to

$$\begin{aligned}
[\![!\Delta, !\Psi, \Gamma_{11}, \Gamma'_{12}, \Gamma_2]\!] &\xrightarrow{Split_{(!\Delta,!\Psi),\Gamma_{11},(\Gamma_{12},\Gamma_2)}} [\![!\Delta, !\Psi, \Gamma_{11}]\!] \otimes [\![!\Delta, !\Psi, \Gamma'_{12}, \Gamma_2]\!] \\
&\xrightarrow{f \otimes id} T[\![C]\!] \otimes [\![!\Delta, !\Psi, \Gamma'_{12}, \Gamma_2]\!] \\
&\xrightarrow{(T(F) \otimes id);\sigma;t} T[\![!\Delta, !\Psi, \Gamma'_{12}, \Gamma_2, \Lambda]\!] \\
&\xrightarrow{T(id \otimes \sigma)} T[\![!\Delta, !\Psi, \Gamma'_{12}, \Lambda, \Gamma_2]\!] \\
&\xrightarrow{\dot{g}^*} T[\![B]\!].
\end{aligned}$$

Using the Equation (11.2.8), this is equal to

$$\begin{aligned}
[\![!\Delta, !\Psi, \Gamma_{11}, \Gamma'_{12}, \Gamma_2]\!] &\xrightarrow{Split_{(!\Delta,!\Psi),\Gamma_{11},(\Gamma_{12},\Gamma_2)}} [\![!\Delta, !\Psi, \Gamma_{11}]\!] \otimes [\![!\Delta, !\Psi, \Gamma'_{12}, \Gamma_2]\!] \\
&\xrightarrow{f \otimes id} T[\![C]\!] \otimes [\![!\Delta, !\Psi, \Gamma'_{12}, \Gamma_2]\!] \\
&\xrightarrow{(T(F) \otimes id);\sigma;t} T[\![!\Delta, !\Psi, \Gamma'_{12}, \Gamma_2, \Lambda]\!] \\
&\xrightarrow{T(id \otimes \sigma)} T[\![!\Delta, !\Psi, \Gamma'_{12}, \Lambda, \Gamma_2]\!] \\
&\xrightarrow{TSplit_{!\Delta,(!\Psi,\Gamma'_{12},\Lambda),\Gamma_2}} T([\![!\Delta, !\Psi, \Gamma'_{12}, \Lambda]\!] \otimes [\![!\Delta, \Gamma_2]\!]) \\
&\xrightarrow{T(id \otimes v)} T([\![!\Delta, !\Psi, \Gamma'_{12}, \Lambda]\!] \otimes [\![A]\!]) \\
&\xrightarrow{g^*} T[\![B]\!].
\end{aligned}$$

Using the same technique as in the case where $x \in |!\Psi|$, one can move up the occurrence of v, and thus get h in the requested form.

Again, the computation for $h' = [\![!\Delta, \Gamma_1, \Gamma_2 \rhd M[V/x] : B]\!]^v$ is similar.

Cases $M \equiv NP$ and $M \equiv \langle N, P \rangle^n$. As in the previous case, the typing judgement $!\Delta, \Gamma_1, x : A \rhd M : B$ comes from a typing rule (X) with hypotheses

$$!\Delta', \Gamma_{11} \rhd N : C, \qquad !\Delta', \Gamma_{12}, \rhd P : D,$$

for some types C and D, where $(!\Delta', \Gamma_{11}, \Gamma_{12}) = (!\Delta, \Gamma_1, x : A)$. If we call f and g the following maps:

$$f = [\![!\Delta', \Gamma_{11} \rhd N : C]\!]^c, \qquad g = [\![!\Delta', \Gamma_{12} \rhd P : d]\!]^c,$$

we have

$$[\![!\Delta, \Gamma_1, x : A \rhd M : B]\!]^c = (f \otimes_{!\Delta} g); F$$

for a given map F, depending if M is of the form NP or of the form $\langle N, P\rangle^n$. If M is an extended value, so are N and P. If we call f' and g' the following maps:

$$f' = [\![!\Delta', \Gamma_{11} \rhd N : C]\!]^v, \qquad g' = [\![!\Delta', \Gamma_{12} \rhd P : D]\!]^v,$$

we have

$$[\![!\Delta, \Gamma_1, x : A \rhd M : B]\!]^v = (f' \otimes_{!\Delta} g); G,$$

for a given map G, depending if M is of the form NP or of the form $\langle N, P\rangle^n$. From Lemma 9.1.19, one can assume that Γ_{11} and Γ_{12} do not contain any duplicable variables, implying that $|!\Delta| \subseteq |!\Delta'|$. Thus there exists some context $!\Psi$ such that $!\Delta' = (!\Psi, !\Delta)$. In particular, this means that

$$(!\Psi, \Gamma_{11}, \Gamma_{12}) = (\Gamma_1, x : A). \tag{11.2.9}$$

The variable $x : A$ belongs to either $!\Psi$, Γ_{11} or Γ_{12}. The study of each one of these three possibilities is similar as in the previous case.

This ends the list of cases: the lemma is verified. □

Theorem 11.2.5 (Soundness)**.** *If $\Delta \rhd M \approx_{ax} M' : A$ then we have $[\![\Delta \rhd M : A]\!]^c_\Theta = [\![\Delta \rhd M' : A]\!]^c_\Theta$ (and $[\![\Delta \rhd M : A]\!]^v_\Theta = [\![\Delta \rhd M' : A]\!]^v_\Theta$ if M and M' are values) for every interpretation Θ.*

Proof. Proof by induction on the size of the derivation of $M \approx_{ax} M'$. Case distinction on the last rule used. The cases corresponding to reflexivity, symmetry, transitivity and congruence are trivial. We prove the result for the other cases:

Rule (β_λ). Here, we are considering $\Delta \rhd \mathit{let}\ x = V\ \mathit{in}\ M \approx_{ax} M[V/x] : A$. We have directly from Lemma 11.2.4 that $[\![M[V/x]]\!]^c = [\![\mathit{let}\ x = V\ \mathit{in}\ M]\!]^c$, and $[\![M[V/x]]\!]^v = [\![\mathit{let}\ x = V\ \mathit{in}\ M]\!]^v$ if they are extended values.

Rule ($\beta_\otimes$). We are considering $\Delta \rhd \mathit{let}\ \langle x^C, y^D\rangle^n = \langle V, W\rangle^n\ \mathit{in}\ M \approx_{ax} M[V/x, W/y] : A$. By α-equivalence one can assume that x is not free in W: we have $M[V/x, W/y] = (M[W/y])[V/x]$.

One can split Δ into $(!\Delta', \Gamma_1, \Gamma_2, \Gamma_3)$, such that $|\Gamma_1|$ contains only the free variables of V, $|\Gamma_2|$ only the free variables of W and $|\Gamma_3|$ only the ones of M. From Lemma 9.1.19, the typing judgements

$$!\Delta', \Gamma_1 \rhd V : !^n C, \quad !\Delta', \Gamma_2 \rhd W : !^n D, \quad !\Delta', \Gamma_3, x : !^n C, y : !^n D \rhd M : A,$$

are valid. Let f, g and h be the denotations

$$\begin{aligned} f &= [\![!\Delta', \Gamma_1 \rhd V : !^n C]\!]^v, \\ g &= [\![!\Delta', \Gamma_2 \rhd W : !^n D]\!]^v, \\ h &= [\![!\Delta', \Gamma_3, x : !^n C, y : !^n D \rhd M : B]\!]^c. \end{aligned}$$

From Lemma 11.2.4,

$$\begin{aligned} &[\![!\Delta', \Gamma_3, x : !^n C, \Gamma_2 \rhd M[W/y] : A]\!]^c \\ &= (id \otimes_{!\Delta'} [\![!\Delta', \Gamma_2 \rhd W : !^n D]\!]^v); [\![!\Delta', \Gamma_3, x : !^n C, y : !^n D \rhd M : A]\!]^c, \end{aligned}$$

thus, still from Lemma 11.2.4,

$$[\![!\Delta', \Gamma_3, \Gamma_2, \Gamma_1 \rhd (M[W/y])[V/x] : A]\!]^c$$

$$= (id_{!\Delta',\Gamma_3,\Gamma_2} \otimes_{!\Delta'} f); [\![!\Delta', \Gamma_3, \Gamma_2, x : !^n C \rhd M[W/y] : A]\!]^c$$
$$= (id_{!\Delta',\Gamma_3,\Gamma_2} \otimes_{!\Delta'} f); (id_{!\Delta,\Gamma_3} \otimes \sigma); (id_{!\Delta,\Gamma_3,x:!^n C} \otimes_{!\Delta'} g); h$$

On another hand, by definition we have

$$\begin{aligned}
&[\![!\Delta', \Gamma_1, \Gamma_2 \rhd \langle V, W\rangle^n : !^n(C \otimes D)]\!]^c \\
&= ((f; \eta_{!^n C}) \otimes_{!\Delta'} (g; \eta_{!^n D})); \Psi_1; T d^{L^n}_{C,D}, \\
&[\![!\Delta', \Gamma_1, \Gamma_2, \Gamma_3 \rhd \mathit{let}\ \langle x^C, y^D\rangle^n = \langle V, W\rangle^n\ \mathit{in}\ M : A]\!]^c \\
&= \left(\left(((f; \eta_{!^n C}) \otimes_{!\Delta'} (g; \eta_{!^n D})); \Psi_1; T d^{L^n}_{C,D}\right) \otimes_{!\Delta} id\right); \left(T\left(d^{L^n}_{C,D}\right)^{-1} \otimes id\right); \sigma; t; h^* \\
&= ((((f;\eta_{!^n C})\otimes_{!\Delta'}(g;\eta_{!^n D}));\Psi_1) \otimes_{!\Delta} id);(T d^{L^n}_{C,D}\otimes id); \left(T\left(d^{L^n}_{C,D}\right)^{-1} \otimes id\right);\sigma;t;h^* \\
&= ((((f; \eta_{!^n C}) \otimes_{!\Delta'} (g; \eta_{!^n D})); \Psi_1) \otimes_{!\Delta} id); \sigma; t; h^* \\
&= (((f; \eta_{!^n C}) \otimes_{!\Delta'} (g; \eta_{!^n D})) \otimes_{!\Delta} id); (\Psi_1 \otimes id); \sigma; t; h^* \\
&= ((f \otimes_{!\Delta'} g) \otimes_{!\Delta} id); ((\eta_{!^n C} \otimes \eta_{!^n D}) \otimes id); (\Psi_1 \otimes id); \sigma; t; h^* \\
&= ((f \otimes_{!\Delta'} g) \otimes_{!\Delta} id); (\eta_{!^n(C\otimes D)} \otimes id); \sigma; t; h^* \\
&= ((f \otimes_{!\Delta'} g) \otimes_{!\Delta} id); \sigma; h
\end{aligned}$$

which makes the same map (modulo permutation of inputs) as in the previous calculation.

The case of the value denotation is similar.

Rule (β_*). We have $\Delta \rhd \mathit{let} * = * \ \mathit{in}\ M \approx_{ax} M : A$.

