ON A CATEGORICALLY SOUND QUANTUM PROGRAMMING LANGUAGE FOR CIRCUIT DESCRIPTION

by

Francisco Rios

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

at

Dalhousie University
Halifax, Nova Scotia
August 2021

Table of Contents

List of Tables

Abstract

This thesis contains contributions to the mathematical foundations of quantum programming languages.

The likely arrival of scalable quantum computers in the not so distant future has resulted in a flurry of activity in the development of quantum programming languages. As in classical computing, the transition from a description of a quantum algorithm found in the literature to a hardware-specific set of instructions run on a quantum device is a complex process, prone to errors. This issue is exacerbated in the quantum setting not only by the complexity of quantum algorithms but also by the fragility of quantum information, which renders ineffective some of the classical techniques used to debug programs.

In this thesis, we contribute to the solution of some of these issues. We introduce Proto-Quipper-M, a new quantum programming language designed to serve as a testbed for the research and development of sound mathematical semantics and reasoning techniques for quantum programs. We first present Proto-Quipper-M as a formalization of a fragment of Quipper, a high-level functional programming language for describing families of quantum circuits. In particular, we define Proto-Quipper-M as a simply-typed lambda calculus with a special type for quantum circuits and a strong type system designed to enforce linearity on quantum data, and thus prevent violations of the no-cloning property of quantum information. We endow Proto-Quipper-M with computational meaning via a big-step operational semantics and prove that the language is type-safe by showing that it enjoys the type-preservation and error-freeness properties. We also give Proto-Quipper-M a denotational semantics in a suitable class of monoidal categories and show that these categories give rise to linear-non-linear models in the sense of Benton, and thus models of intuitionistic linear logic. Finally, we crystallize the connection between the syntax and the semantics of the language by proving the soundness theorem for Proto-Quipper-M.

List of Abbreviations and Symbols Used

$\mathbb{N}$	The set of natural numbers $\{0, 1, 2, \ldots\}$, p.10.
$\mathbb{R}$	The set of real numbers, p.10.
$\mathbb{C}$	The set of complex numbers, p.10.
$\alpha^\dagger$	The complex conjugate of α, p.10.
$\langle \cdot , \cdot \rangle$	The inner product operation on a vector space, p.10.
$\mathscr{H}$	A Hilbert space, p.11.
$\mathscr{H}^*$	The dual of the Hilbert space $\mathscr{H}$, p.11.
$\lVert \cdot \rVert$	The norm of a scalar, vector, or linear operator, p.12.
$T^\dagger$	The adjoint of the linear operator T, p.12.
U^{-1}	The inverse of the linear operator U, p.12.
$\otimes$	The tensor product, p.15.
$\{\lvert 0\rangle, \lvert 1\rangle\}$	The computational basis of $\mathbb{C}^2$, p.17.
$\mathbf{C}(A, B)$	The class of morphisms with domain A and codomain B, p.24.
Set	The category of sets and functions, p.25.
Rel	The category of sets and relations, p.25.
Mon	The category of monoids and monoid homomorphisms, p.25.
FHilb	The category of finite-dimensional Hilbert spaces and linear transformations, p.25.
$\alpha : F \Rightarrow G$	A natural transformation from the functor F to the functor G, p.27.
$[\mathbf{C}, \mathbf{D}]$	The category of functors $F : \mathbf{C} \to \mathbf{D}$ and natural transformations, p.27.
$\widehat{\mathbf{C}}$	The category of presheaves on $\mathbf{C}$, p.28.
H_A	The contravariant representable hom-functor with representing object A, p.28.
$\mathcal{Y}$	The Yoneda embedding, p.29.
$\lim\limits_{\leftarrow \mathbf{I}} D$	The limit of a diagram $D : \mathbf{I} \to \mathbf{C}$, p.30.
$\lim\limits_{\rightarrow \mathbf{I}} D$	The colimit of a diagram $D : \mathbf{I} \to \mathbf{C}$, p.31.

$F \dashv G$	The functor $F : \mathbf{C} \to \mathbf{D}$ is a left adjoint to the functor $G : \mathbf{D} \to \mathbf{C}$, p.32.
SMC	Symmetric monoidal category, p.38.
SMCC	Symmetric monoidal closed category, p.39.
$B \multimap -$	A specified right adjoint to the endofunctor $- \otimes B$, p.39.
CCC	Cartesian closed category, p.39.
(F, m_0, m)	A monoidal functor F with structure morphisms m_0 and m, p.41.
LNL	Linear-non-linear model, p.44.
$\mathbf{Set}^{\mathbf{2}^{op}}$	The presheaf category over $\mathbf{2}$, p.46.
$[\![-]\!]_{\mathcal{W}}$	The interpretation function for wire types in $\mathcal{W}$, p.53.
$\mathbf{M}_{\mathcal{L}}$	The symmetric monoidal category of labeled circuits, p.54.
$\mathrm{Circ}(T, U)$	The type of circuits with inputs of type T and outputs of type U, p.55.
$FV(M)$	The set of free variables of the term M, p.57.
$M[N/x]$	The capture-avoiding substitution of N for x in M, p.59.
$\mathcal{D}$	A derivation, p.61.
$\Gamma; Q \vdash M : A$	A typing judgement with variable context Γ, label context Q, and term M of type A, p.61.
(C, M)	A configuration with labeled circuit C and term M, p.65.
$\Downarrow$	The evaluation relation of Proto-Quipper-M, p.65.
$Q \vdash (C, M) : A; Q'$	A configuration typing judgement with input labels Q, configuration (C, M), type A, and output labels Q', p.86.
$\mathbf{L}$	A category of denotations of Proto-Quipper-M, p.99.
$- \odot L$	The copower functor of L, p.100.
$\flat$	The representable functor $\mathbf{L}(I, -)$, p.101.
$!$	The "bang" modality of linear logic, the boxing comonad, p.108.
$[\![-]\!]_s$	The interpretation function for simple M-types, p.109.
$\vartriangle_{\Gamma_1 \cup \Gamma_2}$	The generalized duplication morphism on the context $\Gamma_1 \cup \Gamma_2$, p.116.

$\diamond_\Phi$ The discarding morphism on the parameter context Φ, p.117.

$\Phi, \mathcal{D}$ The weakening of derivation $\mathcal{D}$ by the parameter context Φ, p.131.

$[\![(C, M)]\!]$ The semantics in $\mathbf{L}$ of a well-typed configuration (C, M), p.152.

Acknowledgements

First of all, I would like to thank my advisor Peter Selinger not only for introducing me to the fascinating world of categorical semantics for quantum programming languages, but more importantly, for his guidance, support, and encouragement during the elaboration of this thesis. I would also like to extend my thanks to the members of my supervisory committee, Robert Paré and Julien Ross, for reading an earlier version of this document and providing valuable comments and corrections. Not to mention for the affable conversations, mathematical and otherwise, during my time at Dalhousie. Special thanks to Michael Mislove for graciously agreeing to serve as my external examiner.

The administrative staff at Dalhousie University, ranging from the Department of Mathematics and Statistics to the Faculty of Graduate Studies, has been remarkably helpful and accommodating. They do make a difference.

Finally, I would like to express my deepest gratitude to my family and friends for being a constant reminder of what is most important in life. Thank you!

This research was supported in part by the Natural Science and Engineering Research Council of Canada (NSERC), by the U.S. Air Force Office of Scientific Research, Air Force Material Command, USAF under Award No. FA9550-15-1-0331, and by Rigetti Computing.

Chapter 1

Introduction

Quantum computing studies the information processing capabilities of systems governed by the laws of quantum mechanics, especially the use of superposition, entanglement, and unitary evolution to carry out computation. The field started in the early 1980s when Feynman first explored the possibility of simulating physical systems with computers [40], and Benioff introduced the first quantum mechanical model of a computer [16]. However, it was not until the mid-1990s, after the work of Shor on integer factorization and of Grover on database search, that quantum computing took off. Indeed, in 1994, Shor showed that integers could be factored efficiently on a quantum computer [100] while in 1996, Grover devised a quantum algorithm for searching a database with a quadratic speedup over known classical algorithms [48]. Since then, quantum computing has been applied to various other fields such as differential equations, quantum simulation, circuit synthesis, and machine learning (e.g, [9], [107], [24], [112]). It is precisely this type of computational advantage and its applications that have given rise to a significant interest in quantum computing.

The *quantum bit* or *qubit*, a generalization of the classical bit, is the basic unit of information in quantum computing. The *state* of a qubit is represented by a unit vector in a 2-dimensional Hilbert space, which we assume to be $\mathbb{C}^2$. The *state space* of a system of n qubits is represented by $\mathbb{C}^{2^n}$, and a state of such a system by a unit vector in it. Two fundamental operations can be performed on qubits: *unitary transformations* and *measurements*. Unitary transformations describe the evolution of closed quantum systems while measurements reflect such systems' interactions with their environment. The application of such operations on a qubit system yields a quantum computation. Thus, we can think of a *quantum algorithm* as a sequence of quantum operations designed to solve a particular problem. Quantum algorithms can be given a graphical representation in the form of *quantum circuits*, similar to that of digital circuits. This formalism can moreover serve as a *language* for the description of quantum algorithms.

Quantum systems exhibit exotic behaviors such as *superposition*, where taking an appropriate linear combination of quantum states yields another quantum state, and *entanglement*, where the quantum state of a component of a system cannot be described independently of the state of the rest of the system. Yet another distinctive trait of quantum systems is the *no-cloning property* of quantum information: in general, quantum data cannot duplicated [109].

1.1 Formal Methods for Quantum Programming Languages

Establishing the correctness of quantum programs can be a difficult task. This is due in no small measure to the nature of quantum information. For example, debugging a classical program by analyzing its state is a useful technique in classical computing. However, such an approach is ineffective in the quantum setting since observing the state of a quantum program collapses the state. Thus, more sophisticated techniques are required. Our approach consists in developing *Proto-Quipper-M*, a new quantum programming language with a strong type system and robust semantics in which quantum program behavior and meaning can be explored. Ultimately, this approach can facilitate not only the specification of quantum programs but also the verification of their properties. This potentially opens the door to formalization in proof assistants, the gold standard when proving program correctness.

To better understand the behavior of programs, it is a powerful technique to give a formal notion of meaning to the language in which such programs are written. There are several approaches to the semantics of programming languages. In this thesis, we will be mostly concerned with operational and denotational semantics.

- An *operational semantics* gives meaning to a program by formally describing the behavior that it induces when run on an abstract machine. This type of semantics focuses on *how* a computation is carried out.

- A *denotational semantics* gives meaning to a program by associating it with a particular object in a mathematical structure. This type of semantics focuses on *what* a computation means, rather than on how it is performed.

A denotational semantics establishes a robust connection between the syntactic nature of a programming language and the structural essence of the mathematical space in which it is interpreted. This connection is an effective tool for understanding the capabilities and limitations of a programming language since it allows us to get a better grasp of the behavior of programs in terms of the mathematical properties of their interpretations.

Of interest to us will be the semantic spaces given by categorical structures since these models are especially well-suited for the study of quantum programming languages. This is because they abstract away from implementation details allowing for a better understanding of quantum programs and high-level reasoning about them. More on this will be discussed in Chapter 4 and subsequent chapters of this thesis.

1.2 The Evolution of Quantum Programming Languages

The promise of the arrival of scalable quantum programmable devices in the near future has propelled the development of a number of quantum programming languages. In the early 1980s, Benioff [16] and Deutsch [35] introduced the first quantum mechanical models of computers. But it was not until 1996 when Knill put forward a set of conventions for describing quantum algorithms using pseudo-code [63] that the development of quantum programming languages started. One of the first such languages was *QCL*, an imperative-style language developed by Ömer in the late 1990s and early 2000s ([78], [79]). Around the same time, Sanders and Zuliani [92] and Bettelli et al. [22] introduced other imperative-style quantum programming languages.

Functional quantum programming languages did not make their appearance until the early 2000s. In 2002, Selinger introduced *Quantum Flow Charts*, a first-order quantum programming language with a strong type system and a denotational semantics given in terms of complete partial orders of superoperators [95]. In 2004, Van Tonder defined λ_q, a lambda calculus for pure quantum computation equipped with an operational semantic [105]. From 2004 to 2009, Selinger and Valiron developed the *Quantum Lambda Calculus* ([104], [98], [99]), a higher-order functional quantum programming language with a strong linear type system and both an operational and a denotational semantics. In 2005, *QML* was introduced by Altenkirch and Grattage [7] as a first-order quantum programming language with the aim of permitting some quantum control in addition to quantum data. In 2011, Selinger and collaborators introduced *Quipper* ([6], [46], [47]), a very expressive high-level functional programming language capable of constructing and manipulating quantum circuits. Quipper is embedded in the functional programming language Haskell [58]. More on Quipper and its relation to Proto-Quipper-M will be discussed in the next section. We note that in 2015, Ross [90] introduced what is now known as *Proto-Quipper-S*, another stand-alone functional quantum programming language.

We do not intend to present a comprehensive history of quantum programming languages here. However, we note that in the last few years, a number of such languages coming from industry and academia have emerged. Those developed by industry include Microsoft's stand-alone quantum programming language *Q#* ([73], [102]), IBM's quantum assembly language *OpenQASM* [34], and Rigetti's quantum instruction language *Quil* [101]. Among the research-oriented languages coming from academia, we have the C-like imperative quantum programming language *Scaffold* [2], the higher-order quantum extension of the PCF language, *qPCF* [80], and the quantum circuit language *QWIRE* [82]. More on the history of quantum programming languages can be found in [94], [43], [91], and [74]. For more recent developments, see [30], [51], and [75].

1.3 Proto-Quipper-M: A Sound Language for Circuit Description

As mentioned in the previous section, Quipper is a high-level functional programming language for quantum computing. One of the main features of Quipper, which sets it apart from other quantum programming languages, is that Quipper is a quantum circuit description language. The language not only provides a syntax for the construction of quantum circuits by applying one gate at a time but also allows for the treatment of circuits as data. These can then be stored in variables and operated upon as a whole. Meta-operations on circuits such as circuit transformations, inversions, and iterations are supported. This ability to manipulate circuits at two-levels of abstraction, namely, locally, at the gate level, and globally, on entire circuits, is not only powerful but also quite useful for programmers as it faithfully reflects how many quantum algorithms are described.

Moreover, Quipper has been used to implement a number of non-trivial quantum algorithms covering a variety of applications (see [28], [8], [50], [107], [54], [87], and [71]). However, the language has some limitations. Quipper is implemented as an embedded language in the host language Haskell. This approach certainly permits an efficient implementation as it takes advantage of all the infrastructure already present in the host language. But it also causes a disparity between the type system of Quipper and that of its host. As a consequence, Quipper is not type-safe: there are well-typed programs that yield run-time errors. In particular, Haskell does not enforce linearity, that is, the constraint that a quantum state cannot be duplicated. Moreover, Quipper does not have a formal semantics since as an embedded language this would require to give a formal semantics to Haskell, a full-blown programming language, an impractical task.

On the other hand, Proto-Quipper-M is a quantum programming language that formalizes a small but useful fragment of Quipper and aims to solve some of the issues above. As a typed lambda calculus, Proto-Quipper-M is a stand-alone, i.e., non-embedded, programming language with a strong type system and sound denotational and operational semantics. The language serves as a stepping stone towards giving Quipper a complete formal definition and semantics as a stand-alone quantum programming language. Proto-Quipper-M is designed to enforce linearity, and so prevent violations of the no-cloning property. As shown later in this thesis, Proto-Quipper-M is indeed an *error-free* language.

As a circuit description language, Quipper has two distinct run-times:

- a *circuit generation time*, when circuits are constructed, and
- a *circuit execution time*, when circuits are run.

This feature is not unique to Quipper, but also present in hardware description languages such as Verilog and VHDL ([52], [103], [10]). As a consequence of the existence of two run-times, a Quipper program handles two types of values, namely, parameters and states:

- a *parameter* is a value known at circuit generation time, and
- a *state* is a value known at circuit execution time.

For example, given a list of qubits $[q_1, \ldots, q_n]$, the length n of the list is typically known at circuit generation time and is, therefore, a parameter while the state of the qubits in the list is known at circuit execution time and is, therefore, a state. As expected, a parameter may not depend on a state as states are unknown at circuit generation time, but a state may depend on a parameter. In the end, Quipper is a language that can describe not only individual quantum circuits, but also parametrized *families* of them; this is one of the key features that Proto-Quipper-M is designed to model.

Another distinctive feature of Quipper is that it has only one kind of variable, which can store both parameters and states, or any combination of these. Moreover, its type system is powerful enough to guarantee that states and parameters are used correctly. This feature makes Quipper a rather flexible language as it allows programmers to work with complex quantum data structures more easily. The distinction between parameters and states is also present in Proto-Quipper-M.

Overall, we use a semantic approach in the design of Proto-Quipper-M. What that means is that we first

- define a categorical model for parameters and states, then
- identify some relevant structures in the model, and finally
- define the language to fit the model.

Note that this is the opposite of the usual methodology, where one first defines the language and then looks for a model for it. However, our approach has the advantage that our language is almost correct-by-construction as we have the structure that models it from the beginning. Moreover, as a consequence of this semantic approach, Proto-Quipper-M is a slightly more general language than Quipper in the sense that it can describe families of morphisms in an arbitrary monoidal category, of which quantum circuits are just one example.

Finally, we endow Proto-Quipper-M with computational meaning via an operational semantics and establish the connection between the language and the model by proving safety and soundness properties.

1.4 Outline of Thesis

The thesis is organized as follows:

- Chapters 2– 4 form the first part of the thesis and present background material. Chapter 2 reviews the basics of quantum computing. We revisit the principles of quantum mechanics and the most relevant notions of Hilbert spaces and linear operators. The Quantum Random Access Machine model and the quantum circuit model of quantum computing are also introduced. Chapter 3 recalls some of the most fundamental concepts of category theory. Adjunctions and monads and their monoidal counterparts are discussed. Representable functors and the Yoneda Lemma are introduced, as well as a characterization of monoidal adjunctions. These will be used to prove that our categorical model of Proto-Quipper-M is a model of intuitionistic linear logic. Chapter 4 introduces a first categorical model of Proto-Quipper-M to motivate the general model introduced later in the thesis. Special emphasis is placed on motivating some of the choices made in the design of Proto-Quipper-M. The character of this chapter is mostly expository.

  We note that the first part of the thesis only contains the most necessary material for later chapters and no proofs are given. However, numerous references are provided here and in the ensuing chapters.

- Chapters 5– 6 form the second part of the thesis and present the more syntactic and computational results of our work. In Chapter 5, we introduce the syntax, type system, and operational semantics of Proto-Quipper-M. In particular, Proto-Quipper-M is defined as a simply-typed lambda calculus with a special type Circ(T, U) for quantum circuits and a *strong type system*. We recall the notion of *generalized circuit* and introduce that of *labeled circuit*. Also, a big-step *operational semantics* for Proto-Quipper-M is given in terms of an evaluation relation on pairs of the form (C, M), where C is a labeled circuit and M is a term of the language. In Chapter 6, we prove that Proto-Quipper-M is a *type-safe language* by showing that it enjoys the *subject-reduction* and *error-freeness* properties. This is accomplished by proving various intricate syntactic lemmas which are also required to establish the fundamental *substitution lemma*.

- Chapters 7–9 form the third part of the thesis and present a detailed account of our semantic results. In Chapter 7, we give a generalization of the *categorical model* of Proto-Quipper-M discussed in Chapter 4 in terms of copower and representable functors. In particular, we show that these functors yield a *linear-non-linear model* in the sense of Benton, and so a model of intuitionistic linear logic. Also, the *denotational semantics* of all the syntactic structures of Proto-Quipper-M is given. Chapter 8 is the semantic analogue of Chapter 6 in the sense that Chapter 8 contains the semantic versions of several of the syntactic results obtained in Chapter 6. These semantic results are used in the proof of the semantic version of the substitution lemma. In Chapter 9, we use the conclusions of previous chapters to prove one of the main results of the thesis, namely, the *soundness theorem* for Proto-Quipper-M .

1.5 Contributions

This thesis advances the mathematical foundations of quantum programming languages. Its main contribution is the development of a new quantum programming language, called Proto-Quipper-M, designed as a proving ground for the research of sound formal semantics and reasoning techniques for quantum programs.

My original work appears in chapters 4–9. Proto-Quipper-M was first defined in the article [89], co-authored with my supervisor Peter Selinger. Besides motivating the design choices of Proto-Quipper-M, this paper introduces the first categorical models of the language and its operational semantics, and describes some of their main properties. Chapter 4 of this thesis provides a summary of some of the contents of this article. Chapters 5–9 extend and refine some of the properties of Proto-Quipper-M mentioned in Chapter 4 and present new results. These results are my original contribution. They include numerous syntactic lemmas leading to the proof of the weakening lemma, which allows our language to not require explicit “discard” operators (not all linear term calculi enjoy this property); a proof of the admissibility of the fundamental substitution operation in Proto-Quipper-M; a demonstration of the type-safety of our language via detailed proofs of the subject reduction and error-freeness theorems; a generalization of previously known linear-non-linear models of

Proto-Quipper-M; several intricate lemmas leading to the semantic versions of the weakening and substitution lemmas; and finally, a detailed, extensive proof of the soundness theorem for Proto-Quipper-M.

Chapter 2

Quantum Computation

In this chapter, we review the basics of quantum computing, starting with linear algebra. More on linear algebra can be found in [42] and [69]. Standard references for quantum computing include [76], [1], and [110]. Our presentation roughly follows those of [60], [95], and [90].

2.1 Linear Algebra

Linear algebra studies vector spaces and their linear transformations. In this section, we revisit the most relevant notions of complex vector spaces as they are used in quantum computing.

2.1.1 Hilbert Spaces

As usual, we write $\mathbb{N}$ for the set of natural numbers, $\mathbb{R}$ for the set of real numbers, and $\mathbb{C}$ for the set of complex numbers. Also, for every $\alpha \in \mathbb{C}$, $\alpha^\dagger$ denotes its complex conjugate. We will be mostly concerned with finite dimensional vector spaces over $\mathbb{C}$. Given such a space V, with basis $\{|b_1\rangle, \ldots, |b_n\rangle\}$ such that $n \in \mathbb{N}$ is the dimension of V, every $|v\rangle \in V$ can be written *uniquely* as a linear combination

$$|v\rangle = \alpha_1 |b_1\rangle + \ldots + \alpha_n |b_n\rangle$$

where $\alpha_1, \ldots, \alpha_n \in \mathbb{C}$. In column vector notation,

$$|v\rangle = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix}.$$

Clearly, $V \cong \mathbb{C}^n$.

A *complex inner product space* is a vector space V over $\mathbb{C}$ equipped with an inner product operation $\langle \cdot , \cdot \rangle : V \times V \to \mathbb{C}$ that satisfies the following conditions for all $|v\rangle, |w\rangle, |w_j\rangle \in V$ and $\lambda_j \in \mathbb{C}$:

1. Linearity in the second argument,

$$\langle |v\rangle, \sum_j \lambda_j |w_j\rangle\rangle = \sum_j \lambda_j \langle |v\rangle, |w_j\rangle\rangle$$

2. Conjugate symmetry,

$$\langle |v\rangle, |w\rangle\rangle = \langle |w\rangle, |v\rangle\rangle^\dagger$$

3. Positive definitiveness,

$$\langle |v\rangle, |v\rangle\rangle \geq 0$$

   with equality if and only if $|v\rangle = 0$.

To simplify notation, we will write $\langle v|w\rangle$ instead of $\langle |v\rangle, |w\rangle\rangle$.

A *Hilbert space* is a complex inner product space that is a complete metric space with respect to the distance function induced by its inner product. In quantum computing, we mostly deal with finite-dimensional complex inner product spaces. These spaces are necessarily complete, and so they are finite-dimensional Hilbert spaces. So from now on, and unless otherwise stated, by Hilbert space we mean finite-dimensional Hilbert space. Moreover, we will be mainly interested in spaces of dimensions 2^n for $n \in \mathbb{N}$. Note that the column vector representation of a basis element of such a space requires 2^n entries, while its Dirac notation only needs a binary string of length n. For example, for the first vector $|b_1\rangle$ of the canonical basis of the 2^n-dimensional vector space $\mathbb{C}^{2^n}$, we have

$$\underbrace{|00\ldots00\rangle}_{n \text{ entries}} = |b_1\rangle = \left.\begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}\right\} 2^n \text{ entries.}$$

Given a Hilbert space $\mathscr{H}$, the *dual vector space* $\mathscr{H}^*$ associated with $\mathscr{H}$ has as its elements all linear transformations $f : \mathscr{H} \to \mathbb{C}$. By the well-known Riesz representation theorem, each such f is of the form $f(|\psi\rangle) = \langle\phi|\psi\rangle$ for some unique $|\phi\rangle \in \mathscr{H}$. We write $f = \langle\phi|$ and call $\langle\phi|$ the *dual* of $|\phi\rangle$. It has a matrix representation given by the conjugate transpose of the matrix representation of $|\phi\rangle$; in other words, $|\phi\rangle$ is represented by a column vector and $\langle\phi|$ by a row vector. The linear transformation $\langle\phi|$ is also called the "bra" of ϕ and the vector $|\phi\rangle$ the "ket" of ϕ.

If the inner product of two vectors is zero, then they are said to be *orthogonal.* The *Euclidean norm* of a vector $|\psi\rangle$, written as $\||\psi\rangle\|$, is given by $\||\psi\rangle\| = \sqrt{\langle\psi|\psi\rangle}$. A *unit vector* is a vector of norm 1, and an *orthonormal set* is a set of mutually orthogonal unit vectors.

Proposition 2.1.1. *If $\{|b_1\rangle, \ldots, |b_n\rangle\}$ is an orthonormal basis for a Hilbert space $\mathscr{H}$, then $\{\langle b_1|, \ldots, \langle b_n|\}$ is an orthonormal basis for $\mathscr{H}^*$.* $\diamond$

2.1.2 Operators and Matrix Representations

If $S : V \to W$ is a linear transformation, $|\zeta\rangle$ a nonzero vector in V, $\lambda \in \mathbb{C}$, and $S(|\zeta\rangle) = \lambda|\zeta\rangle$, then $|\zeta\rangle$ is an *eigenvector* of S with *eigenvalue* λ. A *linear operator* on a vector space V is a linear transformation $T : V \to V$. We sometimes write $T|\delta\rangle$ instead of $T(|\delta\rangle)$ whenever $|\delta\rangle \in V$. Given $|\phi\rangle, |\psi\rangle$ in a Hilbert space $\mathscr{H}$, we can define a linear operator $|\psi\rangle\langle\phi| : \mathscr{H} \to \mathscr{H}$ that maps $|\gamma\rangle \mapsto |\psi\rangle\langle\phi|\,(|\gamma\rangle) := \langle\phi|\gamma\rangle|\psi\rangle$, called the *outer product* of $|\psi\rangle$ with $\langle\phi|$.

Theorem 2.1.2. *Let $B = \{|b_1\rangle, \ldots, |b_n\rangle\}$ be an orthonormal basis for a Hilbert space $\mathscr{H}$. Then every linear operator T on $\mathscr{H}$ can be decomposed as*

$$T = \sum_{k,j=1}^{n} T_{kj}\, |b_k\rangle\langle b_j|$$

where $T_{kj} = \langle b_k|T|b_j\rangle$. $\diamond$

Note that T_{kj} is the matrix entry in the kth row and jth column of the matrix representation of T. Also, by Theorem 2.1.2, the *identity operator* $1_{\mathscr{H}} : \mathscr{H} \to \mathscr{H}$ can be written as

$$1_{\mathscr{H}} = \sum_{j=1}^{n} |b_j\rangle\langle b_j|.$$

Given an operator T on a Hilbert space $\mathscr{H}$, the *adjoint* of T, denoted by $T^\dagger$, is the unique linear operator on $\mathscr{H}$ that satisfies $\langle T^\dagger(|\phi\rangle), |\psi\rangle\rangle = \langle|\phi\rangle, T(|\psi\rangle)\rangle$ for all $|\phi\rangle, |\psi\rangle \in \mathscr{H}$. The matrix representation of $T^\dagger$ is the conjugate transpose of T. An operator U is called *unitary* if $U^\dagger = U^{-1}$ where U^{-1} is the *inverse* operator of U, i.e., the unique linear operator such that $UU^{-1} = U^{-1}U = I$. An operator T is called *Hermitian* if $T^\dagger = T$, that is, if T is *self-adjoint*. Unitary and Hermitian operators are familiar to physicists because they describe the time-evolution and the observables of quantum systems, respectively. A *projector* P on a vector space $\mathscr{H}$ is a linear operator on $\mathscr{H}$ that satisfies $P^2 = P$, and it is called an *orthogonal projector* if it also satisfies $P^\dagger = P$. The eigenvalues of a Hermitian operator are real numbers:

Theorem 2.1.3. *If T is a Hermitian operator on a Hilbert space $\mathscr{H}$ and $T|\psi\rangle = \lambda|\psi\rangle$ for some non-zero $|\psi\rangle \in \mathscr{H}$, then $\lambda \in \mathbb{R}$.* $\diamond$

Given an orthonormal basis $B = \{|b_1\rangle, \ldots, |b_n\rangle\}$ of a Hilbert space $\mathscr{H}$, the *trace* of an operator T on $\mathscr{H}$ is defined as

$$tr(T) = \sum_{j=1}^{n} \langle b_j|T|b_j\rangle.$$

The trace of an operator is independent of the basis in which is expressed.

A linear operator T that satisfies $T^\dagger T = TT^\dagger$ is said to be *normal.* Note that unitary and Hermitian operators are normal. The following is a fundamental theorem of normal operators.

Theorem 2.1.4 (Spectral Decomposition)**.** *If T is a normal operator on a finite-dimensional Hilbert space $\mathscr{H}$, then there is an orthonormal basis $B = \{|b_1\rangle, \ldots, |b_n\rangle\}$ of $\mathscr{H}$ consisting of eigenvectors of T.* $\diamond$

We denote the eigenvalues of $|b_j\rangle$ by λ_j and call the set of eigenvalues of T the *spectrum* of T. Note that the *spectral decomposition* of T is then given by

$$T = \sum_{j=1}^{n} \lambda_j \, |b_j\rangle\langle b_j|.$$

In terms of matrices, the spectral decomposition theorem takes the following form:

Theorem 2.1.5. *If T is a normal matrix, then there exists a unitary matrix U and a diagonal matrix D such that $T = UDU^\dagger$, where the entries of D are the eigenvalues of T and the columns of U are the eigenvectors of T.* $\diamond$

Table 2.1 gives examples of normal matrices representing gates commonly used in the design of quantum circuits. The unitary and Hermitian matrices H, X, $CNOT$,

$$H = \tfrac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} \qquad X = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix} \qquad S = \begin{bmatrix}1 & 0\\ 0 & i\end{bmatrix} \qquad T = \begin{bmatrix}1 & 0\\ 0 & e^{i\pi/4}\end{bmatrix}$$

$$CNOT = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix} \qquad SWAP = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \qquad CS = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & i\end{bmatrix}$$

Table 2.1: Commonly Used Quantum Gates.

and $SWAP$ are known as the *Hadamard*, *not*, *controlled-not*, and *swap* gates, respectively. The unitary matrices S, T, and CS are known as the *phase*, $\pi/8$, and *controlled-phase* gates, respectively.

The spectral decomposition theorem is very useful as it allows us to diagonalize normal operators, and so simplify the expressions for functions of normal operators. If $f : \mathbb{C} \to \mathbb{C}$ is a function with Taylor series expansion given by

$$f(z) = \sum_{m=0}^{\infty} a_m z^m,$$

and T is a normal operator on a Hilbert space $\mathscr{H}$ with basis $B = \{|b_1\rangle, \ldots, |b_n\rangle\}$ consisting of orthonormal eigenvectors of T whose eigenvalues belong to the interval of convergence of the Taylor series for f, then we can define

$$f(T) = \sum_{m=0}^{\infty} a_m T^m.$$

In this case,

$$f(T) = \sum_{j=1}^{n} f(\lambda_j)\, |b_j\rangle\langle b_j|.$$

For example, the exponential of the unitary operator S is given by

$$e^S = \sum_{j=0}^{1} e^{\lambda_j}\, |j\rangle\langle j| = \begin{bmatrix} e & 0\\ 0 & e^i\end{bmatrix}$$

since S has eigenvectors $|0\rangle$ and $|1\rangle$ with respective eigenvalues 1 and i.