The typing judgement $\Delta \rhd \mathit{let} * = * \ \mathit{in}\ M : A$ comes from rule ($\top.E$). The typing judgements $\rhd * : \top$ and $\Delta \rhd M : A$ are valid, and since $[\![\rhd * : \top]\!]^c = id_\top$, if we call

$$g = [\![\Delta \rhd M : A]\!]^c, \qquad h = [\![\Delta \rhd \mathit{let} * = * \ \mathit{in}\ M : A]\!]^c,$$

then $h = (\eta_\top \otimes g); \Psi_1 = g$. Similarly, if M is an extended value, and if we call

$$g' = [\![\Delta \rhd M : A]\!]^v, \qquad h' = [\![\Delta \rhd \mathit{let} * = * \ \mathit{in}\ M : A]\!]^v,$$

then $h' = (id_\top \otimes g') = g'$.

Rule (η_λ). We have $\Delta \rhd \lambda^n x^A.\{V <: A \multimap B\}x^A \approx_{ax} V : !^n(A \multimap B)$. Both V and $\lambda^n x^A.\{V <: A \multimap B\}x^A$ are values.

Let $f = [\![\Delta \rhd V : !^n(A \multimap B)]\!]^v$.

If $n \neq 0$. Then from Lemma 9.1.15, $\Delta = !\Delta'$.

Let $g = [\![\Delta, x : A \rhd \{V <: A \multimap B\}x^A : B]\!]^c$. From Lemma 11.2.2, we have

$$\begin{aligned}
g &= ((f; I_{!^n(A\multimap B),A\multimap B}; \eta_{A\multimap B}) \otimes \eta_A); \Psi_1; \varepsilon^*_{A,B} \\
&= ((f; I_{!^n(A\multimap B),A\multimap B}) \otimes id_A); (\eta_{A\multimap B} \otimes \eta_A); \Psi_1; \varepsilon^*_{A,B} \\
&= ((f; I_{!^n(A\multimap B),A\multimap B}) \otimes id_A); \varepsilon_{A,B}.
\end{aligned}$$

By naturality of Φ,

$$\Phi^{-1}\left(((f; I_{!^n(A\multimap B),A\multimap B}) \otimes id_A); \varepsilon_{A,B}\right) = f; I_{!^n(A\multimap B),A\multimap B}; \Phi^{-1}(\varepsilon_{A,B})$$

$$= f; I_{!^n(A \multimap B), A \multimap B}.$$

Thus, if $h = [\![\Delta \rhd \lambda^n x^A.\{V <: A \multimap B\} x^A : !^n(A \multimap B)]\!]^v$,

$$\begin{aligned} h &= \delta_{\Delta'}; d^L_\Delta; L(\Phi^{-1}(g)); I_{!(A \multimap B), !^n(A \multimap B)} \\ &= \delta_{\Delta'}; d^L_\Delta; L(f; I_{!^n(A \multimap B), A \multimap B}); I_{!(A \multimap B), !^n(A \multimap B)} \\ &= \delta_{\Delta'}; d^L_\Delta; L(f); L(I_{!^n(A \multimap B), A \multimap B}); I_{!(A \multimap B), !^n(A \multimap B)} \\ &= \delta_{\Delta'}; d^L_\Delta; L(f); \epsilon_{!^n(A \multimap B)} \\ &= \delta_{\Delta'}; d^L_\Delta; \epsilon_\Delta; f \\ &= f. \end{aligned}$$

If $n = 0$. The computation is similar, although simpler since there is no reference to ! nor to L. Indeed, $\{V <: A \multimap B\} = V$ from Lemma 9.1.23, and if $g = [\![\Delta, x : A \rhd V x^A : B]\!]^c$,

$$\begin{aligned} g &= (f; \eta_{A \multimap B}) \otimes (\eta_A); \Psi_1; \varepsilon^*_{A,B} \\ &= (f \otimes id_A); (\eta_{A \multimap B}) \otimes (\eta_A); \Psi_1; \varepsilon^*_{A,B} \\ &= (f \otimes id_A); \varepsilon_{A,B}. \end{aligned}$$

By naturality of Φ, if $h = [\![\Delta \rhd \lambda^0 x^A.\{V <: A \multimap B\} x^A : A \multimap B]\!]^v$,

$$h = L(\Phi^{-1}(g)) = f$$

Finally, in the case of the computational denotation, since both terms are core values, we have the result by definition.

Rule (β^2_λ). We have $\Delta \rhd \mathit{let}\ x^A = N\ \mathit{in}\ x^A \approx_{ax} N : A$. By α-equivalence one can assume that x is not a free variable of N.

Suppose that N is an extended value, and let f be $[\![\Delta \rhd N : A]\!]^v$. Then if h is the denotation $[\![\Delta \rhd \mathit{let}\ x^A = N\ \mathit{in}\ x^A : A]\!]^v$, we have

$$h = (f \otimes id_\top); \sigma_{A,\top}; id_A = f.$$

Now, suppose that N is any term, and let f' be $[\![\Delta \rhd N : A]\!]^c$. From Lemma 11.1.15, if $h' = [\![\Delta \rhd \mathit{let}\ x^A = N\ \mathit{in}\ x^A : A]\!]^c$,

$$h' = (f' \otimes id_\top); \sigma_{T([\![A]\!]),\top}; t; id^*_A = (f' \otimes id_\top); \sigma_{A,\top}; id_A = f'.$$

Rule $(\eta_\otimes)$. We have $\Delta \rhd \mathit{let}\ \langle x^A, y^B \rangle^n = N\ \mathit{in}\ \langle x^{!^n A}, y^{!^n B} \rangle^n \approx_{ax} N : !^n(A \otimes B)$. By α-equivalence one can assume that neither x nor y are free variables of N.

Suppose that N is an extended value, and let f be $[\![\Delta \rhd N : !^n(A \otimes B)]\!]^v$. Then if h is the denotation $[\![\Delta \rhd \mathit{let}\ \langle x^A, y^B \rangle^n = N\ \mathit{in}\ \langle x^{!^n A}, y^{!^n B} \rangle^n : !^n(A \otimes B)]\!]^v$, using the fact that the category is strongly monoidal and monoidal strict,

$$\begin{aligned} h &= (f \otimes id_\top); \left(\left(d^{L^n}_{[\![A]\!],[\![B]\!]} \right)^{-1} \otimes id_\top \right); \sigma_{!^n(A \otimes B),\top}; d^{L^n}_{[\![A]\!],[\![B]\!]} \\ &= f; \left(d^{L^n}_{[\![A]\!],[\![B]\!]} \right)^{-1}; d^{L^n}_{[\![A]\!],[\![B]\!]} \\ &= f. \end{aligned}$$

Now, suppose that N is any term, and let f' be $[\![\Delta \rhd N : !^n(A \otimes B)]\!]^c$. If h' is the denotation $[\![\Delta \rhd \mathit{let}\ \langle x^A, y^B\rangle^n = N\ \mathit{in}\ \langle x^{!^n A}, y^{!^n B}\rangle^n : !^n(A \otimes B)]\!]^c$,

$$\begin{aligned} h' &= (f' \otimes id_\top); \left(T\left(d^{L^n}_{[\![A]\!],[\![B]\!]}\right)^{-1} \otimes id_\top\right); \sigma_{T[\![!^n(A\otimes B)]\!],\top}; t; \left(d^{L^n}_{[\![A]\!],[\![B]\!]}; \eta\right)^* \\ &= f'; T\left(d^{L^n}_{[\![A]\!],[\![B]\!]}\right)^{-1}; T d^{L^n}_{[\![A]\!],[\![B]\!]} \\ &= f'. \end{aligned}$$

Rule (η_*). We have $\Delta \rhd \mathit{let}\ * = \{N <: \top\}\ \mathit{in}\ *^n \approx_{ax} N : !^n\top$.

Suppose that N is an extended value, and let f be $[\![\Delta \rhd N : !^n\top]\!]^v$. Then if h is the denotation $[\![\Delta \rhd \mathit{let}\ * = \{N <: \top\}\ \mathit{in}\ *^n : !^n\top]\!]^v$, using Lemma 11.2.2 and Lemma 8.3.3,

$$h = ((f; I_{!^n\top,\top}) \otimes d^{L^n}_\top) = f; I_{!^n\top,\top}; d^{L^n}_\top = f.$$

Now, suppose that N is any term, and let f' be the denotation $[\![\Delta \rhd N : !^n\top]\!]^c$. If h' is the denotation $[\![\Delta \rhd \mathit{let}\ * = \{N <: \top\}\ \mathit{in}\ *^n : !^n\top]\!]^c$,

$$\begin{aligned} h' &= ((f'; T I_{!^n\top,\top}) \otimes (d^{L^n}_\top; \eta_{!^n\top})); \Psi_1 \\ &= ((f'; T I_{!^n\top,\top}) \otimes (\eta_\top; T d^{L^n}_\top)); \Psi_1 \\ &= (f' \otimes \eta_\top); (T I_{!^n\top,\top} \otimes T d^{L^n}_\top); \Psi_1 \\ &= (f' \otimes \eta_\top); \Psi_1; T(I_{!^n\top,\top} \otimes d^{L^n}_\top) \\ &= f'; T(I_{!^n\top,\top}; d^{L^n}_\top) \\ &= f'. \end{aligned}$$

Rule (let_1). We have

$$\Delta \rhd (\mathit{let}\ \boxdot^C = (\mathit{let}\ \boxdot^D = M\ \mathit{in}\ N)\ \mathit{in}\ P \approx_{ax} \mathit{let}\ \boxdot^D = M\ \mathit{in}\ \mathit{let}\ \boxdot^C = N\ \mathit{in}\ P : A.$$

Using Lemma 9.1.19, there exist contexts $!\Delta'$, Γ_1, Γ_2, Γ_3, Λ_1 and Λ_2 such that Λ_1 contains the variables in $\boxdot$, Λ_2 the variables in $\boxdot$, such that $\Delta = (!\Delta', \Gamma_1, \Gamma_2, \Gamma_3)$ and such that

$$!\Delta', \Gamma_1 \rhd M : D, \quad !\Delta', \Gamma_2, \Lambda_1 \rhd N : C, \quad !\Delta', \Gamma_3, \Lambda_2 \rhd P : A.$$

Let h_1 and h_2 be

$$\begin{aligned} h_1 &= [\![!\Delta', \Gamma_1, \Gamma_2, \Gamma_3 \rhd (\mathit{let}\ \boxdot^C = (\mathit{let}\ \boxdot^D = M\ \mathit{in}\ N)\ \mathit{in}\ P : A]\!]^c, \\ h_2 &= [\![!\Delta', \Gamma_1, \Gamma_2, \Gamma_3 \rhd \mathit{let}\ \boxdot^D = M\ \mathit{in}\ \mathit{let}\ \boxdot^C = N\ \mathit{in}\ P : A]\!]^c. \end{aligned}$$

Let us define the following maps:

$$\begin{aligned} f_M &= [\![!\Delta', \Gamma_1 \rhd M : D]\!]^c \\ f_N &= [\![!\Delta', \Gamma_2, \Lambda_1 \rhd N : C]\!]^c \\ f_P &= [\![!\Delta', \Gamma_3, \Lambda_2 \rhd P : A]\!]^c \\ g_1 &= [\![!\Delta', \Gamma_1, \Gamma_2 \rhd \mathit{let}\ \boxdot^D = M\ \mathit{in}\ N : C]\!]^c \\ g_2 &= [\![!\Delta', \Gamma_2, \Lambda_1, \Gamma_3 \rhd \mathit{let}\ \boxdot^C = N\ \mathit{in}\ P : A]\!]^c \end{aligned}$$

There exists a map $F : [\![D]\!] \to [\![\Lambda_1]\!]$ and a map $G : [\![C]\!] \to [\![\Lambda_2]\!]$, respectively based upon the value of $\boxdot$ and $\boxdot$, such that

$$\begin{aligned}
g_1 &= (f_M \otimes_{!\Delta'} id_{!\Delta',\Gamma_2}); (TF \otimes id_{!\Delta',\Gamma_2}); \sigma_{T[\![\Lambda_1]\!],[\![!\Delta',\Gamma_2]\!]}; t; f_N^*,\\
g_2 &= (f_N \otimes_{!\Delta'} id_{!\Delta',\Gamma_3}); (TG \otimes id_{!\Delta',\Gamma_3}); \sigma_{T[\![\Lambda_2]\!],[\![!\Delta',\Gamma_3]\!]}; t; f_P^*,\\
h_1 &= (g_1 \otimes_{!\Delta'} id_{!\Delta',\Gamma_3}); (TG \otimes id_{!\Delta',\Gamma_3}); \sigma_{T[\![\Lambda_2]\!],[\![!\Delta',\Gamma_3]\!]}; t; f_P^*,\\
h_2 &= (f_M \otimes_{!\Delta'} id_{!\Delta',\Gamma_2,\Gamma_3}); (TF \otimes id_{!\Delta',\Gamma_2,\Gamma_3}); \sigma_{T[\![\Lambda_1]\!],[\![!\Delta',\Gamma_2,\Gamma_3]\!]}; t; T(id \otimes \sigma_{[\![\Gamma_3]\!],[\![\Lambda_1]\!]}); g_2^*.
\end{aligned}$$

Using Lemma 11.1.11, one can rewrite h_1 as

$$h_1 = (((f_M; TF) \otimes_{!\Delta'} id_{!\Delta',\Gamma_2}) \otimes_{!\Delta'} id_{!\Delta',\Gamma_3}); ((\sigma; t; f_N^*; TG) \otimes id_{!\Delta',\Gamma_3}); \sigma; t; f_P^*.$$

Using Lemma 11.1.12, one can show that

$$\begin{aligned}
T[\![\Lambda_1]\!] \otimes [\![!\Delta',\Gamma_2,\Gamma_3]\!] &\xrightarrow{\sigma\otimes t} T[\![!\Delta',\Gamma_2,\Gamma_3,\Lambda_1]\!] \xrightarrow{T(id\otimes\sigma)} T[\![!\Delta',\Gamma_2,\Lambda_1,\Gamma_3]\!]\\
&\xrightarrow{T((f_N;TG)\otimes_{!\Delta'} id)} T(T[\![\Lambda_2]\!] \otimes [\![!\Delta',\Gamma_3]\!])
\end{aligned}$$

is equal to

$$\begin{aligned}
T[\![\Lambda_1]\!] \otimes [\![!\Delta',\Gamma_2,\Gamma_3]\!] &\xrightarrow{(id\otimes\sigma);\sigma} [\![!\Delta',\Gamma_3,\Gamma_2]\!] \otimes T[\![\Lambda_1]\!] \xrightarrow{t} T[\![!\Delta',\Gamma_3,\Gamma_2,\Lambda_1]\!]\\
&\xrightarrow{T(id\otimes_{!\Delta'}(f_N;TG))} T([\![!\Delta',\Gamma_3]\!] \otimes T[\![\Lambda_2]\!]) \xrightarrow{T\sigma} T(T[\![\Lambda_2]\!] \otimes [\![!\Delta',\Gamma_3]\!]).
\end{aligned}$$

Now, using Equations (5.4.5) and (5.4.4), the following diagram is commutative:

$$\begin{tikzcd}
 & {[\![!\Delta',\Gamma_3,\Gamma_2]\!] \otimes T[\![\Lambda_1]\!]} \arrow[d, "Split\otimes id"'] \arrow[dr, "t"] & \\
 & {[\![!\Delta',\Gamma_3,!\Delta',\Gamma_2]\!] \otimes T[\![\Lambda_1]\!]} \arrow[d, "id\otimes t"] \arrow[dr, "t"] & {T[\![!\Delta',\Gamma_3,\Gamma_2,\Lambda_1]\!]} \arrow[d, "TSplit=T(Split\otimes id)"] \\
 & {[\![!\Delta',\Gamma_3]\!] \otimes T[\![!\Delta',\Gamma_2,\Lambda_1]\!]} \arrow[r, "t"] \arrow[d, "id\otimes T(f_N;TG)"] \arrow[dd, bend right=60, "id\otimes(f_N^*;TG)"'] & {T[\![!\Delta',\Gamma_3,!\Delta',\Gamma_2,\Lambda_1]\!]} \arrow[d, "T(id\otimes(f_N;TG))"] \\
 & {[\![!\Delta',\Gamma_3]\!] \otimes T^2[\![\Lambda_2]\!]} \arrow[r, "t"] \arrow[d, "id\otimes\mu"] & {T([\![!\Delta',\Gamma_3]\!] \otimes T[\![\Lambda_2]\!])} \arrow[d, "T(t)"] \\
 & {[\![!\Delta',\Gamma_3]\!] \otimes T[\![\Lambda_2]\!]} \arrow[dr, "t"'] & {T^2[\![!\Delta',\Gamma_3,\Lambda_2]\!]} \arrow[dl, "\mu"] \\
 & & {T[\![!\Delta',\Gamma_3,\Lambda_2]\!]}
\end{tikzcd}$$

This is enough to say that $h_1 = h_2$.

The case where M, N and P are extended values is similar, without the need for the equations of strong monadicity.

Rule (let_2). We have

$$\Delta \rhd let\ \boxdot = V\ in\ let\ \boxdot = W\ in\ M \approx_{ax} let\ \boxdot = W\ in\ let\ \boxdot = V\ in\ M : A,$$

where V and W are core values. Using Lemma 9.1.19, there exist contexts $!\Delta'$, Γ_1, Γ_2, Γ_3, Λ_1 and Λ_2 such that Λ_1 contains the variables in $\boxdot$, Λ_2 the variables in $\boxdot$, such that $\Delta = (!\Delta', \Gamma_1, \Gamma_2, \Gamma_3)$ and such that

$$!\Delta', \Gamma_3, \Lambda_1, \Lambda_2 \rhd M : A, \quad !\Delta', \Gamma_1 \rhd V : C, \quad !\Delta', \Gamma_2 \rhd W : D.$$

Let h_1 and h_2 be

$$\begin{aligned} h_1 &= [\![!\Delta', \Gamma_1, \Gamma_2, \Gamma_3 \rhd \mathit{let}\ \boxdot = V\ \mathit{in}\ \mathit{let}\ \boxdot = W\ \mathit{in}\ M : A]\!]^c, \\ h_2 &= [\![!\Delta', \Gamma_1, \Gamma_2, \Gamma_3 \rhd \mathit{let}\ \boxdot = W\ \mathit{in}\ \mathit{let}\ \boxdot = V\ \mathit{in}\ M : A]\!]^c. \end{aligned}$$

Let us define the following maps:

$$\begin{aligned} f_M &= [\![!\Delta', \Gamma_3, \Lambda_1, \Lambda_2 \rhd M : A]\!]^c \\ f_V &= [\![!\Delta', \Gamma_1 \rhd V : C]\!]^v \\ f_W &= [\![!\Delta', \Gamma_2 \rhd W : D]\!]^v \\ g_1 &= [\![!\Delta', \Gamma_1, \Gamma_3, \Lambda_2 \rhd \mathit{let}\ \boxdot = V\ \mathit{in}\ M : A]\!]^c \\ g_2 &= [\![!\Delta', \Gamma_2, \Gamma_3, \Lambda_1 \rhd \mathit{let}\ \boxdot = W\ \mathit{in}\ M : A]\!]^c \end{aligned}$$

There exists a map $F : [\![C]\!] \to [\![\Lambda_1]\!]$ and a map $G : [\![D]\!] \to [\![\Lambda_2]\!]$, respectively based upon the value of $\boxdot$ and $\boxdot$, such that

$$\begin{aligned} g_1 &= ((f_V; \eta_C; TF) \otimes_{!\Delta'} id_{!\Delta', \Gamma_3, \Lambda_2}); \sigma_{T[\![\Lambda_1]\!], [\![!\Delta', \Gamma_3, \Lambda_2]\!]}; t; (id \otimes \sigma_{[\![\Lambda_2]\!], [\![\Lambda_1]\!]}); f_M^*, \\ g_2 &= ((f_W; \eta_D; TG) \otimes_{!\Delta'} id_{!\Delta', \Gamma_3, \Lambda_1}); \sigma_{T[\![\Lambda_2]\!], [\![!\Delta', \Gamma_3, \Lambda_1]\!]}; t; f_M^*, \\ h_1 &= ((f_V; \eta_C; TF) \otimes_{!\Delta'} id_{!\Delta', \Gamma_2, \Gamma_3}); \sigma_{T[\![\Lambda_1]\!], [\![!\Delta', \Gamma_2, \Gamma_3]\!]}; t; g_2^*, \\ h_2 &= id \otimes \sigma_{[\![\Gamma_1]\!], [\![\Gamma_2]\!]} \otimes id; ((f_W; \eta_D; TG) \otimes_{!\Delta'} id_{!\Delta', \Gamma_1, \Gamma_3}); \sigma_{T[\![\Lambda_2]\!], [\![!\Delta', \Gamma_1, \Gamma_3]\!]}; t; g_1^*. \end{aligned}$$

We now want to show that $h_1 = h_2$. Using Equation (5.4.4) stating that $\eta_{A \otimes B} = (id_A \otimes \eta_B); t_{A,B}$, we rewrite h_1:

$$\begin{aligned} h_1 &= ((f_V; \eta_C; TF) \otimes_{!\Delta'} id_{!\Delta', \Gamma_2, \Gamma_3}); \sigma_{T[\![\Lambda_1]\!], [\![!\Delta', \Gamma_2, \Gamma_3]\!]}; t; g_2^* \\ &= ((f_V; F; \eta_{[\![\Lambda_1]\!]}) \otimes_{!\Delta'} id_{!\Delta', \Gamma_2, \Gamma_3}); \sigma_{T[\![\Lambda_1]\!], [\![!\Delta', \Gamma_2, \Gamma_3]\!]}; t; g_2^* \\ &= ((f_V; F) \otimes_{!\Delta'} id_{!\Delta', \Gamma_2, \Gamma_3}); \sigma_{[\![\Lambda_1]\!], [\![!\Delta', \Gamma_2, \Gamma_3]\!]}; (id_{!\Delta', \Gamma_2, \Gamma_3} \otimes \eta_{[\![\Lambda_1]\!]}) t; g_2^* \\ &= ((f_V; F) \otimes_{!\Delta'} id_{!\Delta', \Gamma_2, \Gamma_3}); \sigma_{[\![\Lambda_1]\!], [\![!\Delta', \Gamma_2, \Gamma_3]\!]}; \eta; g_2^* \\ &= ((f_V; F) \otimes_{!\Delta'} id_{!\Delta', \Gamma_2, \Gamma_3}); \sigma_{[\![\Lambda_1]\!], [\![!\Delta', \Gamma_2, \Gamma_3]\!]}; g_2. \end{aligned}$$