2.1.3 Tensor Products

The *tensor product* is a construction of paramount importance in quantum computing as it can be used to build complex quantum systems from simple ones. Suppose that $\mathscr{H}_1$ and $\mathscr{H}_2$ are finite-dimensional Hilbert spaces with orthonormal bases $B_1 = \{|b_1\rangle, \dots, |b_n\rangle\}$ and $B_2 = \{|c_1\rangle, \dots, |c_m\rangle\}$, respectively. Then, the tensor product space $\mathscr{H}_1 \otimes \mathscr{H}_2$ is the Hilbert space with basis $\{|b_i\rangle \otimes |c_j\rangle\}_{i\in\{1,\dots,n\}, j\in\{1,\dots,m\}}$ characterized by the following axioms:

1. For all $\alpha \in \mathbb{C}$, $|\psi_1\rangle \in \mathscr{H}_1$, and $|\psi_2\rangle \in \mathscr{H}_2$,

$$\alpha(|\psi_1\rangle \otimes |\psi_2\rangle) = (\alpha|\psi_1\rangle) \otimes |\psi_2\rangle = |\psi_1\rangle \otimes (\alpha|\psi_2\rangle).$$

2. For all $|\psi_1\rangle, |\phi_1\rangle \in \mathscr{H}_1$ and $|\psi_2\rangle \in \mathscr{H}_2$,

$$(|\psi_1\rangle + |\phi_1\rangle) \otimes |\psi_2\rangle = (|\psi_1\rangle \otimes |\psi_2\rangle) + (|\phi_1\rangle \otimes |\psi_2\rangle).$$

3. For all $|\psi_1\rangle \in \mathscr{H}_1$ and $|\psi_2\rangle, |\phi_2\rangle \in \mathscr{H}_2$,

$$|\psi_1\rangle \otimes (|\psi_2\rangle + |\phi_2\rangle) = (|\psi_1\rangle \otimes |\psi_2\rangle) + (|\psi_1\rangle \otimes |\phi_2\rangle).$$

Given $|\psi\rangle \in \mathscr{H}_1$ and $|\phi\rangle \in \mathscr{H}_2$, we sometimes write $|\psi\phi\rangle$ or $|\psi\rangle|\phi\rangle$ for $|\psi\rangle \otimes |\phi\rangle$.

Suppose that T_1 and T_2 are linear operators on $\mathscr{H}_1$ and $\mathscr{H}_2$, respectively. Then $T_1 \otimes T_2$ is the linear operator on $\mathscr{H}_1 \otimes \mathscr{H}_2$ given by

$$(T_1 \otimes T_2)\left(\sum_{i,j} \alpha_{ij}|b_i\rangle \otimes |c_j\rangle\right) = \sum_{i,j} \alpha_{ij}\, T_1|b_i\rangle \otimes T_2|c_j\rangle$$

for all $|\psi\rangle = \sum_{i,j} \alpha_{ij}|b_i\rangle \otimes |c_j\rangle \in \mathscr{H}_1 \otimes \mathscr{H}_2$. Moreover, if A and B are the matrix representations of T_1 and T_2, respectively, then the matrix representation of $T_1 \otimes T_2$ is given by the *Kronecker product* of A with B:

$$A \otimes B = \begin{bmatrix} A_{11}B & A_{12}B & \dots & A_{1n}B \\ A_{21}B & A_{22}B & \dots & A_{2n}B \\ \vdots & \vdots & \vdots & \vdots \\ A_{n1}B & A_{n2}B & \dots & A_{nm}B \end{bmatrix},$$

where A_{ij} denotes the entry of A in row i, column j, and $A_{ij}B$ denotes the scalar multiplication of A_{ij} with B. For example, the tensor product of the Hadamard gate with the 2×2 identity gate is

$$\begin{aligned} H \otimes I &= \tfrac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes I \\ &= \tfrac{1}{\sqrt{2}} \begin{bmatrix} 1 \cdot I & 1 \cdot I \\ 1 \cdot I & -1 \cdot I \end{bmatrix} \\ &= \tfrac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}. \end{aligned}$$

2.2 The Elements of Quantum Computing

Quantum computing is the study of the type of information processing that can be performed by physical systems governed by the laws of quantum mechanics. The discovery of quantum mechanics at the beginning of the 20th century was due to the physics crisis of the time: the failure of classical mechanics to describe many phenomena at the atomic level. And so quantum mechanics was born as a new mathematical framework for the development of physical theories, yielding a rather accurate description of the physical world at a small scale.

The laws of quantum mechanics are simple but counter-intuitive. One of the goals of quantum computing is to develop tools that improve our understanding and intuition of quantum mechanics. Moreover, quantum computers potentially offer impressive computational speed-up over classical computers. For example, in 1994 Peter Shor showed that the factoring problem for integers could be solved efficiently on a quantum computer [100], and in 1996 Lov Grover showed that the search problem through an unstructured space could be sped up as well [48]. Since then many other problems from fields as diverse as quantum circuit synthesis (see [53], [24], [62], and [97]), differential equations ([68], [21], [9]), quantum simulation ([111], [41], [29]), and machine learning ([70], [38], [112]), have been shown to have more efficient solutions when solved using techniques from quantum computing.

2.2.1 Quantum Bits and Quantum Systems

The *quantum bit* or *qubit* is the fundamental unit of information in quantum computing; it is a generalization of the *classical bit.* While a classical bit can be in either of two *states*, 0 or 1, the state of a qubit can be any linear combination $\alpha|0\rangle + \beta|1\rangle$, where $\alpha, \beta \in \mathbb{C}$ are *amplitudes* such that $|\alpha|^2 + |\beta|^2 = 1$, and $\{|0\rangle, |1\rangle\}$ is an orthonormal basis of a 2-dimensional Hilbert space, which we will assume to be $\mathbb{C}^2$. The basis $\{|0\rangle, |1\rangle\}$ is also known as the *computational basis.* Two states are considered equivalent if one is a scalar multiple of the other.

By analogy with the states of a classical bit, the qubit states $|0\rangle = 1|0\rangle + 0|1\rangle$ and $|1\rangle = 0|0\rangle + 1|1\rangle$ are sometimes called *classical states*, while states $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ with non-zero amplitudes are called *quantum superpositions* of $|0\rangle$ and $|1\rangle$. In general, the *state space* of a system of n qubits is $\mathbb{C}^{2^n}$ and a *state* of such a system is a unit vector of the form

$$|\psi\rangle = \sum_{b_1,\ldots,b_n \in \{0,1\}} \alpha_{b_1\ldots b_n} |b_1 \ldots b_n\rangle,$$

where $\alpha_{b_1\ldots b_n} \in \mathbb{C}$, and $|b_1 \ldots b_n\rangle$ is the j-th canonical basis vector of $\mathbb{C}^{2^n}$, where j is the natural number whose binary representation is $b_1 \ldots b_n$.

For any qubit states $|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$ and $|\phi\rangle = \beta_0|0\rangle + \beta_1|1\rangle$, the *product state* of $|\psi\rangle$ with $|\phi\rangle$ is given by $|\psi\rangle \otimes |\phi\rangle = \alpha_0\beta_0|00\rangle + \alpha_0\beta_1|01\rangle + \alpha_1\beta_0|10\rangle + \alpha_1\beta_1|11\rangle$. Note that, in general, the state of a pair of qubits is not of the form $|\psi\rangle \otimes |\phi\rangle$. For example, the *Bell state* $\frac{|01\rangle + |10\rangle}{\sqrt{2}}$ cannot be written as a tensor product $|\psi\rangle \otimes |\phi\rangle$ for any $|\psi\rangle, |\phi\rangle \in \mathbb{C}^2$. Such states are called *entangled* and they play a fundamental role in quantum computing.

2.2.2 Evolution of Quantum Systems

One of the fundamental operations that can be performed on qubits is a *unitary transformation.* Such a transformation describes the evolution of the state space of a quantum system, and in the context of quantum computing, it is represented by a unitary matrix. A unitary transformation on n qubits is called an *n-ary quantum gate.*

Important examples of unary and binary quantum gates are given in Table 2.1 on page 14. Like any other linear operators, quantum gates are determined by their actions on a basis. For example, the not gate X swaps the basis vectors, sending $|0\rangle$ to $|1\rangle$, and $|1\rangle$ to $|0\rangle$. The Hadamard gate H can create quantum superpositions, for example, $H|0\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and $H|1\rangle = \frac{|0\rangle-|1\rangle}{\sqrt{2}}$. The controlled-not gate $CNOT$ is a 2-qubit gate that does not change the first qubit and applies the not gate X to the second qubit only when the first qubit is $|1\rangle$. These gates can be used to create entanglement between qubits, for example,

$$\begin{aligned}
(CNOT \circ (H \otimes I) \circ (X \otimes I))(|0\rangle \otimes |0\rangle) &= (CNOT \circ (H \otimes I))(X|0\rangle \otimes I|0\rangle) \\
&= (CNOT \circ (H \otimes I))(|1\rangle \otimes |0\rangle) \\
&= CNOT(H|1\rangle \otimes I|0\rangle) \\
&= CNOT((\tfrac{|0\rangle-|1\rangle}{\sqrt{2}}) \otimes |0\rangle) \\
&= CNOT(\tfrac{|00\rangle-|10\rangle}{\sqrt{2}}) \\
&= CNOT(\tfrac{|00\rangle}{\sqrt{2}}) - CNOT(\tfrac{|10\rangle}{\sqrt{2}}) \\
&= \tfrac{|00\rangle-|11\rangle}{\sqrt{2}}.
\end{aligned} \tag{2.1}$$

Moreover, any n-ary quantum gate can be approximated arbitrarily closely by a composition of a finite number of gates from Table 2.1. A gate set with this property is said to be *universal* for quantum computing. Many such gate sets have been found and investigated (see [36], [13], [61], [32], and [4]). As expected, not all potentially desirable behaviors of a quantum system can be achieved through unitary evolution. Of particular interest to us is the impossibility to clone an arbitrary quantum state using a unitary transformation. This is the content of the no-cloning theorem proved by Wootters and Zurek [109], and independently by Dieks [37], in 1982. Similarly, it is impossible to delete an unknown quantum state with a unitary operation, as shown by Pati and Braunstein [81] in what is now known as the no-deleting theorem. These two properties of quantum systems, i.e., no-cloning and no-deleting of quantum information, play a pivotal role in the development of quantum programming languages, which in principle should enforce these constrains when dealing with quantum data. Proto-Quipper-M is designed to consistently enforce such restrictions. More on this will be explored in later chapters of this thesis.

2.2.3 Measurements

In the previous section, we discussed the evolution of closed quantum systems via unitary transformations. However, to better understand such systems, we must let them interact with their environment and measure the results of such interactions. A *measurement* is the other fundamental operation that can be performed on the state of a quantum system. Unlike unitary transformations, measurement is a probabilistic operation.

In quantum computing at its most basic level, a measurement takes place when we observe a qubit to determine its state. For example, when we measure a qubit in state $\alpha_0|0\rangle + \alpha_1|1\rangle$, the result is either 0 with probability $|\alpha_0|^2$ and final state $|0\rangle$, or 1 with probability $|\alpha_1|^2$ and final state $|1\rangle$. To get a better idea of how measurements work in general, let us look at the measurement of a quantum system of two qubits in state

$$\alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle.$$

In this case, if we measure the left qubit, the result is either 0 with probability $|\alpha_{00}|^2 + |\alpha_{01}|^2$ and final state

$$\frac{\alpha_{00}|00\rangle + \alpha_{01}|01\rangle}{\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}},$$

or 1 with probability $|\alpha_{10}|^2 + |\alpha_{11}|^2$ and final state

$$\frac{\alpha_{10}|10\rangle + \alpha_{11}|11\rangle}{\sqrt{|\alpha_{10}|^2 + |\alpha_{11}|^2}}.$$

If after this measurement, we measure the right qubit, the result is either 00 with overall probability $|\alpha_{00}|^2$ and final state $|00\rangle$, or 01 with overall probability $|\alpha_{01}|^2$ and final state $|01\rangle$, or 10 with overall probability $|\alpha_{10}|^2$ and final state $|10\rangle$, or 11 with overall probability $|\alpha_{11}|^2$ and final state $|11\rangle$. Now it is easy to see that if we had measured the right qubit first and then the left qubit, the result would have been the same. The measurement of quantum systems with more than two qubits is similar.

2.2.4 The Quantum Random Access Machine Model

So far, our treatment has mostly focused on the mathematical aspects of quantum computing. To bridge the gap between this abstract framework and a more concrete

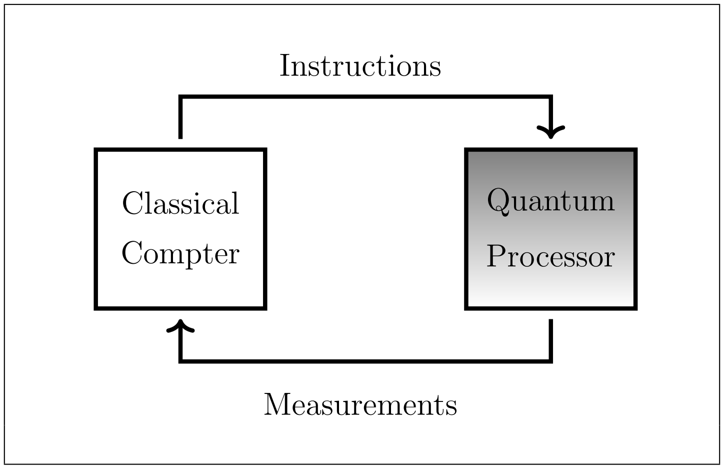


Table 2.2: The QRAM Model.

set-up for the implementation of quantum programming languages, we recall a pair of useful computational models in this section and the next.

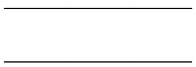
Several quite different, but equivalent, models of quantum computing have been developed (see [36], [39], [86], and [5]). Here we discuss a convenient architecture, called the *Quantum Random Access Machine*, or *QRAM*, introduced by Knill [63], that is rather fitting for the design and interpretation of quantum programming languages. In a QRAM model, two devices interact with each other: a classical computer and a quantum processor, as in Table 2.2. The latter is assumed to have several addressable quantum bits on which it can operate. In particular, the quantum processor can initialize, apply unitary transformations to, and measure any of its qubits.

With this architecture in mind, it is not too difficult to see how to run a quantum program. Namely, the classical computer stores the program where it is compiled and executed. During the run, the classical device may send a sequence of instructions to the quantum processor requesting the performance of quantum operations on some of its qubits. Once these are completed, the results of the measurements are sent back to the classical computer for further processing. This feedback loop can be executed as many times as necessary.

2.2.5 Quantum Circuits

In the previous section, we presented a rather schematic, but natural, architecture for a quantum computer: the QRAM model. Here we introduce what is perhaps the most well-known model of quantum computation, namely, the *quantum circuit model* of Deutsch [36]. In this model, a quantum circuit is a sequence of unitary operations with measurements at the end. We can think of these circuits as sequences of instructions to be executed by the quantum processor of a QRAM. And so the quantum circuit formalism can also serve as a language for the implementation of quantum algorithms.

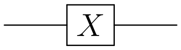
Quantum circuits have graphical representations that resemble those of digital circuits. The identity gate on n qubits I_n is represented by n horizontal *wires*. For example, I_2 is depicted as



.

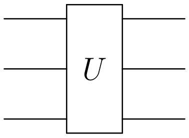
We also refer to these wires as *quantum wires* or *wires of qubit type* as they serve as references to qubits in a computation.

A non-identity unitary transformation operating on n qubits is represented by a labeled box with n *input wires* and n *output wires*. For example, the 1-qubit not gate X is represented by



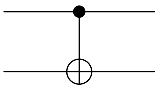
,

while an arbitrary 3-qubit gate U by



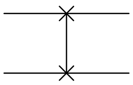
.

Some special gates have distinctive representations, i.e., without boxes surrounding their labels. For example, the 2-qubit controlled-not gate $CNOT$ is represented by



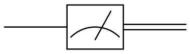
,

while the swap gate $SWAP$ by



.

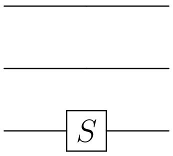
The gate implementing the measurement of a qubit in the computational basis is depicted as



.

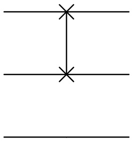
Here we refer to wires drawn with double lines as *classical wires* or *wires of bit type* as they serve as references to bits in a computation. So the measurement gate inputs a qubit and outputs a bit.

The graphical representation of the tensor product of quantum gates is given by the vertical concatenation of their corresponding representations. For example, the tensor product $I_2 \otimes S$ is depicted as



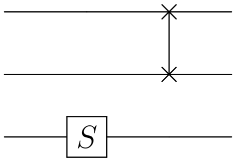
,

while the product $SWAP \otimes I_1$ as



.

The composition of quantum gates on n qubits is represented by the horizontal concatenation of their corresponding representations. More precisely, the concatenation occurs by joining the output wires of the representation of the first gate being applied with the input wires of the representation of the second one. For example, the composition $(SWAP \otimes I_1) \circ (I_2 \otimes S)$ is depicted as



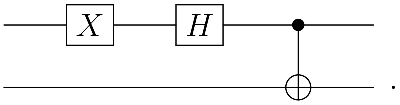
.

By the functoriality of $\otimes$, we have that

$$(SWAP \circ I_2) \otimes (I_1 \circ S) = (SWAP \otimes I_1) \circ (I_2 \otimes S),$$

and by the discussion above, we see that the graphical representation of the tensor product $(SWAP \circ I_2) \otimes (I_1 \circ S)$ is the same as that of the composition $(SWAP \otimes I_1) \circ (I_2 \otimes S)$.

Naturally, using the concatenation of diagrams, we can construct graphical representations of more interesting quantum operations. For example, recall from Equation (2.1) that the gate $CNOT \circ (H \otimes I) \circ (X \otimes I)$ creates entanglement between pairs of qubits. This gate can be represented by



We finish this section by noting that we can obtain the matrix representation of a quantum gate from its circuit representation by interpreting the vertical and horizontal concatenation of the graphical representations of its components as the tensor product and composition of the corresponding matrix representations, respectively. For example, the matrix representation of the last quantum circuit depicted above is

$$
\begin{aligned}
CNOT \circ (H \otimes I) \circ (X \otimes I) &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \\
&= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & -1 & 0 & 1 \\ -1 & 0 & 1 & 0 \end{bmatrix}.
\end{aligned}
$$

Chapter 3

Category Theory

3.1 Elementary Category Theory

In this chapter, we recall some of the most fundamental concepts of category theory, including adjunctions, monads, and monoidal categories. More on the basics of category theory can be found in [56], [67], [11], and [88]. For connections of category theory with computer science and logic, see [15], [33], [3], and [65]. A nice account of quantum theory in terms of category theory and diagrammatic languages is given in [57]. More advanced treatments of category theory include [66], [25], and [26].

3.1.1 Categories

We can think of a category as a system of related entities of the same kind satisfying some natural conditions. In particular, a category is an abstraction of the collection of sets and functions, and their composition and associativity and unit laws. In this section, we introduce categories formally, provide some elementary examples, and fix some notation.

Definition 3.1.1. A *category* $\mathbf{C}$ consists of the following:

- a class $\mathrm{Ob}(\mathbf{C})$ of *objects*;
- for any two objects $A, B \in \mathrm{Ob}(\mathbf{C})$, a class $\mathbf{C}(A, B)$ of *morphisms* with *domain* A and *codomain* B;
- for any object $A \in \mathrm{Ob}(\mathbf{C})$, a morphism $\mathrm{id}_A \in \mathbf{C}(A, A)$ called the *identity morphism* on A;
- for any objects $A, B, C \in \mathrm{Ob}(\mathbf{C})$, an assignment called *composition*

$$\begin{array}{rcl} \circ : \mathbf{C}(B, C) \times \mathbf{C}(A, B) & \longrightarrow & \mathbf{C}(A, C). \\ (g, f) & \longmapsto & g \circ f \end{array}$$

The morphism $g \circ f$, also written as gf, is called the composition of g with f.

This data is required to satisfy the following two axioms:

1. *Associativity*: For any morphisms $f \in \mathbf{C}(A, B)$, $g \in \mathbf{C}(B, C)$, and $h \in \mathbf{C}(C, D)$,

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

2. *Identity*: For any morphism $f \in \mathbf{C}(A, B)$,

$$\mathrm{id}_B \circ f = f = f \circ \mathrm{id}_A$$

◇

The class of objects of a category $\mathbf{C}$ may also be written as $\mathbf{C}_0$, and the class of its morphisms as $\mathbf{C}_1$. We sometimes write $A \in \mathbf{C}$ instead of $A \in \mathbf{C}_0$, and 1_A instead of id_A. If for any two objects $A, B \in \mathbf{C}_0$, the class $\mathbf{C}(A, B)$ is a set, then we say that $\mathbf{C}$ is *locally small*. Morphisms can also be referred to as *maps* or *arrows*. We may write $A \xrightarrow{f} B$ or $f : A \to B$ whenever $f \in \mathbf{C}(A, B)$ and say that the *source* of f is A and its *target* is B. A morphism $f : A \to B$ is called an *isomorphism* if there exists a morphism $f^{-1} : B \to A$ such that $f^{-1} \circ f = \mathrm{id}_A$ and $f \circ f^{-1} = \mathrm{id}_B$. We then say that A and B are *isomorphic*.

Categorical structures abound in mathematics. The prototypical example is the category **Set**, whose objects are sets and whose morphisms are functions. Other examples include the category **Rel**, with sets as objects and binary relations as morphisms; the category **FHilb**, whose objects are finite-dimensional Hilbert spaces and whose morphisms are linear maps; and the category **Mon**, with monoids as objects and monoid homomorphisms as morphisms.

We can construct new categories from others. Given a category $\mathbf{C}$, the *opposite category* $\mathbf{C}^{op}$ has the same objects as $\mathbf{C}$, but with morphisms $\mathbf{C}^{op}(A, B)$ equal to $\mathbf{C}(B, A)$. Given categories $\mathbf{C}$ and $\mathbf{D}$, the *product category* $\mathbf{C} \times \mathbf{D}$ has as objects ordered pairs (A, B), where A is an object in $\mathbf{C}$ and B is an object in $\mathbf{D}$, and as morphisms, the pairs $(f, g) : (A, B) \to (A', B')$, where f is a morphism in $\mathbf{C}(A, A')$ and g is a morphism in $\mathbf{D}(B, B')$.

3.1.2 Functors and Natural Transformations

Just as objects in a category are related by morphisms, categories are related by functors.

Definition 3.1.2. Let $\mathbf{C}$ and $\mathbf{D}$ be categories.

- A (*covariant*) *functor* $F : \mathbf{C} \to \mathbf{D}$ consists of
  - an operation on objects
  $$F_0 : \mathbf{C}_0 \to \mathbf{D}_0$$
  that assigns an object $F_0(A)$ in $\mathbf{D}$ to each object A in $\mathbf{C}$, and
  - for each $A, B \in \mathbf{C}_0$, an operation on arrows
  $$F_{A,B} : \mathbf{C}(A, B) \to \mathbf{D}(F_0(A), F_0(B))$$
  that assigns an arrow $F_0(A) \xrightarrow{F_{A,B}(f)} F_0(B)$ in $\mathbf{D}$ to each arrow $A \xrightarrow{f} B$ in $\mathbf{C}$.

  These operations satisfy the following axioms:

  1. *Compositionality*: $F_{A,C}(g \circ f) = F_{B,C}(g) \circ F_{A,B}(f)$ for all arrows $A \xrightarrow{f} B$ and $B \xrightarrow{g} C$ in $\mathbf{C}$.
  2. *Identity*: $F_{A,A}(\mathrm{id}_A) = \mathrm{id}_{F_0(A)}$ for every object A in $\mathbf{C}$.

- A *contravariant functor* $F : \mathbf{C} \to \mathbf{D}$ consists of
  - an operation on objects
  $$F_0 : \mathbf{C}_0 \to \mathbf{D}_0$$
  that assigns an object $F_0(A)$ in $\mathbf{D}$ to each object A in $\mathbf{C}$, and
  - for each $A, B \in \mathbf{C}_0$, an operation on arrows
  $$F_{A,B} : \mathbf{C}(A, B) \to \mathbf{D}(F_0(B), F_0(A))$$
  that assigns an arrow $F_0(B) \xrightarrow{F_{A,B}(f)} F_0(A)$ in $\mathbf{D}$ to each arrow $A \xrightarrow{f} B$ in $\mathbf{C}$.

  These operations satisfy the following axioms:

1. *Compositionality*: $F_{A,C}(g \circ f) = F_{A,B}(f) \circ F_{B,C}(g)$ for all arrows $A \xrightarrow{f} B$ and $B \xrightarrow{g} C$ in $\mathbf{C}$.
2. *Identity*: $F_{A,A}(\mathrm{id}_A) = \mathrm{id}_{F_0(A)}$ for every object A in $\mathbf{C}$. ◇

We may drop the subscripts of the operations defining a functor when no confusion arises. Note that contravariant functors from $\mathbf{C}$ to $\mathbf{D}$ are in one-to-one correspondence with covariant functors from $\mathbf{C}^{op}$ to $\mathbf{D}$.

The notions of injective function and surjective function can be lifted to the level of functors:

Definition 3.1.3. A functor $F : \mathbf{C} \to \mathbf{D}$ is

- *faithful* if each operation $F_{A,B} : \mathbf{C}(A, B) \to \mathbf{D}(F(A), F(B))$ is injective,
- *full* if each operation $F_{A,B} : \mathbf{C}(A, B) \to \mathbf{D}(F(A), F(B))$ is surjective, and
- an *embedding* if F is faithful and injective on objects. ◇

Just as functors relate categories, natural transformations relate functors that have the same source and target.

Definition 3.1.4. Given functors $F, G : \mathbf{C} \to \mathbf{D}$, a *natural transformation* $\alpha : F \Rightarrow G$ from F to G is a family of morphisms $\alpha_A : F(A) \to G(A)$ in $\mathbf{D}$ indexed by objects A in $\mathbf{C}$ such that for each morphism $f : A \to B$ in $\mathbf{C}$ the following diagram commutes:

$$\begin{tikzcd}
F(A) \arrow[r, "\alpha_A"] \arrow[d, "F(f)"'] & G(A) \arrow[d, "G(f)"] \\
F(B) \arrow[r, "\alpha_B"'] & G(B).
\end{tikzcd}$$

The morphisms α_A are called the *components* of α. ◇

Functors and natural transformations yield categories.

Definition 3.1.5. Given categories $\mathbf{C}$ and $\mathbf{D}$, the *functor category* from $\mathbf{C}$ to $\mathbf{D}$, written as $[\mathbf{C}, \mathbf{D}]$ or $\mathbf{D}^{\mathbf{C}}$, has as objects the functors from $\mathbf{C}$ to $\mathbf{D}$ and as morphisms the natural transformations between them. ◇

Functor category isomorphisms are rather relevant in category theory, and so have their own name.

Definition 3.1.6. Given categories $\mathbf{C}$ and $\mathbf{D}$, a *natural isomorphism* between functors from $\mathbf{C}$ to $\mathbf{D}$ is an isomorphism in the functor category $[\mathbf{C}, \mathbf{D}]$. $\diamond$

The following characterization of a natural isomorphism is useful.

Lemma 3.1.7. *Let $F, G : \mathbf{C} \to \mathbf{D}$ be functors and $\alpha : F \Rightarrow G$ a natural transformation. Then α is a natural isomorphism if and only if for all objects $A \in \mathbf{C}$, $\alpha_A : F(A) \to G(A)$ is an isomorphism in $\mathbf{D}$.* $\diamond$

As expected, when there is a natural isomorphism from F to G, we say that F and G are naturally isomorphic, and write $F \cong G$, $F(A) \cong G(A)$ *naturally in* $A \in \mathbf{C}$, or even $F(A) \cong_A G(A)$.

3.1.3 Representable Functors and the Yoneda Lemma

In this section, we recall a fundamental result of category theory, the Yoneda lemma. This lemma characterizes the natural transformations from a representable hom-functor to a presheaf, and it has as a consequence a representation that allows us to embed any category into the category of its presheaves. We begin by defining some of the structures just mentioned.

Definition 3.1.8. Let $\mathbf{C}$ be a category. A *presheaf* on $\mathbf{C}$ is a functor $\mathbf{C}^{op} \to \mathbf{Set}$. $\diamond$

Presheaves can be organized into categories.

Definition 3.1.9. Given a category $\mathbf{C}$, we define the *category of presheaves* on $\mathbf{C}$, sometimes written as $\widehat{\mathbf{C}}$, as the functor category $[\mathbf{C}^{op}, \mathbf{Set}]$ whose objects are presheaves on $\mathbf{C}$ and morphisms natural transformations between them. $\diamond$

Certain types of presheaves on a given category serve as faithful representatives of the objects in such category.

Definition 3.1.10. Given a locally small category $\mathbf{C}$ and an object $A \in \mathbf{C}$, we define the *contravariant representable hom-functor with representing object* A to be the presheaf

$$H_A : \mathbf{C}^{op} \to \mathbf{Set}$$

such that

- for every object $B \in \mathbf{C}$, $H_A(B) = \mathbf{C}(B, A)$, and
- for every morphism $f : B' \to B$ in $\mathbf{C}$,

  $$H_A(f) : \mathbf{C}(B, A) \to \mathbf{C}(B', A)$$

  is given by the assignment $g \mapsto g \circ f$ for all $g : B \to A$ in $\mathbf{C}(B, A)$. $\diamond$

Note that $H_A(f)$ is sometimes written as $\mathbf{C}(f, A)$ or $- \circ f$. Also, there is a covariant version of representable hom-functors defined in the natural way.

Definition 3.1.11. Let $\mathbf{C}$ be a locally small category. A functor $F : \mathbf{C}^{op} \to \mathbf{Set}$ is *representable* if $F \cong H_A$ for some object $A \in \mathbf{C}$. A *representation* of F is a choice of an object $A \in \mathbf{C}$ and a natural isomorphism between F and H_A. $\diamond$

We now present the main result of this section.

Theorem 3.1.12 (The Yoneda Lemma)**.** *Let $\mathbf{C}$ be a locally small category. Then there exists a bijective correspondence*

$$[\mathbf{C}^{op}, \mathbf{Set}](H_A, F) \cong F(A)$$

natural in $A \in \mathbf{C}$ and $F \in [\mathbf{C}^{op}, \mathbf{Set}]$. $\diamond$

The following functor yields a representation of $\mathbf{C}$ in $\widehat{\mathbf{C}}$.

Definition 3.1.13. Given a locally small category $\mathbf{C}$, we define the functor

$$\mathcal{Y} : \mathbf{C} \to [\mathbf{C}^{op}, \mathbf{Set}]$$

as follows:

- for every object $A \in \mathbf{C}$, let $\mathcal{Y}(A) = H_A$, and
- for every morphism $f : A \to A'$ in $\mathbf{C}$, let

  $$\mathcal{Y}(f) : H_A \Rightarrow H_{A'}$$

  be the natural transformation whose component at an object $B \in \mathbf{C}$ is given by the function $\mathcal{Y}(f)_B : H_A(B) = \mathbf{C}(B, A) \to H_{A'}(B) = \mathbf{C}(B, A')$ that sends a morphism $g : B \to A$ to the morphism $f \circ g : B \to A'$. $\diamond$

Corollary 3.1.14. *For any locally small category $\mathbf{C}$, the functor*

$$\mathcal{Y} : \mathbf{C} \to [\mathbf{C}^{op}, \mathbf{Set}]$$

is a full embedding, called the Yoneda embedding. $\diamond$

3.1.4 Limits and Colimits

So far in this chapter, we have mostly focused on structures that relate categories, namely, functors; in this section, we turn our attention to structures that can be built inside categories, namely, limits and colimits. These structures are of utmost importance as they capture many constructions in mathematics.

Typical examples of limits include the cartesian product of sets in the category **Set**, the direct product of groups in the category **Grp**, and the meet of a set in a poset $(A, \leq)$, when seen as a category.

Definition 3.1.15. Given categories $\mathbf{I}$ and $\mathbf{C}$, a *diagram* in $\mathbf{C}$ of shape $\mathbf{I}$ is a functor $D : \mathbf{I} \to \mathbf{C}$. $\diamond$

The general definition of limit is a follows.

Definition 3.1.16. Let $\mathbf{I}$ and $\mathbf{C}$ be categories, and $D : \mathbf{I} \to \mathbf{C}$ a diagram.

- A *cone* on D consists of an object $A \in \mathbf{C}$, called the *vertex* of the cone, along with a family

$$\{f_i : A \to D(i) : i \in \mathbf{I}_0\}$$

of morphisms in $\mathbf{C}$ such that for all morphisms $\alpha : i \to j$ in $\mathbf{I}$, the diagram

$$\begin{tikzcd} & A \arrow[dl, "f_i"'] \arrow[dr, "f_j"] & \\ D(i) \arrow[rr, "D(\alpha)"'] & & D(j) \end{tikzcd}$$

commutes.