Similarly one can rewrite g_1, g_2 and h_2:

$$\begin{aligned} g_1 &= ((f_V; F) \otimes_{!\Delta'} id_{!\Delta', \Gamma_3, \Lambda_2}); \sigma_{[\![\Lambda_1]\!], [\![!\Delta', \Gamma_3, \Lambda_2]\!]}; (id \otimes \sigma_{[\![\Lambda_2]\!], [\![\Lambda_1]\!]}); f_M, \\ g_2 &= ((f_W; G) \otimes_{!\Delta'} id_{!\Delta', \Gamma_3, \Lambda_1}); \sigma_{[\![\Lambda_2]\!], [\![!\Delta', \Gamma_3, \Lambda_1]\!]}; f_M, \\ h_2 &= id \otimes \sigma_{[\![\Gamma_1]\!], [\![\Gamma_2]\!]} \otimes id; ((f_W; G) \otimes_{!\Delta'} id_{!\Delta', \Gamma_1, \Gamma_3}); \sigma_{[\![\Lambda_2]\!], [\![!\Delta', \Gamma_1, \Gamma_3]\!]}; g_1. \end{aligned}$$

The term h_1 becomes:

$$\begin{aligned} h_1 &= ((f_V; F) \otimes_{!\Delta'} id_{!\Delta', \Gamma_2, \Gamma_3}); \sigma_{[\![\Lambda_1]\!], [\![!\Delta', \Gamma_2, \Gamma_3]\!]}; g_2 \\ &= ((f_V; F) \otimes_{!\Delta'} id_{!\Delta', \Gamma_2, \Gamma_3}); \sigma_{[\![\Lambda_1]\!], [\![!\Delta', \Gamma_2, \Gamma_3]\!]}; ((f_W; G) \otimes_{!\Delta'} id_{!\Delta', \Gamma_3, \Lambda_1}); \\ &\quad \sigma_{[\![\Lambda_2]\!], [\![!\Delta', \Gamma_3, \Lambda_1]\!]}; f_M \\ &= ((f_V; F) \otimes_{!\Delta'} id_{!\Delta', \Gamma_2, \Gamma_3}); (id_{\Lambda_1} \otimes ((f_W; G) \otimes_{!\Delta'} id_{!\Delta', \Gamma_3}); \sigma_{[\![\Lambda_1, \Lambda_2]\!], [\![!\Delta', \Gamma_3]\!]}; f_M \end{aligned}$$

$$= ((f_V; F) \otimes_{!\Delta'} ((f_W; G) \otimes_{!\Delta'} id_{!\Delta',\Gamma_3})); \sigma_{[\![\Lambda_1,\Lambda_2]\!],[\![!\Delta',\Gamma_3]\!]}; f_M.$$

A similar computation shows that h_2 also yields this term.

In the case where M is an extended value, the only computation needed is the last one.

Rules (let^{app}) *and* $(let^{\otimes})$. We have

$$\begin{aligned}
&\Delta \rhd let\ x^{A\multimap B} = M\ in\ let\ y^A = N\ in\ x^{A\multimap B} y^A && \approx_{ax} MN && : B,\\
&\Delta \rhd let\ x^{!^n A} = M\ in\ let\ y^{!^n B} = N\ in\ \langle x^{!^n A}, y^{!^n B}\rangle^n && \approx_{ax} \langle M, N\rangle^n && : !^n(A \otimes B).
\end{aligned}$$

Note that both of these equations are of the form

$$\Delta \rhd let\ x^C = M\ in\ let\ y^D = N\ in\ P \qquad \approx_{ax} P(M, N) : E,$$

for some types C, D and E and some term construct $P(-,-)$.

We can compute the computational interpretation in both cases. The value interpretation can be done only in the second case. However, the proof being a subset of the one required for the computational interpretation, we omit it.

Using α-equivalence, one can assume that x and y does not occur in M nor in N. Using Lemma 9.1.19, there exist contexts $!\Delta'$, Γ_1 and Γ_2 such that $\Delta = (!\Delta', \Gamma_1, \Gamma_2)$ and such that

$$!\Delta', \Gamma_1 \rhd M : C, \qquad\qquad !\Delta', \Gamma_2 \rhd N : D.$$

Let h_1 and h_2 be

$$\begin{aligned}
h_1 &= [\![!\Delta', \Gamma_1, \Gamma_2 \rhd let\ x^C = M\ in\ let\ y^D = N\ in\ P(x, y) : B]\!]^c,\\
h_2 &= [\![!\Delta', \Gamma_1, \Gamma_2 \rhd P(M, N) : B]\!]^c.
\end{aligned}$$

Let us define the following maps:

$$\begin{aligned}
f_M &= [\![!\Delta', \Gamma_1 \rhd M : C]\!]^c,\\
f_N &= [\![!\Delta', \Gamma_2 \rhd N : D]\!]^c,\\
g &= [\![!\Delta', \Gamma_2, x : C \rhd let\ y^D = N\ in\ P(x, y) : B]\!]^c,\\
k &= [\![x : C, y : D \rhd P(x, y) : B]\!]^c.
\end{aligned}$$

By definition and from Lemma 11.1.15, there exists a map $F : [\![C \otimes D]\!] \to T(B)$ such that

$$\begin{aligned}
k &= (\eta_C \otimes \eta_D); \Psi_1; F^*,\\
g &= (f_N \otimes id_C); \sigma_{T[\![D]\!],[\![C]\!]}; t; k^*,\\
h_1 &= (f_M \otimes_{!\Delta'} id_{!\Delta',\Gamma_2}); \sigma_{T[\![C]\!],[\![!\Delta',\Gamma_2]\!]}; t; g^*,\\
h_2 &= (f_M \otimes_{!\Delta'} f_N); \Psi_1; F^*,
\end{aligned}$$

Using Equation (5.4.4) of the tensorial strength, let us rewrite k:

$$\begin{aligned}
k &= (\eta_C \otimes \eta_D); \Psi_1; F^*\\
&= (\eta_C \otimes \eta_D); \sigma_{T[\![C]\!],T[\![D]\!]}; t_{T[\![D]\!],[\![C]\!]}; T\sigma_{T[\![D]\!],[\![C]\!]}; Tt_{[\![C]\!],[\![D]\!]}; \mu; F^*\\
&= (\eta_D \otimes \eta_C); t_{T[\![D]\!],[\![C]\!]}; T\sigma_{T[\![D]\!],[\![C]\!]}; Tt_{[\![C]\!],[\![D]\!]}; \mu; F^*\\
&= (\eta_D \otimes id_C); \eta_{T[\![D]\!]\otimes[\![C]\!]}; T\sigma_{T[\![D]\!],[\![C]\!]}; Tt_{[\![C]\!],[\![D]\!]}; \mu; F^*
\end{aligned}$$

$$
\begin{aligned}
&= (\eta_D \otimes id_C); \sigma_{T[\![D]\!],[\![C]\!]}; t_{[\![C]\!],[\![D]\!]}; F^* \\
&= (id_C \otimes \eta_D); t_{[\![C]\!],[\![D]\!]}; F^* \\
&= \eta_{[\![C]\!]\otimes[\![D]\!]}; F^* \\
&= F.
\end{aligned}
$$

Let us rewrite g:

$$g = (f_N \otimes id_C); \sigma_{T[\![D]\!],[\![C]\!]}; t; k^* = (f_N \otimes id_C); \sigma_{T[\![D]\!],[\![C]\!]}; t; F^*$$

Let us rewrite h_1:

$$
\begin{aligned}
h_1 &= (f_M \otimes_{!\Delta'} id_{!\Delta',\Gamma_2}); \sigma_{T[\![C]\!],[\![!\Delta',\Gamma_2]\!]}; t; g^* \\
&= (f_M \otimes_{!\Delta'} id_{!\Delta',\Gamma_2}); \sigma_{T[\![C]\!],[\![!\Delta',\Gamma_2]\!]}; t; T(f_N \otimes id_C); T\sigma_{T[\![D]\!],[\![C]\!]}; Tt; F^* \\
&= (f_M \otimes_{!\Delta'} f_N); \sigma_{T[\![C]\!],T[\![D]\!]}; t; T(id_{T[\![D]\!]} \otimes id_C); T\sigma_{T[\![D]\!],[\![C]\!]}; Tt; F^* \\
&= (f_M \otimes_{!\Delta'} f_N); \sigma_{T[\![C]\!],T[\![D]\!]}; t; T\sigma_{T[\![D]\!],[\![C]\!]}; Tt; F^* \\
&= (f_M \otimes_{!\Delta'} f_N); \Psi_1; F^* \\
&= h_2.
\end{aligned}
$$

Rule (let^λ). We have

$$\Delta \rhd let\ x^D = V\ in\ \lambda^n y^A.M \approx_{ax} \lambda^n y^A.\ let\ \ x^D = V\ in\ M : !^n(A \multimap B),$$

where V is a core value. Using α-equivalence, one can assume that x does not occur in M nor in V, and that $y \neq x$. Using Lemma 9.1.19, there exist contexts $!\Delta'$, Γ_1 and Γ_2 such that $\Delta = (!\Delta', \Gamma_1, \Gamma_2)$ and such that

$$!\Delta', \Gamma_2, x : D, y : A \rhd M : B, \qquad !\Delta', \Gamma_1 \rhd V : D,$$

Let h_1 and h_2 be

$$
\begin{aligned}
h_1 &= [\![!\Delta', \Gamma_1, \Gamma_2 \rhd let\ x^D = V\ in\ \lambda^n y^A.M : !^n(A \multimap B)]\!]^c, \\
h_2 &= [\![!\Delta', \Gamma_1, \Gamma_2 \rhd \lambda^n y^A.\ let\ \ x^D = V\ in\ M : !^n(A \multimap B)]\!]^c.
\end{aligned}
$$

Let us define the following maps:

$$
\begin{aligned}
f_M &= [\![!\Delta', \Gamma_2, x : D, y : A \rhd M : B]\!]^c \\
f_V &= [\![!\Delta', \Gamma_1 \rhd V : D]\!]^v \\
g_1 &= [\![!\Delta', \Gamma_2, x : D \rhd \lambda^n y^A.M : !^n(A \multimap B)]\!]^c \\
g_2 &= [\![!\Delta', \Gamma_1, \Gamma_2, y : A \rhd let\ x^D = V\ in\ M : B]\!]^c
\end{aligned}
$$

There are four cases: n can be null or not, and M can be an extended value or not (allowing us to compute the value interpretation). We fully develop the computational interpretation of the case $n = 0$ since the other three cases are done using similar calculations.