- A *limit* of D is a cone on D with a vertex L and a family

$$\{p_i : L \to D(i) : i \in \mathbf{I}_0\}$$

of morphisms, called the *projections* of the limit, such that for every cone $(A, \{f_i : A \to D(i) : i \in \mathbf{I}_0\})$ on D, there exists a unique morphism $\overline{f} : A \to L$ such that for every $i \in \mathbf{I}_0$, $f_i = p_i \circ \overline{f}$. The vertex L is called the *limit object* of the limit cone $(L, \{p_i : L \to D(i) : i \in \mathbf{I}_0\})$ and is denoted by $\lim D$ or $\lim\limits_{\leftarrow \mathbf{I}} D$. $\diamond$

The dual notion of a limit is a colimit. The sum of sets is an example of a colimit in **Set**, as is the free product of groups in **Grp**, and the join of a set in a poset $(A, \leq)$. Here is the general definition.

Definition 3.1.17. Let $\mathbf{I}$ and $\mathbf{C}$ be categories, and $D : \mathbf{I} \to \mathbf{C}$ a diagram.

- A *cocone* on D consists of an object $A \in \mathbf{C}$, called the *vertex* of the cocone, along with a family

  $$\{f_i : D(i) \to A : i \in \mathbf{I}_0\}$$

  of morphisms in $\mathbf{C}$ such that for all morphisms $\alpha : i \to j$ in $\mathbf{I}$, the diagram

  $$\begin{tikzcd} & A & \\ D(i) \arrow[ur, "f_i"] \arrow[rr, "D(\alpha)"'] & & D(j) \arrow[ul, "f_j"'] \end{tikzcd}$$

  commutes.

- A *colimit* of D is a cocone on D with a vertex C and a family

  $$\{c_i : D(i) \to C : i \in \mathbf{I}_0\}$$

  of morphisms, called the *coprojections* of the colimit, such that for every cocone $(A, \{f_i : D(i) \to A : i \in \mathbf{I}_0\})$ on D, there exists a unique morphism $\overline{f} : C \to A$ such that for every $i \in \mathbf{I}_0$, $f_i = \overline{f} \circ c_i$. The vertex C is called the *colimit object* of the colimit cone $(C, \{c_i : D(i) \to C : i \in \mathbf{I}_0\})$ and is denoted by $\operatorname{colim} D$ or $\lim\limits_{\to\mathbf{I}} D$. ◇

3.1.5 Adjunctions

In this section, we introduce one of the fundamental notions of category theory, namely, adjunctions. Adjunctions are pervasive in mathematics as they reflect naturally occurring patterns. These patterns appear in several different, but equivalent, ways. Here we recall some of the main definitions and results about adjunctions as used in this thesis.

Definition 3.1.18. Given categories and functors $\begin{tikzcd} \mathbf{C} \arrow[r, shift left, "F"] & \mathbf{D} \arrow[l, shift left, "G"] \end{tikzcd}$, we say that F is a *left adjoint* to G, and G is a *right adjoint* to F, and write $F \dashv G$, if there is a family φ of isomorphisms

$$\varphi_{A,B} : \mathbf{C}(A, G(B)) \cong \mathbf{D}(F(A), B)$$

natural in $A \in \mathbf{C}$ and $B \in \mathbf{D}$. $\diamond$

An *adjunction* between F and G is a choice of such a natural isomorphism φ, and we write $\varphi : F \dashv G$, or $F \dashv G$ when φ is clear from the context. Given a morphism $f : A \to GB$ of $\mathbf{C}$, the morphism $\varphi_{A,B}(f) : F(A) \to B$ of $\mathbf{D}$ is called the *transpose* of f and is denoted by f^*. Similarly, given a morphism $g : F(A) \to B$ of $\mathbf{D}$, the morphism $\varphi^{-1}_{A,B}(g) : A \to G(B)$ of $\mathbf{C}$ is called the transpose of g and is denoted by g_*.

Forgetful functors usually have adjoints. For example, consider the category **Mon** of monoids and monoid homomorphisms. The forgetful functor $U : \mathbf{Mon} \to \mathbf{Set}$ has a left adjoint functor $F : \mathbf{Set} \to \mathbf{Mon}$ that sends each set X to the free monoid on X. Similarly, consider the category $\mathbf{Vect}_{\mathbb{C}}$ of vector spaces over $\mathbb{C}$ and linear transformations. The forgetful functor $U : \mathbf{Vect}_{\mathbb{C}} \to \mathbf{Set}$ has a left adjoint functor $F : \mathbf{Set} \to \mathbf{Vect}_{\mathbb{C}}$ that sends each set X to the complex vector space with basis X.

We now introduce some additional structures used in alternative definitions of adjunction. First, we have the unit and counit of an adjunction.

Definition 3.1.19. Given an adjunction $\begin{tikzcd} \mathbf{C} \arrow[r, shift left, "F"] \arrow[r, phantom, "\perp"] & \mathbf{D} \arrow[l, shift left, "G"] \end{tikzcd}$ whose natural isomorphism φ has components $\varphi_{A,B} : \mathbf{C}(A, G(B)) \to \mathbf{D}(F(A), B)$ for objects $A \in \mathbf{C}$ and $B \in \mathbf{D}$, we define

- the *unit* of the adjunction to be the natural transformation $\eta : 1_{\mathbf{C}} \Rightarrow G \circ F$ with components $\eta_A : A \to G(F(A))$ given by $\eta_A = \varphi^{-1}_{A,F(A)}(1_{F(A)})$ for each object $A \in \mathbf{C}$, and

- the *counit* of the adjunction to be the natural transformation $\epsilon : F \circ G \Rightarrow 1_{\mathbf{D}}$ with components $\epsilon_B : F(G(B)) \to B$ given by $\epsilon_B = \varphi_{G(B),B}(1_{G(B)})$ for each object $B \in \mathbf{D}$. $\diamond$

For example, consider the adjunction above, $\begin{tikzcd} \mathbf{Set} \arrow[r, shift left, "F"] \arrow[r, phantom, "\perp"] & \mathbf{Vect}_{\mathbb{C}} \arrow[l, shift left, "U"] \end{tikzcd}$. The components of its unit $\eta : 1_{\mathbf{Set}} \Rightarrow U \circ F$ are given by the functions $\eta_X : X \to U(F(X))$

that send each element x of a set X to itself while the components of its counit $\epsilon : F \circ U \Rightarrow 1_{\mathbf{Vect}_{\mathbb{C}}}$ are given by the linear transformations $\epsilon_V : F(U(V)) \to V$ that send each (finite) formal $\mathbb{C}$-linear sum $\sum_{v \in V} \alpha_v v$ in $F(U(V))$ to its evaluation in the complex vector space V.

The unit and counit satisfy some useful identities, which will be used in the second formulation of adjunctions in Theorem 3.1.23 below.

Lemma 3.1.20. *Given an adjunction* $\begin{tikzcd}\mathbf{C} \arrow[r, shift left, "F"] \arrow[r, phantom, "\perp"] & \mathbf{D} \arrow[l, shift left, "G"]\end{tikzcd}$ *with unit* η *and counit* ϵ*, then for all objects* $A \in \mathbf{C}$ *and* $B \in \mathbf{D}$*, the following diagrams commute:*

$$\begin{tikzcd}
F(A) \arrow[r, "F(\eta_A)"] \arrow[dr, "1_{F(A)}"'] & F(G(F(A))) \arrow[d, "\epsilon_{F(A)}"] & & G(B) \arrow[r, "\eta_{G(B)}"] \arrow[dr, "1_{G(B)}"'] & G(F(G(B))) \arrow[d, "G(\epsilon_B)"] \\
& F(A) & & & G(B)
\end{tikzcd} \tag{3.1}$$

◇

The identities (3.1) are called the *triangle identities*. We now introduce the notion of universal arrow. Universal arrows will be used in the third and last formulation of adjunction.

Definition 3.1.21. Given a functor $G : \mathbf{D} \to \mathbf{C}$ and an object $A \in \mathbf{C}$, a *universal arrow* from A to G is a pair $(F(A), \eta_A)$ consisting of an object $F(A) \in \mathbf{D}$ and a morphism $\eta_A : A \to G(F(A))$ of $\mathbf{C}$ such that for every pair (B, f) where B is an object in $\mathbf{D}$ and $f : A \to G(B)$ a morphism of $\mathbf{C}$, there is a unique morphism $f^* : F(A) \to B$ of $\mathbf{D}$ such that the following diagram commutes in $\mathbf{C}$:

$$\begin{tikzcd}
A \arrow[r, "\eta_A"] \arrow[dr, "f"'] & G(F(A)) \arrow[d, "G(f^*)"] \\
& G(B)
\end{tikzcd}$$

◇

Going back to the adjunction $\begin{tikzcd}\mathbf{Set} \arrow[r, shift left, "F"] \arrow[r, phantom, "\perp"] & \mathbf{Vect}_{\mathbb{C}} \arrow[l, shift left, "U"]\end{tikzcd}$, it is the case that for any set X, vector space V, and function $f : X \to U(V)$, there is a unique linear transformation $f^* : F(X) \to V$ such that the diagram

$$\begin{tikzcd}
X \arrow[r, "\eta_X"] \arrow[dr, "f"'] & U(F(X)) \arrow[d, "U(f^*)"] \\
& U(V)
\end{tikzcd}$$

commutes. This well-known fact precisely states that the component η_X of the unit of the adjunction is a universal arrow from X to U. Actually, this situation holds in general.

Lemma 3.1.22. *Given an adjunction* $\begin{tikzcd} \mathbf{C} \arrow[r, shift left, "F"] \arrow[r, phantom, "\perp"] & \mathbf{D} \arrow[l, shift left, "G"] \end{tikzcd}$ *and an object* $A \in \mathbf{C}$*. Then the unit component* $\eta_A : A \to G(F(A))$ *is a universal arrow from* A *to* G*.* $\diamond$

The following theorem shows that each of the two previous lemmas yields an equivalent definition of adjunction.

Theorem 3.1.23. *Given functors* $\begin{tikzcd} \mathbf{C} \arrow[r, shift left, "F"] & \mathbf{D} \arrow[l, shift left, "G"] \end{tikzcd}$ *, there is a one-to-one correspondence between:*

1. *adjunctions* $\varphi : F \dashv G$,
2. *natural transformations* $\eta : 1_{\mathbf{C}} \Rightarrow G \circ F$ *and* $\epsilon : F \circ G \Rightarrow 1_{\mathbf{D}}$ *satisfying the triangle identities, and*
3. *natural transformations* $\eta : 1_{\mathbf{C}} \Rightarrow G \circ F$ *such that* $\eta_A : A \to G(F(A))$ *is a universal arrow from* A *to* G *for every object* $A \in \mathbf{C}$. $\diamond$

We finish this section by recalling a result that will be useful in later chapters when dealing with the categorical semantics of Proto-Quipper-M. It tells us that left adjoints are *cocontinuous* functors while right adjoints are *continuous* functors.

Theorem 3.1.24. *Given an adjunction* $\begin{tikzcd} \mathbf{C} \arrow[r, shift left, "F"] \arrow[r, phantom, "\perp"] & \mathbf{D} \arrow[l, shift left, "G"] \end{tikzcd}$ *, then* F *preserves colimits and* G *preserve limits.* $\diamond$

3.1.6 Monads and Comonads

We finish Section 3.1 by introducing some relevant structures, namely, monads and comonads, also known as triples and cotriples, respectively. We can think of these structures as abstractions of universal algebras; however, they have found applications in fields beyond pure mathematics such as theoretical computer science and logic.

Monads and comonads are based on endofunctors and, as we will see below, they are closely related to adjoint functors.

Definition 3.1.25. Given a category $\mathbf{C}$, a *monad* on $\mathbf{C}$ consists of a endofunctor $T : \mathbf{C} \to \mathbf{C}$ equipped with a pair of natural transformations:

- $\mu : T^2 \Rightarrow T$, called the *multiplication*, and
- $\eta : 1_{\mathbf{C}} \Rightarrow T$, called the *unit*,

such that the following coherence conditions are satisfied:

1. *Associativity*:

$$\begin{tikzcd} T^3 \arrow[r, Rightarrow, "\mu T"] \arrow[d, Rightarrow, "T\mu"'] & T^2 \arrow[d, Rightarrow, "\mu"] \\ T^2 \arrow[r, Rightarrow, "\mu"'] & T \end{tikzcd}$$

2. *Unity*:

$$\begin{tikzcd} T \arrow[r, Rightarrow, "\eta T"] \arrow[dr, Rightarrow, "1_T"'] & T^2 \arrow[d, Rightarrow, "\mu"] & T \arrow[l, Rightarrow, "T\eta"'] \arrow[dl, Rightarrow, "1_T"] \\ & T & \end{tikzcd}$$

◇

As an example of a monad we have the endofunctor $T : \mathbf{Set} \to \mathbf{Set}$ that sends a set A to its *Kleene closure* A^* (that is, $T(A)$ is the set of strings over A) and a function $f : A \to B$ to the function $T(f) : T(A) \to T(B)$ that sends a string $(a_1, a_2, \ldots, a_n)$ to the string $(f(a_1), f(a_2), \ldots, f(a_n))$. Also, for each set A, $\eta_A : A \to T(A)$ is the function that sends an element a of A to the string (a), and $\mu_A : T^2(A) \to T(A)$ is the function that sends a string of strings $((a_{11}, \ldots, a_{1m}), (a_{21}, \ldots, a_{2n}), \ldots, (a_{l1}, \ldots, a_{lp}))$ to the concatenated string $(a_{11}, a_{12}, \ldots, a_{lp})$.

Examples of monads can be easily generated, as adjoint functors give rise to them:

Proposition 3.1.26. *Given an adjunction* $\begin{tikzcd} \mathbf{C} \arrow[r, shift left, "F"] \arrow[r, phantom, "\perp"] & \mathbf{D} \arrow[l, shift left, "U"] \end{tikzcd}$ *with unit* η *and counit* ϵ, *the endofunctor* $T := U \circ F$ *along with the unit* η *and the natural transformation* $\mu := U\epsilon F$ *yield a monad on* $\mathbf{C}$. ◇

For example, the free monoid adjunction $\mathbf{Set} \underset{U}{\overset{F}{\underset{\longleftarrow}{\overset{\longrightarrow}{\perp}}}} \mathbf{Mon}$ gives rise to the Kleene closure monad described above. Moreover, every monad arises from an adjunction:

Proposition 3.1.27. *Given a monad T on a category $\mathbf{C}$ with unit η and multiplication μ, there is a category $\mathbf{D}$ and an adjunction $\mathbf{C} \underset{U}{\overset{F}{\underset{\longleftarrow}{\overset{\longrightarrow}{\perp}}}} \mathbf{D}$ with unit η and counit ϵ that generates it.* $\diamond$

The dual notion of a monad is that of a comonad.

Definition 3.1.28. Given a category $\mathbf{D}$, a *comonad* on $\mathbf{D}$ consists of an endofunctor $K : \mathbf{D} \to \mathbf{D}$ equipped with a pair of natural transformations:

- $\delta : K \Rightarrow K^2$, called the *comultiplication*, and
- $\epsilon : K \Rightarrow 1_{\mathbf{D}}$, called the *counit*,

such that the following coherence conditions are satisfied:

1. *Coassociativity*:

$$\begin{tikzcd}
K^3 & K^2 \arrow[l, Rightarrow, "\delta K"'] \\
K^2 \arrow[u, Rightarrow, "K\delta"] & K \arrow[l, Rightarrow, "\delta"] \arrow[u, Rightarrow, "\delta"']
\end{tikzcd}$$

2. *Counity*:

$$\begin{tikzcd}
K & K^2 \arrow[l, Rightarrow, "\epsilon K"'] \arrow[r, Rightarrow, "K\epsilon"] & K \\
 & K \arrow[ul, Rightarrow, "1_K"] \arrow[u, Rightarrow, "\delta"'] \arrow[ur, Rightarrow, "1_K"'] &
\end{tikzcd}$$

$\diamond$

As with monads, adjunctions yield comonads:

Proposition 3.1.29. *Given an adjunction $\mathbf{C} \underset{U}{\overset{F}{\underset{\longleftarrow}{\overset{\longrightarrow}{\perp}}}} \mathbf{D}$ with unit η and counit ϵ, the endofunctor $K := F \circ U$ along with the counit ϵ and the natural transformation $\delta := F\eta U$ yield a comonad on $\mathbf{D}$.* $\diamond$

Also, every comonad arises from an adjunction:

Proposition 3.1.30. *Given a comonad K on a category $\mathbf{D}$ with counit ϵ and comultiplication δ, there is a category $\mathbf{C}$ and an adjunction $\begin{tikzcd} \mathbf{C} \arrow[r, shift left, "F"] \arrow[r, phantom, "\perp"] & \mathbf{D} \arrow[l, shift left, "U"] \end{tikzcd}$ with unit η and counit ϵ that generates it.* $\diamond$

Comonads play an important role in the categorical semantics of intuitionistic linear logic as they are used to construct models of intuitionistic logic inside those of intuitionistic linear logic. See [20], [23], [72], and Chapter 7 for more details.

3.2 Monoidal Category Theory

We may think of monoidal category theory as the extension of category theory with the idea of parallelism. Informally, a monoidal category is a category in which objects and morphisms can be combined in parallel. Such categories are particularly well suited to represent situations where objects can be interpreted as systems and morphisms as processes transforming such systems.

3.2.1 Monoidal Categories

We begin this section by formally defining monoidal categories.

Definition 3.2.1. A *monoidal category* is a category $\mathbf{M}$ equipped with the following data:

- a functor $\otimes : \mathbf{M} \times \mathbf{M} \to \mathbf{M}$, called the *tensor product*;
- an object $I \in \mathbf{M}$, called the *unit object*;
- a natural isomorphism α whose components $\alpha_{A,B,C} : (A \otimes B) \otimes C \to A \otimes (B \otimes C)$ for all objects $A, B, C \in \mathbf{M}$ are called the *associators*;
- a natural isomorphism λ whose components $\lambda_A : I \otimes A \to A$ for all objects $A \in \mathbf{M}$ are called the *left unitors*; and
- a natural isomorphism ρ whose components $\rho_A : A \otimes I \to A$ for all objects $A \in \mathbf{M}$ are called the *right unitors.*

This data must satisfy the *triangle identity*

$$\begin{tikzcd}
(A \otimes I) \otimes B \arrow[rr, "\alpha_{A,I,B}"] \arrow[dr, "\rho_A \otimes \mathrm{id}_B"'] & & A \otimes (I \otimes B) \arrow[dl, "\mathrm{id}_A \otimes \lambda_B"] \\
& A \otimes B &
\end{tikzcd}$$

for all objects $A, B \in \mathbf{M}$ as well as the *pentagon identity*

$$\begin{tikzcd}
(A \otimes (B \otimes C)) \otimes D \arrow[rr, "\alpha_{A,B \otimes C,D}"] & & A \otimes ((B \otimes C) \otimes D) \arrow[d, "\mathrm{id}_A \otimes \alpha_{B,C,D}"] \\
((A \otimes B) \otimes C) \otimes D \arrow[u, "\alpha_{A,B,C} \otimes \mathrm{id}_D"] \arrow[dr, "\alpha_{A \otimes B,C,D}"'] & & A \otimes (B \otimes (C \otimes D)). \\
& (A \otimes B) \otimes (C \otimes D) \arrow[ur, "\alpha_{A,B,C \otimes D}"'] &
\end{tikzcd}$$

for all objects $A, B, C, D \in \mathbf{M}$. ◇

As customary, we may drop the subscripts of objects and morphisms when no confusion arises. Some natural examples of monoidal categories are the category **Set**, with a tensor product given by the cartesian product of sets, and a unit object given by a singleton; and the category **FHilb**, with a tensor product given by the usual tensor product of Hilbert spaces, as defined in Section 2.1.3, and a unit object given by the set of complex numbers $\mathbb{C}$ when seen as a Hilbert space.

A *strict monoidal category* is a monoidal category in which all the associators and unitors are identities. For example, a commutative monoid when regarded as a discrete category yields a strict monoidal category in the natural way. Moreover, it can be shown that $\lambda_I = \rho_I$ holds in any monoidal category. A much deeper result is given by the *Coherence Theorem for Monoidal Categories*:

Theorem 3.2.2. *In a monoidal category* $\mathbf{M}$*, every well-formed equation built from* $\circ, \otimes, \mathrm{id}, \alpha, \alpha^{-1}, \lambda, \lambda^{-1}, \rho,$ *and* ρ^{-1} *holds.* ◇

Definition 3.2.3. A *symmetric monoidal category* (SMC) is a monoidal category $\mathbf{M}$ equipped with a natural isomorphism σ whose components $\sigma_{A,B} : A \otimes B \to B \otimes A$, called the *symmetrors*, for all objects $A, B \in \mathbf{M}$ satisfy the following coherence conditions:

1. *Compatibility with the associators*: The diagram

$$\begin{tikzcd}
& A \otimes (B \otimes C) \arrow[r, "\sigma_{A,B\otimes C}"] & (B \otimes C) \otimes A \arrow[dr, "\alpha_{B,C,A}"] & \\
(A \otimes B) \otimes C \arrow[ur, "\alpha_{A,B,C}"] \arrow[dr, "\sigma_{A,B}\otimes\mathrm{id}"'] & & & B \otimes (C \otimes A) \\
& (B \otimes A) \otimes C \arrow[r, "\alpha_{B,A,C}"'] & B \otimes (A \otimes C) \arrow[ur, "\mathrm{id}\otimes\sigma_{A,C}"'] &
\end{tikzcd}$$

   commutes for all objects $A, B, C \in \mathbf{M}$.

2. *Compatibility with the unitors*: The diagram

$$\begin{tikzcd}
I_{\mathbf{M}} \otimes A \arrow[rr, "\sigma_{I_{\mathbf{M}},A}"] \arrow[dr, "\lambda_A"'] & & A \otimes I_{\mathbf{M}} \arrow[dl, "\rho_A"] \\
& A &
\end{tikzcd}$$

   commutes for all objects $A \in \mathbf{M}$.

3. *Compatibility with the symmetrors*: The diagram

$$\begin{tikzcd}
A \otimes B \arrow[rr, "\sigma_{A,B}"] \arrow[dr, "\mathrm{id}"'] & & B \otimes A \arrow[dl, "\sigma_{B,A}"] \\
& A \otimes B &
\end{tikzcd}$$

   commutes for all objects $A, B \in \mathbf{M}$. ◇

Definition 3.2.4. A *symmetric monoidal closed category* (SMCC) is a symmetric monoidal category $\mathbf{M}$ such that for each object $B \in \mathbf{M}$, the functor $- \otimes B : \mathbf{M} \to \mathbf{M}$ has a specified right adjoint $B \multimap - : \mathbf{M} \to \mathbf{M}$. ◇

Definition 3.2.5. A *cartesian closed category* (CCC) is a symmetric monoidal closed category in which the tensor product is a categorical product. ◇

In a monoidal category, arrows with source the tensor unit I play the role of *states* or *points* in their respective targets. We can also think of such arrows $p : I \to A$ as preparing the initial fixed system I into a particular system p of type A.

Definition 3.2.6. Given a monoidal category $\mathbf{M}$, a *state* or *point* of an object A in $\mathbf{M}$ is a morphism of the form $s : I \to A$. ◇

In **FHilb**, the points of a Hilbert space $\mathscr{H}$ are the linear functions $\varphi : \mathbb{C} \to \mathscr{H}$; we can identify the point φ with the element h of $\mathscr{H}$ via $h = \varphi(1)$. In **Set**, the points of a set A are the functions $f : \{*\} \to A$; we can identify the point f with the element a of A via $a = f(*)$. In **Rel**, the points of a set A are the relations $R : \{*\} \to A$; we can identify the point R with the subset S of A of all elements related to $*$ according to R.

Definition 3.2.7. Given a monoidal category $\mathbf{M}$ and objects A, B in $\mathbf{M}$,

- a morphism $j : I \to A \otimes B$ is called a *joint state* of A and B;
- a joint state is a *product state*, or *separable*, if it is of the form

$$I \xrightarrow{\lambda_I^{-1}} I \otimes I \xrightarrow{a \otimes b} A \otimes B$$

  for some states $a : I \to A$ and $b : I \to B$; and
- a joint state is an *entangled* state if it is not a product state. ◇

In **FHilb**, joint states of Hilbert spaces $\mathscr{H}_1$ and $\mathscr{H}_2$ correspond to elements of $\mathscr{H}_1 \otimes \mathscr{H}_2$, product states correspond to elements of the form $h_1 \otimes h_2$, and entangled states correspond to elements of $\mathscr{H}_1 \otimes \mathscr{H}_2$ which cannot be written in this form. In **Set**, joint states of sets A and B correspond to elements of $A \times B$, product states correspond to elements $(a, b) \in A \times B$ coming from given $a \in A$ and $b \in B$, and there are no entangled states. In **Rel**, joint states of sets A and B correspond to subsets of $A \times B$, product states correspond to subsets $P \subseteq A \times B$ of the form $P = \{(a, b) \in A \times B : a \in Q \text{ and } b \in R\}$ for given states $Q \subseteq A$ and $R \subseteq B$, and entangled states correspond to subsets of $A \times B$ that are not of this form.

3.2.2 Monoidal Functors

Just as functors relate categories, monoidal functors relate monoidal categories.

Definition 3.2.8. Given monoidal categories $\mathbf{M}$ and $\mathbf{N}$, a (*lax*) *monoidal functor* (F, m_0, m) from $\mathbf{M}$ to $\mathbf{N}$ is a functor $F : \mathbf{M} \to \mathbf{N}$ equipped with

- a morphism

  $$m_0 : I_{\mathbf{N}} \to F(I_{\mathbf{M}})$$

  where $I_{\mathbf{M}}$ and $I_{\mathbf{N}}$ are the unit objects of $\mathbf{M}$ and $\mathbf{N}$, respectively, and

- a natural transformation m with components

  $$m_{A,B} : F(A) \otimes F(B) \to F(A \otimes B)$$

  for all objects $A, B \in \mathbf{M}$.

This data is required to satisfy the following coherence conditions:

1. *Compatibility with the associators*: The diagram

$$\begin{tikzcd}
(F(A) \otimes F(B)) \otimes F(C) \arrow[r, "\alpha_{F(A),F(B),F(C)}"] \arrow[d, "m_{A,B}\otimes\mathrm{id}"'] & F(A) \otimes (F(B) \otimes F(C)) \arrow[d, "\mathrm{id}\otimes m_{B,C}"] \\
F(A \otimes B) \otimes F(C) \arrow[d, "m_{A\otimes B,C}"'] & F(A) \otimes F(B \otimes C) \arrow[d, "m_{A,B\otimes C}"] \\
F((A \otimes B) \otimes C) \arrow[r, "F(\alpha_{A,B,C})"'] & F(A \otimes (B \otimes C))
\end{tikzcd}$$

   commutes for all objects $A, B, C \in \mathbf{M}$.

2. *Compatibility with the unitors*: The diagrams

$$\begin{tikzcd}
I_{\mathbf{N}} \otimes F(A) \arrow[r, "\lambda_{F(A)}"] \arrow[d, "m_0\otimes\mathrm{id}"'] & F(A) \arrow[dd, "\mathrm{id}"] \\
F(I_{\mathbf{M}}) \otimes F(A) \arrow[d, "m_{I,A}"'] & \\
F(I_{\mathbf{M}} \otimes A) \arrow[r, "F(\lambda_A)"'] & F(A)
\end{tikzcd}
\qquad
\begin{tikzcd}
F(A) \otimes I_{\mathbf{N}} \arrow[r, "\rho_{F(A)}"] \arrow[d, "\mathrm{id}\otimes m_0"'] & F(A) \arrow[dd, "\mathrm{id}"] \\
F(A) \otimes F(I_{\mathbf{M}}) \arrow[d, "m_{A,I}"'] & \\
F(A \otimes I_{\mathbf{M}}) \arrow[r, "F(\rho_A)"'] & F(A)
\end{tikzcd}$$

   commute for all objects $A \in \mathbf{M}$.

The morphism m_0 and the natural transformation m and its components are called the *mediating* or *structure morphisms* of F. If these morphisms are isomorphisms, then F is said to be a *strong monoidal functor*. If they are identities, then F is said to be a *strict monoidal functor*. To simplify notation, we usually write (F, m) instead of (F, m_0, m). ◇

The notion of monoidal functor can be naturally extended to contexts involving symmetric monoidal categories.

Definition 3.2.9. Given symmetric monoidal categories $\mathbf{M}$ and $\mathbf{N}$, a *symmetric monoidal functor* (F, m) from $\mathbf{M}$ to $\mathbf{N}$ is a monoidal functor $F : \mathbf{M} \to \mathbf{N}$ whose mediating morphisms are compatible with the symmetrors, that is, the diagram

$$\begin{tikzcd}
F(A) \otimes F(B) \arrow[r, "\sigma_{F(A),F(B)}"] \arrow[d, "m_{A,B}"'] & F(B) \otimes F(A) \arrow[d, "m_{B,A}"] \\
F(A \otimes B) \arrow[r, "F(\sigma_{B,A})"'] & F(B \otimes A)
\end{tikzcd}$$

commutes for all objects $A, B \in \mathbf{M}$. ◇

3.2.3 Monoidal Natural Transformations

Just as monoidal functors relate monoidal categories, monoidal natural transformations relate monoidal functors.

Definition 3.2.10. Given monoidal categories $\mathbf{M}$ and $\mathbf{N}$ and monoidal functors (F, m) and (G, n) from $\mathbf{M}$ to $\mathbf{N}$, a *monoidal natural transformation* η from (F, m) to (G, n) is a natural transformation $\eta : F \Rightarrow G$ whose components are compatible with the structure morphisms of F and G, that is, the diagrams

$$\begin{tikzcd}
F(A) \otimes F(B) \arrow[r, "\eta_A \otimes \eta_B"] \arrow[d, "m_{A,B}"'] & G(A) \otimes G(B) \arrow[d, "n_{A,B}"] \\
F(A \otimes B) \arrow[r, "\eta_{A \otimes B}"'] & G(A \otimes B)
\end{tikzcd}$$

and

$$\begin{tikzcd}
 & I_{\mathbf{N}} \arrow[dl, "m_0"'] \arrow[dr, "n_0"] & \\
F(I_{\mathbf{M}}) \arrow[rr, "\eta_{I_{\mathbf{M}}}"'] & & G(I_{\mathbf{M}})
\end{tikzcd}$$

commute for all objects $A, B \in \mathbf{M}$. ◇

3.2.4 Monoidal Adjunctions

There is a natural extension of the notion of adjunction to the context of (symmetric) monoidal categories.

Definition 3.2.11. Given (symmetric) monoidal categories $\mathbf{M}$ and $\mathbf{N}$, a (*symmetric*) *monoidal adjunction* between them is an ordinary adjunction in which both of the functors are (symmetric) monoidal functors and both the unit and the counit of the adjunction are monoidal natural transformations. ◇

There is a convenient characterization of symmetric monoidal adjunctions. We will use this result later to show that our categorical model of Proto-Quipper-M is actually an LNL model, defined in the next section, and so a model of intuitionistic linear logic.

Theorem 3.2.12. *Given symmetric monoidal categories* $\mathbf{C}$ *and* $\mathbf{D}$*, a symmetric monoidal functor* $(F, m) : \mathbf{C} \to \mathbf{D}$*, and a functor* $G : \mathbf{D} \to \mathbf{C}$ *such that* $F \dashv G$*. Then the following two statements are equivalent:*

1. *The adjunction* $F \dashv G$ *lifts to a symmetric monoidal adjunction* $(F, m) \dashv (G, n)$*.*

2. *The symmetric monoidal functor* (F, m) *is strong.* ◇

3.2.5 LNL Models and Monoidal Comonads

Relevant examples of monoidal adjunctions are given by the linear-non-linear models of Benton [17].

Definition 3.2.13. A *linear-non-linear model* (LNL model) consists of

- a cartesian closed category $\mathbf{C}$,
- a symmetric monoidal closed category $\mathbf{L}$, and
- a pair of symmetric monoidal functors $(F, m) : \mathbf{C} \to \mathbf{L}$ and $(G, n) : \mathbf{L} \to \mathbf{C}$ that form a symmetric monoidal adjunction with $(F, m) \dashv (G, n)$. $\diamond$

Comonads can be extended to the context of (symmetric) monoidal categories. These extensions have proved to be quite useful in the categorical interpretation of the "bang" modality ! of linear logic.

Definition 3.2.14. Given a (symmetric) monoidal category $\mathbf{L}$, a (*symmetric*) *monoidal comonad* on $\mathbf{L}$ consists of a (symmetric) monoidal endofunctor $(K, q) : \mathbf{L} \to \mathbf{L}$ equipped with a pair of monoidal natural transformations:

- $\delta : K \Rightarrow K^2$, called the *comultiplication*, and
- $\epsilon : K \Rightarrow 1_{\mathbf{L}}$, called the *counit*,

such that the following coherence conditions are satisfied:

1. *Coassociativity*:

$$\begin{tikzcd}
K^3 & K^2 \arrow[l, Rightarrow, "\delta K"'] \\
K^2 \arrow[u, Rightarrow, "K\delta"] & K \arrow[l, Rightarrow, "\delta"] \arrow[u, Rightarrow, "\delta"']
\end{tikzcd}$$

2. *Counity*:

$$\begin{tikzcd}
K & K^2 \arrow[l, Rightarrow, "\epsilon K"'] \arrow[r, Rightarrow, "K\epsilon"] & K \\
& K \arrow[ul, Rightarrow, "1_K"] \arrow[u, Rightarrow, "\delta"'] \arrow[ur, Rightarrow, "1_K"'] &
\end{tikzcd}$$

$\diamond$

As expected, LNL models yield symmetric monoidal comonads.

Lemma 3.2.15. *Given a LNL model* $\begin{tikzcd} \mathbf{C} \arrow[r, shift left, "F"] \arrow[r, phantom, "\perp"] & \mathbf{D} \arrow[l, shift left, "G"] \end{tikzcd}$ *with unit* η *and counit* ϵ*, the endofunctor* $! := F \circ G$ *along with the counit* ϵ *and the natural transformation* $\delta := F\eta G$ *yield a symmetric monoidal comonad.* $\diamond$

LNL models play a central role in the categorical semantics of the intuitionistic fragment of Girard's linear logic ([44], [45]) as they subsume several other well-known categorical models such as Lafont categories [64], Seely categories [93], and linear categories [20]. More on the categorical semantics of linear logic can be found in [72], [23], and [20].

Chapter 4

Towards a Quantum Circuit Description Language

Besides motivating the definition of Proto-Quipper-M, the purpose of this chapter is to introduce a first categorical model of the language. In Section 4.1, we present a toy presheaf category that is not yet a model of Proto-Quipper-M but is particularly well-suited to illustrate the distinction between parameters and states first discussed in Section 1.3. In Section 4.2, we generalize this cartesian category to a monoidal context to obtain a categorical structure in which Proto-Quipper-M can be soundly modeled. This model is further generalized later in the thesis. Since this chapter mainly serves as motivation for the structures introduced in the subsequent chapters, we only sketch some of the main ideas here and provide details later. Most of the material presented in this chapter first appeared in [89].

4.1 Modeling Parameters and States

In this section, we introduce a categorical model of the notions of parameter and state. Being cartesian, this structure may serve as a model for a classical circuit description language. However, as discussed in Section 4.2, this category is not suitable for modeling quantum circuits. Nonetheless, this model has a rich enough structure to illustrate several relevant constructions which will also be present in the more general models introduced later in the thesis.

4.1.1 A Presheaf Model

Presheaf categories were introduced in Chapter 3. These categories have a very rich structure which makes them ideal candidates for giving meaning to programming languages. Our first categorical model of parameters and states is one such category, namely, the category $\mathbf{Set}^{\mathbf{2}^{op}}$ described below.

Consider the two-object category $\mathbf{2}$ with objects 0 and 1, and unique non-identity morphism $u : 0 \to 1$, then the presheaf category $\mathbf{Set}^{\mathbf{2}^{op}} = [\mathbf{2}^{op}, \mathbf{Set}]$ has as objects functors of the form $A : \mathbf{2}^{op} \to \mathbf{Set}$ and as arrows natural transformations $f : A \to B$ between them. More concretely, we can think of an object as a triple $A = (A_0, A_1, a)$, where $a : A_1 \to A_0$ is a function, and of a morphism $f : A \to B$ as pair of functions f_0, f_1 making the following diagram commute:

$$\begin{tikzcd}
A_1 \arrow[r, "f_1"] \arrow[d, "a"'] & B_1 \arrow[d, "b"] \\
A_0 \arrow[r, "f_0"'] & B_0
\end{tikzcd} \tag{4.1}$$

For each $x \in A_0$, we define the *fiber* of A over x as $A_x = \{s \in A_1 \mid a(s) = x\}$. The elements of A_0 are called *parameters* and those of A_x *states*. An object $A = (A_0, A_1, a)$ naturally describes a family of sets $(A_x)_{x \in A_0}$ that uniquely determines it, up to isomorphism. As a consequence, there is a one-to-one correspondence between the elements of A_1 and the ordered pairs (x, s), called the *generalized elements* of A, where x is a parameter in A_0 and s is a state in the fiber of A over x.

We now have the following key observation:

> The commutativity of square (4.1) precisely models the feature that a state may depend on a parameter, but a parameter may not depend on a state.

To justify this, it suffices to examine the action of a morphism $f : A \to B$ on a generalized element (x, s) of A, for which we write $(y, t) = f(x, s)$. Let (x, s) be such a generalized element. So, $x \in A_0$ and $s \in A_x$. Let $y = f_0(x) \in B_0$ and $t = f_1(s) \in B_1$, we then have

$$\begin{aligned}
y &= f_0(x) \\
&= f_0(a(s)) \\
&= b(f_1(s)) \\
&= b(t),
\end{aligned}$$

where the second equation holds since $s \in A_x$ and the third since $f : A \to B$ is a natural transformation. So, $t \in B_y = B_{f_0(x)}$ and the restriction of f_1 to A_x yields a function $f_x : A_x \to B_{f_0(x)}$ that sends the state $s \in A_x$ to the state $t = f_x(s) \in B_{f_0(x)}$. Clearly, the state t depends on the parameter x (and the state s) and the parameter

$y = f_0(x)$ depends only on the parameter x. Hence, states may depend on parameters, but parameters may not depend on states.

4.1.2 Parameter and State Objects

The following example further illustrates how the category $\mathbf{Set}^{\mathbf{2}^{op}}$ models the distinction between parameters and states.

Example 4.1.1. Consider the objects **bool** and **bit** of $\mathbf{Set}^{\mathbf{2}^{op}}$ defined below:

$$\mathbf{bool} = \begin{tikzcd} 2 \arrow[d, "\mathrm{id}"] \\ 2 \end{tikzcd} \qquad \mathbf{bit} = \begin{tikzcd} 2 \arrow[d, "v"] \\ 1 \end{tikzcd}$$

where $2 = \{0, 1\}$, id is the identity function on 2, $1 = \{*\}$, and v is the unique function from the 2-element set 2 to the singleton 1.

Note that **bool** has $(0, 0)$ and $(1, 1)$ as generalized elements, which we may think of as `false` and `true`, respectively, while the generalized elements of **bit** are $(*, 0)$ and $(*, 1)$, which we may think of as simply 0 and 1, respectively. Intuitively, a boolean is only a parameter. This is reflected in the generalized elements $(0, 0)$ and $(1, 1)$ of **bool** by the fact that a state in a generalized element is the same as its corresponding parameter, and so we may think of these elements as having no state at all, just parameters. Similarly, intuitively, a bit is only a state. This is reflected by all the generalized elements of **bit** having the same parameter, and so we may think of these elements as having no parameter at all, just states. Not only do the generalized elements of **bool** and **bit** formally look like parameters and states, respectively, but most importantly, they do behave as such, as illustrated below.

Note that there is a morphism $f : \mathbf{bool} \to \mathbf{bit}$ that assigns 1 to `true` and 0 to `false` as shown by the commutative diagram

$$\begin{tikzcd} 2 \arrow[r, "f_1 = \mathrm{id}"] \arrow[d, "\mathrm{id}"'] & 2 \arrow[d, "v"] \\ 2 \arrow[r, "f_0 = v"'] & 1. \end{tikzcd}$$

However, no morphism $g : \mathbf{bit} \to \mathbf{bool}$ assigning `true` to 1 and `false` to 0 exists since the diagram

$$\begin{tikzcd}
2 \arrow[r, "g_1=\mathrm{id}"] \arrow[d, "v"'] & 2 \arrow[d, "\mathrm{id}"] \\
1 \arrow[r, "g_0"'] & 2
\end{tikzcd}$$

does not commute for any function $g_0 : 1 \to 2$. Thus, a bit can be initialized by a boolean, but a boolean cannot be initialized by a bit, that is, a state may depend on a parameter, but a parameter may not depend on a state. $\diamond$

This example suggests the following definition.

Definition 4.1.2. Given an object $A = (A_0, A_1, a) \in \mathbf{Set}^{\mathbf{2}^{op}}$, we say that

- A is a *parameter object* if $A_1 = A_0$ and $a = \mathrm{id}_{A_1}$, and
- A is a *state object* or *simple* if $A_0 \cong 1$. $\diamond$

4.1.3 A Language for Parameters and States

The simply typed lambda calculus can be given semantics in a cartesian closed category. Since our presheaf model $\mathbf{Set}^{\mathbf{2}^{op}}$ is cartesian closed as well as cocomplete, we can then interpret the simply typed lambda calculus with, for example, base types **bool** and **bit**, constants **true**, **false** : **bool**, operations **and** : **bit** × **bit** → **bit**, **not** : **bit** → **bit**, and **init** : **bool** → **bit**, sum types, and inductive data types such as **list**(A) and **nat**. Thus the simply typed lambda calculus can be extended with these types to yield a categorically sound calculus. In this language, we can describe parametrized circuits. For example, the term $f = \lambda b.\lambda x.$ if b then **not** x else x, of type f : **bool** → **bit** → **bit**, yields the *NOT* gate when applied to the boolean **true**, but the identity gate when applied to **false**. Note, moreover, that the term $g = \lambda x.$ if x then **true** else **false**, of type g : **bit** → **bool**, cannot be soundly interpreted in our presheaf model, as shown in Example 4.1.1, and so the type system forbids such terms.

4.2 Modeling Families of Circuits

As we have seen, parameters and states can be modeled in $\mathbf{Set}^{\mathbf{2}^{op}}$. However, families of quantum circuits cannot be adequately modeled in this category since it is cartesian. In particular,

- for each object $A \in \mathbf{Set}^{\mathbf{2}^{op}}$, including state objects, there is a *diagonal morphism* $\Delta_A : A \to A \times A$.

This is indeed problematic if we want to model quantum circuits since in this category we can duplicate quantum states, violating the no-cloning property. Nonetheless, this model serves as a bridge towards an appropriate generalization to the quantum setting. Moreover, such a generalization is rather natural. From Section 4.1.1, an object $A = (A_0, A_1, a) \in \mathbf{Set}^{\mathbf{2}^{op}}$ can be seen as a pair $(A_0, (A_x)_{x \in A_0})$ and a morphism $f : A \to B$ as a pair $(f_0, (f_x)_{x \in A_0})$, where f_0 is a function between parameter sets, and for each parameter x in A_0, f_x is a function from a fiber of A to a fiber of B. We can now generalize this to a monoidal setting by assuming each A_x to be an object of a fixed monoidal category, and each $f_x : A_x \to B_{f_0(x)}$ to be a morphism of this category.

4.2.1 Generalized Circuits

As an initial step in the design of a general circuit description language, we elaborate on what we mean by a circuit. Instead of specifying circuits as graphical representations of sequences of gates, as is usually done, we adopt a more abstract point of view and define a circuit as nothing more than a morphism in a given symmetric monoidal category $\mathbf{M}$. We call these morphisms *generalized circuits*. Thus, in this context, we can think of Proto-Quipper-M as a language for describing families of morphisms in $\mathbf{M}$. Such morphisms can be considered as concrete data that our language will be able to further manipulate.

We extend the category of generalized circuits $\mathbf{M}$ by fully embedding it into some symmetric monoidal closed, product-complete category $\overline{\mathbf{M}}$. This can always be done, for example, by the Yoneda embedding, but since our results do not depend on any particular construction of such an extension, but only on the fact that it exists, we won't elaborate on it any further.

Since $\overline{\mathbf{M}}$ is monoidal closed, it has higher-order objects such as $((A \otimes B) \multimap C) \multimap D$ which serve as support for the more concrete operations of $\mathbf{M}$. Thus, we think of the category $\overline{\mathbf{M}}$ as an abstract category of states, in contrast to the category $\mathbf{M}$ which we consider as a concrete category of circuits.

4.2.2 A Model of Proto-Quipper-M.

We now formally introduce parameters in our construction to get a model of Proto-Quipper-M. We will denote this model by $\overline{\overline{\mathbf{M}}}$.

Definition 4.2.1. Given a symmetric monoidal category $\mathbf{M}$ with a symmetric monoidal closed, product-complete extension $\overline{\mathbf{M}}$, the category $\overline{\overline{\mathbf{M}}}$ consists of:

- objects given by pairs of the form $A = (A_0, (A_x)_{x \in A_0})$, where A_0 is a set and $(A_x)_{x \in A_0}$ is an A_0-indexed family of objects of $\overline{\mathbf{M}}$, and
- morphisms $f : (A_0, (A_x)_{x \in A_0}) \to (B_0, (B_y)_{y \in B_0})$ given by pairs of the form $(f_0, (f_x)_{x \in A_0})$, where $f_0 : A_0 \to B_0$ is a function and $(f_x)_{x \in A_0}$ is an A_0-indexed family of morphisms $f_x : A_x \to B_{f_0(x)}$ of $\overline{\mathbf{M}}$. $\diamond$

This construction is also known as the "families" construction. As before, the first component of an object $A = (A_0, (A_x)_{x \in A_0})$, namely A_0, is called the *parameter set* of A and its elements are called *parameters*. For each parameter x in A_0, A_x is called the *fiber* of A over x. The parameter and state objects of $\overline{\overline{\mathbf{M}}}$ are defined analogously to those of $\mathbf{Set}^{\mathbf{2}^{op}}$:

Definition 4.2.2. Given a category $\overline{\overline{\mathbf{M}}}$ as above and an object $A = (A_0, (A_x)_{x \in A_0})$ of $\overline{\overline{\mathbf{M}}}$, we say that

- A is a *parameter object* if each of its fibers is the tensor unit $I_{\mathbf{M}}$, that is, if $A = (A_0, (I_x)_{x \in A_0})$ where $I_x = I_{\mathbf{M}}$ for each $x \in A_0$,
- A is a *state object* or *simple* if $A_0 \cong 1$, and
- A is an *M-object* if each of its fibers belongs to $\mathbf{M}$. $\diamond$

Not only does $\overline{\overline{\mathbf{M}}}$ have coproducts, but also a symmetric monoidal closed structure:

Proposition 4.2.3. *The category $\overline{\overline{\mathbf{M}}}$ has coproducts. The initial object is given by $0 = (\emptyset, \emptyset)$, where the first component denotes the empty set and the second the empty family. Binary coproducts are given by $A + B = (A_0 + B_0, (C_i)_{i \in A_0 + B_0})$, where $A_0 + B_0 = \{(0, x) \mid x \in A_0\} \cup \{(1, y) \mid y \in B_0\}$ is the disjoint union of sets, and $C_{(0,x)} = A_x$ and $C_{(1,y)} = B_y$. Infinite coproducts are defined in a similar fashion.* $\diamond$

Proposition 4.2.4. *The category $\overline{\overline{\mathbf{M}}}$ has a symmetric monoidal closed structure given by:*

$$\begin{array}{lcl} I & = & (1, (I)) \\ A \otimes B & = & (A_0 \times B_0, (A_x \otimes B_y)_{(x,y) \in A_0 \times B_0}) \\ A \multimap B & = & (A_0 \to B_0, (C_f)_{f \in A_0 \to B_0}), \end{array}$$

where $C_f = \prod_{x \in A_0}(A_x \multimap B_{f(x)})$, $A_0 \to B_0$ is the set of all functions from A_0 to B_0, and $A_x \multimap B_y$ is an exponential object in the product-complete monoidal closed category $\overline{\mathbf{M}}$. ◇

As we will see in Chapter 7, the category $\overline{\overline{\mathbf{M}}}$ has indeed the required structure to be a model of Proto-Quipper-M. It is moreover a part of a linear-non-linear adjunction, and so a model of intuitionistic linear logic.

Chapter 5

The Proto-Quipper-M Language

In this chapter, we introduce the syntax, type system, and operational semantics of Proto-Quipper-M. Proto-Quipper-M is defined relative to a notion of circuit. We elaborate on this notion before introducing the syntax of the language.

5.1 Circuits, Wires, and Labels

Categorical considerations influenced the design of Proto-Quipper-M, and thus it is no surprise that even at the level of syntax, categorical entities appear. This will become more evident as we proceed.

5.1.1 Circuits and Wire Types

Recall from Section 4.2.1 that a circuit is simply a morphism in a symmetric monoidal category; we call such morphisms *generalized circuits*. From now on, we assume that a locally small symmetric monoidal category $\mathbf{M}$ of generalized circuits has been given along with a set $\mathcal{W}$ of *wire types* and an interpretation function $[\![-]\!]_{\mathcal{W}} : \mathcal{W} \to \mathbf{M}_0$, assigning an object of $\mathbf{M}$ to every wire type. We denote wire types by Greek letters such as α, β, and γ. We will usually assume that the set of wire types contains types relevant to quantum circuits such as the type of bits, **bit**, and the type of qubits, **qubit**. However, the set $\mathcal{W}$ is arbitrary in general.

5.1.2 Labeled Circuits

To enable our programming language to manipulate morphisms of $\mathbf{M}$, we equip $\mathbf{M}$ with an additional *labeling structure*, which we now describe. Let $\mathcal{L}$ be a countably infinite set of *labels*. We denote labels by letters such as k and ℓ.

Definition 5.1.1. A *label context* is a function from a finite set of labels to wire types. We write label contexts as $Q = \ell_1 : \alpha_1, \ldots, \ell_n : \alpha_n$. Expressions of the form

$\ell : \alpha$ in a label context are called *label type declarations*. $\diamond$

To give meaning to the terms of Proto-Quipper-M, we begin by interpreting the label contexts of the language.

Definition 5.1.2. To each label context $Q = \ell_1 : \alpha_1, \ldots, \ell_n : \alpha_n$, we associate an object of $\mathbf{M}$, namely, $[\![Q]\!]_s = [\![\alpha_1]\!]_{\mathcal{W}} \otimes \ldots \otimes [\![\alpha_n]\!]_{\mathcal{W}}$. It will be useful to think of such an object as a particular bracketing of the tensor product of $[\![\alpha_1]\!]_{\mathcal{W}}, \ldots, [\![\alpha_n]\!]_{\mathcal{W}}$. In case $Q = \emptyset$, we set $[\![Q]\!]_s = I_{\mathbf{M}}$. $\diamond$

The subscript s in the semantic brackets $[\![-]\!]_s$ indicates that this function assigns meaning to entities involving *simple M-types*. Simple M-types are introduced in Definition 5.2.1 below.

We can now define the category of labeled circuits.

Definition 5.1.3. Let $\mathbf{M}$, $\mathcal{W}$, $\mathcal{L}$, and the function $[\![-]\!]_{\mathcal{W}} : \mathcal{W} \to \mathbf{M}_0$ be given as above. The symmetric monoidal category $\mathbf{M}_{\mathcal{L}}$ of *labeled circuits* is defined as follows:

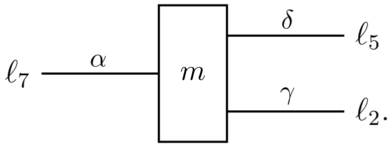
- The objects of $\mathbf{M}_{\mathcal{L}}$ are label contexts.
- A morphism $f : Q \to Q'$ in $\mathbf{M}_{\mathcal{L}}$ is a morphism $[\![f]\!]_s : [\![Q]\!]_s \to [\![Q']\!]_s$ in $\mathbf{M}$.

Identities and composition are defined in the unique way to make the function $[\![-]\!]_s$ into a functor $[\![-]\!]_s : \mathbf{M}_{\mathcal{L}} \to \mathbf{M}$. Note that this functor is full and faithful. Moreover, we endow $\mathbf{M}_{\mathcal{L}}$ with the unique (up to natural isomorphism) symmetric monoidal structure making this functor strong symmetric monoidal. $\diamond$

5.1.3 Visualizing Labeled Circuits

Note that if label contexts Q and Q' have disjoint domains, then $Q \otimes Q' \cong Q, Q'$, i.e., we can think of the tensor product of disjoint label contexts as given by their union. We will make use of this observation throughout, as two label contexts can always be made disjoint up to isomorphism by renaming their labels. We can depict the morphisms of $\mathbf{M}_{\mathcal{L}}$ as generalized circuits with labeled and typed inputs and outputs. For example, if $m : Q \to Q'$ is one such morphism with $Q = \ell_7 : \alpha$ and

$Q' = \ell_2 : \gamma, \ell_5 : \delta$, then we can visualize m as follows:



5.2 The Syntax of Proto-Quipper-M

In this section, we give the formal definition of Proto-Quipper-M and its type system. Proto-Quipper-M is a simply-typed lambda calculus that supports not only usual programming constructs such as disjoint unions, pairs, and higher-order functions, but also linearity and circuit description. References for the lambda calculus include [96], [14], and [45]. More on linearity can be found in [106], [44], [93], and [72]. For an introduction to circuit description languages, see [52]. Accessible accounts of type systems are given in [27] and [83]. We now present each of the main components of our language.

5.2.1 Types

A type is a property of a program construct that reflects how the construct should be interpreted. It is sometimes useful to think of a type as a set of related values that a program may compute.

Definition 5.2.1. The *types* of Proto-Quipper-M are defined by

$$A, B \ ::= \ \alpha \mid 0 \mid A + B \mid I \mid A \otimes B \mid A \multimap B \mid !A \mid \mathbf{nat} \mid \mathbf{list}\ A \mid \mathrm{Circ}(T, U).$$

Here, α ranges over the set $\mathcal{W}$ of *wire types*, and T and U are *simple M-types* defined by

$$T, U \ ::= \ \alpha \mid I \mid T \otimes U.$$

The subset of *parameter types* is defined by

$$P, R \ ::= \ 0 \mid P + R \mid I \mid P \otimes R \mid !A \mid \mathbf{nat} \mid \mathbf{list}\ P \mid \mathrm{Circ}(T, U).$$

The type system of Proto-Quipper-M is based on propositional intuitionistic linear logic (see [23], [19], [20], [18] or [12]), thus the similarity between the types of the system and the connectives of the logic. As expected, $A + B$ is the type of sums, $A \otimes B$ the type of pairs, and $A \multimap B$ the type of functions from A to B. 0 is the empty type and I the unit type. Also, **list** A and **nat** are the usual inductive types of lists of elements of A and natural numbers, respectively.

The type $!A$ represents the duplicable and discardable elements of type A while the simple M-types represent the inputs and outputs of circuits. $\mathrm{Circ}(T, U)$ is the type of circuits with inputs of type T and outputs of type U. The parameter types represent values that are known at circuit generation time; we usually think of parameters as classical data.

5.2.2 Terms and Values

We now introduce the language constructs.

Definition 5.2.2. The *terms* of Proto-Quipper-M are defined by

$$\begin{array}{rcl} M, N & ::= & x \mid \ell \mid c \mid \mathrm{let}\ x = M \ \mathrm{in}\ N \mid \\ & & \Box_A M \mid \mathrm{left}_{A,B}\, M \mid \mathrm{right}_{A,B}\, M \mid \mathrm{case}\ M \ \mathrm{of}\ \{\mathrm{left}\, x \to N \mid \mathrm{right}\, y \to P\} \mid \\ & & * \mid M; N \mid \langle M, N \rangle \mid \mathrm{let}\ \langle x, y \rangle = M \ \mathrm{in}\ N \mid \lambda x^A.M \mid MN \mid \\ & & \mathrm{lift}\ M \mid \mathrm{force}\ M \mid \mathrm{box}_T\, M \mid \mathrm{apply}(M, N) \mid (\vec{\ell}, C, \vec{\ell'}). \end{array}$$

Here, x ranges over a countably infinite set of *variables*, ℓ ranges over the set $\mathcal{L}$ of *labels*, and c ranges over a given set of *constants*. We assume that these sets are mutually disjoint. The subset of *label tuples* is defined by

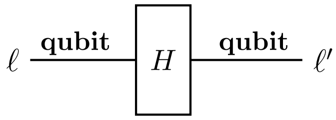
$$\vec{k}, \vec{\ell} \ ::= \ * \mid k \mid \langle \vec{k}, \vec{\ell} \rangle$$

and the subset of *values* by

$$V, W \ ::= \ x \mid \ell \mid c \mid \mathrm{left}_{A,B}\, V \mid \mathrm{right}_{A,B}\, V \mid * \mid \langle V, W \rangle \mid \lambda x^A.M \mid \mathrm{lift}\ M \mid (\vec{\ell}, C, \vec{\ell'}).$$

Now we present the intended meaning of the terms. A label is a pointer to an input or output of a labeled circuit, i.e., a wire identifier. Constants have a fixed interpretation in suitable categories and some represent operations that can be performed by a quantum device. It will be useful to assume that the category of generalized circuits contains some distinguished objects and morphisms that may serve in the interpretation of relevant gates; for example, $H : \mathbf{qubit} \to \mathbf{qubit}$ for the

Hadamard gate and $CNOT$: $\mathbf{qubit} \otimes \mathbf{qubit} \rightarrow \mathbf{qubit} \otimes \mathbf{qubit}$ for the controlled-not gate. We can visualize the Hadamard gate as the following labeled circuit, for example:



The interpretation of most other terms is as in the standard lambda calculus. We illustrate with the following cases. See Table 5.2 for more details. For example, the term let $x = M$ in N defines the program that first evaluates M, then assigns that value to x, and finally runs N. The term $\lambda x^A.M$ stands for the function that maps x to M while the term MN represents the application of the function M to N. More unusual terms include $(\vec{\ell}, C, \vec{\ell'})$, which represents a *boxed circuit*, i.e., a quantum circuit regarded as Proto-Quipper-M data of type $\mathrm{Circ}(T, U)$. In particular, C is a labeled circuit and $\vec{\ell}$ and $\vec{\ell'}$ are label tuples that provide an interface between the inputs and outputs of C and the types T and U, respectively. This will be made more precise in the typing rules below. We further observe that terms of the form $(\vec{\ell}, C, \vec{\ell'})$ are not intended to be written by the users of the programming language as no concrete syntax for the circuits C has been provided. Instead, such terms represent values internally computed during the evaluation of a Proto-Quipper-M program. On the other hand, we do have a syntax for a term of the form $\mathrm{box}_T\, M$ which represents a boxed circuit of type $\mathrm{Circ}(T, U)$ generated by a duplicable function of type $!(T \multimap U)$ denoted by the term M.

The term lift M stands for a duplicable version of the program M provided that M has no quantum data embedded. Thus, if M has type A, lift M has type $!A$. Conversely, given a term M of type $!A$, the term force M has type A. This conveys the idea that a program M that can be used an arbitrary number of times can also be used exactly once. Finally, $\mathrm{apply}(M, N)$ is the program that evaluates the term M to a boxed circuit and then appends such circuit to the current circuit at the wires specified by the term N.

We now introduce the notions of free variable and capture-avoiding substitution.

Definition 5.2.3. The set of *free variables* of a term M, denoted by $FV(M)$, is defined by

- $FV(x) = x$,
- $FV(\ell) = \emptyset$,
- $FV(c) = \emptyset$,
- $FV(\text{let } x = M \text{ in } N) = FV(M) \cup (FV(N) \backslash \{x\})$,
- $FV(\Box_A M) = FV(M)$,
- $FV(\text{left}_{A,B}\, M) = FV(M)$,
- $FV(\text{right}_{A,B}\, M) = FV(M)$,
- $FV(\text{case } M \text{ of } \{\text{left}\, x \to N \mid \text{right}\, y \to P\}) = FV(M) \cup (FV(N) \backslash \{x\}) \cup (FV(P) \backslash \{y\})$,
- $FV(*) = \emptyset$,
- $FV(M; N) = FV(M) \cup FV(N)$,
- $FV(\langle M, N \rangle) = FV(M) \cup FV(N)$,
- $FV(\text{let } \langle x, y \rangle = M \text{ in } N) = FV(M) \cup (FV(N) \backslash \{x, y\})$,
- $FV(\lambda x^A.M) = FV(M) \backslash \{x\}$,
- $FV(MN) = FV(M) \cup FV(N)$,
- $FV(\text{lift } M) = FV(M)$,
- $FV(\text{force } M) = FV(M)$,
- $FV(\text{box}_T\, M) = FV(M)$,
- $FV(\text{apply}(M, N)) = FV(M) \cup FV(N)$, and
- $FV((\vec{\ell}, C, \vec{\ell'})) = \emptyset$. ◇

Note that this definition reflects our intended interpretation of labels as pointers or wire identifiers rather than as variables; this is accomplished by letting $FV(\ell) = \emptyset$. In particular, there are no binders for labels.

The notions of *α-equivalence* and *bound variable* are just as in the standard lambda calculus, see [96] or [14]. So terms that differ only in the name of their bound variables are identified; for example, $\lambda y^A.\langle y, y\rangle$ and $\lambda z^A.\langle z, z\rangle$ are considered equal. In the sequel, we will assume *Barendregt's variable convention*, i.e., we will always assume without loss of generality that bound variables have been renamed to be distinct from any other variables in a given context.

Definition 5.2.4. The capture-avoiding *substitution* of a term N for the free occurrences of a variable x in a term M, written as $M[N/x]$, is defined as follows:

- $x[N/x] = N$,
- $y[N/x] = y \quad \text{if } x \neq y$,
- $\ell[N/x] = \ell$,
- $c[N/x] = c$,
- $(\text{let } y = R \text{ in } S)[N/x] = (\text{let } y = R[N/x] \text{ in } S[N/x])$,
- $(\Box_A R)[N/x] = \Box_A R[N/x]$,
- $(\text{left}_{A,B}\, R)[N/x] = \text{left}_{A,B}\, R[N/x]$,
- $(\text{right}_{A,B}\, R)[N/x] = \text{right}_{A,B}\, R[N/x]$,
- $(\text{case } Q \text{ of } \{\text{left}\, y \to R \mid \text{right}\, z \to S\})[N/x] =$

  $\text{case } Q[N/x] \text{ of } \{\text{left}\, y \to R[N/x] \mid \text{right}\, z \to S[N/x]\}$,
- $*[N/x] = *$,
- $(R; S)[N/x] = R[N/x]; S[N/x]$,
- $(\langle R, S\rangle)[N/x] = \langle R[N/x], S[N/x]\rangle$,
- $(\text{let } \langle y, z\rangle = R \text{ in } S)[N/x] = (\text{let } \langle y, z\rangle = R[N/x] \text{ in } S[N/x])$,
- $(\lambda y^A.R)[N/x] = \lambda y^A.R[N/x]$,
- $(RS)[N/x] = R[N/x]\, S[N/x]$,

- $(\text{lift } R)[N/x] = \text{lift } R[N/x]$,
- $(\text{force } R)[N/x] = \text{force } R[N/x]$,
- $(\text{box}_T\, R)[N/x] = \text{box}_T\, R[N/x]$,
- $(\text{apply}(R, S))[N/x] = \text{apply}(R[N/x], S[N/x])$, and
- $(\vec{\ell}, C, \vec{\ell}')[N/x] = (\vec{\ell}, C, \vec{\ell}')$. ◇

5.2.3 The Type System of Proto-Quipper-M

Since terms are intended to represent computations that can be performed by a computer, it is natural to expect that not every term will be meaningful in every context. For example, the term M should denote a pair of terms for the term $\text{let } \langle x, y\rangle = M \text{ in } N$ to be meaningful; the term M should denote a function for the term MN to be meaningful; and the term M should represent a circuit and N should represent a corresponding label tuple for the term $\text{apply}(M, N)$ to be meaningful.

Moreover, the meaning of a term depends on the environment in which it occurs. Syntactic representations of such environments are captured by the notions of variable and label contexts.

Definition 5.2.5. A *variable context* is a function from a finite set of variables to types. We write a variable context as $\Gamma = x_1 : A_1, \dots, x_n : A_n$. Expressions of the form $x : A$ in a variable context are called *variable type declarations.* A variable context in which all types are parameter types is called a *parameter context*; we usually denote parameter contexts by Φ. ◇

Given contexts Γ_1 and Γ_2, we write Γ_1, Γ_2 for their union provided their domains are disjoint. We adopt the following convention and say that contexts Γ_1 and Γ_2 are disjoint and write $\Gamma_1 \cap \Gamma_2 = \emptyset$ whenever their domains are. For variable contexts Γ_1 and Γ_2 such that $\Gamma_1 = \Gamma_1', \Phi$ and $\Gamma_2 = \Gamma_2', \Phi$, where Φ is a parameter context, Γ_1' and Γ_2' are arbitrary variable contexts, and Γ_1', and Γ_2' and Φ are mutually disjoint, we define $\Gamma_1 \cup \Gamma_2 = \Gamma_1', \Gamma_2', \Phi$. Otherwise, $\Gamma_1 \cup \Gamma_2$ is undefined.