From Lemma 9.1.15, Γ_1 and Γ_2 are respectively of the form $!\Gamma_1'$ and $!\Gamma_2'$, and $D = !D'$. By definition, we have

$$
\begin{aligned}
g_1 &= \delta_{!\Delta',!\Gamma_2',x:!D'}; d^L_{!\Delta',!\Gamma_2',x:!D'}; L\Phi^{-1}(f_M); I_{!(A\multimap B),!^n(A\multimap B)}; \eta_{!^n(A\multimap B)} \\
g_2 &= ((f_V; \eta_D) \otimes_{!\Delta'} id_{!\Delta',\Gamma_2,y:A}); \sigma_{T[\![D]\!],[\![!\Delta',\Gamma_2,y:A]\!]}; t; T(id \otimes \sigma_{A,D}); f^*_M,
\end{aligned}
$$

$$h_1 = ((f_V;\eta_D)\otimes_{!\Delta'} id_{!\Delta',\Gamma_2});\sigma_{T[\![D]\!],[\![!\Delta',\Gamma_2]\!]};t;g_1^*,$$
$$h_2 = \delta_{!\Delta',!\Gamma_1',!\Gamma_2'};d^L_{!\Delta',!\Gamma_1',!\Gamma_1'};L\Phi^{-1}(g_2);I_{!(A\multimap B),!^n(A\multimap B)};\eta_{!^n(A\multimap B)}.$$

We want to show that $h_1 = h_2$. Let us rewrite g_2, using Equation (5.4.4) of strong monadicity of T:

$$\begin{aligned}
g_2 &= ((f_V;\eta_D)\otimes_{!\Delta'} id_{!\Delta',\Gamma_2,y:A});\sigma_{T[\![D]\!],[\![!\Delta',\Gamma_2,y:A]\!]};t;T(id\otimes\sigma_{A,D});f_M^*\\
&= (f_V\otimes_{!\Delta'} id_{!\Delta',\Gamma_2,y:A});\sigma_{[\![D]\!],[\![!\Delta',\Gamma_2,y:A]\!]};(id\otimes\eta_D);t;T(id\otimes\sigma_{A,D});f_M^*\\
&= (f_V\otimes_{!\Delta'} id_{!\Delta',\Gamma_2,y:A});\sigma_{[\![D]\!],[\![!\Delta',\Gamma_2,y:A]\!]};\eta_{!\Delta',\Gamma_2,y:A,x:D};T(id\otimes\sigma_{A,D});f_M^*\\
&= (f_V\otimes_{!\Delta'} id_{!\Delta',\Gamma_2,y:A});\sigma_{[\![D]\!],[\![!\Delta',\Gamma_2,y:A]\!]};(id\otimes\sigma_{A,D});\eta_{!\Delta',\Gamma_2,x:D,y:A};f_M^*\\
&= (f_V\otimes_{!\Delta'} id_{!\Delta',\Gamma_2,y:A});\sigma_{[\![D]\!],[\![!\Delta',\Gamma_2,y:A]\!]};(id\otimes\sigma_{A,D});f_M\\
&= (f_V\otimes_{!\Delta'} id_{!\Delta',\Gamma_2,y:A});(\sigma_{[\![D]\!],[\![!\Delta',\Gamma_2]\!]}\otimes id_A);f_M\\
&= ((f_V\otimes_{!\Delta'} id_{!\Delta',\Gamma_2})\otimes id_A);(\sigma_{[\![D]\!],[\![!\Delta',\Gamma_2]\!]}\otimes id_A);f_M.
\end{aligned}$$

By naturality of Φ,

$$\Phi^{-1}(g_2) = (f_V\otimes_{!\Delta'} id_{!\Delta',\Gamma_2});\sigma_{[\![D]\!],[\![!\Delta',\Gamma_2]\!]};\Phi^{-1}(f_M).$$

The map h_2 is reformatted using the naturality of d^L and of δ:

$$\begin{aligned}
h_2 &= \delta_{!\Delta',!\Gamma_1',!\Gamma_2'};d^L_{!\Delta',!\Gamma_1',!\Gamma_1'};L\Phi^{-1}(g_2);I_{!(A\multimap B),!^n(A\multimap B)};\eta_{!^n(A\multimap B)}\\
&= \delta_{!\Delta',!\Gamma_1',!\Gamma_2'};d^L_{!\Delta',!\Gamma_1',!\Gamma_1'};L(f_V\otimes_{!\Delta'} id_{!\Delta',\Gamma_2});L\sigma_{[\![D]\!],[\![!\Delta',\Gamma_2]\!]};L\Phi^{-1}(f_M);\\
&\quad I_{!(A\multimap B),!^n(A\multimap B)};\eta_{!^n(A\multimap B)}\\
&= (f_V\otimes_{!\Delta'} id_{!\Delta',\Gamma_2});\sigma_{[\![D]\!],[\![!\Delta',\Gamma_2]\!]};\delta_{!\Delta',!\Gamma_2',y:!D'};d^L_{!\Delta',!\Gamma_2',y:!D'};L\Phi^{-1}(f_M);\\
&\quad I_{!(A\multimap B),!^n(A\multimap B)};\eta_{!^n(A\multimap B)}.
\end{aligned}$$

Note that g_1 is of the form $g_1';\eta_{!^n(A\multimap B)}$. Again using Equation (5.4.4):

$$\begin{aligned}
h_1 &= ((f_V;\eta_D)\otimes_{!\Delta'} id_{!\Delta',\Gamma_2});\sigma_{T[\![D]\!],[\![!\Delta',\Gamma_2]\!]};t;g_1^*\\
&= (f_V\otimes_{!\Delta'} id_{!\Delta',\Gamma_2});\sigma_{[\![D]\!],[\![!\Delta',\Gamma_2]\!]};(id\otimes\eta_D);t;g_1^*\\
&= (f_V\otimes_{!\Delta'} id_{!\Delta',\Gamma_2});\sigma_{[\![D]\!],[\![!\Delta',\Gamma_2]\!]};\eta_{!\Delta',\Gamma_2,y:D};g_1^*\\
&= (f_V\otimes_{!\Delta'} id_{!\Delta',\Gamma_2});\sigma_{[\![D]\!],[\![!\Delta',\Gamma_2]\!]};\eta_{!\Delta',\Gamma_2,y:D};(g_1';\eta)^*\\
&= (f_V\otimes_{!\Delta'} id_{!\Delta',\Gamma_2});\sigma_{[\![D]\!],[\![!\Delta',\Gamma_2]\!]};\eta_{!\Delta',\Gamma_2,y:D};T(g_1')\\
&= (f_V\otimes_{!\Delta'} id_{!\Delta',\Gamma_2});\sigma_{[\![D]\!],[\![!\Delta',\Gamma_2]\!]};g_1';\eta_{!^n(A\multimap B)}\\
&= (f_V\otimes_{!\Delta'} id_{!\Delta',\Gamma_2});\sigma_{[\![D]\!],[\![!\Delta',\Gamma_2]\!]};\delta_{!\Delta',!\Gamma_2',y:!D'};d^L_{!\Delta',!\Gamma_2',y:!D'};L\Phi^{-1}(f_M);\\
&\quad I_{!(A\multimap B),!^n(A\multimap B)};\eta_{!^n(A\multimap B)}\\
&= h_2.
\end{aligned}$$

Rule ($app_{<:}$). The setting is the following: $A <: B$, $\Delta = (!\Delta',\Gamma_1,\Gamma_2)$ and

$$!\Delta',\Gamma_1 \rhd N : B\multimap C, \qquad\qquad !\Delta',\Gamma_2 \rhd P : A.$$

We are considering the relation $\Delta \rhd N\{P <: B\} \approx_{ax} \{N <: A\multimap C\}P : C$.

In this situation, only the computational interpretation makes sense. If

$$f = [\![!\Delta',\Gamma_1 \rhd N : B\multimap C]\!]^c, \quad g = [\![!\Delta',\Gamma_2 \rhd P : A]\!]^c,$$

then, since $I_{B \multimap C, A \multimap C} = (I_{A,B}) \multimap C$, we have

$$\begin{aligned}
[\![\Delta \rhd N\{P <: B\} : B]\!]^c &= (f \otimes_{!\Delta'} (g; TI_{A,B})); \Psi_1; \varepsilon^*_{B,C} \\
&= (f \otimes_{!\Delta'} g); (id \otimes TI_{A,B}); \Psi_1; \varepsilon^*_{B,C} \\
&= (f \otimes_{!\Delta'} g); \Psi_1; ((id \otimes I_{A,B}); \varepsilon_{B,C})^*,
\end{aligned}$$

and

$$\begin{aligned}
[\![\Delta \rhd \{N <: A \multimap C\}P : C]\!]^c &= ((f; TI_{B \multimap C, A \multimap C}) \otimes_{!\Delta'} g); \Psi_1; \varepsilon^*_{A,C} \\
&= (f \otimes_{!\Delta'} g); (TI_{B \multimap C, A \multimap C} \otimes id); \Psi_1; \varepsilon^*_{A,C} \\
&= (f \otimes_{!\Delta'} g); \Psi_1; ((I_{B \multimap C, A \multimap C} \otimes id); \varepsilon_{A,C})^* \\
&= (f \otimes_{!\Delta'} g); \Psi_1; (((I_{A,B} \multimap C) \otimes id); \varepsilon_{A,C})^*.
\end{aligned}$$

The two denotations are equal by naturality of Φ. Indeed, since

$$\begin{tikzcd}
\mathcal{C}(A \multimap C, A \multimap C) \arrow[r, "\Phi"] \arrow[d, "(I_{A,B} \multimap C);-"'] & \mathcal{C}((A \multimap C) \otimes A, TC) \arrow[d, "((I_{A,B} \multimap C) \otimes A);-"] \\
\mathcal{C}(B \multimap C, A \multimap C) \arrow[r, "\Phi"'] & \mathcal{C}((B \multimap C) \otimes A, TC),
\end{tikzcd}$$

we have $\Phi(I_{A,B} \multimap C) = ((I_{A,B} \multimap C) \otimes A); \varepsilon_{A,C}$. Since

$$\begin{tikzcd}
\mathcal{C}(B \multimap C, B \multimap C) \arrow[r, "\Phi"] \arrow[d, "-;(I_{A,B} \multimap C)"'] & \mathcal{C}((B \multimap C) \otimes B, TC) \arrow[d, "((B \multimap C) \otimes I_{A,B});-"] \\
\mathcal{C}(B \multimap C, A \multimap C) \arrow[r, "\Phi"'] & \mathcal{C}((B \multimap C) \otimes A, TC),
\end{tikzcd}$$

we have $\Phi(I_{A,B} \multimap C) = ((B \multimap C) \otimes I_{A,B}); \varepsilon_{B,C}$. Thus,

$$((B \multimap C) \otimes I_{A,B}); \varepsilon_{B,C} = ((I_{A,B} \multimap C) \otimes A); \varepsilon_{A,C}.$$

Rule ($let^{\otimes}_{<:}$). In this case, $A <: A'$, $B <: B'$, and $n' = 0$ if $n = 0$. We have $\Delta = (!\Delta', \Gamma_1, \Gamma_2)$ with

$$!\Delta', \Gamma_1 \rhd N : !^n(A \otimes B), \qquad !\Delta', \Gamma_2, x : !^{n'}A', y : !^{n'}B' \rhd P : E,$$

and we consider the following relation:

$$\Delta \rhd \Big(let\ \langle x^{A'}, y^{B'}\rangle^{n'} = \{N <: !^{n'}(A' \otimes B')\}\ in\ P\Big) \approx_{ax} \big(let\ \langle x^A, y^B\rangle^n = N\ in\ P\big) : E.$$