Proper circuit construction depends on adequate wiring of circuits. For this, we introduce the notion of valid label tuple.

$$\frac{}{\emptyset \vdash_{\mathcal{L}} * : I}\ (*)_{\mathcal{L}} \qquad \frac{}{\ell : \alpha \vdash_{\mathcal{L}} \ell : \alpha}\ (\mathit{label})_{\mathcal{L}} \qquad \frac{Q \vdash_{\mathcal{L}} \vec{\ell} : T \quad Q' \vdash_{\mathcal{L}} \vec{\ell'} : U}{Q, Q' \vdash_{\mathcal{L}} \langle \vec{\ell}, \vec{\ell'} \rangle : T \otimes U}\ (\mathit{pair})_{\mathcal{L}}$$

Table 5.1: Rules for Label Tuple Judgements.

Definition 5.2.6. A *label tuple judgement* is an expression of the form

$$Q \vdash_{\mathcal{L}} \vec{\ell} : T$$

where Q is a label context, $\vec{\ell}$ is a label tuple, and T is a simple M-type. A label tuple judgement is *valid* if it can be derived from the rules in Table 5.1. We denote its derivation by $\mathcal{D} : Q \vdash_{\mathcal{L}} \vec{\ell} : T$, or even $\mathcal{D}$ when no confusion arises. A label tuple is valid if it occurs in a valid label tuple judgement. Note that the $(\mathit{pair})_{\mathcal{L}}$ rule has the side condition that Q and Q' must have disjoint domains. ◇

It is not hard to see that in a valid label tuple judgement $Q \vdash_{\mathcal{L}} \vec{\ell} : T$, the context Q is uniquely determined by $\vec{\ell}$ and T, and conversely, T is uniquely determined by $\vec{\ell}$ and Q.

To avoid meaningless terms such as $\vec{\ell}x$ and $\langle M, N \rangle(\lambda x^A.R)$, we endow Proto-Quipper-M with a strong type system. This system assigns a type to each well-formed term and enjoys the *error-freeness property*: Executing a well-typed program never yields a run-time error. This and other interesting properties of the type system of Proto-Quipper-M will be explored in the following chapters.

Definition 5.2.7. A *typing judgement* is an expression of the form

$$\Gamma; Q \vdash M : A$$

where Γ is a variable context, Q is a label context, M is a term, and A is a type. A typing judgement is *valid* if it can be derived from the rules in Table 5.2. A rule containing a variable context of the form $\Gamma_1 \cup \Gamma_2$ in the conclusion assumes that $\Gamma_1 \cup \Gamma_2$ is defined, and similarly for contexts of the form Q_1, Q_2. A term M is *well-typed* of type A in the context $\Gamma; Q$ if the typing judgement $\Gamma; Q \vdash M : A$ is valid. ◇

Many of the typing rules of Proto-Quipper-M are similar to those of a linear lambda calculus ([19], [20], [23]); we now highlight some of the differences. In the

$$\frac{}{\Phi, x : A; \emptyset \vdash x : A}\ (\mathit{var}) \qquad \frac{}{\Phi; \ell : \alpha \vdash \ell : \alpha}\ (\mathit{label}) \qquad \frac{}{\Phi; \emptyset \vdash c : A_c}\ (\mathit{const})$$

$$\frac{\Gamma_1; Q_1 \vdash M : A \quad \Gamma_2, x : A; Q_2 \vdash N : B}{\Gamma_1 \cup \Gamma_2; Q_1, Q_2 \vdash \mathrm{let}\ x = M\ \mathrm{in}\ N : B}\ (\mathit{let})$$

$$\frac{\Gamma; Q \vdash M : 0}{\Gamma; Q \vdash \Box_A M : A}\ (\mathit{initial})$$

$$\frac{\Gamma; Q \vdash M : A}{\Gamma; Q \vdash \mathrm{left}_{A,B}\, M : A + B}\ (\mathit{left}) \qquad \frac{\Gamma; Q \vdash M : B}{\Gamma; Q \vdash \mathrm{right}_{A,B}\, M : A + B}\ (\mathit{right})$$

$$\frac{\Gamma_1; Q_1 \vdash M : A + B \quad \Gamma_2, x : A; Q_2 \vdash N : C \quad \Gamma_2, y : B; Q_2 \vdash P : C}{\Gamma_1 \cup \Gamma_2; Q_1, Q_2 \vdash \mathrm{case}\ M\ \mathrm{of}\ \{\mathrm{left}\, x \to N \mid \mathrm{right}\, y \to P\} : C}\ (\mathit{case})$$

$$\frac{}{\Phi; \emptyset \vdash * : I}\ (*) \qquad \frac{\Gamma_1; Q_1 \vdash M : I \quad \Gamma_2; Q_2 \vdash N : A}{\Gamma_1 \cup \Gamma_2; Q_1, Q_2 \vdash M; N : A}\ (\mathit{seq})$$

$$\frac{\Gamma_1; Q_1 \vdash M : A \quad \Gamma_2; Q_2 \vdash N : B}{\Gamma_1 \cup \Gamma_2; Q_1, Q_2 \vdash \langle M, N \rangle : A \otimes B}\ (\mathit{pair})$$

$$\frac{\Gamma_1; Q_1 \vdash M : A \otimes B \quad \Gamma_2, x : A, y : B; Q_2 \vdash N : C}{\Gamma_1 \cup \Gamma_2; Q_1, Q_2 \vdash \mathrm{let}\ \langle x, y \rangle = M\ \mathrm{in}\ N : C}\ (\mathit{let\text{-}pair})$$

$$\frac{\Gamma, x : A; Q \vdash M : B}{\Gamma; Q \vdash \lambda x^A . M : A \multimap B}\ (\mathit{abs}) \qquad \frac{\Gamma_1; Q_1 \vdash M : A \multimap B \quad \Gamma_2; Q_2 \vdash N : A}{\Gamma_1 \cup \Gamma_2; Q_1, Q_2 \vdash MN : B}\ (\mathit{app})$$

$$\frac{\Phi; \emptyset \vdash M : A}{\Phi; \emptyset \vdash \mathrm{lift}\ M : !A}\ (\mathit{lift}) \qquad \frac{\Gamma; Q \vdash M : !A}{\Gamma; Q \vdash \mathrm{force}\ M : A}\ (\mathit{force})$$

$$\frac{\Gamma; Q \vdash M : !(T \multimap U)}{\Gamma; Q \vdash \mathrm{box}_T\, M : \mathrm{Circ}(T, U)}\ (\mathit{box})$$

$$\frac{\Gamma_1; Q_1 \vdash M : \mathrm{Circ}(T, U) \quad \Gamma_2; Q_2 \vdash N : T}{\Gamma_1 \cup \Gamma_2; Q_1, Q_2 \vdash \mathrm{apply}(M, N) : U}\ (\mathit{apply})$$

$$\frac{Q \vdash_{\mathcal{L}} \vec{\ell} : T \quad Q' \vdash_{\mathcal{L}} \vec{\ell'} : U \quad C \in \mathbf{M}_{\mathcal{L}}(Q, Q')}{\Phi; \emptyset \vdash (\vec{\ell}, C, \vec{\ell'}) : \mathrm{Circ}(T, U)}\ (\mathit{circ})$$

Table 5.2: The Typing Rules of Proto-Quipper-M.

typing judgements of Proto-Quipper-M variables and labels are kept separate, which allows the language to handle circuit wiring properly. Also, several of the typing rules generate judgements with variable contexts of the form $\Gamma_1 \cup \Gamma_2$; as a result, only variables of parameter type can be duplicated or discarded, as can be seen in the following example.

Example 5.2.8. Consider the following type derivation where we have omitted annotating the applications of the (*var*) rule for readability

$$\dfrac{\dfrac{}{x : P; \emptyset \vdash * : I}\,(*) \qquad \dfrac{\dfrac{\dfrac{}{x : P; \emptyset \vdash x : P} \quad \dfrac{}{y : A; \emptyset \vdash y : A}}{x : P, y : A, \emptyset \vdash \langle x, y\rangle : P \otimes A}\,(\mathit{pair}) \quad \dfrac{}{x : P; \emptyset \vdash x : P}}{x : P, y : A; \emptyset \vdash \langle\langle x, y\rangle, x\rangle : (P \otimes A) \otimes P}\,(\mathit{pair})}{x : P,\, y : A;\, \emptyset \vdash \langle *, \langle\langle x, y\rangle, x\rangle\rangle : I \otimes ((P \otimes A) \otimes P)}\,(\mathit{pair})$$

Note that the conclusion is valid provided that P is a parameter type for then each variable context in the derivation is well defined. As a result, the variable x can be duplicated, as in the middle application of the (*pair*) rule; discarded, as in the application of the $(*)$ rule; and both duplicated and discarded, as in the bottom application of the (*pair*) rule. $\diamond$

We call a variable of parameter type a *parametric* variable and a variable whose type is not known to be a parameter type a *linear* variable. The design of our type system ensures that linear variables and labels are used exactly once, whereas parametric variables may be used any number of times or not at all, as shown above. This is different from other approaches. For example, Benton's LNL term assignment [17] has judgements of the form $\Theta; \Gamma \vdash M : A$, where Θ is a *classical* context, akin to our parameter contexts, and Γ is a linear context. However, these contexts are syntactically distinct and contain two different sorts of variables: classical variables such as x, y, and z, and linear variables such as a, b, and c. As a consequence, two different sorts of lambda abstractions are present. In contrast, the parameter types of Proto-Quipper-M form a subset of the set of all types and there is only one kind of variable and lambda abstraction. This makes the Proto-Quipper-M approach more flexible as it allows the programmer to mix parameters and states more freely. Also, Proto-Quipper-M does not need explicit "copy" and "discard" term operators like the linear lambda calculus of Benton et al. does [20].

Moreover, the strong type system of Proto-Quipper-M guarantees the absence of run-time errors as discussed in Chapter 6. In particular, the type system enforces the no-cloning property of quantum information as it forbids the derivation of terms of the form $\lambda x^{\mathbf{qubit}}.\langle x, x\rangle : \mathbf{qubit} \multimap \mathbf{qubit} \otimes \mathbf{qubit}$, for example.

The (*const*) rule assumes that each constant has a fixed type A_c. For example, we could have a constant $c_n : \mathbf{nat}$ for each number $n \in \mathbb{N}$ and a constant succ : $\mathbf{nat} \multimap \mathbf{nat}$ for the successor function. The typing rule (*lift*) ensures that only terms with no reference to non-duplicable entities such as linear variables and labels can generate duplicable values of exponential type. We illustrate the meaning of the (*circ*) rule in the example below.

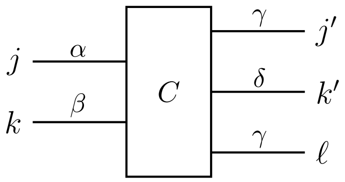
Example 5.2.9. Consider the following rule instance:

$$\frac{Q \vdash_{\mathcal{L}} \langle j, k\rangle : \alpha \otimes \beta \quad Q' \vdash_{\mathcal{L}} \langle\langle j', k'\rangle, \ell\rangle : (\gamma \otimes \delta) \otimes \gamma \quad C \in \mathbf{M}_{\mathcal{L}}(Q, Q')}{\Phi; \emptyset \vdash (\langle j, k\rangle, C, \langle\langle j', k'\rangle, \ell\rangle) : \mathrm{Circ}(\alpha \otimes \beta, (\gamma \otimes \delta) \otimes \gamma)} \; (\mathit{circ})$$

Assuming $C \in \mathbf{M}_{\mathcal{L}}(Q, Q')$, the conclusion is valid if and only if the label tuple judgements $Q \vdash_{\mathcal{L}} \langle j, k\rangle : \alpha \otimes \beta$ and $Q' \vdash_{\mathcal{L}} \langle\langle j', k'\rangle, \ell\rangle : (\gamma \otimes \delta) \otimes \gamma$ are valid. This indeed occurs if and only if

$$Q = j : \alpha,\ k : \beta$$
$$Q' = j' : \gamma,\ k' : \delta,\ \ell : \gamma.$$

Thus, the morphism $C : Q \to Q'$ of $\mathbf{M}_{\mathcal{L}}$ is a labeled circuit of the form



which yields a generalized circuit $[\![\alpha]\!] \otimes [\![\beta]\!] \to [\![\gamma]\!] \otimes [\![\delta]\!] \otimes [\![\gamma]\!]$ when interpreted in the appropriate category. In the end, the goal of the valid label tuple judgements is to properly identify particular inputs and outputs of C with particular occurrences of wire types α, β and γ in the simple-M types $\alpha \otimes \beta$ and $(\gamma \otimes \delta) \otimes \gamma$, respectively. ⋄

5.3 Operational Semantics

So far, we have only informally described the intended behavior of Proto-Quipper-M terms. We now give the language a more rigorous mathematical foundation and computational meaning by endowing it with a *big-step operational semantics*, through which the *overall* execution of a program is described directly rather than by translation as in categorical semantics. Big-step operational semantics, also known as *natural semantics* and *evaluation semantics*, was introduced by Gilles Khan in the 1980s ([31], [59]). This approach was derived from the pioneering work of Gordon Plotkin on *structural operational semantics* ([84], [85]), through which the *individual steps* of the computations that take place during the execution of a program are described. More on operational semantics can be found in [55], [77], and [83]. For more comprehensive treatments of the semantics of programming languages, see [49] or [108].

To construct the operational semantics of Proto-Quipper-M, we introduce two useful relations: an *evaluation relation*, which describes the evolution of correct programs, and an *error relation*, which captures the ill-behaved ones.

5.3.1 Evaluation Relation

First, we present the main component of the operational semantics: the *configuration*.

Definition 5.3.1. A *configuration* is a pair (C, M) where C is a labeled circuit and M is a term. If the term $M = V$ is a value, then we call (C, V) a *value configuration*. $\diamond$

We can think of C as the circuit under construction when the program M is executed.

Definition 5.3.2. The *evaluation relation* of Proto-Quipper-M, denoted by $\Downarrow$, is the relation on the set of configurations generated by the rules in Table 5.3. We usually write $(C, M) \Downarrow (C', V)$ for $((C, M), (C', V)) \in \Downarrow$. $\diamond$

Intuitively, the meaning of $(C, M) \Downarrow (C', V)$ is as follows: when the term M is executed in the context of the partially constructed circuit C, it generates a circuit C', by possibly appending some gates to C, along with a value V. Several of the rules involve terms of the form $N[V/x]$, which, as we recall, denotes capture-avoiding substitution as in Definition 5.2.4. For the most part, the evaluation rules involving terms that have a type annotation, such as $\Box_A M$, $\lambda x^A.M$, and $\mathrm{left}_{A,B}\, M$, do not take into account the types when they are applied. However, there is one exception:

$$\frac{}{(C,\ell)\Downarrow(C,\ell)}\ (\textit{label})_{\Downarrow} \qquad \frac{}{(C,c)\Downarrow(C,c)}\ (\textit{const})_{\Downarrow}$$

$$\frac{(C,M)\Downarrow(C',V) \quad (C',N[V/x])\Downarrow(C'',W)}{(C,\text{let } x=M \text{ in } N)\Downarrow(C'',W)}\ (\textit{let})_{\Downarrow}$$

$$\frac{(C,M)\Downarrow(C',V)}{(C,\text{left}\,M)\Downarrow(C',\text{left}\,V)}\ (\textit{left})_{\Downarrow} \qquad \frac{(C,M)\Downarrow(C',V)}{(C,\text{right}\,M)\Downarrow(C',\text{right}\,V)}\ (\textit{right})_{\Downarrow}$$

$$\frac{(C,M)\Downarrow(C',\text{left}\,V) \quad (C',N[V/x])\Downarrow(C'',W)}{(C,\text{case } M \text{ of } \{\text{left}\,x\to N \mid \text{right}\,y\to P\})\Downarrow(C'',W)}\ (\textit{case-left})_{\Downarrow}$$

$$\frac{(C,M)\Downarrow(C',\text{right}\,V) \quad (C',P[V/y])\Downarrow(C'',W)}{(C,\text{case } M \text{ of } \{\text{left}\,x\to N \mid \text{right}\,y\to P\})\Downarrow(C'',W)}\ (\textit{case-right})_{\Downarrow}$$

$$\frac{}{(C,*)\Downarrow(C,*)}\ (*)_{\Downarrow} \qquad \frac{(C,M)\Downarrow(C',*) \quad (C',N)\Downarrow(C'',W)}{(C,M;N)\Downarrow(C'',W)}\ (\textit{seq})_{\Downarrow}$$

$$\frac{(C,M)\Downarrow(C',V) \quad (C',N)\Downarrow(C'',V')}{(C,\langle M,N\rangle)\Downarrow(C'',\langle V,V'\rangle)}\ (\textit{pair})_{\Downarrow}$$

$$\frac{(C,M)\Downarrow(C',\langle V,V'\rangle) \quad (C',N[V/x,V'/y])\Downarrow(C'',W)}{(C,\text{let}\,\langle x,y\rangle=M \text{ in } N)\Downarrow(C'',W)}\ (\textit{let-pair})_{\Downarrow}$$

$$\frac{}{(C,\lambda x.M)\Downarrow(C,\lambda x.M)}\ (\textit{abs})_{\Downarrow}$$

$$\frac{(C,M)\Downarrow(C',\lambda x.M') \quad (C',N)\Downarrow(C'',V) \quad (C'',M'[V/x])\Downarrow(C''',W)}{(C,MN)\Downarrow(C''',W)}\ (\textit{app})_{\Downarrow}$$

$$\frac{}{(C,\text{lift } M)\Downarrow(C,\text{lift } M)}\ (\textit{lift})_{\Downarrow} \qquad \frac{(C,M)\Downarrow(C',\text{lift } M') \quad (C',M')\Downarrow(C'',V)}{(C,\text{force } M)\Downarrow(C'',V)}\ (\textit{force})_{\Downarrow}$$

$$\frac{(C,M)\Downarrow(C',\text{lift } N) \quad \textit{freshlabels}(T)=(Q,\vec{\ell}) \quad (\text{id}_Q,N\vec{\ell})\Downarrow(D,\vec{\ell}')}{(C,\text{box}_T\,M)\Downarrow(C',(\vec{\ell},D,\vec{\ell}'))}\ (\textit{box})_{\Downarrow}$$

$$\frac{(C,M)\Downarrow(C',(\vec{\ell},D,\vec{\ell}')) \quad (C',N)\Downarrow(C'',\vec{k}) \quad \textit{append}(C'',\vec{k},\vec{\ell},D,\vec{\ell}')=(C''',\vec{k}')}{(C,\text{apply}(M,N))\Downarrow(C''',\vec{k}')}\ (\textit{apply})_{\Downarrow}$$

$$\frac{}{(C,(\vec{\ell},D,\vec{\ell}'))\Downarrow(C,(\vec{\ell},D,\vec{\ell}'))}\ (\textit{circ})_{\Downarrow}$$

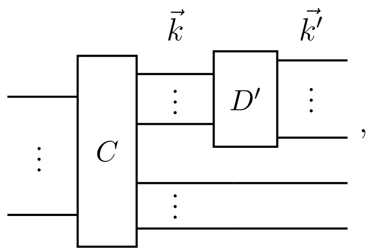
Table 5.3: The Evaluation Relation of Proto-Quipper-M.

the annotation T in the term $\mathrm{box}_T\, M$ does have a run-time effect. For example, if $T = \mathbf{qubit}$, then we get the identity circuit on one qubit when boxing the identity function, as in $\mathrm{box}_T(\lambda x^T.x)$, but we get the identity circuit on two qubits when $T = \mathbf{qubit} \otimes \mathbf{qubit}$.

The evaluation of Proto-Quipper-M terms follows a *call-by-value* reduction strategy. Roughly speaking, to apply a function, it is necessary that all its arguments (subterms) be evaluated first. Most of the rules of the language are standard, except for the $(\mathit{box})_{\Downarrow}$ and $(\mathit{apply})_{\Downarrow}$ rules. However, these two rules perform one of the most distinctive tasks of Proto-Quipper-M, namely, circuit construction.

The $(\mathit{box})_{\Downarrow}$ rule uses an ancillary map *freshlabels* that assigns to each simple M-type T an ordered pair $(Q, \vec{\ell})$, where Q is a label context and $\vec{\ell}$ is a label tuple such that $Q \vdash_{\mathcal{L}} \vec{\ell} : T$ is valid, and such that the labels in $\vec{\ell}$ are *fresh*, i.e., that they have not occurred in any other term used so far. Under these circumstances, Q and $\vec{\ell}$ are unique up to renaming of fresh labels. Now the $(\mathit{box})_{\Downarrow}$ rule is to be understood as follows. To execute the program $\mathrm{box}_T\, M$ in the context of circuit C, start by running the term M in the context of circuit C to generate a circuit C' and a value lift N, where N is a term of type $T \multimap U$. Apply the *freshlabels* function to T to obtain a label context Q and a label tuple $\vec{\ell}$ such that $Q \vdash_{\mathcal{L}} \vec{\ell} : T$ is valid. Now execute the program $N\vec{\ell}$ in the context of the identity circuit on Q to generate a circuit D along with a label tuple $\vec{\ell'}$. Form the value configuration $(C', (\vec{\ell}, D, \vec{\ell'}))$. This is the result of the execution of the program $\mathrm{box}_T\, M$ in the context of the circuit C.

Before we proceed with the $(\mathit{apply})_{\Downarrow}$ rule, we introduce a useful equivalence relation on terms of the form $(\vec{\ell}, D, \vec{\ell'})$, where, as usual, $\vec{\ell}$ and $\vec{\ell'}$ are label tuples and D is a labeled circuit. We say that terms $(\vec{\ell}, D, \vec{\ell'})$ and $(\vec{k}, D', \vec{k'})$ are equivalent, and write $(\vec{\ell}, D, \vec{\ell'}) \cong (\vec{k}, D', \vec{k'})$, if they are equal up to renaming of labels. Similarly to the $(\mathit{box})_{\Downarrow}$ rule, the $(\mathit{apply})_{\Downarrow}$ rule uses an ancillary map *append* that we now describe. For any labeled circuits $C : Q_0 \to Q_1$ and $D : Q_2 \to Q_3$, and label tuples $\vec{k}$, $\vec{\ell}$, and $\vec{\ell'}$, we define the operation $\mathit{append}(C, \vec{k}, \vec{\ell}, D, \vec{\ell'})$ as follows. First, verify that the outputs of C contain the labels in $\vec{k}$ and that the labels occurring in the inputs and outputs of D are precisely those occurring in $\vec{\ell}$ and $\vec{\ell'}$, respectively. Then, choose a labeled circuit D' and a fresh label tuple $\vec{k'}$ such that $(\vec{k}, D', \vec{k'}) \cong (\vec{\ell}, D, \vec{\ell'})$. Finally, output the configuration $(C', \vec{k'})$, where C' is the labeled circuit



provided that the composition is defined. Note that there are several situations in which $append(C, \vec{k}, \vec{\ell}, D, \vec{\ell'})$ is not defined, each yielding a run-time error. For example, when the shapes of $\vec{k}$ and $\vec{\ell}$ are distinct, when $\vec{k}$, $\vec{\ell}$, or $\vec{\ell'}$ are not compatible with the outputs of C or the inputs or outputs of D', respectively, or when the wire types of the outputs of C do not agree with those of the corresponding inputs of D'.

We now describe the $(apply)_{\Downarrow}$ rule. To execute the program apply(M, N) in the context of the circuit C, start by running the term M in the context of the circuit C to generate a circuit C' and a boxed circuit $(\vec{\ell}, D, \vec{\ell'})$. Now execute the program N in the context of the circuit C' to generate a circuit C'' along with a label tuple $\vec{k}$. Apply the *append* function to $(C'', \vec{k}, \vec{\ell}, D, \vec{\ell'})$ to wire the circuit C'' to the circuit D through the interface provided by the label tuples $\vec{k}$ and $\vec{\ell}$, as described above, to obtain the configuration $(C''', \vec{k'})$. This configuration is the result of the execution of the program apply(M, N) in the context of the circuit C.

5.3.2 Run-Time Errors

Up to this point, we have mostly dealt with well-behaved configurations; now we turn our attention to those with less desirable behaviors. As hinted in Section 5.2.3, the main purpose of the type system of Proto-Quipper-M is to avoid the occurrence of run-time errors during the execution of a program. There are several types of run-time errors, for example:

- *Run-time type errors.* The most common kind of error that a type system aims to avoid. These include evaluating a let statement such as let $\langle x, y\rangle = M$ in N, when M is not a pair, or executing an application MN, where M is not a function.

- *Label or unbound variable errors.* These include trying to append a gate through a wire ℓ that is not part of the output of the circuit under construction, or trying to use a variable x that has not been declared.

- *Deleting or cloning errors.* These types of errors are rather characteristic of quantum programming languages, as they usually represent violations of the no-deleting or no-cloning properties of quantum mechanics. For example, trying to discard or duplicate a *linear* variable.

When designing a programming language, specifying what constitutes a run-time error may be subtle. Informally, an error is a configuration (C, M) that is not the left-hand side of any evaluation rule—for then, the evaluation would get "stuck" as there is nothing for (C, M) to evaluate to whenever such a configuration is reached. However, defining this precisely is not entirely obvious. For example, consider the configuration $(C, M; N)$. Even though this is the left-hand side of an evaluation rule, namely $(\mathit{seq})_{\Downarrow}$, the rule only applies if M evaluates to a value of the form $*$. If M evaluates to a different kind of value, the rule does not apply, which must be an error. Still, we cannot simply define (C, M) to be an error configuration if there is no value configuration (C', V) such that $(C, M) \Downarrow (C', V)$, because non-terminating configurations are not in the domain of $\Downarrow$ even though they may not be errors. (Note that the version of Proto-Quipper-M defined in this thesis does not have non-terminating programs; however, these could be included by extending the language with recursive functions, for example.) And so, to deal with this issue, we endow Proto-Quipper-M with a formal notion of error.

Definition 5.3.3. The *error relation* of Proto-Quipper-M, denoted also by $\Downarrow$, is given by the set of ordered pairs of the form $((C, M), \mathrm{Error})$ derived from the error-generation rules in Table 5.4 and the error-propagation rules in Table 5.5 and Table 5.6. As usual, we write $(C, M) \Downarrow \mathrm{Error}$ for $((C, M), \mathrm{Error}) \in \Downarrow$. $\diamond$

The error-generation rules tell us precisely under what circumstances a Proto-Quipper-M run-time error is produced, while the error-propagation rules tell us that errors are global, i.e., if an error occurs at any point during the execution of a program, then the error immediately propagates to the top level.

$$\frac{}{(C, x) \Downarrow \text{Error}}\ (\textit{var-err-g})_{\Downarrow} \qquad \frac{(C, M) \Downarrow (C', V)}{(C, \Box_A M) \Downarrow \text{Error}}\ (\textit{initial-err-g})_{\Downarrow}$$

$$\frac{(C, M) \Downarrow (C', W) \text{ where } W \text{ is not of the form left}\, V \text{ or right}\, V}{(C, \text{case } M \text{ of } \{\text{left}\, x \to N \mid \text{right}\, y \to P\}) \Downarrow \text{Error}}\ (\textit{case-err-g})_{\Downarrow}$$

$$\frac{(C, M) \Downarrow (C', V) \text{ where } V \neq *}{(C, M; N) \Downarrow \text{Error}}\ (\textit{seq-err-g})_{\Downarrow}$$

$$\frac{(C, M) \Downarrow (C', W) \text{ where } W \text{ is not of the form } \langle V, V' \rangle}{(C, \text{let}\, \langle x, y \rangle = M \text{ in } N) \Downarrow \text{Error}}\ (\textit{let-pair-err-g})_{\Downarrow}$$

$$\frac{(C, M) \Downarrow (C', V) \text{ where } V \text{ is not of the form } \lambda x^A.M'}{(C, MN) \Downarrow \text{Error}}\ (\textit{app-err-g})_{\Downarrow}$$

$$\frac{(C, M) \Downarrow (C', V) \text{ where } V \text{ is not of the form lift } M'}{(C, \text{force } M) \Downarrow \text{Error}}\ (\textit{force-err-g})_{\Downarrow}$$

$$\frac{(C, M) \Downarrow (C', V) \text{ where } V \text{ is not of the form lift } N}{(C, \text{box}_T\, M) \Downarrow \text{Error}}\ (\textit{box-err-g-1})_{\Downarrow}$$

$$\frac{(C, M) \Downarrow (C', \text{lift } N) \quad \textit{freshlabels}(T) = (Q, \vec{\ell}) \quad (\text{id}_Q, N\vec{\ell}) \Downarrow (D, V) \text{ where } V \neq \vec{\ell}'}{(C, \text{box}_T\, M) \Downarrow \text{Error}}\ (\textit{box-err-g-2})_{\Downarrow}$$

$$\frac{(C, M) \Downarrow (C', V) \text{ where } V \text{ is not of the form } (\vec{\ell}, D, \vec{\ell}')}{(C, \text{apply}(M, N)) \Downarrow \text{Error}}\ (\textit{apply-err-g-1})_{\Downarrow}$$

$$\frac{(C, M) \Downarrow (C', (\vec{\ell}, D, \vec{\ell}')) \quad (C', N) \Downarrow (C'', V) \text{ where } V \neq \vec{k}}{(C, \text{apply}(M, N)) \Downarrow \text{Error}}\ (\textit{apply-err-g-2})_{\Downarrow}$$

$$\frac{(C, M) \Downarrow (C', (\vec{\ell}, D, \vec{\ell}')) \quad (C', N) \Downarrow (C'', \vec{k}) \quad \textit{append}(C'', \vec{k}, \vec{\ell}, D, \vec{\ell}') \text{ undefined}}{(C, \text{apply}(M, N)) \Downarrow \text{Error}}\ (\textit{apply-err-g-3})_{\Downarrow}$$

Table 5.4: Error-Generation Rules of Proto-Quipper-M.

5.3.3 Big-Step Operational Semantics

Given the evaluation and error relations of the previous section, we can now equip Proto-Quipper-M with an operational semantics.

Definition 5.3.4. The *big-step operational semantics* of Proto-Quipper-M, also denoted by $\Downarrow$, is given by the union of the evaluation and error relations defined above. $\diamond$

We note that when restricted to value configurations (C, V) where the value V is closed (i.e., $FV(V) = \emptyset$), the operational semantics is a reflexive relation:

$$\frac{(C, M) \Downarrow \text{Error}}{(C, \text{let } x = M \text{ in } N) \Downarrow \text{Error}} \; (\textit{let-err-p-1})_{\Downarrow}$$

$$\frac{(C, M) \Downarrow (C', V) \quad (C', N[V/x]) \Downarrow \text{Error}}{(C, \text{let } x = M \text{ in } N) \Downarrow \text{Error}} \; (\textit{let-err-p-2})_{\Downarrow}$$

$$\frac{(C, M) \Downarrow \text{Error}}{(C, \Box_A M) \Downarrow \text{Error}} \; (\textit{initial-err-p})_{\Downarrow}$$

$$\frac{(C, M) \Downarrow \text{Error}}{(C, \text{left}_{A,B}\, M) \Downarrow \text{Error}} \; (\textit{left-err-p})_{\Downarrow} \qquad \frac{(C, M) \Downarrow \text{Error}}{(C, \text{right}_{A,B}\, M) \Downarrow \text{Error}} \; (\textit{right-err-p})_{\Downarrow}$$

$$\frac{(C, M) \Downarrow \text{Error}}{(C, \text{case } M \text{ of } \{\text{left}\, x \to N \mid \text{right}\, y \to P\}) \Downarrow \text{Error}} \; (\textit{case-err-p})_{\Downarrow}$$

$$\frac{(C, M) \Downarrow (C', \text{left}_{A,B}\, V) \quad (C', N[V/x]) \Downarrow \text{Error}}{(C, \text{case } M \text{ of } \{\text{left}\, x \to N \mid \text{right}\, y \to P\}) \Downarrow \text{Error}} \; (\textit{case-left-err-p})_{\Downarrow}$$

$$\frac{(C, M) \Downarrow (C', \text{right}_{A,B}\, V) \quad (C', N[V/x]) \Downarrow \text{Error}}{(C, \text{case } M \text{ of } \{\text{left}\, x \to N \mid \text{right}\, y \to P\}) \Downarrow \text{Error}} \; (\textit{case-right-err-p})_{\Downarrow}$$

$$\frac{(C, M) \Downarrow \text{Error}}{(C, M; N) \Downarrow \text{Error}} \; (\textit{seq-err-p-1})_{\Downarrow} \qquad \frac{(C, M) \Downarrow (C', *) \quad (C', N) \Downarrow \text{Error}}{(C, M; N) \Downarrow \text{Error}} \; (\textit{seq-err-p-2})_{\Downarrow}$$

$$\frac{(C, M) \Downarrow \text{Error}}{(C, \langle M, N \rangle) \Downarrow \text{Error}} \; (\textit{pair-err-p-1})_{\Downarrow}$$

$$\frac{(C, M) \Downarrow (C', V) \quad (C', N) \Downarrow \text{Error}}{(C, \langle M, N \rangle) \Downarrow \text{Error}} \; (\textit{pair-err-p-2})_{\Downarrow}$$

$$\frac{(C, M) \Downarrow \text{Error}}{(C, \text{let } \langle x, y \rangle = M \text{ in } N) \Downarrow \text{Error}} \; (\textit{let-pair-err-p-1})_{\Downarrow}$$

$$\frac{(C, M) \Downarrow (C', \langle V, V' \rangle) \quad (C', N[V/x, V'/y]) \Downarrow \text{Error}}{(C, \text{let } \langle x, y \rangle = M \text{ in } N) \Downarrow \text{Error}} \; (\textit{let-pair-err-p-2})_{\Downarrow}$$

Table 5.5: Error-Propagation Rules of Proto-Quipper-M.

$$\frac{(C, M) \Downarrow \text{Error}}{(C, MN) \Downarrow \text{Error}} \; (\textit{app-err-p-1})_{\Downarrow}$$

$$\frac{(C, M) \Downarrow (C', \lambda x.M') \quad (C', N) \Downarrow \text{Error}}{(C, MN) \Downarrow \text{Error}} \; (\textit{app-err-p-2})_{\Downarrow}$$

$$\frac{(C, M) \Downarrow (C', \lambda x.M') \quad (C', N) \Downarrow (C'', V) \quad (C'', M'[V/x]) \Downarrow \text{Error}}{(C, MN) \Downarrow \text{Error}} \; (\textit{app-err-p-3})_{\Downarrow}$$

$$\frac{(C, M) \Downarrow \text{Error}}{(C, \text{force } M) \Downarrow \text{Error}} \; (\textit{force-err-p-1})_{\Downarrow}$$

$$\frac{(C, M) \Downarrow (C', \text{lift } M') \quad (C', M') \Downarrow \text{Error}}{(C, \text{force } M) \Downarrow \text{Error}} \; (\textit{force-err-p-2})_{\Downarrow}$$

$$\frac{(C, M) \Downarrow \text{Error}}{(C, \text{box}_T\, M) \Downarrow \text{Error}} \; (\textit{box-err-p-1})_{\Downarrow}$$

$$\frac{(C, M) \Downarrow (C', \text{lift } N) \quad \textit{freshlabels}(T) = (Q, \vec{\ell}) \quad (\text{id}_Q, N\vec{\ell}) \Downarrow \text{Error}}{(C, \text{box}_T\, M) \Downarrow \text{Error}} \; (\textit{box-err-p-2})_{\Downarrow}$$

$$\frac{(C, M) \Downarrow \text{Error}}{(C, \text{apply}(M, N)) \Downarrow \text{Error}} \; (\textit{apply-err-p-1})_{\Downarrow}$$

$$\frac{(C, M) \Downarrow (C', (\vec{\ell}, D, \vec{\ell'})) \quad (C', N) \Downarrow \text{Error}}{(C, \text{apply}(M, N)) \Downarrow \text{Error}} \; (\textit{apply-err-p-2})_{\Downarrow}$$

Table 5.6: Error-Propagation Rules of Proto-Quipper-M (Continuation).

Proposition 5.3.5. *If* (C, V) *is a value configuration where* V *is closed, then we have that* $(C, V) \Downarrow (C, V)$.

Proof. By structural induction on the set of closed values. Assume that (C, V) is a value configuration where V is closed.

The result follows immediately by the definition of the corresponding rule whenever V is a label, a constant, $*$, a lambda abstraction, a lift of a term, or a boxed circuit.

If $V = \mathrm{left}_{A,B}\, W$, where W is a closed value, then, by the induction hypothesis, $(C, W) \Downarrow (C, W)$, and by the $(\mathit{left})_{\Downarrow}$ rule, $(C, \mathrm{left}_{A,B}\, W) \Downarrow (C, \mathrm{left}_{A,B}\, W)$. The remaining cases, namely, when $V = \mathrm{right}_{A,B}\, W$ or $V = \langle W, W' \rangle$, where W and W' are closed values, are as in the previous case. □

We finish this chapter by observing that now that we have a formal definition of what constitutes an error, we will be able to show that our language is indeed *error-free*. This and other interesting properties of the Proto-Quipper-M language will be further explored in subsequent chapters.

Chapter 6

Type Safety

In this chapter, we show that Proto-Quipper-M is *type-safe*. This is accomplished by proving that the language satisfies both the *subject reduction* and the *error-freeness* properties. Even though these properties are easily stated, their proofs require quite intricate inductive arguments that depend on several syntactic lemmas. What is less obvious is the fact that much of the work lies in defining the type system and the operational semantics in just the right way so that these properties may hold in the first place.

6.1 Properties of the Type System

We begin our treatment of type safety by proving a number of lemmas necessary to establish the fundamental *Substitution Lemma*. Throughout this section (and the rest of the thesis), we will be making heavy use of different variants of the principle of well-founded induction such as structural induction on terms, rule induction and special rule induction on valid typing judgements, and induction on derivations.

Our first lemma tells us that the free variables of a well-typed term must belong to its variable context.

Lemma 6.1.1. *If* $\Gamma; Q \vdash M : A$ *is a valid typing judgement and* $x \notin \Gamma$*, then* $x \notin FV(M)$*.*

Proof. By rule induction on the set of valid typing judgements. Let $\Gamma; Q \vdash M : A$ be a valid typing judgement. Consider the following cases:

Axiom Rules: The result follows immediately for the (*circ*) rule and the axiom cases (*var*), (*label*), (*const*), and $(*)$ by the definition of such rules.

Purely Inductive Rules: These rules are the non-axiom rules with no distinguished variables. They are treated in a similar fashion and include (*initial*), (*left*), (*right*), (*seq*), (*pair*), (*app*), (*lift*), (*force*), (*box*) and (*apply*). In these cases, the result follows by direct applications of the induction hypothesis. We illustrate with the (*app*) case:

If the typing rule is (*app*) with $M = RS$, then the rule is

$$\frac{\Gamma_1; Q_1 \vdash R : A \multimap B \quad \Gamma_2; Q_2 \vdash S : A}{\Gamma_1 \cup \Gamma_2; Q_1, Q_2 \vdash RS : B}\ (\mathit{app})$$

where $\Gamma = \Gamma_1 \cup \Gamma_2$. Assume that $x \in FV(M)$. We want to show that $x \in \Gamma$.

Since $FV(M) = FV(RS) = FV(R) \cup FV(S)$, we have that $x \in FV(R)$ or $x \in FV(S)$. If $x \in FV(R)$, then the induction hypothesis implies that $x \in \Gamma_1$ since $\Gamma_1; Q_1 \vdash R : A \multimap B$ is valid. Similarly, if $x \in FV(S)$, then $x \in \Gamma_2$. In any case, $x \in \Gamma$.

Distinguished Variables Rules: These are the rules (*let*), (*case*), (*let-pair*), and (*abs*). In these cases, the result follows by applications of the induction hypothesis and case distinction. We present the (*let-pair*) case:

If the typing rule is (*let-pair*) with $M = (\text{let}\,\langle y, z\rangle = R \text{ in } S)$, then the rule is

$$\frac{\Gamma_1; Q_1 \vdash R : A \otimes B \quad \Gamma_2, y : A, z : B; Q_2 \vdash S : C}{\Gamma_1 \cup \Gamma_2; Q_1, Q_2 \vdash \text{let}\,\langle y, z\rangle = R \text{ in } S : C}\ (\mathit{let\text{-}pair})$$

where $\Gamma = \Gamma_1 \cup \Gamma_2$. Assume that $x \in FV(M)$. We want to show that $x \in \Gamma$.

Since $FV(M) = FV(\text{let}\,\langle y, z\rangle = R \text{ in } S) = FV(R) \cup (FV(S)\backslash\{y, z\})$, we have that $x \in FV(R)$ or $x \in FV(S)\backslash\{y, z\}$. If $x \in FV(R)$, then the induction hypothesis implies that $x \in \Gamma_1$ since $\Gamma_1; Q_1 \vdash R : A \otimes B$ is valid. Similarly, if $x \in FV(S)\backslash\{y, z\}$, then $x \in FV(S)$, $x \neq y$, and $x \neq z$. Since $\Gamma_2, y : A, z : B; Q_2 \vdash S : C$ is valid, the induction hypothesis implies that $x \in (\Gamma_2, y : A, z : B)$. But $x \neq y$ and $x \neq z$, and so, $x \in \Gamma_2$. In any case, $x \in \Gamma$.

□

We now show that non-trivial substitutions can occur in a term only when it has free variables.

Lemma 6.1.2. *If $x \notin FV(M)$, then $M[N/x] = M$.*

Proof. By structural induction on terms. Let x be a variable and let N be a term. Suppose that $x \notin FV(M)$.

- If M is a variable, a label, a constant, $*$, or a boxed circuit term, then the result is immediate from the definition of substitution in such terms.

- All the cases involving terms with bound variables are treated in a similar fashion. Each invokes Barendregt's convention and assumes that all the bound variables are fresh to avoid unintended variable capture. We illustrate with the *let* case:

  Suppose $M = (\text{let } y = R \text{ in } S)$. Since $FV(M) = FV(R) \cup (FV(S)\backslash\{y\})$ and $x \notin FV(M)$, we have that $x \notin FV(R)$ and $x \notin FV(S)\backslash\{y\}$. By Barendregt's convention, y is fresh, and so, $x \neq y$. Thus, $x \notin FV(S)$. The induction hypothesis then implies that $R[N/x] = R$ and $S[N/x] = S$. And thus,

$$\begin{aligned} M[N/x] &= (\text{let } y = R \text{ in } S)[N/x] \\ &= (\text{let } y = R[N/x] \text{ in } S[N/x]) \\ &= (\text{let } y = R \text{ in } S) \\ &= M. \end{aligned}$$

- The remaining cases follow immediately from the definition of capture-avoiding substitution and the induction hypothesis. We present the *pair* case:

  Suppose $M = \langle R, S\rangle$. Since $FV(M) = FV(R) \cup FV(S)$ and $x \notin FV(M)$, we have that $x \notin FV(R)$ and $x \notin FV(S)$. The induction hypothesis then implies that $R[N/x] = R$ and $S[N/x] = S$. And thus,

$$M[N/x] = \langle R, S\rangle[N/x] = \langle R[N/x], S[N/x]\rangle = \langle R, S\rangle = M.$$

□

We can often derive the form of a value term from its type. Here we assume that there are no constant symbols in the language; otherwise, the constants of course have their assigned types.

Lemma 6.1.3 (Closed Value Lemma)**.** *If $\emptyset; Q \vdash V : A$ is a valid typing judgement with V a value term, then the following hold:*

- *if $A = I$, then $V = *$;*
- *if $A = B \otimes C$, then $V = \langle V_1, V_2 \rangle$ for some value terms V_1 of type A and V_2 of type B;*
- *if $A = T$ where T is a simple M-type, then $V = \vec{\ell}$, a label tuple;*
- *if $A = B + C$, then $V = \mathrm{left}_{B,C}\, W$ for some value term W of type B or $V = \mathrm{right}_{B,C}\, W'$ for some value term W' of type C;*
- *if $A = B \multimap C$, then $V = \lambda x^B.M$ for some term M of type C;*
- *if $A = !B$, then $V = \mathrm{lift}\ M$ for some term M of type B, and*
- *if $A = \mathrm{Circ}(T, U)$, then $V = (\vec{\ell}, D, \vec{\ell'})$, a boxed circuit term, where $Q \vdash_{\mathcal{L}} \vec{\ell} : T$ and $Q' \vdash_{\mathcal{L}} \vec{\ell'} : U$ are valid label judgements and $D \in \mathbf{M}_{\mathcal{L}}(Q, Q')$ for some label contexts Q and Q'.*

Proof. By induction on the typing derivation $\mathcal{D}$ of $\emptyset; Q \vdash V : A$ with V a value term.

The result follows directly by inspection of the last rule applied in $\mathcal{D}$. The only instance that uses the induction hypothesis is the one where $A = T$, a simple M-type. In this case, the last rule of $\mathcal{D}$ is $(*)$, (*label*), or (*pair*). In the $(*)$ and (*label*) cases, the result is immediate. In the (*pair*) case, $V = \langle V_1, V_2 \rangle$ for some value terms V_1 and V_2, and $\mathcal{D}$ is

$$\dfrac{\begin{array}{cc} \vdots & \vdots \\ \emptyset; Q_1 \vdash V_1 : T_1 & \emptyset; Q_2 \vdash V_2 : T_2 \end{array}}{\emptyset; Q_1, Q_2 \vdash \langle V_1, V_2 \rangle : T_1 \otimes T_2}\ (\mathit{pair}).$$

where $T = T_1 \otimes T_1$ with T_1 and T_2 simple M-types.

By applying the induction hypothesis to the subderivation of the typing judgement $\emptyset; Q_1 \vdash V_1 : T_1$, we have that $V_1 = \vec{k}$, a label tuple. Similarly, the induction hypothesis applied to the subderivation of $\emptyset; Q_2 \vdash V_2 : T_2$ implies that $V_2 = \vec{\ell}$, also a label tuple. Thus, $V = \langle \vec{k}, \vec{\ell} \rangle$ is a label tuple as well. □

The context of a value of parameter type is a parameter context.

Lemma 6.1.4 (Parameter Value Lemma)**.** *If $\Gamma; Q \vdash V : P$ is a valid typing judgement where V is a value term and P is a parameter type, then $Q = \emptyset$ and Γ is a parameter context.*

Proof. By induction on the typing derivation $\mathcal{D}$ of $\Gamma; Q \vdash V : P$.

- If the last rule of $\mathcal{D}$ is any of (*var*), (*const*), ($*$), (*lift*), or (*circ*), then $Q = \emptyset$ and Γ is a parameter context by the definition of such rule.

- Note that since wire types and arrow types are not parameter types, the result is vacuously true when V is a label or a lambda abstraction.

- For the remaining cases, namely, (*left*), (*right*), and (*pair*), the result follows immediately from the induction hypothesis.

□

6.2 Weakening Lemma

The *Weakening Lemma* tells us that the validity of a term and its type do not change when its variable context is extended with a parameter context. First, we prove a special version of it.

Lemma 6.2.1 (Special Weakening Lemma)**.** *If $\Gamma; Q \vdash M : A$ is a valid typing judgement, and Φ a parameter context such that $\Phi \cap \Gamma = \emptyset$, then $\Phi, \Gamma; Q \vdash M : A$ is valid.*

Proof. By rule induction on the set of valid typing judgements.

- The (*circ*) case as well as the axiom cases, namely, (*var*), (*label*), (*const*), and ($*$) are treated in a similar fashion. We illustrate with the (*var*) case:

  The typing rule is
  $$\frac{}{\Phi, x : A; \emptyset \vdash x : A}\ (\mathit{var}).$$

  Let Φ' be a parameter context such that $\Phi' \cap (\Phi, x : A) = \emptyset$. Then, Φ', Φ is a well-defined parameter context not containing x. By the (*var*) rule,

  $$\Phi', \Phi, x : A; \emptyset \vdash x : A$$

  is a valid typing judgement.

- The result follows directly from the induction hypothesis in the cases in which the rule has typing judgements with no distinguished variables in their contexts, namely, (*initial*), (*left*), (*right*), (*seq*), (*pair*), (*app*), (*lift*), (*force*), (*box*), and (*apply*). We present the (*seq*) case:

  The typing rule is

  $$\frac{\Gamma_1; Q_1 \vdash M : I \quad \Gamma_2; Q_2 \vdash N : A}{\Gamma_1 \cup \Gamma_2; Q_1, Q_2 \vdash M; N : A}\ (\mathit{seq})$$

  where $\Gamma_1; Q_1 \vdash M : I$ and $\Gamma_2; Q_2 \vdash N : A$ are valid typing judgements. Let Φ be a parameter context such that $\Phi \cap (\Gamma_1 \cup \Gamma_2) = \emptyset$. Then, $\Phi \cap \Gamma_1 = \emptyset$ and $\Phi \cap \Gamma_2 = \emptyset$. By the induction hypothesis, $\Phi, \Gamma_1; Q_1 \vdash M : I$ and $\Phi, \Gamma_2; Q_2 \vdash N : A$ are valid. Since $(\Phi, \Gamma_1) \cup (\Phi, \Gamma_2) = \Phi, (\Gamma_1 \cup \Gamma_2)$, the (*seq*) rule then implies that $\Phi, (\Gamma_1 \cup \Gamma_2); Q \vdash M; N : A$ is valid as well.

- The cases where the typing rule has a context with distinguished variables, namely, (*let*), (*case*), (*let-pair*), and (*abs*), are treated in a similar fashion. All of them invoke Barendregt's convention to avoid unintended variable clashes. We illustrate with the (*abs*) case:

  The typing rule is

  $$\frac{\Gamma, x : A; Q \vdash M : B}{\Gamma; Q \vdash \lambda x^A.M : A \multimap B}\ (\mathit{abs})$$

  where $\Gamma, x : A; Q \vdash M : B$ is a valid typing judgement. Let Φ be a parameter context such that $\Phi \cap \Gamma = \emptyset$. By Barendregt's convention, we may assume that $x \notin \Phi$. Then, $\Phi \cap (\Gamma, x : A) = \emptyset$, and since $\Gamma, x : A; Q \vdash M : B$ is valid, the induction hypothesis implies that $\Phi, \Gamma, x : A; Q \vdash M : B$ is valid. By the (*abs*) rule, $\Phi, \Gamma; Q \vdash \lambda x^A.M : A \multimap B$ is valid as well.

□

The general *Weakening Lemma* easily follows now.

Lemma 6.2.2. *If $\Gamma; Q \vdash M : A$ is a valid typing judgement, and Φ a parameter context such that $\Phi \cup \Gamma$ is defined, then $\Phi \cup \Gamma; Q \vdash M : A$ is valid.*

Proof. Since $\Phi \cup \Gamma$ is defined, there exist parameter contexts Φ', Φ'', and a variable context Δ, all mutually disjoint, such that $\Phi = \Phi', \Phi''$, and $\Phi \cap \Gamma = \Phi''$, with

$\Gamma = \Phi'', \Delta$, and so, $\Phi \cup \Gamma = \Phi', \Phi'', \Delta$. Since Φ' is a parameter context such that $\Phi' \cap \Gamma = \Phi' \cap (\Phi'', \Delta) = \emptyset$, the Special Weakening Lemma 6.2.1 implies that $\Phi', \Phi'', \Delta; Q \vdash M : A$ is valid, that is, $\Phi \cup \Gamma; Q \vdash M : A$ is valid. □

We finish this section by showing that in appropriate contexts, the validity and type of a label tuple are the same in both typing systems $\vdash_{\mathcal{L}}$ and $\vdash$.

Lemma 6.2.3. *If T is a simple M-type, then $Q \vdash_{\mathcal{L}} \vec{\ell} : T$ is a valid label tuple judgement if and only if for every parameter context Φ, the typing judgement $\Phi; Q \vdash \vec{\ell} : T$ is valid.*

Proof. To prove the left-to-right direction, we proceed by induction on the typing derivation $\mathcal{D}$ of $Q \vdash_{\mathcal{L}} \vec{\ell} : T$. Let Φ be a parameter context. We want to show that $\Phi; Q \vdash \vec{\ell} : T$ is a valid typing judgement. Consider the following cases:

- If the last rule of $\mathcal{D}$ is $(*)_{\mathcal{L}}$ or $(label)_{\mathcal{L}}$, then the result is immediate.
- If the last rule of $\mathcal{D}$ is $(pair)_{\mathcal{L}}$ with $\vec{\ell} = \langle \vec{\ell'}, \vec{\ell''} \rangle$, then $\mathcal{D}$ is

$$\dfrac{\begin{array}{cc} \vdots & \vdots \\ Q' \vdash_{\mathcal{L}} \vec{\ell'} : T' & Q'' \vdash_{\mathcal{L}} \vec{\ell''} : T'' \end{array}}{Q', Q'' \vdash_{\mathcal{L}} \langle \vec{\ell'}, \vec{\ell''} \rangle : T' \otimes T''} \, (pair)_{\mathcal{L}}$$

  where $Q = Q', Q''$, and $T = T' \otimes T''$ for some simple M-types T' and T''. Since $Q' \vdash_{\mathcal{L}} \vec{\ell'} : T'$ and $Q'' \vdash_{\mathcal{L}} \vec{\ell''} : T''$ are valid, the induction hypothesis implies that $\Phi; Q' \vdash \vec{\ell'} : T'$ and $\Phi; Q'' \vdash \vec{\ell''} : T''$ are valid, and so, by the (*pair*) rule, $\Phi; Q', Q'' \vdash \langle \vec{\ell'}, \vec{\ell''} \rangle : T' \otimes T''$ is valid as well.

The proof of the converse is similar. □

6.3 Substitution Lemma

We now prove one of the main results of this chapter: the admissibility of the substitution operation in Proto-Quipper-M.

Lemma 6.3.1 (Substitution Lemma)**.** *If $\Gamma, x : A; Q \vdash N : B$ and $\Gamma'; Q' \vdash V : A$ are valid typing judgements such that $\Gamma \cup \Gamma'$ is defined, $Q \cap Q' = \emptyset$, and V is a value term, then $\Gamma \cup \Gamma'; Q, Q' \vdash N[V/x] : B$ is valid.*

Proof. By induction on the typing derivation $\mathcal{D}$ of $\Gamma, x : A; Q \vdash N : B$. Assume that $\Gamma'; Q' \vdash V : A$ is valid and such that $\Gamma \cup \Gamma'$ is defined, $Q \cap Q' = \emptyset$, and V is a value term. We want to show that $\Gamma \cup \Gamma'; Q, Q' \vdash N[V/x] : B$ is valid.

- If the last (and only) rule of $\mathcal{D}$ is (*var*) with $N = x$, then $\mathcal{D}$ is

  $$\frac{}{\Phi, x : A; \emptyset \vdash x : A}\ (\mathit{var})$$

  with $\Gamma = \Phi$, a parameter context, $Q = \emptyset$, and $B = A$. Since $N[V/x] = x[V/x] = V$, we want to show that $\Phi \cup \Gamma'; Q' \vdash V : A$ is valid.

  Since $\Gamma \cup \Gamma' = \Phi \cup \Gamma'$ is defined, Φ is a parameter context, and $\Gamma'; Q' \vdash V : A$ is valid, the Weakening Lemma 6.2.2 then implies that $\Phi \cup \Gamma'; Q' \vdash V : A$ is valid as well.

- If the last rule of $\mathcal{D}$ is (*var*) with $N = y \neq x$, then $\mathcal{D}$ is

  $$\frac{}{\Phi, x : A, y : B; \emptyset \vdash y : B}\ (\mathit{var})$$

  with $\Gamma = (\Phi, y : B)$ and $Q = \emptyset$, where $(\Phi, x : A)$ is a parameter context. Since $N[V/x] = y[V/x] = y$, we want to show that $(\Phi, y : B) \cup \Gamma'; Q' \vdash y : B$ is valid.

  Since $\Gamma'; Q' \vdash V : A$ is a valid typing judgement and A is a parameter type, the Parameter Value Lemma 6.1.4 implies that $Q' = \emptyset$ and that Γ' is a parameter context. Since $(\Phi, y : B) \cup \Gamma'$ is a parameter context except possibly for $y : B$, the (*var*) rule implies that $(\Phi, y : B) \cup \Gamma'; \emptyset \vdash y : B$ is valid.

- The proofs for the cases where the last (and only) rule of $\mathcal{D}$ is any of (*label*), (*const*), ($*$), or (*circ*), are similar. We present the (*label*) case:

  If the last rule of $\mathcal{D}$ is (*label*) with $N = \ell$, then $\mathcal{D}$ is

  $$\frac{}{\Phi; \ell : \alpha \vdash \ell : \alpha}\ (\mathit{label})$$

  with $(\Gamma, x : A) = \Phi$, a parameter context, $Q = \ell : \alpha$, and $B = \alpha$, a wire type. Since $N[V/x] = \ell[V/x] = \ell$, we want to show that $\Gamma \cup \Gamma'; \ell : \alpha, Q' \vdash \ell : \alpha$ is valid.

Since $\Gamma'; Q' \vdash V : A$ is valid and A is a parameter type, the Parameter Value Lemma 6.1.4 implies that $Q' = \emptyset$ and Γ' is a parameter context. Thus, $\Gamma \cup \Gamma'$ is a parameter context as well, and by the (*label*) rule, $\Gamma \cup \Gamma'; \ell : \alpha \vdash \ell : \alpha$ is valid.

- If the last rule of $\mathcal{D}$ is any of (*initial*), (*left*), (*right*), (*force*), or (*box*), we obtain the result by a direct application of the induction hypothesis. We illustrate with the (*left*) case:

  If the last rule of $\mathcal{D}$ is (*left*) with $N = \mathrm{left}_{C,D}\, M$, then $\mathcal{D}$ is

$$\dfrac{\begin{array}{c}\vdots\\ \overline{\Gamma}; Q \vdash M : C\end{array}}{\overline{\Gamma}; Q \vdash \mathrm{left}_{C,D}\, M : C + D}\ (\textit{left})$$

  with $(\Gamma, x : A) = \overline{\Gamma}$ and $B = C + D$. Since $N[V/x] = (\mathrm{left}_{C,D}\, M)[V/x] = \mathrm{left}_{C,D}\, M[V/x]$, we want to prove that $\Gamma \cup \Gamma'; Q, Q' \vdash \mathrm{left}_{C,D}\, M[V/x] : C + D$ is valid.

  Since $\Gamma, x : A; Q \vdash M : C$ is valid, the induction hypothesis implies that $\Gamma \cup \Gamma'; Q, Q' \vdash M[V/x] : C$ is also valid. By the (*left*) rule, we get the validity of $\Gamma \cup \Gamma'; Q, Q' \vdash \mathrm{left}_{C,D}\, M[V/x] : C + D$.

- If the last rule of $\mathcal{D}$ is (*lift*) with $N = \mathrm{lift}\ M$, then $\mathcal{D}$ is

$$\dfrac{\begin{array}{c}\vdots\\ \Phi; \emptyset \vdash M : C\end{array}}{\Phi; \emptyset \vdash \mathrm{lift}\ M : !C}\ (\textit{lift})$$

  with $(\Gamma, x : A) = \Phi$, a parameter context, $Q = \emptyset$, and $B = !C$. Since $N[V/x] = (\mathrm{lift}\ M)[V/x] = \mathrm{lift}\ M[V/x]$, we want to show the validity of the judgement $\Gamma \cup \Gamma'; Q' \vdash \mathrm{lift}\ M[V/x] : !C$.

  Since $\Gamma'; Q' \vdash V : A$ is valid and A is a parameter type, the Parameter Value Lemma 6.1.4 implies that $Q' = \emptyset$ and Γ' is a parameter context, and so, $\Gamma \cup \Gamma'$ is a parameter context as well. Since $\Gamma, x : A; \emptyset \vdash M : C$ is valid, the induction hypothesis implies that $\Gamma \cup \Gamma'; \emptyset \vdash M[V/x] : C$ is valid. By the (*lift*) rule, $\Gamma \cup \Gamma'; \emptyset \vdash \mathrm{lift}\ M[V/x] : !C$ is valid as well.

- If the last rule of $\mathcal{D}$ is any of (*seq*), (*pair*), (*app*), or (*apply*), we obtain the result by applications of the induction hypothesis, the Parameter Value Lemma 6.1.4, and Lemmas 6.1.1 and 6.1.2. We illustrate with the (*pair*) case:

  If the last rule of $\mathcal{D}$ is (*pair*) with $N = \langle R, S\rangle$, then $\mathcal{D}$ is

$$\dfrac{\begin{array}{c}\vdots \\ \Gamma_1; Q_1 \vdash R : C\end{array} \quad \begin{array}{c}\vdots \\ \Gamma_2; Q_2 \vdash S : D\end{array}}{\Gamma_1 \cup \Gamma_2; Q_1, Q_2 \vdash \langle R, S\rangle : C \otimes D}\ (\mathit{pair})$$

  with $(\Gamma, x : A) = \Gamma_1 \cup \Gamma_2$, label context $Q = Q_1, Q_2$, and $B = C \otimes D$. Since $N[V/x] = (\langle R, S\rangle)[V/x] = \langle R[V/x], S[V/x]\rangle$, we want to show that $\Gamma \cup \Gamma'; Q_1, Q_2, Q' \vdash \langle R[V/x], S[V/x]\rangle : C \otimes D$ is valid.

  We consider the following cases:

  - $x \in \Gamma_1 \backslash \Gamma_2$: In this case, $\Gamma_1 \backslash \{x : A\}, x : A; Q_1 \vdash R : C$ is valid. By the induction hypothesis, $(\Gamma_1 \backslash \{x : A\}) \cup \Gamma'; Q_1, Q' \vdash R[V/x] : C$ is also valid. Since $x \notin \Gamma_2$, by Lemma 6.1.1, $x \notin FV(S)$. Lemma 6.1.2 then implies that $S = S[V/x]$, and so, $\Gamma_2; Q_2 \vdash S[V/x] : D$ is valid as well. Since

$$\begin{aligned} ((\Gamma_1 \backslash \{x : A\}) \cup \Gamma') \cup \Gamma_2 &= ((\Gamma_1 \backslash \{x : A\}) \cup \Gamma_2) \cup \Gamma' \\ &= ((\Gamma_1 \cup \Gamma_2) \backslash \{x : A\}) \cup \Gamma' \\ &= \Gamma \cup \Gamma', \end{aligned}$$

    the validity of $(\Gamma_1 \backslash \{x : A\}) \cup \Gamma'; Q_1, Q' \vdash R[V/x] : C$ and $\Gamma_2; Q_2 \vdash S[V/x] : D$ yields that, by the (*pair*) rule, $\Gamma \cup \Gamma'; Q_1, Q_2, Q' \vdash \langle R[V/x], S[V/x]\rangle : C \otimes D$ is valid valid as well.

  - $x \in \Gamma_2 \backslash \Gamma_1$: This case is analogous to the previous one.

  - $x \in \Gamma_1 \cap \Gamma_2$: In this case, $\Gamma_1 \backslash \{x : A\}, x : A; Q_1 \vdash R : C$ is valid. By the induction hypothesis, $(\Gamma_1 \backslash \{x : A\}) \cup \Gamma'; Q_1, Q' \vdash R[V/x] : C$ is valid. Similarly, $(\Gamma_2 \backslash \{x : A\}) \cup \Gamma'; Q_2, Q' \vdash S[V/x] : C$ is valid. Now, $x \in \Gamma_1 \cap \Gamma_2$ implies that A is a parameter type, and since $\Gamma'; Q' \vdash V : A$ is valid, the Parameter Value Lemma 6.1.4 implies that $Q' = \emptyset$ and Γ' is a parameter context. Since

$$\begin{aligned}((\Gamma_1 \backslash \{x : A\}) \cup \Gamma') \cup ((\Gamma_2 \backslash \{x : A\}) \cup \Gamma') &= ((\Gamma_1 \cup \Gamma_2) \backslash \{x : A\}) \cup \Gamma' \\ &= ((\Gamma, x : A) \backslash \{x : A\}) \cup \Gamma' \\ &= \Gamma \cup \Gamma',\end{aligned}$$

the validity of the judgements $(\Gamma_1 \backslash \{x : A\}) \cup \Gamma'; Q_1, Q' \vdash R[V/x] : C$ and $(\Gamma_2 \backslash \{x : A\}) \cup \Gamma'; Q_2, Q' \vdash S[V/x] : C$ yields that, by the (*pair*) rule, $\Gamma \cup \Gamma'; Q_1, Q_2 \vdash \langle R[V/x], S[V/x] \rangle : C \otimes D$ is valid as well.