By α-equivalence, one can assume that x, y do not occur in N.
Let f, g, h_1 and h_2 be the following maps:

$$\begin{aligned}
f &= [\![!\Delta', \Gamma_1 \rhd N : !^n(A \otimes B)]\!]^c, \\
g &= [\![!\Delta', \Gamma_2, x : !^{n'}A', y : !^{n'}B' \rhd P : E]\!]^c, \\
h_1 &= [\![!\Delta', \Gamma_1, \Gamma_2 \rhd let\ \langle x^{A'}, y^{B'}\rangle^{n'} = \{N <: !^{n'}(A' \otimes B')\}\ in\ P : E]\!]^c, \\
h_2 &= [\![!\Delta', \Gamma_1, \Gamma_2 \rhd let\ \langle x^A, y^B\rangle^n = N\ in\ P : E]\!]^c.
\end{aligned}$$

By definition, we have

$$\begin{aligned}
h_1 &= ((f; I_{!^n(A\otimes B),!^{n'}(A'\otimes B')}) \otimes_{!\Delta'} id_{!\Delta',\Gamma_2}); (T(d^{L^{n'}}_{A',B'})^{-1} \otimes id);\\
&\quad \sigma_{T[\![x:!^{n'}A',y:!^{n'}B']\!],[\![!\Delta',\Gamma_2]\!]}; t; g^*,\\
h_2 &= (f \otimes_{!\Delta'} id_{!\Delta',\Gamma_2}); (T(d^{L^n}_{A,B})^{-1} \otimes id);\\
&\quad \sigma_{T[\![x:!^nA,y:!^nB]\!],[\![!\Delta',\Gamma_2]\!]}; t; (I_{(!\Delta',\Gamma_2,x:!^nA,y:!^nB),(!\Delta',\Gamma_2,x:!^{n'}A',y:!^{n'}B')}; g)^*.
\end{aligned}$$

These two maps are the same since

$$I_{!^n(A\otimes B),!^{n'}(A'\otimes B')} = L^n I_{A,A'} \otimes L^{n'} I_{B,B'}$$

and

$$I_{(!\Delta',\Gamma_2,x:!^nA,y:!^nB),(!\Delta',\Gamma_2,x:!^{n'}A',y:!^{n'}B')} = id_{!\Delta',\Gamma_2} \otimes L^n I_{A,A'} \otimes L^{n'} I_{B,B'}.$$

For the same reasons the value interpretation of the two terms in relation are the same.

This ends the list of possible cases. Theorem 11.2.5 is then verified. □

Corollary 11.2.6. *If Erase(M) = Erase(M') and if $\Delta \rhd M, M' : A$ are valid typing judgements, then $[\![M]\!]^c = [\![M']\!]^c$ (and $[\![M]\!]^v = [\![M']\!]^v$ if M and M' are values).*

Proof. Corollary of Theorems 9.2.7 and 11.2.5. □

11.3 Completeness

Recall the category $\mathcal{C}_\lambda$ from Section 9.3. Since the category $\mathcal{C}_\lambda$ is a linear category for duplication, one can interpret the language in it. This section states that the defined lambda calculus is an *internal language* of linear categories for duplication.

Since the category $\mathcal{C}_\lambda$ is a monoidal category, one can w.l.o.g. generalize the notion of pairing to finite tensor products of terms.

Definition 11.3.1. We define the following shortcut notations for terms:

$$\langle\rangle^n = *^n, \qquad \langle x^A\rangle^n = x^{!^nA}, \qquad \langle x_1, x_2 \ldots\rangle^n = \langle x_1, \langle x_2 \ldots\rangle^0\rangle^n,$$

$$\begin{aligned}
(let\ \langle\rangle^n = M\ in\ N) &= (let\ * = \{M <: \top\}\ in\ v\ N),\\
(let\ \langle x^A\rangle^n = M\ in\ N) &= (let\ x^{!^nA} = M\ in\ N),\\
(let\ \langle x_1, x_2 \ldots\rangle^n = M\ in\ N) &= (let\ \langle x_1, z_2\rangle^n = M\ in\ let\ \langle x_2 \ldots\rangle^n = z_2\ in\ N),
\end{aligned}$$

and for types:

$$A_1 \otimes A_2 \otimes \cdots \otimes A_n = A_1 \otimes (A_2 \otimes (\cdots \otimes A_n) \cdots).$$

Lemma 11.3.2. *The following derived typing rules are correct:*

$$\frac{\begin{array}{l}!\Delta, \Gamma_1 \rhd M : !^n(A_1 \otimes \cdots \otimes A_k)\\ !\Delta, \Gamma_2, x_1 : !^nA_1, \ldots, x_k : !^nA_k \rhd N : B\end{array}}{!\Delta, \Gamma_1, \Gamma_2 \rhd let\ \langle x_1, \ldots, x_k\rangle^n = M\ in\ N : B,} \tag{11.3.1}$$

and

$$\frac{\begin{array}{l}!\Delta,\Gamma_1 \rhd M_1 : !^n A_1\\ !\Delta,\Gamma_2 \rhd M_2 : !^n A_2\\ \dots\\ !\Delta,\Gamma_k \rhd M_k : !^n A_k\end{array}}{!\Delta,\Gamma_1,\dots,\Gamma_k \rhd \langle M_1,\dots M_k\rangle^n : !^n(A_1\otimes\dots\otimes A_k).} \tag{11.3.2}$$

Proof. Proof by induction on k. □

Lemma 11.3.3. *In the setting of Equation 11.3.2 in Lemma 11.3.2, if for all i we have $f_i = [\![!\Delta,\Gamma_i \rhd M_i : !^n A_i]\!]^c$, then*

$$[\![\Gamma_1,\dots,\Gamma_k \rhd \langle M_1,\dots M_k\rangle^n : !^n(A_1\otimes\dots\otimes A_k)]\!]^c = (f_1\otimes_{!\Delta}\dots\otimes_{!\Delta} f_k);\Psi_1;Td^{L^n}_{A_1,\dots,A_k}.$$

In the case where all the M_i are extended values, if f'_i is $f'_i = [\![!\Delta,\Gamma_i \rhd M_i : !^n A_i]\!]^v$, then

$$[\![\Gamma_1,\dots,\Gamma_k \rhd \langle M_1,\dots M_k\rangle^n : !^n(A_1\otimes\dots\otimes A_k)]\!]^c = (f_1\otimes_{!\Delta}\dots\otimes_{!\Delta} f_k);d^{L^n}_{A_1,\dots,A_k}.$$

Proof. Proof by induction on k. □

Lemma 11.3.4. *In the setting of Equation 11.3.1 in Lemma 11.3.2, if*

$$\begin{aligned} f &= [\![!\Delta,\Gamma_1 \rhd M : !^n(A_1\otimes\dots\otimes A_k)]\!]^c,\\ g &= [\![!\Delta,\Gamma_2, x_1 : !^n A_1,\dots,x_k : !^n A_k \rhd N : B]\!]^c, \end{aligned}$$

then

$$[\![!\Delta,\Gamma_1,\Gamma_2 \rhd \mathit{let}\ \langle x_1,\dots,x_k\rangle^n = M\ \mathit{in}\ N : B]\!]^c = (f\otimes_{!\Delta} id);(T(d^{L^n}_{A_1,\dots,A_n})^{-1}\otimes id);\sigma;t;g^*.$$

In the case where M and N are extended values, if

$$\begin{aligned} f &= [\![!\Delta,\Gamma_1 \rhd M : !^n(A_1\otimes\dots\otimes A_k)]\!]^v,\\ g &= [\![!\Delta,\Gamma_2, x_1 : !^n A_1,\dots,x_k : !^n A_k \rhd N : B]\!]^v, \end{aligned}$$

then

$$[\![!\Delta,\Gamma_1,\Gamma_2 \rhd \mathit{let}\ \langle x_1,\dots,x_k\rangle^n = M\ \mathit{in}\ N : B]\!]^v = (f\otimes_{!\Delta} id);(T(d^{L^n}_{A_1,\dots,A_n})^{-1}\otimes id);\sigma;g.$$

Proof. Proof by induction on n. □

Lemma 11.3.5. *In $\mathcal{C}_\lambda$, a valid typing judgement $x_1 : A_1,\dots x_n : A_n \rhd M : B$ has for computational denotation $(t : A_1\otimes\dots\otimes A_n \rhd \mathit{let}\ \langle x_1,\dots x_n\rangle = t\ \mathit{in}\ \lambda*.M : \top\multimap B)$. If $M = V$ is a value, the value interpretation is $(t : A_1\otimes\dots\otimes A_n \rhd \mathit{let}\ \langle x_1,\dots x_n\rangle = t\ \mathit{in}\ V : B)$.*

Proof. Proof by structural induction on M, using Lemmas 11.3.3 and 11.3.4. For each case, one writes the categorical morphism corresponding to the term and one interprets it in $\mathcal{C}_\lambda$. □

Theorem 11.3.6 (Completeness)**.** *In $\mathcal{C}_\lambda$, Θ being the identity, one has $[\![x : A \rhd M : B]\!]^c_\Theta \approx_{ax} (x : A \rhd \lambda*.M : \top\multimap B)$ and $[\![x : A \rhd V : B]\!]^v_\Theta \approx_{ax} (x : A \rhd V : B)$.*

Proof. Corollary of Lemma 11.3.5. □

11.4 Towards a Denotational Model

In Chapters 8-11, we developed the categorical requirements for modeling a generic call-by-value linear lambda calculus, i.e., its type system (which includes subtyping) and equational laws. We have not yet specialized the language to a particular set of built-in operators, for example, those that are required for quantum computation.

However, since the quantum lambda calculus of Chapter 6 is the main motivation behind our work, we will comment very briefly on what additional properties would be required to interpret its primitives. The quantum lambda calculus of Chapter 6 is obtained by instantiating and extending the call-by-value language of Chapter 9 with the following primitive types, constants, and operations:

Types:	bit, $qbit$
Constants:	$0 : !bit$, $1 : !bit$
	$new : !(bit \multimap qbit)$, $U : !(qbit^n \multimap qbit^n)$, $meas : !(qbit \multimap !bit)$
Operations:	$\dfrac{\Gamma_1, !\Delta \rhd P : bit \quad \Gamma_2, !\Delta \rhd M : A \quad \Gamma_2, !\Delta \rhd N : A}{\Gamma_1, \Gamma_2, !\Delta \rhd if\, P\ then\ M\ else\ N : A}\ (if)$

In the intended semantics, $!bit \cong bit$, while $!qbit$ is empty. *new* creates a new qubit, and *meas* measures a qubit.

The denotational semantics of these operations is already well-understood (see Section 4.3) in the absence of higher-order types. They can all be interpreted in the category $\mathbf{Q}$ of superoperators described in Section 4.3. The part that is not yet well-understood is how these features interact with higher-order types.

In light of our present work, we can conclude that a model of the quantum lambda calculus consists of a linear category for duplication $(\mathcal{C}, L, T, \multimap)$, such that the associated category of computations $\mathcal{C}_T$ contains the category $\mathbf{Q}$ as a full monoidal subcategory. To construct an actual instance of such a model is still an open problem.

Bibliography

Samson Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 111(1-2):3–57, April 1993.

Samson Abramsky and Bob Coecke. A categorical semantics of quantum protocols. In *Proceedings of the 19th Symposium on Logic in Computer Science, LICS'04*, pages 415–425, Turku, Finland, July 2004. IEEE, IEEE Computer Society Press.

Dorit Aharonov. Quantum computation. In Dietrich Stauffer, editor, *Annual Reviews of Computational Physics VI*. World Scientific, 1999. Also available on arXiv as quant-ph/9812037.