- If the last rule of $\mathcal{D}$ is (*abs*) with $N = \lambda y^C.M$, then $\mathcal{D}$ is

$$\dfrac{\begin{matrix}\vdots \\ \overline{\Gamma}, y : C; Q \vdash M : D\end{matrix}}{\overline{\Gamma}; Q \vdash \lambda y^C.M : C \multimap D}\ (abs)$$

with $(\Gamma, x : A) = \overline{\Gamma}$ and $B = C \multimap D$. By Barendregt's convention, we may assume that $y \notin \Gamma'$. Since $x \in \overline{\Gamma}$, $y \neq x$, and so $N[V/x] = (\lambda y^C.M)[V/x] = \lambda y^C.M[V/x]$. We want to prove that $\Gamma \cup \Gamma'; Q, Q' \vdash \lambda y^C.M[V/x] : C \multimap D$ is a valid typing judgment.

Since $\Gamma, x : A, y : C; Q \vdash M : D$ is valid, the induction hypothesis implies that $(\Gamma, y : C) \cup \Gamma'; Q, Q' \vdash M[V/x] : D$ is valid. But $(\Gamma, y : C) \cup \Gamma' = ((\Gamma \cup \Gamma'), y : C)$, so the (*abs*) rule implies that $\Gamma \cup \Gamma'; Q, Q' \vdash \lambda y^C.M[V/x] : C \multimap D$ is valid as well.

- The proofs for the remaining cases, namely, (*let*), (*case*), and (*let-pair*), are similar. We illustrate with the (*let-pair*) case:

If the last rule of $\mathcal{D}$ is (*let-pair*) with $N = (\text{let}\,\langle y, z\rangle = R \text{ in } S)$, then $\mathcal{D}$ is

$$\dfrac{\begin{matrix}\vdots \\ \Gamma_1; Q_1 \vdash R : C \otimes D\end{matrix} \quad \begin{matrix}\vdots \\ \Gamma_2, y : C, z : D; Q_2 \vdash S : B\end{matrix}}{\Gamma_1 \cup \Gamma_2; Q_1, Q_2 \vdash \text{let}\,\langle y, z\rangle = R \text{ in } S : B}\ (let\text{-}pair)$$

with $(\Gamma, x : A) = \Gamma_1 \cup \Gamma_2$, and $Q = Q_1, Q_2$. By Barendregt's convention, we may assume that $x \neq y$, $x \neq z$, and $y, z \notin \Gamma'$. By Lemma 6.1.1, $y, z \notin FV(V)$. Since $N[V/x] = (\text{let}\,\langle y, z\rangle = R \text{ in } S)[V/x] = (\text{let}\,\langle y, z\rangle = R[V/x] \text{ in } S[V/x])$, we want to show that $\Gamma \cup \Gamma'; Q_1, Q_2, Q' \vdash \text{let}\,\langle y, z\rangle = R[V/x] \text{ in } S[V/x] : B$ is a valid typing judgment.

We consider the following cases:

- $x \in \Gamma_1 \backslash \Gamma_2$: In this case, $\Gamma_1 \backslash \{x : A\}, x : A; Q_1 \vdash R : C \otimes D$ is valid, and by the induction hypothesis, $(\Gamma_1 \backslash \{x : A\}) \cup \Gamma'; Q_1, Q' \vdash R[V/x] : C \otimes D$ is also valid. Since $x \notin \Gamma_2$, $x \neq y$ and $x \neq z$, we have that $x \notin FV(S)$ by Lemma 6.1.1. By Lemma 6.1.2, $S = S[V/x]$, and so, we have that $\Gamma_2, y : C, z : D; Q_2 \vdash S[V/x] : B$ is valid. Note that

  $$\begin{aligned} ((\Gamma_1 \backslash \{x : A\}) \cup \Gamma') \cup \Gamma_2 &= ((\Gamma_1 \backslash \{x : A\}) \cup \Gamma_2) \cup \Gamma' \\ &= ((\Gamma_1 \cup \Gamma_2) \backslash \{x : A\}) \cup \Gamma' \\ &= ((\Gamma, x : A) \backslash \{x : A\}) \cup \Gamma' \\ &= \Gamma \cup \Gamma', \end{aligned}$$

  and since the judgements $(\Gamma \backslash \{x : A\}) \cup \Gamma'; Q_1, Q' \vdash R[V/x] : C \otimes D$ and $\Gamma_2, y : C, z : D; Q_2 \vdash S[V/x] : B$ are valid, then by the (*let-pair*) rule, $\Gamma \cup \Gamma'; Q_1, Q_2, Q' \vdash \text{let}\,\langle y, z\rangle = R[V/x] \text{ in } S[V/x] : B$ is valid as well.

- $x \in \Gamma_2 \backslash \Gamma_1$: This case is analogous to the previous one.

- $x \in \Gamma_1 \cap \Gamma_2$: In this case, A is a parameter type, and since $\Gamma'; Q' \vdash V : A$ is valid, the Parameter Value Lemma 6.1.4 implies that $Q' = \emptyset$ and Γ' is a parameter context. Since $\Gamma_1 \backslash \{x : A\}, x : A; Q_1 \vdash R : C \otimes D$ and $\Gamma_2 \backslash \{x : A\}, x : A, y : C, z : D; Q_2 \vdash S : B$ are valid, the induction hypothesis implies that the judgements $(\Gamma_1 \backslash \{x : A\}) \cup \Gamma'; Q_1 \vdash R[V/x] : C \otimes D$ and $((\Gamma_2 \backslash \{x : A\}), y : C, z : D) \cup \Gamma'; Q_2 \vdash S[V/x] : B$ are valid as well. Note that

  $$((\Gamma_2 \backslash \{x : A\}), y : C, z : D) \cup \Gamma' = ((\Gamma_2 \backslash \{x : A\}) \cup \Gamma'), y : C, z : D$$

  since $y, z \notin \Gamma'$, and so we can rewrite the latter typing judgement as $((\Gamma_2 \backslash \{x : A\}) \cup \Gamma'), y : C, z : D; Q_2 \vdash S[V/x] : B$.

  Let $\widehat{\Gamma_1} = (\Gamma_1 \backslash \{x : A\}) \cup \Gamma'$ and let $\widehat{\Gamma_2} = (\Gamma_2 \backslash \{x : A\}) \cup \Gamma'$, then

$$\begin{aligned}\widehat{\Gamma_1} \cup \widehat{\Gamma_2} &= ((\Gamma_1 \backslash \{x : A\}) \cup (\Gamma_2 \backslash \{x : A\})) \cup \Gamma' \\ &= ((\Gamma_1 \cup \Gamma_2) \backslash \{x : A\}) \cup \Gamma' \\ &= ((\Gamma, x : A) \backslash \{x : A\}) \cup \Gamma' \\ &= \Gamma \cup \Gamma',\end{aligned}$$

and so

$$((\Gamma_1 \backslash \{x : A\}) \cup \Gamma') \cup ((\Gamma_2 \backslash \{x : A\}) \cup \Gamma') = \Gamma \cup \Gamma'.$$

Since the judgements $(\Gamma_1 \backslash \{x : A\}) \cup \Gamma'; Q_1 \vdash R[V/x] : C \otimes D$ and $((\Gamma_2 \backslash \{x : A\}) \cup \Gamma'), y : C, z : D; Q_2 \vdash S[V/x] : B$ are valid, then by the (*let-pair*) rule,

$$\Gamma \cup \Gamma'; Q_1, Q_2 \vdash \text{let } \langle y, z \rangle = R[V/x] \text{ in } S[V/x] : B$$

is valid as well.

□

6.4 Type Preservation

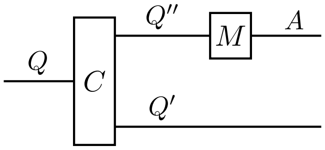
We now make more explicit the relationship between the syntax and the operational semantics of Proto-Quipper-M by showing that the evaluation relation satisfies the subject reduction property. Subject reduction, also known as type preservation, says that if a well-typed configuration evaluates to a value configuration, the latter is well-typed of the same type. Since the evaluation relation of Proto-Quipper-M is defined on configurations while the typing rules only apply to terms, we first extend the notion of typing and validity to configurations.

Definition 6.4.1. Let Q, Q' be label contexts, (C, M) a configuration, and A a type. We say that (C, M) is *well-typed* or *valid* with input labels Q, output labels Q', and type A, in symbols

$$Q \vdash (C, M) : A; Q',$$

if there exists a label context Q'' disjoint from Q' such that $C : Q \to Q'', Q'$ and $\emptyset; Q'' \vdash M : A$. ◇

Note that in a well-typed configuration, M may contain some free labels, but never free variables, and so, M is always a *closed* term with labels bound to a subset of the outputs of the labeled morphism C. Sometimes it may be helpful to picture this situation as follows:



This will become more formal in Chapter 9 where we discuss soundness.

Theorem 6.4.2 (Subject Reduction Theorem)**.** *If $Q \vdash (C, M) : A; Q'$ is a valid configuration and $(C, M) \Downarrow (C', V)$, then $Q \vdash (C', V) : A; Q'$ is also valid.*

Proof. We proceed by rule induction on the evaluation relation $\Downarrow$. See Table 5.3.

Axiom Rules: The result is immediate for the axiom cases, namely: $(\mathit{label})_\Downarrow$, $(\mathit{const})_\Downarrow$, $(*)_\Downarrow$, $(\mathit{abs})_\Downarrow$, $(\mathit{lift})_\Downarrow$, and $(\mathit{circ})_\Downarrow$, because in these cases, $(C, M) = (C', V)$.

Purely Inductive Rules: These rules are all treated in a similar fashion and include $(\mathit{left})_\Downarrow$, $(\mathit{right})_\Downarrow$, $(\mathit{seq})_\Downarrow$, $(\mathit{pair})_\Downarrow$, and $(\mathit{force})_\Downarrow$. In these cases, the result follows by direct applications of the induction hypothesis and the corresponding *typing* rules. We treat each case in turn:

- $(\mathit{left})_\Downarrow$: The evaluation rule is

  $$\frac{(C, M) \Downarrow (C', V)}{(C, \mathrm{left}_{A,B}\, M) \Downarrow (C', \mathrm{left}_{A,B}\, V)}\ .$$

  Assume that $Q \vdash (C, \mathrm{left}_{A,B}\, M) : A + B; Q'$ is valid. We want to show the validity of $Q \vdash (C', \mathrm{left}_{A,B}\, V) : A + B; Q'$.

  By assumption, there exists a label context Q'' such that $C : Q \to Q'', Q'$ and $\emptyset; Q'' \vdash \mathrm{left}_{A,B}\, M : A + B$ is valid. By the (*left*) rule, $\emptyset; Q'' \vdash M : A$ is valid. It follows that $Q \vdash (C, M) : A; Q'$ is valid as well. Applying the induction hypothesis to $(C, M) \Downarrow (C', V)$ yields that $Q \vdash (C', V) : A; Q'$ is valid. Thus, there exists a label context Q''' such that $C' : Q \to Q''', Q'$ and $\emptyset; Q''' \vdash V : A$. By the (*left*) rule, we have that $\emptyset; Q''' \vdash \mathrm{left}_{A,B}\, V : A + B$ is valid as well. Thus, $Q \vdash (C', \mathrm{left}_{A,B}\, V) : A + B; Q'$ is valid too.

- $(\mathit{right})_{\Downarrow}$: This case is similar to the $(\mathit{left})_{\Downarrow}$ case.
- $(\mathit{seq})_{\Downarrow}$: The evaluation rule is

$$\frac{(C, M) \Downarrow (C', *) \quad (C', N) \Downarrow (C'', W)}{(C, M; N) \Downarrow (C'', W)}\ .$$

Assume that $Q \vdash (C, M; N) : A; Q'$ is valid. We want to show the validity of $Q \vdash (C'', W) : A; Q'$.

By assumption, there exists a label context Q'' such that $C : Q \to Q'', Q'$ and $\emptyset; Q'' \vdash M; N : A$ is valid. By the typing rule (*seq*), there are label contexts Q''_1 and Q''_2 such that $Q'' = Q''_1, Q''_2$, and both $\emptyset; Q''_1 \vdash M : I$ and $\emptyset; Q''_2 \vdash N : A$ are valid.

Since $C : Q \to Q''_1, Q''_2, Q'$ and $\emptyset; Q''_1 \vdash M : I$ is valid, it follows that $Q \vdash (C, M) : I; Q''_2, Q'$ is valid as well. Applying the induction hypothesis to $(C, M) \Downarrow (C', *)$ yields that $Q \vdash (C', *) : I; Q''_2, Q'$ is valid. Thus, there exists a label context Q''' such that $C' : Q \to Q''', Q''_2, Q'$ and $\emptyset; Q''' \vdash * : I$ is valid. By the typing rule $(*)$, $Q''' = \emptyset$, and so, $C' : Q \to Q''_2, Q'$. This and the validity of $\emptyset; Q''_2 \vdash N : A$ imply that $Q \vdash (C', N) : A; Q'$ is valid too. By the induction hypothesis applied to $(C', N) \Downarrow (C'', W)$, we have that $Q \vdash (C'', W) : A; Q'$ is valid as well.

- $(\mathit{pair})_{\Downarrow}$: The evaluation rule is

$$\frac{(C, M) \Downarrow (C', V) \quad (C', N) \Downarrow (C'', V')}{(C, \langle M, N\rangle) \Downarrow (C'', \langle V, V'\rangle)}\ .$$

Assume that $Q \vdash (C, \langle M, N\rangle) : A \otimes B; Q'$ is valid. We want to show the validity of $Q \vdash (C'', \langle V, V'\rangle) : A \otimes B; Q'$.

By assumption, there exists a label context Q'' such that $C : Q \to Q'', Q'$ and $\emptyset; Q'' \vdash \langle M, N\rangle : A \otimes B$ is valid. By the typing rule (*pair*), there are label contexts Q''_1 and Q''_2 such that $Q'' = Q''_1, Q''_2$, and both $\emptyset; Q''_1 \vdash M : A$ and $\emptyset; Q''_2 \vdash N : B$ are valid.

Since $C : Q \to Q_1'', Q_2'', Q'$ and $\emptyset; Q_1'' \vdash M : A$ is valid, it follows that $Q \vdash (C, M) : A; Q_2'', Q'$ is valid as well. Applying the induction hypothesis to $(C, M) \Downarrow (C', V)$ yields that $Q \vdash (C', V) : A; Q_2'', Q'$ is valid. Thus, there exists a label context Q_1''' such that $C' : Q \to Q_1''', Q_2'', Q'$ and $\emptyset; Q_1''' \vdash V : A$ is valid. Since $\emptyset; Q_2'' \vdash N : B$ is valid, it follows that $Q \vdash (C', N) : B; Q_1''', Q'$ is valid as well. By the induction hypothesis applied to $(C', N) \Downarrow (C'', V')$, we have that $Q \vdash (C'', V') : B; Q_1''', Q'$ is valid too. Thus, there exists a label context Q_2''' such that $C'' : Q \to Q_2''', Q_1''', Q'$ and $\emptyset, Q_2''' \vdash V' : B$ is valid. By the typing rule (*pair*), $\emptyset, Q_1''', Q_2''' \vdash \langle V, V' \rangle : A \otimes B$ is valid, and so, $Q \vdash (C'', \langle V, V' \rangle) : A \otimes B; Q'$ is valid as well.

- $(\textit{force})_{\Downarrow}$: The evaluation rule is

  $$\frac{(C, M) \Downarrow (C', \text{lift } M') \quad (C', M') \Downarrow (C'', V)}{(C, \text{force } M) \Downarrow (C'', V)}\ .$$

  Assume that $Q \vdash (C, \text{force } M) : A; Q'$ is valid. We want to show the validity of $Q \vdash (C'', V) : A; Q'$.

  By assumption, there exists a label context Q'' such that $C : Q \to Q'', Q'$ and $\emptyset; Q'' \vdash \text{force } M : A$ is valid. By the typing rule (*force*), we have that $\emptyset; Q'' \vdash M : !A$ is valid, and so, $Q \vdash (C, M) : !A; Q'$ is valid as well. Applying the induction hypothesis to $(C, M) \Downarrow (C', \text{lift } M')$ yields that $Q \vdash (C', \text{lift } M') : !A, Q'$ is valid. Thus, there exists a label context Q''' such that $C' : Q \to Q''', Q'$ and $\emptyset; Q''' \vdash \text{lift } M' : !A$ is valid. By the typing rule (*lift*), we have that $Q''' = \emptyset$ and $\emptyset; \emptyset \vdash M' : A$ is valid, and so, $Q \vdash (C', M') : A; Q'$ is valid as well. By the induction hypothesis applied to $(C', M') \Downarrow (C'', V)$, we have that $Q \vdash (C'', V) : B; Q'$ is valid too.

Substitution Rules: These are rules with a configuration containing a term in which a substitution has occurred, namely, $(\textit{let})_{\Downarrow}$, $(\textit{case-left})_{\Downarrow}$, $(\textit{case-right})_{\Downarrow}$, $(\textit{let-pair})_{\Downarrow}$, and $(\textit{app})_{\Downarrow}$. In these cases, the result follows by the induction hypothesis and the Substitution Lemma 6.3.1. We treat each case in turn:

- $(\textit{let})_{\Downarrow}$: The evaluation rule is

  $$\frac{(C, M) \Downarrow (C', V) \quad (C', N[V/x]) \Downarrow (C'', W)}{(C, \text{let } x = M \text{ in } N) \Downarrow (C'', W)}\ .$$

Assume that $Q \vdash (C, \text{let } x = M \text{ in } N) : B; Q'$ is valid. We want to show the validity of $Q \vdash (C'', W) : B; Q'$.

By assumption, there exists a label context Q'' such that $C : Q \to Q'', Q'$ and $\emptyset; Q'' \vdash \text{let } x = M \text{ in } N : B$ is valid. By the typing rule (*let*), there are label contexts Q''_1 and Q''_2 such that $Q'' = Q''_1, Q''_2$, and both $\emptyset; Q''_1 \vdash M : A$ and $x : A; Q''_2 \vdash N : B$ are valid.

Since $C : Q \to Q''_1, Q''_2, Q'$ and $\emptyset; Q''_1 \vdash M : A$ is valid, it follows that $Q \vdash (C, M) : A; Q''_2, Q'$ is valid as well. Applying the induction hypothesis to $(C, M) \Downarrow (C', V)$ yields that $Q \vdash (C', V) : A; Q''_2, Q'$ is also valid. Thus, there exists a label context Q''' such that $C' : Q \to Q''', Q''_2, Q'$ and $\emptyset; Q''' \vdash V : A$ is valid. But $x : A; Q''_2 \vdash N : B$ is valid, and so $\emptyset; Q''', Q''_2 \vdash N[V/x] : B$ is also valid by the Substitution Lemma 6.3.1. This together with $C' : Q \to Q''', Q''_2, Q'$ implies that $Q \vdash (C', N[V/x]) : B; Q'$ is valid. By the induction hypothesis applied to $(C', N[V/x]) \Downarrow (C'', W)$, we have that $Q \vdash (C'', W) : B; Q'$ is valid as well.

- (*let-pair*)$_\Downarrow$: This case is similar to the (*let*)$_\Downarrow$ case.
- (*case-left*)$_\Downarrow$: The evaluation rule is

$$\frac{(C, M) \Downarrow (C', \text{left}_{A,B}\, V) \quad (C', N[V/x]) \Downarrow (C'', W)}{(C, \text{case } M \text{ of } \{\text{left}\, x \to N \mid \text{right}\, y \to P\}) \Downarrow (C'', W)}\ .$$

Assume that $Q \vdash (C, \text{case } M \text{ of } \{\text{left}\, x \to N \mid \text{right}\, y \to P\}) : D; Q'$ is valid. We want to show that $Q \vdash (C'', W) : D; Q'$ is also valid.

By assumption, there exists a label context Q'' such that $C : Q \to Q'', Q'$ and $\emptyset; Q'' \vdash \text{case } M \text{ of } \{\text{left}\, x \to N \mid \text{right}\, y \to P\} : D$ is valid. By the typing rule (*case*), there are label contexts Q''_1 and Q''_2 such that $Q'' = Q''_1, Q''_2$, and $\emptyset; Q''_1 \vdash M : A + B$ as well as $x : A; Q''_2 \vdash N : D$ and $y : B; Q''_2 \vdash P : D$ are valid.

Since $C : Q \to Q''_1, Q''_2, Q'$ and $\emptyset; Q''_1 \vdash M : A + B$ is valid, it follows that $Q \vdash (C, M) : A + B; Q''_2, Q'$ is valid as well. By the induction hypothesis, $(C, M) \Downarrow (C', \text{left}_{A,B}\, V)$ implies that $Q \vdash (C', \text{left}_{A,B}\, V) : A + B; Q''_2, Q'$ is valid. Thus, there exists a label context Q''' such that $C' : Q \to Q''', Q''_2, Q'$ and $\emptyset; Q''' \vdash \text{left}_{A,B}\, V : A + B$ is valid. By the (*left*) rule, $\emptyset; Q''' \vdash V : A$ is valid as well. But $x : A; Q_2'' \vdash N : D$ is valid, and so, $\emptyset; Q''', Q_2 \vdash N[V/x] : D$ is also valid by the Substitution Lemma 6.3.1. Since $C' : Q \to Q''', Q_2'', Q'$, $Q \vdash (C', N[V/x]) : D; Q'$ is valid too. By the induction hypothesis applied to $(C', N[V/x]) \Downarrow (C'', W)$, we have that $Q \vdash (C'', W) : D; Q'$ is valid as well.

- $(\textit{case-right})_{\Downarrow}$: This case is similar to the $(\textit{case-left})_{\Downarrow}$ case.
- $(\textit{app})_{\Downarrow}$: The evaluation rule is

$$\frac{(C, M) \Downarrow (C', \lambda x^A.M') \quad (C', N) \Downarrow (C'', V) \quad (C'', M'[V/x]) \Downarrow (C''', W)}{(C, MN) \Downarrow (C''', W)}\,.$$

Assume that $Q \vdash (C, MN) : B; Q'$ is valid. We want to show the validity of $Q \vdash (C''', W) : B; Q'$.

By assumption, there exists a label context Q'' such that $C : Q \to Q'', Q'$ and $\emptyset; Q'' \vdash MN : B$ is valid. By the typing rule $(\textit{app})$, there are label contexts Q_1'' and Q_2'' such that $Q'' = Q_1'', Q_2''$, and both $\emptyset; Q_1'' \vdash M : A \multimap B$ and $\emptyset; Q_2'' \vdash N : A$ are valid.

Since $C : Q \to Q_1'', Q_2'', Q'$ and $\emptyset; Q_1'' \vdash M : A \multimap B$ is valid, it follows that $Q \vdash (C, M) : A \multimap B; Q_2'', Q'$ is valid as well. Applying the induction hypothesis to $(C, M) \Downarrow (C', \lambda x^A.M')$ yields that $Q \vdash (C', \lambda x^A.M') : A \multimap B; Q_2'', Q'$ is valid. Thus, there exists a label context Q''' such that $C' : Q \to Q''', Q_2'', Q'$ and $\emptyset; Q''' \vdash \lambda x^A.M' : A \multimap B$ is valid as well. By the typing rule $(\textit{abs})$, $x : A; Q''' \vdash M' : B$ is also valid.

Since $C' : Q \to Q''', Q_2'', Q'$ and $\emptyset; Q_2'' \vdash N : A$ is valid, we have that the judgement $Q \vdash (C', N) : A; Q''', Q'$ is also valid. Applying the induction hypothesis to $(C', N) \Downarrow (C'', V)$ yields that $Q \vdash (C'', V) : A; Q''', Q'$ is valid too. Thus, there exists a label context $\overline{Q}$ such that $C'' : Q \to \overline{Q}, Q''', Q'$ and $\emptyset; \overline{Q} \vdash V : A$ is valid. But $x : A; Q''' \vdash M' : B$ is valid as well, and so, $\emptyset; \overline{Q}, Q''' \vdash M'[V/x] : B$ is also valid by the Substitution Lemma 6.3.1. Since $C'' : Q \to \overline{Q}, Q''', Q'$, it follows that $Q \vdash (C'', M'[V/x]) : B; Q'$ is valid too. By the induction hypothesis applied to $(C'', M'[V/x]) \Downarrow (C''', W)$, we have that $Q \vdash (C''', W) : B; Q'$ is valid as well.

Circuit Rules: These are the rules $(\mathit{box})_{\Downarrow}$ and $(\mathit{apply})_{\Downarrow}$ and they are the most interesting rules of Proto-Quipper-M as they generate circuits. We treat each in turn:

- $(\mathit{box})_{\Downarrow}$: The evaluation rule is
$$\frac{(C, M) \Downarrow (C', \text{lift } N) \quad \mathit{freshlabels}(T) = (\overline{Q}, \vec{\ell}) \quad (\text{id}_{\overline{Q}}, N\vec{\ell}) \Downarrow (D, \vec{\ell'})}{(C, \text{box}_T\, M) \Downarrow (C', (\vec{\ell}, D, \vec{\ell'}))}\,.$$
Assume that $Q \vdash (C, \text{box}_T\, M) : A; Q'$ is valid. We want to show the validity of $Q \vdash (C', (\vec{\ell}, D, \vec{\ell'})) : A; Q'$.

  By assumption, there exists a label context Q'' such that $C : Q \to Q'', Q'$ and $\emptyset; Q'' \vdash \text{box}_T\, M : A$ is valid. By the typing rule (box), $A = \text{Circ}(T, U)$ for some simple M-type U, and $\emptyset; Q'' \vdash M : !(T \multimap U)$ is valid. This and $C : Q \to Q'', Q'$ imply that $Q \vdash (C, M) : !(T \multimap U); Q'$ is valid as well. Applying the induction hypothesis to $(C, M) \Downarrow (C', \text{lift } N)$ yields that $Q \vdash (C', \text{lift } N) : !(T \multimap U); Q'$ is also valid. Thus, there exists a label context Q''' such that $C' : Q \to Q''', Q'$ and $\emptyset; Q''' \vdash \text{lift } N : !(T \multimap U)$ is valid. By the typing rule (lift), $Q''' = \emptyset$ and $\emptyset; \emptyset \vdash N : T \multimap U$ is also valid. This and $C' : Q \to \emptyset, Q'$ imply the validity of $Q \vdash (C', N) : T \multimap U; Q'$. Since $\mathit{freshlabels}(T) = (\overline{Q}, \vec{\ell})$, we have that $\overline{Q} \vdash_{\mathcal{L}} \vec{\ell} : T$ is valid, and by Lemma 6.2.3, $\emptyset; \overline{Q} \vdash \vec{\ell} : T$ is valid as well. This and $\emptyset; \emptyset \vdash N : T \multimap U$ yield that $\emptyset; \overline{Q} \vdash N\vec{\ell} : U$ is valid by the (app) rule.

  Since $\mathit{id}_{\overline{Q}} : \overline{Q} \to \overline{Q}, \emptyset$ and $\emptyset; \overline{Q} \vdash N\vec{\ell} : U$ is valid, $\overline{Q} \vdash (\mathit{id}_{\overline{Q}}, N\vec{\ell}) : U; \emptyset$ is also valid. Applying the induction hypothesis to $(\text{id}_{\overline{Q}}, N\vec{\ell}) \Downarrow (D, \vec{\ell'})$ yields that $\overline{Q} \vdash (D, \vec{\ell'}) : U; \emptyset$ is valid too. Thus, there exists a label context $\overline{\overline{Q}}$ such that $D : \overline{Q} \to \overline{\overline{Q}}, \emptyset$ and $\emptyset; \overline{\overline{Q}} \vdash \vec{\ell'} : U$ is valid. Then, Lemma 6.2.3 implies that $\overline{\overline{Q}} \vdash_{\mathcal{L}} \vec{\ell'} : U$ is valid as well. Since $\overline{Q} \vdash_{\mathcal{L}} \vec{\ell} : T$ and $\overline{\overline{Q}} \vdash_{\mathcal{L}} \vec{\ell'} : U$ are valid and $D : \overline{Q} \to \overline{\overline{Q}} \in \mathbf{M}_{\mathcal{L}}(\overline{Q}, \overline{\overline{Q}})$, the (circ) rule implies that $\emptyset; \emptyset \vdash (\vec{\ell}, D, \vec{\ell'}) : \text{Circ}(T, U)$ is valid. This and $C' : Q \to \emptyset, Q'$ then imply that $Q \vdash (C', (\vec{\ell}, D, \vec{\ell'})) : \text{Circ}(T, U); Q'$ is valid as well.

- $(\mathit{apply})_{\Downarrow}$: The evaluation rules is
$$\frac{(C, M) \Downarrow (C', (\vec{\ell}, D, \vec{\ell'})) \quad (C', N) \Downarrow (C'', \vec{k}) \quad \mathit{append}(C'', \vec{k}, \vec{\ell}, D, \vec{\ell'}) = (C''', \vec{k'})}{(C, \text{apply}(M, N)) \Downarrow (C''', \vec{k'})}\,.$$

Assume that $Q \vdash (C, \mathrm{apply}(M, N)) : U; Q'$ is valid. We want to show the validity of $Q \vdash (C''', \vec{k'}) : U; Q'$.

By assumption, there exists a label context Q'' such that $C : Q \to Q'', Q'$ and $\emptyset; Q'' \vdash \mathrm{apply}(M, N) : U$ is valid. By the (*apply*) rule, there exist label contexts Q''_1 and Q''_2 such that $Q'' = Q''_1, Q''_2$ and both $\emptyset; Q''_1 \vdash M : \mathrm{Circ}(T, U)$ and $\emptyset; Q''_2 \vdash N : T$ are valid. The former and $C : Q \to Q''_1, Q''_2, Q'$ imply that $Q \vdash (C, M) : \mathrm{Circ}(T, U); Q''_2, Q'$ is valid as well. Applying the induction hypothesis to $(C, M) \Downarrow (C', (\vec{\ell}, D, \vec{\ell'}))$ yields the validity of $Q \vdash (C', (\vec{\ell}, D, \vec{\ell'})) : \mathrm{Circ}(T, U); Q''_2, Q'$. Thus, there is a label context Q''' such that $C' : Q \to Q''', Q''_2, Q'$ and $\emptyset; Q''' \vdash (\vec{\ell}, D, \vec{\ell'}) : \mathrm{Circ}(T, U)$ is valid. By the typing rule (*circ*), $Q''' = \emptyset$, there exist label contexts $\widetilde{Q}$ and $\widetilde{Q}'$ such that $D \in \mathbf{M}_{\mathcal{L}}(\widetilde{Q}, \widetilde{Q}')$, and both $\widetilde{Q} \vdash_{\mathcal{L}} \vec{\ell} : T$ and $\widetilde{Q}' \vdash_{\mathcal{L}} \vec{\ell'} : U$ are valid. Since $C' : Q \to Q''_2, Q'$ and $\emptyset; Q''_2 \vdash N : T$ is valid, we have that $Q \vdash (C', N) : T; Q'$ is valid. Applying the induction hypothesis to $(C', N) \Downarrow (C'', \vec{k})$ yields that $Q \vdash (C'', \vec{k}) : T; Q'$ is valid as well. Thus, there exists a label context $\overline{Q}$ such that $C'' : Q \to \overline{Q}, Q'$ and $\emptyset; \overline{Q} \vdash \vec{k} : T$ is valid.

Since $\mathit{append}(C'', \vec{k}, \vec{\ell}, D, \vec{\ell'}) = (C''', \vec{k'})$, we have that $C''' = (D' \otimes id_{Q'}) \circ C''$ where $D' : \overline{Q} \to \overline{Q}'$ is the labeled circuit derived from $D : \widetilde{Q} \to \widetilde{Q}'$ with $\overline{Q}$ being the label context obtained by the relabeling of $\vec{\ell}$ by $\vec{k}$, and $\overline{Q}'$ being the label context obtained by the relabeling of $\vec{\ell'}$ by fresh $\vec{k'}$; this yields that $\overline{Q}' \vdash_{\mathcal{L}} \vec{k'} : U$ is valid, and by Lemma 6.2.3, $\emptyset; \overline{Q}' \vdash \vec{k'} : U$ is valid as well. This and $C''' : Q \to \overline{Q}', Q'$ then imply that $Q \vdash (C''', \vec{k'}) : U; Q'$ is also valid.

□

6.5 Error-Freeness

We finish this chapter by showing that Proto-Quipper-M satisfies the error-freeness property: a well-typed configuration never yields a run-time error.