Thorsten Altenkirch and Jonathan Grattage. QML: Quantum data and control. Draft, February 2005a.

Thorsten Altenkirch and Jonathan Grattage. A functional quantum programming language. In Prakash Panangaden, editor, *Proceedings of the 20th Symposium on Logic in Computer Science, LICS'05*, pages 249–258, Chicago, Illinois, US., June 2005b. IEEE, IEEE Computer Society Press.

Andrew G. Barber. *Linear Type Theories, Semantics and Action Calculi*. PhD thesis, Laboratory for Foundation of Computer Sciences, Edinburgh University, Scotland, UK., 1997. Also available as report ECS-LFCS-97-371.

Henk P. Barendregt. *The Lambda-Calculus, its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundation of Mathematics*. North Holland, 1984.

Michael Barr. Accessible categories and models of linear logic. *Journal of Pure and Applied Algebra*, 69:219–232, 1990.

John S. Bell. On the Einstein Podolsky Rosen paradox. *Physics*, 1:195–200, 1964.

Paul Benioff. The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. *Journal of Statistical Physics*, 22(5): 563–591, May 1980.

Nick Benton. A mixed linear and non-linear logic: Proofs, terms and models (extended abstract). In Leszek Pacholski and Jerzy Tiuryn, editors, *Computer Science Logic, Eighth International Workshop, CSL'94, Selected Papers*, volume 933 of *Lecture Notes in Computer Science*, pages 121–135, Kazimierz, Poland, September 1994. European Association for Computer Science Logic, Springer Verlag.

Nick Benton and Philip Wadler. Linear logic, monads and the lambda calculus. In *Proceedings of the 11th Symposium on Logic in Computer Science, LICS'96*, pages 420–431, New Brunswick, New Jersey, US., July 1996. IEEE, IEEE Computer Society Press.

Nick Benton, Gavin Bierman, Martin Hyland, and Valeria C. V. de Paiva. Linear lambda-calculus and categorical models revisited. In *Computer Science Logic, Sixth International Workshop, CSL'92, Selected Papers*, volume 702 of *Lecture Notes in Computer Science*, San Miniato, Italy, September 1992. European Association for Computer Science Logic, Springer Verlag.

Nick Benton, Gavin Bierman, Valeria C. V. de Paiva, and Martin Hyland. A term calculus for intuitionistic linear logic. In Marc Bezem and Jan Friso Groote, editors, *Proceedings of the International Conference on Typed Lambda Calculi and Applications, TLCA'93*, volume 664 of *Lecture Notes in Computer Science*, pages 75–90, Ultrech, Netherlands, March 1993. Springer Verlag.

Stefano Bettelli, Tommaso Calarco, and Luciano Serafini. Toward an architecture for quantum programming. *The European Physical Journal D - Atomic, Molecular and Optical Physics*, 25(2): 181–200, August 2003. Also found on arXiv:cs.PL/0103009.

Gavin Bierman. *On Intuitionistic Linear Logic*. PhD thesis, Computer Science department, Cambridge University, England, UK., December 1993. Available as Technical Report 346, August 1994.

Gavin Bierman. What is a categorical model of intuitionistic linear logic. In Mariangiola Dezani-Ciancaglini and Gordon D. Plotkin, editors, *Proceedings of the Second International Conference on Typed Lambda Calculi and Applications, TLCA'95*, volume 902 of *Lecture Notes in Computer Science*, pages 78–93, Edinburgh, Scotland, UK., April 1995. Springer Verlag.

Olivier Bournez and Mathieu Hoyrup. Rewriting logic and probabilities. In *14th International Conference on Rewriting Techniques and Applications*, pages 61–75, Valencia, Spain, june 2003.

Val Breazu-Tannen, Thierry Coquand, Carl A. Gunter, and Andre Scedrov. Inheritance and explicit coercion. In *Proceedings of the Fourth Symposium on Logic in Computer Science, LICS'89* IEE (1989), pages 112–129.

Hans J. Briegel and Robert Raussendorf. Computational model for the one-way quantum computer: Concepts and summary. To be found on arXiv: quant-ph/0207183, July 2002.

Man-Duen Choi. Completely positive linear maps on complex matrices. *Linear Algebra and its Applications*, 10(3):285–290, June 1975.

Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2):345–363, 1936.

Bob Coecke. Quantum information-flow, concretely, abstractly. In Peter Selinger, editor, *Proceedings of the Second International Workshop on Quantum Programming Languages*, volume 33 of *TUCS General Publication*, pages 57–73, Turku, Finland, July 2004. TUCS.

Bob Coecke and Éric O. Paquette. POVMs and Naimark's theorem without sums. In Peter Selinger, editor, *Preliminary Proceedings of the Fourth International Workshop on Quantum Programming Languages*, Oxford, UK., July 2006. To appear in ENTCS.

Bob Coecke and Dusko Pavlovic. Quantum measurements without sums. In Goong Chen, Louis Kauffman, and Samuel J. Lomonaco, editors, *Mathematics of Quantum Computation and Quantum Technology*, pages 559–592. Taylor and Francis CRC Press, 2007. Also arXiv:quant-ph/0608035.

Vincent Danos and Russ S. Harmer. Probabilistic game semantics. *ACM Transactional on Computational Logic*, 3(3):359–382, July 2002.

Vincent Danos, Elham Kashefi, and Prakash Panangaden. The measurement calculus. *Journal of the ACM*, 54(2), 2007.

David Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 400 (1818):97–117, July 1985.

David Deutsch. Quantum computational networks. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 425(1868):73–90, 1989.

David S. Dummit and Richard M. Foote. *Abstract Algebra*. John Wiley and Sons, second edition, 1999. ISBN 0-471-36857-1.

Samuel Eilenberg and Gregory M. Kelly. Closed categories. In Samuel Eilenberg, David K. Harrison, Saunders Mac Lane, and Helmut Röhrl, editors, *Proceedings of the Conference on Categorical Algebra, La Jolla 1965*, pages 421–562, University of California, San Diego, California, US., June 1965. Springer Verlag.

Albert Einstein, Boris Podolsky, and Nathan Rosen. Can quantum-mechanical description of physical reality be considered complete? *Phys. Rev.*, 47(10):777–780, May 1935.

Andrzej Filinski. Normalization by evaluation for the computational lambda-calculus. In Samson Abramsky, editor, *Proceedings of the Fifth International Conference on Typed Lambda Calculi and Applications, TLCA'01*, volume 2044 of *Lecture Notes in Computer Science*, pages 151–165, Krakow, Poland, May 2001. Springer Verlag.

Simon J. Gay. Quantum programming languages: Survey and bibliography. *Mathematical Structures in Computer Science*, 16:581–600, 2006.

Simon J. Gay and Rajagopal Nagarajan. Communicating quantum processes. In *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 145–157, Long Beach, California, US., January 2005. ACM, ACM Press. Preliminary version in Selinger (2004a); also arXiv:quant-ph/0409052.

Jean-Yves Girard. Between logic and quantic: a tract. In Thomas Ehrhard, Jean-Yves Girard, Paul Ruet, and Philip Scott, editors, *Linear Logic in Computer Science*, volume 316 of *London Mathematical Society Lecture Note Series*, chapter 10. Cambridge, UK., 2004.

Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987.

Jean-Yves Girard, editor. *La Machine de Turing*, volume 131 of *Points Sciences*. Editions du Seuil, 1995. Contains (Turing, 1936) and (Turing, 1950) in integrality, with comments.

Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts In Theoretical Computer Science*. Cambridge University Press, 1990.

Martin Hyland and Andrea Schalk. Glueing and orthogonality for models of linear logic. *Theoretical Computer Science*, 294(1–2):183–231, February 2003.

Proceedings of the Fourth Symposium on Logic in Computer Science, LICS'89, Pacific Grove, California, US., June 1989. IEEE, IEEE Computer Society Press.

Gregory M. Kelly and Miguel L. Laplaza. Coherence for compact closed categorie. *Journal of Pure and Applied Algebra*, 19:193–213, December 1980.

Stephen C. Kleene. A theory of positive integers in formal logic, part I. *American Journal of Mathematics*, 57(1):153–173, January 1935a.

Stephen C. Kleene. A theory of positive integers in formal logic, part II. *American Journal of Mathematics*, 57(2):219–244, April 1935b.

Emanuel H. Knill. Conventions for quantum pseudocode. Technical Report LAUR-96-2724, Los Alamos National Laboratory, Los Alamos, New Mexico, US., 1996.

Yves Lafont. The linear abstract machine. *Theoretical Computer Science*, 59:157–180, 1988a.

Yves Lafont. *Logiques, Catégories et Machines*. PhD thesis, Université Paris 7, 1988b.

Marie Lalire and Philippe Jorrand. A process algebraic approach to concurrent and distributed computation: operational semantics. In Peter Selinger, editor, *Proceedings of the Second International Workshop on Quantum Programming Languages*, volume 33 of *TUCS General Publication*, pages 109–126, Turku, Finland, July 2004. TUCS.

Joachim Lambek and Philip Scott. *Introduction to Higher Order Categorical Logic*, volume 7 of *Cambridge studies in advanced mathematics*. Cambridge University Press, 1989.

Serge Lang. *Algebra*, volume 211 of *Graduate Texts in Mathematics*. Springer Verlag, third edition, 2002.

Serge Lang. *Real and Functional Analysis*, volume 142 of *Graduate Texts in Mathematics*. Springer Verlag, third edition, 1993.

Saunders Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer Verlag, 1998.

Paola Maneggia. *Models of Linear Polymorphisms*. PhD thesis, University of Birmingham, 2004.

Ernest G. Manes. *Algebraic Theories*, volume 26 of *Graduate Texts in Mathematics*. Springer Verlag, 1976.

Paul-André Melliès. Categorical models of linear logic revisited. Preprint, 2002.

Eugenio Moggi. Computational lambda-calculus and monads. Technical Report ECS-LFCS-88-66, Laboratory for Foundation of Computer Sciences, Edinburgh University, Scotland, UK., 1988.

Eugenio Moggi. Computational lambda-calculus and monads. In *Proceedings of the Fourth Symposium on Logic in Computer Science, LICS'89* IEE (1989), pages 14–23.

Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, July 1991.

Peter Naur et al. Report on the algorithmic language ALGOL 60. *Communications of the ACM*, 3 (5):299–314, May 1960.

Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2002.

Harumichi Nishimura and Masanao Ozawa. Computational complexity of uniform quantum circuit families and quantum turing machines. *Theoretical Computer Science*, 276(1–2):147–181, April 2002.

Atsushi Ohori. A Curry-Howard isomorphism for compilation and program execution. In Jean-Yves Girard, editor, *Proceedings of the Fourth International Conference on Typed Lambda Calculi and Applications, TLCA'99*, volume 1581 of *Lecture Notes in Computer Science*, pages 280–294, L'Aquilla, Italy, April 1999. Springer Verlag.

Bernhard Ömer. Quantum programming in QCL. Master's thesis, Institute of Information Systems, Technical University of Vienna, 2000.

Benjamin C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

John Power and Hiroshi Watanabe. Combining a monad and a comonad. *Theoretical Computer Science*, 280(1–2):137–162, 2002.

John Preskill. Lecture notes for quantum computation. Available[1] online, 1999.

Robert Raussendorf and Hans J. Briegel. A one-way quantum computer. *Phys. Rev. Lett.*, 86(22): 5188–5191, 2001.

Jeff W. Sanders and Paolo Zuliani. Quantum programming. In Roland Backhouse and Jose N. Oliveira, editors, *Proceedings of the Fifth International Conference on Mathematics of Program Construction*, volume 1837 of *Lecture Notes in Computer Science*, pages 80–99, Ponte de Lima, Portugal, July 2000. Springer Verlag.

Andrea Schalk. What is a model for linear logic. Unpublished manuscript, 2004.

Robert A.G. Seely. Linear logic, *-autonomous categories and cofree coalgebras. In John W. Gray and Andre Scedrov, editors, *Categories in Computer Science and Logic (Boulder, CO, 1987)*, volume 92 of *Contemporary Mathematics*, pages 371–382. Amer. Math. Soc., 1989.

Peter Selinger, editor. *Proceedings of the Second International Workshop on Quantum Programming Languages*, volume 33 of *TUCS General Publication*, Turku, Finland, July 2004a. TUCS.

Peter Selinger. Control categories and duality: on the categorical semantics of the lambda-mu calculus. *Mathematical Structures in Computer Science*, 11(2):207–260, 2001.

Peter Selinger. Towards a quantum programming language. *Mathematical Structures in Computer Science*, 14(4):527–586, August 2004b.

Peter Selinger. Towards a semantics for higher-order quantum computation. In Peter Selinger, editor, *Proceedings of the Second International Workshop on Quantum Programming Languages*, volume 33 of *TUCS General Publication*, pages 127–143, Turku, Finland, July 2004c. TUCS.

Peter Selinger. Dagger compact closed categories and completely positive maps. In Peter Selinger, editor, *Proceedings of the Third International Workshop on Quantum Programming Languages*, Chicago, Illinois, US., July 2005.

Peter Selinger and Benoît Valiron. A lambda calculus for quantum computation with classical control. *Mathematical Structures in Computer Science*, 16:527–552, 2006a.

Peter Selinger and Benoît Valiron. A lambda calculus for quantum computation with classical control. In Pawel Urzyczyn, editor, *Proceedings of the Seventh International Conference on Typed Lambda Calculi and Applications, TLCA'05*, volume 3461 of *Lecture Notes in Computer Science*, pages 354–368, Nara, Japan, April 2005. Springer Verlag. Journal version appeared in (Selinger and Valiron, 2006a).

[1] http://www.theory.caltech.edu/people/preskill/ph229/

Peter Selinger and Benoît Valiron. On a fully abstract model for a quantum linear functional language. In Peter Selinger, editor, *Preliminary Proceedings of the Fourth International Workshop on Quantum Programming Languages*, pages 103–115, Oxford, UK., July 2006b. To appear in ENTCS. Available[2] online.

Peter Selinger and Benoît Valiron. Linear-non-linear model for a computational call-by-value lambda calculus. In *Proceedings of the Eleventh International Conference on Foundations of Software Science and Computation Structures (FOSSACS 2008)*, volume 4962 of *Lecture Notes in Computer Science*, pages 81–96, Budapest, Hungaria, April 2008.

Koushik Sen, Nirman Kumar, Jose Meseguer, and Gul Agha. Probabilistic rewrite theories: Unifying models, logics and tools. Technical Report UIUCDCS-R-2003-2347, University of Illinois, 2003.

Peter W. Shor. Algorithms for quantum computation: Discrete log and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, Santa Fe, New Mexico, US., November 1994. IEEE, IEEE Computer Society Press.

Paul Taylor. *Practical Foundations of Mathematics*, volume 59 of *Cambridge studies in advanced mathematics*. Cambridge University Press, 1999.

Anne S. Troelstra. *Lectures in Linear Logic*, volume 29 of *CSLI Lecture Notes*. Center for the Study of Language and Information, Stanford, California, US., 1992.

Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, Series 2*, 42, 1936. Can be found integrally, and commented, in (Girard, 1995).

Alan M. Turing. Computing machinery and intelligence. *Journal of the Mind Association*, 59(236): 433–460, 1950. Can be found integrally, and commented, in (Girard, 1995).

Benoît Valiron. A functional programming language for quantum computation with classical control. Master's thesis, University of Ottawa, 2004a.

Benoît Valiron. Quantum typing. In Peter Selinger, editor, *Proceedings of the Second International Workshop on Quantum Programming Languages*, volume 33 of *TUCS General Publication*, Turku, Finland, July 2004b. TUCS.

André van Tonder. Quantum computation, categorical semantics and linear logic. On arXiv: quant-ph/0312174, 2003.

André van Tonder. A lambda calculus for quantum computation. *SIAM Journal of Computing*, 33 (5):1109–1135, 2004. available on arXiv as quant-ph/0307150.

Philip Wadler. There is no substitute for linear logic. Manuscript, presented at Mathematical Foundations of Programming Semantics: Eighth International Workshop, Oxford, UK., 1992.

William K. Wootters and Wojciech H. Zurek. A single quantum cannot be cloned. *Nature*, 299: 802–803, October 1982.

[2] http://www.mathstat.dal.ca/∼selinger/qpl2006/proceedings.html

Index

$\Upsilon^A_{x\Downarrow}$, 91
$\Upsilon^A_{x\Uparrow}$, 91

adjunction, 8
    composition, 9
    left/right adjoint, 8
    unit,counit, 8
alpha-equivalence, 42
arrow, 6
axiomatic equivalence, 45, 94, 122

B_σ, 37
basis along x,y,z, 23
Bell's algorithm, 27
Bell's inequalities, 27
Bloch sphere, 22, 27
BNF, 41
bound variable, 42, 55
bra, 23

$\mathcal{C}_\lambda$, 127
call-by-name, 49, 57
call-by-value, 49, 57, 75
canonical first-order representation, 91
Cat, 7
categorical semantics, 173
category, 6
    cartesian category, 10
    cartesian closed, 47
    equivalence of, 8
    functor category, 8
    linear, 53
    linear-non-linear, 53
    opposite category, 7
    product category, 7
    small,locally small, 6
    trivial category, 6
characteristic matrix, 20, 38
Church-Rosser theorem, 43
Church-style, 45
classical object, 99
classical objects, 37
co-Eilenberg-Moore category, 12
co-Kleisli category, 11
coalgebra, 12
coherence map, 14
commutation of diagrams, 8
commutative comonoid, 13
comonad, 11
    idempotent, 100
        coherence properties, 101
completely positive map, 20
completeness, 48, 198
composition, 6
computation
    category of, 50
concurrent quantum computation, 34
consistency, 59
context, 43
contraction, 20, 39, 51
controlled gate, 24
controlled-not gate, 24
coproduct, 10
    binary, 10
copying, 26, 99
CPM, 37
Curry-Howard isomorphism, 46
Curry-style, 45

dagger category, 36
    compact-closed, 36
degree of
    a term, 160
    a type, 160
denotation, 84, 85
denotational equivalence, 96
denotational semantics, 83
dense-coding algorithm, 30, 65
density matrix, 18
Deutsch Algorithm, 64
Deutsch algorithm, 31
diagonal structure, 14

Dirac convention, 23
dummy variable, 114

Einstein locality criterion, 27
entanglement, 26
equational logic, 122
erasure operation, 109
error state, 59
evaluation strategy, 56
exchange gate, 24

faithful, 7
fixed point, 76
free coalgebra, 12
free variable, 42, 55, 71
full, 7
full abstraction, 96
full embedding, 7
fullness, 89
functional, 18
functor, 7
    diagonal functor, 7
    identity functor, 7
    terminal functor, 7

H, 24
Hadamard gate, 24
height of a term, 159
hermitian form, 17
hidden variable theory, 27
Hilbert space, 18
homset, 6

identity, 6
induced tensor, 16
initial object, 9
intermediate neutral value, 152
interpretation, 173
intuitionistic Linear Logic, 51
intuitionistic logic, 46
isomorphism, 6

ket, 23
Kleisli category, 10
Kleisli exponential, 50
Kleisli triple, 10
Kraus Representation Theorem, 20, 39

lambda calculus, 41
lambda-term, 54, 70, 109
    closed, 42
    combinator, 42
    indexed, 109
    open, 42
    operator, 42
    variables, 41
length, 79
linear category, 51
linear category for duplication, 107, 146
linear exponential comonad, 52
linear lambda calculus, 70
linear logic, 51
Löwner order, 20, 38

map, 6
measure of a term, 155
measurement operator, 39
measurements, 23
mixed state, 26
mixed strategy, 57
monad, 10
    commutative strong, 50
    computational, 50
    strong, 49
monoidal category, 13
    strict, 13
    symmetric, 13
monoidal comonad, 16
    symmetric, 16
monoidal functor, 14
    lax, symmetric, strong, 14
morphism, 6

Naimark Theorem, 21, 37
natural isomorphism, 7
natural transformation, 7
    composition, 8
    identity, 8
    monoidal, 16
$\mathbf{N}_C$, 24
neutral term, 148
no-cloning, 26
norm, 17
    induced norm, 18
normal form, 43
normalization, 79
    Theorem, 81
normed vector space, 18
not gate, 24
i-th number of a term, 162

objects, 6

observable, 23
operational context, 95
    formal, 95
    typed, 95
operational equivalence, 96
operational semantics, 75
operator, 18
    adjoint, 18
    hermitian, 18
    positive, 18
    self adjoint, 18

partial trace, 21, 39
Pauli matrices, 21
phase-shift gate, 24
Φ, 38
placeholder, 109
positive definite, 17
positive map, 19
probabilistic reduction system, 58
product, 9
    binary, 10
product category, 7
program, 62
progress, 45, 63, 78

Q, 38
QML, 33
QRAM model, 35
quantum array, 56
quantum bit, 23
quantum circuit, 25, 32
quantum closure, 56
    closed, 62
    well-typed, 62
quantum computation, 17
    measurement based, 34
    models of, 32
quantum flow-chart language, 37
quantum Turing machine, 33
quantum value state, 59
qubit, 23

ray, 21
reachability, 58
reduction rules, 43
    β-reduction, 43
    η-reduction, 43
reduction strategy, 49
rewrite system
    number one, 152
    number two, 157

safety properties, 62, 77
scalar product, 17
sesquilinear form, 17
Set, 6
side effect, 48
signature, 37
§-size, 91
small step semantics, 75
Solovay-Kitaev Theorem, 24
soundness, 48, 89, 96, 187
Spectral Theorem, 18
subject reduction, 44, 63, 77
substitution, 43, 55, 72, 117
substitution Lemma, 44, 62, 73, 86, 119
subtyping relation, 61, 100
superoperator, 20, 38

T-exponential, 50
teleportation algorithm, 28, 64
tensor product of vector spaces, 19
tensorial strength, 49
terminal object, 9
$\mathbf{tr}_n^k$, 21
$\mathbf{tr}_n$, 21
trace characteristic matrix tuple, 39
type casting, 115
type inference algorithm, 63
type system, 44, 60, 70, 108
typing context, 44, 110
typing derivation, 44
typing judgement, 44, 61, 110
    valid, 44
typing rules, 61, 110
typing tree, 44

U_C, 24
unitary, 18
unitary gates, 24

value, 59
    category of, 50
van Tonder's lambda calculus, 33

weakening, 51