Theorem 6.5.1 (Error-Freeness Theorem). *If $Q \vdash (C, M) : A; Q'$ is a valid configuration, then $(C, M) \not\Downarrow \mathrm{Error}$.*

Proof. We will show by rule induction on the error relation that when $(C, M) \Downarrow \text{Error}$, the assumption that $Q \vdash (C, M) : A; Q'$ is a valid configuration leads to a contradiction. Consider the following cases:

Error-Generation Rules: These are the rules in Table 5.4. Proving the result in these cases requires mostly applications of the Subject Reduction Theorem 6.4.2, the Parameter Value Lemma 6.1.4, and the Closed Value Lemma 6.1.3. Proofs involving these rules are similar. We illustrate with the following cases:

- $(\textit{initial-err-g})_{\Downarrow}$: The evaluation rule is

  $$\frac{(C, M) \Downarrow (C', V)}{(C, \Box_A M) \Downarrow \text{Error}} .$$

  Assume that $Q \vdash (C, \Box_A M) : A; Q'$ is valid. We want to reach a contradiction.

  By assumption, there exists a label context Q'' such that $C : Q \to Q'', Q'$ and $\emptyset; Q'' \vdash \Box_A M : A$ is valid. By the typing rule (*initial*), $\emptyset; Q'' \vdash M : 0$ is valid and therefore so is $Q \vdash (C, M) : 0; Q'$. This and $(C, M) \Downarrow (C', V)$ imply that $Q \vdash (C', V) : 0; Q'$ is valid by the Subject Reduction Theorem 6.4.2. Thus, there exists a label context Q''' such that $C' : Q \to Q''', Q'$ and $\emptyset; Q''' \vdash V : 0$ is valid. By the Parameter Value Lemma 6.1.4, $Q''' = \emptyset$, and so $\emptyset; \emptyset \vdash V : 0$ is valid. Since V is a value term of type 0, we have that V is either a variable or a label (by convention there are no constants of type 0 as 0 represents the empty set). Since the variable context is $\emptyset$, V is not a variable. Similarly, since the label context is $\emptyset$, V is not a label, a contradiction.

- $(\textit{seq-err-g})_{\Downarrow}$: The evaluation rule is

  $$\frac{(C, M) \Downarrow (C', V) \text{ where } V \neq *}{(C, M; N) \Downarrow \text{Error}} .$$

  Assume that $Q \vdash (C, M; N) : A; Q'$ is valid. We want to reach a contradiction.

  By assumption, there exists a label context Q'' such that $C : Q \to Q'', Q'$ and $\emptyset; Q'' \vdash M; N : A$ is valid. By the typing rule (*seq*), there are label contexts Q''_1 and Q''_2 such that $Q'' = Q''_1, Q''_2$, and both $\emptyset; Q''_1 \vdash M : I$ and $\emptyset; Q''_2 \vdash N : A$ are valid. Since $C : Q \to Q''_1, Q''_2, Q'$ and $\emptyset; Q''_1 \vdash M : I$ is valid, it follows that $Q \vdash (C, M) : I; Q''_2, Q'$ is valid as well. This and $(C, M) \Downarrow (C', V)$ imply that $Q \vdash (C', V) : I; Q''_2, Q'$ is valid by the Subject Reduction Theorem 6.4.2. Thus, there exists a label context Q''' such that $C' : Q \to Q''', Q''_2, Q'$ and $\emptyset; Q''' \vdash V : I$ is valid. By the Parameter Value Lemma 6.1.4, $Q''' = \emptyset$, and so, $\emptyset; \emptyset \vdash V : I$. But by the Closed Value Lemma 6.1.3, $V = *$, a contradiction.

- $(\textit{apply-err-g-2})_{\Downarrow}$: The evaluation rule is

$$\frac{(C, M) \Downarrow (C', (\vec{\ell}, D, \vec{\ell'})) \quad (C', N) \Downarrow (C'', V) \text{ where } V \neq \vec{k}}{(C, \text{apply}(M, N)) \Downarrow \text{Error}} .$$

Assume that $Q \vdash (C, \text{apply}(M, N)) : U; Q'$ is valid. We want to reach a contradiction.

By assumption, there exists a label context Q'' such that $C : Q \to Q'', Q'$ and $\emptyset; Q'' \vdash \text{apply}(M, N) : U$ is valid. By the (*apply*) rule, there exist label contexts Q''_1 and Q''_2 such that $Q'' = Q''_1, Q''_2$ and both $\emptyset; Q''_1 \vdash M : \text{Circ}(T, U)$ and $\emptyset; Q''_2 \vdash N : T$ are valid where T and U are simple M-types. The former and $C : Q \to Q''_1, Q''_2, Q'$ imply that $Q \vdash (C, M) : \text{Circ}(T, U); Q''_2, Q'$ is valid as well. This and $(C, M) \Downarrow (C', (\vec{\ell}, D, \vec{\ell'}))$ imply that the judgement $Q \vdash (C', (\vec{\ell}, D, \vec{\ell'})) : \text{Circ}(T, U); Q''_2, Q'$ is also valid by the Subject Reduction Theorem 6.4.2. Thus, there is a label context Q''' such that $C' : Q \to Q''', Q''_2, Q'$ and $\emptyset; Q''' \vdash (\vec{\ell}, D, \vec{\ell'}) : \text{Circ}(T, U)$ is valid. By the typing rule (*circ*), $Q''' = \emptyset$. Since $C' : Q \to Q''_2, Q'$ and $\emptyset; Q''_2 \vdash N : T$ is valid, we have that $Q \vdash (C', N) : T; Q'$ is valid. This and $(C', N) \Downarrow (C'', W)$ imply that $Q \vdash (C'', W) : T; Q'$ is valid as well by the Subject Reduction Theorem 6.4.2. Thus, there exists a label context $\overline{Q}$ such that $C'' : Q \to \overline{Q}, Q'$ and $\emptyset; \overline{Q} \vdash W : T$ is valid. But T is a simple M-type, and thus, the Closed Value Lemma 6.1.3 implies that W is a *label* tuple, a contradiction.

- $(\textit{box-err-g-2})_{\Downarrow}$: The evaluation rule is

$$\frac{(C, M) \Downarrow (C', \text{lift } N) \quad \textit{freshlabels}(T) = (Q, \vec{\ell}) \quad (\text{id}_Q, N\vec{\ell}) \Downarrow (D, V) \text{ where } V \neq \vec{\ell'}}{(C, \text{box}_T\, M) \Downarrow \text{Error}} .$$

Assume that $Q \vdash (C, \mathrm{box}_T\, M) : A; Q'$ is valid. We want to reach a contradiction.

By assumption, there exists a label context Q'' such that $C : Q \to Q'', Q'$ and $\emptyset; Q'' \vdash \mathrm{box}_T\, M : A$ is valid. By the typing rule (*box*), $A = \mathrm{Circ}(T, U)$ for some simple M-type U, and $\emptyset; Q'' \vdash M : !(T \multimap U)$ is valid. This and $C : Q \to Q'', Q'$ imply that $Q \vdash (C, M) : !(T \multimap U); Q'$ is valid as well, which along with $(C, M) \Downarrow (C', \mathrm{lift}\ N)$ yields the validity of $Q \vdash (C', \mathrm{lift}\ N) : !(T \multimap U); Q'$ by the Subject Reduction Theorem 6.4.2. Thus, there exists a label context Q''' such that $C' : Q \to Q''', Q'$ and $\emptyset; Q''' \vdash \mathrm{lift}\ N : !(T \multimap U)$ is valid. By the typing rule (*lift*), $Q''' = \emptyset$ and $\emptyset; \emptyset \vdash N : T \multimap U$ is also valid.

Since *freshlabels*$(T) = (\overline{Q}, \vec{\ell})$, we have that $\overline{Q} \vdash_{\mathcal{L}} \vec{\ell} : T$ is valid, and by Lemma 6.2.3, $\emptyset; \overline{Q} \vdash \vec{\ell} : T$ is valid as well. This and $\emptyset; \emptyset \vdash N : T \multimap U$ yield that $\emptyset; \overline{Q} \vdash N\vec{\ell} : U$ is valid by the (*app*) rule, which along with $id_{\overline{Q}} : \overline{Q} \to \overline{Q}, \emptyset$ imply that $\overline{Q} \vdash (id_{\overline{Q}}, N\vec{\ell}) : U; \emptyset$ is also valid. But then $(\mathrm{id}_{\overline{Q}}, N\vec{\ell}) \Downarrow (D, V)$ imply that $\overline{Q} \vdash (D, V) : U; \emptyset$ is valid by the Subject Reduction Theorem 6.4.2. Thus, there exists a label context $\overline{\overline{Q}}$ such that $D : \overline{Q} \to \overline{\overline{Q}}, \emptyset$ and $\emptyset; \overline{\overline{Q}} \vdash V : U$ is valid. Now U is a simple M-type, and thus, the Closed Value Lemma 6.1.3 implies that V is a *label* tuple, a contradiction.

Error-Propagation Rules: These are the rules with an error-configuration in the hypothesis. See tables 5.5 and 5.6. Proving the result for half of them requires mostly an immediate application of the induction hypothesis. For the other half, besides the induction hypothesis, the Subject Reduction Theorem 6.4.2 and the Substitution Lemma 6.3.1 are needed. We illustrate with the following cases:

- (*left-err-p*)$_\Downarrow$: The evaluation rule is

$$\frac{(C, M) \Downarrow \mathrm{Error}}{(C, \mathrm{left}_{A,B}\, M) \Downarrow \mathrm{Error}}\ .$$

Assume that $Q \vdash (C, \mathrm{left}_{A,B}\, M) : A + B; Q'$ is valid. We want to reach a contradiction.

By assumption, there exists a label context Q'' such that $C : Q \to Q'', Q'$ and $\emptyset; Q'' \vdash \mathrm{left}_{A,B}\, M : A + B$ is valid. By the (*left*) rule, $\emptyset; Q'' \vdash M : A$ is valid. So $Q \vdash (C, M) : A; Q'$ is valid as well. But since $(C, M) \Downarrow \mathrm{Error}$, the induction hypothesis implies a contradiction.

- ($let\text{-}err\text{-}p\text{-}2$)$_{\Downarrow}$: The evaluation rule is

$$\frac{(C, M) \Downarrow (C', V) \quad (C', N[V/x]) \Downarrow \mathrm{Error}}{(C, \mathrm{let}\ x = M\ \mathrm{in}\ N) \Downarrow \mathrm{Error}}\ .$$

  Assume that $Q \vdash (C, \mathrm{let}\ x = M\ \mathrm{in}\ N) : B; Q'$ is valid. We want to reach a contradiction.

  By assumption, there exists a label context Q'' such that $C : Q \to Q'', Q'$ and $\emptyset; Q'' \vdash \mathrm{let}\ x = M\ \mathrm{in}\ N : B$ is valid. By the (*let*) rule, there are label contexts Q''_1 and Q''_2 such that $Q'' = Q''_1, Q''_2$, and both $\emptyset; Q''_1 \vdash M : A$ and $x : A; Q''_2 \vdash N : B$ are valid. The former and $C : Q \to Q''_1, Q''_2, Q'$ imply that $Q \vdash (C, M) : A; Q''_2, Q'$ is valid as well. This and $(C, M) \Downarrow (C', V)$ imply the validity of $Q \vdash (C', V) : A; Q''_2, Q'$ by the Subject Reduction Theorem 6.4.2. Thus, there is a label context Q''' such that $C' : Q \to Q''', Q''_2, Q'$ and $\emptyset; Q''' \vdash V : A$ is valid. This and $x : A; Q''_2 \vdash N : B$ imply that $\emptyset; Q''', Q''_2 \vdash N[V/x] : B$ is valid by the Substitution Lemma 6.3.1. So $Q \vdash (C', N[V/x]) : B; Q'$ is valid as well. But since $(C', N[V/x]) \Downarrow \mathrm{Error}$, the induction hypothesis implies a contradiction.

- ($app\text{-}err\text{-}p\text{-}3$)$_{\Downarrow}$: The evaluation rule is

$$\frac{(C, M) \Downarrow (C', \lambda x^A.M') \quad (C', N) \Downarrow (C'', V) \quad (C'', M'[V/x]) \Downarrow \mathrm{Error}}{(C, MN) \Downarrow \mathrm{Error}}\ .$$

  Assume that $Q \vdash (C, MN) : B; Q'$ is valid. We want to reach a contradiction.

  By assumption, there exists a label context Q'' such that $C : Q \to Q'', Q'$ and $\emptyset; Q'' \vdash MN : B$ is valid. By the (*app*) rule, there are label contexts Q''_1 and Q''_2 such that $Q'' = Q''_1, Q''_2$, and both $\emptyset; Q''_1 \vdash M : A \multimap B$ and $\emptyset; Q''_2 \vdash N : A$ are valid. The former and $C : Q \to Q''_1, Q''_2, Q'$ imply that $Q \vdash (C, M) : A \multimap B; Q''_2, Q'$ is valid. This and $(C, M) \Downarrow (C', \lambda x^A.M')$ imply that $Q \vdash (C', \lambda x^A.M') : A \multimap B; Q''_2, Q'$ is valid by the Subject

Reduction Theorem 6.4.2. Thus, there is a label context Q''' such that $C' : Q \to Q''', Q''_2, Q'$ and $\emptyset; Q''' \vdash \lambda x^A.M' : A \multimap B$ is valid. By the (abs) rule, $x : A; Q''' \vdash M' : B$ is also valid.

Since $C' : Q \to Q''', Q''_2, Q'$ and $\emptyset; Q''_2 \vdash N : A$ is a valid typing judgement, we have that $Q \vdash (C', N) : A; Q''', Q'$ is valid as well. This and $(C', N) \Downarrow (C'', V)$ imply that $Q \vdash (C'', V) : A; Q''', Q'$ is also valid by the Subject Reduction Theorem 6.4.2. Thus, there exists a label context $\overline{Q}$ such that $C'' : Q \to \overline{Q}, Q''', Q'$ and $\emptyset; \overline{Q} \vdash V : A$ is valid. This and $x : A; Q''' \vdash M' : B$ imply that $\emptyset; \overline{Q}, Q''' \vdash M'[V/x] : B$ is valid by the Substitution Lemma 6.3.1. But since $(C'', M'[V/x]) \Downarrow$ Error, the induction hypothesis implies a contradiction.

□

Chapter 7

The Semantics of Proto-Quipper-M

In Chapter 4, we gave an informal introduction to Proto-Quipper-M and its categorical semantics, mostly to motivate and put into context the contents of this thesis. In this chapter, we provide a generalization of the categorical model discussed there and show that it yields a linear-non-linear model, thus a model of intuitionistic linear logic. We also introduce the complete categorical semantics of Proto-Quipper-M in full detail. Further categorical properties of our language, such as soundness, will be discussed in later chapters.

7.1 A General Categorical Model

In this section, we introduce an LNL model of Proto-Quipper-M. This model is given in terms of well-known categorical structures such as copower and representable functors. But first, we should be more specific about what we mean by a categorical model of Proto-Quipper-M. As in Chapter 4, we fix a locally small symmetric monoidal category $\mathbf{M}$, called the category of *generalized circuits.*

Definition 7.1.1. A *model of Proto-Quipper-M* is given by

- a locally small symmetric monoidal closed category $\mathbf{L}$ with coproducts, called the category of *denotations* of Proto-Quipper-M; and
- a full strong symmetric monoidal *embedding* $\mathcal{Y} : \mathbf{M} \to \mathbf{L}$ with structure given by isomorphisms $r_{A,B} : \mathcal{Y}(A) \otimes \mathcal{Y}(B) \to \mathcal{Y}(A \otimes B)$ and $r_{I_\mathbf{M}} : I_\mathbf{L} \to \mathcal{Y}(I_\mathbf{M})$ natural in $A, B \in \mathbf{M}$.

7.1.1 Copowers and Representables

We begin by introducing the main components of our LNL model, namely, a pair of functors between a category $\mathbf{L}$ of denotations and the cartesian category $\mathbf{Set}$.

Definition 7.1.2. Suppose that $\mathbf{L}$ is a category with coproducts and L is an object of $\mathbf{L}$. We define a functor $-\odot L : \mathbf{Set} \to \mathbf{L}$, called the *copower functor* of L. It assigns to each set X, the object $X \odot L$ given by $\coprod_{x\in X} L_x$, where $L_x = L$ for each $x \in X$, and to each function $f : X \to Y$, the unique induced morphism $f \odot L : X \odot L \to Y \odot L$ given by $[\iota_{f(x)}]_{x\in X} : \coprod_{x\in X} L_x \to \coprod_{y\in Y} L_y$, where $[\iota_{f(x)}]_{x\in X} \circ \iota_x = \iota_{f(x)}$ for each $x \in X$. We call $X \odot L$ the *X-indexed copower* of L. $\diamond$

Note that by the universal mapping property of coproducts, $- \odot L : \mathbf{Set} \to \mathbf{L}$ is indeed a functor for each object L of $\mathbf{L}$. To simplify notation, we may write $[\iota_{f(x)}]_x$ instead of $[\iota_{f(x)}]_{x\in X}$. The following result will be useful.

Lemma 7.1.3. *Suppose that* $\mathbf{L}$ *is a locally small category with coproducts and* L *is an object of* $\mathbf{L}$*. Then the copower functor* $- \odot L : \mathbf{Set} \to \mathbf{L}$ *is a left adjoint of the representable functor* $\mathbf{L}(L, -) : \mathbf{L} \to \mathbf{Set}$.

Proof. By Theorem 3.1.23, to show that $- \odot L \dashv \mathbf{L}(L, -)$, it suffices to find a natural transformation $\eta : 1_{\mathbf{Set}} \Rightarrow \mathbf{L}(L, -) \circ (- \odot L)$ such that for every set X, $\eta_X : X \to \mathbf{L}(L, X \odot L)$ is a universal arrow from X to $\mathbf{L}(L, -)$.

Let X be a set and $\eta_X : X \to \mathbf{L}(L, X \odot L)$ a function defined for each $x \in X$ by $\eta_X(x) = \iota_x : L \to \coprod_{x\in X} L_x$. To prove naturality of η, let Y be a set and $f : X \to Y$ a function. We show that the following diagram commutes:

$$\begin{tikzcd}
X \arrow[r, "\eta_X"] \arrow[d, "f"'] & \mathbf{L}(L, X \odot L) \arrow[d, "\mathbf{L}(L, f\odot L)"] \\
Y \arrow[r, "\eta_Y"'] & \mathbf{L}(L, Y \odot L)
\end{tikzcd} \tag{7.1}$$

Let $x \in X$, then, by unraveling definitions, we have that

$$\begin{aligned}
(\mathbf{L}(L, f \odot L) \circ \eta_X)(x) &= \mathbf{L}(L, f \odot L)(\eta_X(x)) \\
&= (f \odot L) \circ \eta_X(x) \\
&= [\iota_{f(x)}]_{x\in X} \circ \eta_X(x) \\
&= [\iota_{f(x)}]_{x\in X} \circ \iota_x \\
&= \iota_{f(x)} \\
&= \eta_Y(f(x)) \\
&= (\eta_Y \circ f)(x),
\end{aligned}$$

and so $\mathbf{L}(L, f \odot L) \circ \eta_X = \eta_Y \circ f$, that is, (7.1) commutes.

To prove universality of $\eta_X : X \to \mathbf{L}(L, X \odot L)$, let M be an object of $\mathbf{L}$ and $f : X \to \mathbf{L}(L, M)$ a function. We show that there is a unique arrow $\overline{f} : X \odot L \to M$ in $\mathbf{L}$ such that the following diagram commutes:

$$\begin{tikzcd}
X \arrow[r, "\eta_X"] \arrow[d, "\mathrm{id}"'] & \mathbf{L}(L, X \odot L) \arrow[d, "\mathbf{L}(L, \overline{f})"] \\
X \arrow[r, "f"'] & \mathbf{L}(L, M)
\end{tikzcd} \tag{7.2}$$

Define $\overline{f}$ to be the coparing morphism $[f(x)]_{x \in X} : X \odot L \to M$. Let $x \in X$, then, by unraveling definitions,

$$\begin{aligned}
(\mathbf{L}(L, \overline{f}) \circ \eta_X)(x) &= \mathbf{L}(L, \overline{f})(\eta_X(x)) \\
&= \overline{f} \circ \eta_X(x) \\
&= [f(x)]_{x \in X} \circ \eta_X(x) \\
&= [f(x)]_{x \in X} \circ \iota_x \\
&= f(x),
\end{aligned}$$

and so $\mathbf{L}(L, \overline{f}) \circ \eta_X = f$, that is, (7.2) commutes.

To show uniqueness of $\overline{f}$, suppose that $g : X \odot L \to M$ is a morphism in $\mathbf{L}$ such that $\mathbf{L}(L, g) \circ \eta_X = f$ as well. Thus, for each $x \in X$,

$$\begin{aligned}
(\mathbf{L}(L, g) \circ \eta_X)(x) &= \mathbf{L}(L, g)(\eta_X(x)) \\
&= g \circ \eta_X(x) \\
&= g \circ \iota_x,
\end{aligned}$$

but then $g \circ \iota_x = f(x) = [f(x)]_x \circ \iota_x$, which implies that $g = [f(x)]_x = \overline{f}$. The result now follows. □

Convention 7.1.4. Let $\mathbf{L}$ be a locally small symmetric monoidal closed category with coproducts and unit object I. We denote the copower functor $- \odot I : \mathbf{Set} \to \mathbf{L}$ by p and the representable functor $\mathbf{L}(I, -) : \mathbf{L} \to \mathbf{Set}$ by $\flat$, which we pronounce "flat." ◇

7.1.2 An LNL Model

We now show that the adjunction of the previous section lifts to a symmetric monoidal one, and thus yields an LNL model of Proto-Quipper-M. Though elementary, this result is significant as LNL models subsume other well-known categorical models of intuitionistic linear logic such as Lafont categories, Seely categories, and linear categories. Moreover, assuming that $\mathbf{M}$ is a locally small symmetric monoidal category, this result also implies that the families construction $\overline{\overline{\overline{\mathbf{M}}}}$ of Chapter 4 indeed yields an instance of an LNL model of Proto-Quipper-M, and thus an instance of a model of intuitionistic linear logic.

Theorem 7.1.5. *The cartesian closed category* $\mathbf{Set}$*, the locally small symmetric monoidal closed category* $\mathbf{L}$*, and the functors* p *and* $\flat$ *yield a linear-non-linear model.*

Proof. By Lemma 7.1.3, $p \dashv \flat$, and so to show that this adjunction lifts to a symmetric monoidal one, it suffices, by Proposition 3.2.12, the characterization of symmetric monoidal adjunctions, to endow p with a structure that makes it a strong symmetric monoidal functor.

Since $\mathbf{L}$ is a symmetric monoidal closed category, the endofunctor $A \otimes -$ is a left adjoint, and so preserves colimits; in particular, we have that $A \otimes (B + C) \cong A \otimes B + A \otimes C$ for all $A, B, C \in \mathbf{L}$. More generally, let X, Y and Z be sets, we observe that

$$(\coprod_{x \in X} A_x \otimes \coprod_{y \in Y} B_y, \{\iota_x \otimes \iota_y : A_x \otimes B_y \to \coprod_{x \in X} A_x \otimes \coprod_{y \in Y} B_y \mid (x,y) \in X \times Y\})$$

and

$$(\coprod_{(x,y) \in X \times Y} (A_x \otimes B_y), \{\bar{\iota}_{(x,y)} : A_x \otimes B_y \to \coprod_{(x,y) \in X \times Y} (A_x \otimes B_y) \mid (x,y) \in X \times Y\})$$

are naturally isomorphic coproducts in $\mathbf{L}$ with mediating isomorphisms given by the corresponding copairing maps

$$[\bar{\iota}_{(x,y)}]'_{(x,y) \in X \times Y} : \coprod_{x \in X} A_x \otimes \coprod_{y \in Y} B_y \longrightarrow \coprod_{(x,y) \in X \times Y} (A_x \otimes B_y)$$

and

$$[\iota_x \otimes \iota_y]_{(x,y) \in X \times Y} : \coprod_{(x,y) \in X \times Y} (A_x \otimes B_y) \longrightarrow \coprod_{x \in X} A_x \otimes \coprod_{y \in Y} B_y.$$

Now consider the special case where $A_x = I$ and $B_y = I$ for all $x \in X$ and $y \in Y$. Since $\lambda_I : I \otimes I \to I$ is a natural isomorphism in I, we have that the coproducts

$$(\textstyle\coprod_{(x,y)\in X\times Y}(I_x\otimes I_y), \{\bar{\iota}_{(x,y)} : I_x\otimes I_y \to \coprod_{(x,y)\in X\times Y}(I_x\otimes I_y) \mid (x,y)\in X\times Y\})$$

and

$$(\textstyle\coprod_{(x,y)\in X\times Y} I_{(x,y)}, \{\iota_{(x,y)} : I_{(x,y)} \to \coprod_{(x,y)\in X\times Y} I_{(x,y)} \mid (x,y)\in X\times Y\})$$

are naturally isomorphic with mediating isomorphisms

$$[\iota_{(x,y)}\circ\lambda_I]_{(x,y)\in X\times Y} : \textstyle\coprod_{(x,y)\in X\times Y}(I_x\otimes I_y) \longrightarrow \coprod_{(x,y)\in X\times Y} I_{(x,y)}$$

and

$$[\bar{\iota}_{(x,y)}\circ\lambda_I^{-1}]_{(x,y)\in X\times Y} : \textstyle\coprod_{(x,y)\in X\times Y} I_{(x,y)} \longrightarrow \coprod_{(x,y)\in X\times Y}(I_x\otimes I_y).$$

For all sets X and Y, let $m_{X,Y} : p(X)\otimes p(Y) \to p(X\times Y)$ be the composition $[\iota_{(x,y)}\circ\lambda_I]_{(x,y)}\circ[\bar{\iota}_{(x,y)}]'_{(x,y)}$, and let $m_1 : I \to p(1)$ be the identity morphism on I. We show that these morphisms endow p with the required structure by proving that the following coherence conditions hold:

- **associator:** We will show that the diagram

$$\begin{tikzcd}
((p(X)\otimes p(Y))\otimes p(Z) \arrow[r, "\alpha_{p(X),p(Y),p(Z)}"] \arrow[d, "m_{X,Y}\otimes \mathrm{id}"'] & p(X)\otimes(p(Y)\otimes p(Z)) \arrow[d, "\mathrm{id}\otimes m_{Y,Z}"] \\
p(X\times Y)\otimes p(Z) \arrow[d, "m_{X\times Y,Z}"'] & p(X)\otimes p(Y\times Z) \arrow[d, "m_{X,Y\times Z}"] \\
p((X\times Y)\times Z) \arrow[r, "p(\alpha_{X,Y,Z})"'] & p(X\times(Y\times Z))
\end{tikzcd} \tag{7.3}$$

commutes. Since $(p(X)\otimes p(Y))\otimes p(Z) = (\coprod_{x\in X} I_x \otimes \coprod_{y\in Y} I_y)\otimes\coprod_{z\in Z} I_z$ is a coproduct, it suffices to show that the top and bottom paths of (7.3) yield the same morphism when precomposed with an injection

$$(\iota_x\otimes\iota_y)\otimes\iota_z : (I_x\otimes I_y)\otimes I_z \longrightarrow (\textstyle\coprod_{x\in X} I_x\otimes\coprod_{y\in Y} I_y)\otimes\coprod_{z\in Z} I_z.$$

Consider the diagram

$$\begin{tikzcd}
I_{(x,y)} \otimes I_z \arrow[r, "\lambda_I^{-1}\otimes \mathrm{id}"] \arrow[ddd, "\mathrm{id}"'] & (I_x \otimes I_y) \otimes I_z \arrow[r, "(\iota_x\otimes\iota_y)\otimes\iota_z"] \arrow[d, "\mathrm{id}\otimes\iota_z"'] & (\coprod_{x\in X} I_x \otimes \coprod_{y\in Y} I_y) \otimes \coprod_{z\in Z} I_z \arrow[ddd, "m_{X,Y}\otimes\mathrm{id}"] \\
 & (I_x \otimes I_z) \otimes \coprod_{z\in Z} I_z \arrow[ur, "(\iota_x\otimes\iota_y)\otimes\mathrm{id}"'] \arrow[d, "\lambda_I\otimes\mathrm{id}"'] & \\
 & I_{(x,y)} \otimes \coprod_{z\in Z} I_z \arrow[dr, "\iota_{(x,y)}\otimes\mathrm{id}"] & \\
I_{(x,y)} \otimes I_z \arrow[ur, "\mathrm{id}\otimes\iota_z"] \arrow[rr, "\iota_{(x,y)}\otimes\iota_z"'] & & \coprod_{(x,y)\in X\times Y} I_{(x,y)} \otimes \coprod_{z\in Z} I_z.
\end{tikzcd}$$

(7.4)

The region on the left as well as the top and bottom triangles commute by functoriality of $\otimes$. The region on the right commutes by definition of $m_{X,Y}$ and functoriality of $\otimes$.

The commutativity of

$$\begin{tikzcd}
I_{(x,y)} \otimes I_z \arrow[rr, "\iota_{(x,y)}\otimes\iota_z"] \arrow[dr, "\bar{\iota}_{((x,y),z)}"] \arrow[dd, "\lambda_I"'] & & \coprod_{(x,y)\in X\times Y} I_{(x,y)} \otimes \coprod_{z\in Z} I_z \arrow[dl, "{[\bar{\iota}_{((x,y),z)}]_{((x,y),z)}}"'] \arrow[dd, "m_{X\times Y,Z}"] \\
 & \coprod_{((x,y),z)\in(X\times Y)\times Z} I_{(x,y)} \otimes I_z \arrow[dr, "{[\iota_{((x,y),z)}\circ\lambda_I]_{((x,y),z)}}"] & \\
I_{((x,y),z)} \arrow[rr, "\iota_{((x,y),z)}"'] & & \coprod_{((x,y),z)\in(X\times Y)\times Z} I_{((x,y),z)}
\end{tikzcd}$$

(7.5)

follows immediately: The top and right triangles commute by definition of morphisms $[\bar{\iota}_{((x,y),z)}]_{((x,y),z)}$ and $m_{X\times Y,Z}$, respectively, while the bottom diagram does by definition of $[\iota_{((x,y),z)} \circ \lambda_I]_{((x,y),z)}$.

Now consider

$$\begin{tikzcd}
I_{(x,y)} \otimes I_z \arrow[rrr, "\mathrm{id}"] \arrow[dr, "\lambda_I^{-1}\otimes \mathrm{id}"] \arrow[ddd, "\mathrm{id}"'] & & & I_x \otimes I_{(y,z)} \arrow[dl, "\mathrm{id}\otimes\lambda_I^{-1}"'] \arrow[ddd, "\mathrm{id}"] \\
& (I_x \otimes I_y) \otimes I_z \arrow[r, "\alpha_{I_x,I_y,I_z}"] \arrow[d, "(\iota_x\otimes\iota_y)\otimes\iota_z"'] & I_x \otimes (I_y \otimes I_z) \arrow[d, "\iota_x\otimes(\iota_y\otimes\iota_z)"] & \\
& (\coprod_{x\in X} I_x \otimes \coprod_{y\in Y} I_y) \otimes \coprod_{z\in Z} I_z \arrow[r, "\alpha_{p(X),p(Y),p(Z)}"] \arrow[d, "m_{X,Y}\otimes\mathrm{id}"'] & \coprod_{x\in X} I_x \otimes (\coprod_{y\in Y} I_y \otimes \coprod_{z\in Z} I_z) \arrow[d, "\mathrm{id}\otimes m_{Y,Z}"] & \\
I_{(x,y)} \otimes I_z \arrow[r, "\iota_{(x,y)}\otimes\iota_z"] \arrow[dd, "\lambda_I"'] & \coprod_{(x,y)\in X\times Y} I_{(x,y)} \otimes \coprod_{z\in Z} I_z \arrow[d, "m_{X\times Y,Z}"'] & \coprod_{x\in X} I_x \otimes (\coprod_{(y,z)\in Y\times Z} I_{(y,z)}) \arrow[d, "m_{X,Y\times Z}"] & I_x \otimes I_{(y,z)} \arrow[l, "\iota_x\otimes\iota_{(y,z)}"'] \arrow[dd, "\lambda_I"] \\
& \coprod_{((x,y),z)\in(X\times Y)\times Z} I_{((x,y),z)} \arrow[r, "p(\alpha_{X,Y,Z})"] & \coprod_{(x,(y,z))\in X\times(Y\times Z)} I_{(x,(y,z))} & \\
I_{((x,y),z)} \arrow[rrr, "\mathrm{id}"'] \arrow[ur, "\iota_{((x,y),z)}"] & & & I_{(x,(y,z))}. \arrow[ul, "\iota_{(x,(y,z))}"]
\end{tikzcd}$$

The top diagram commutes because **L** is a symmetric monoidal category, while the one below it does by naturality of α. The top left diagram commutes by (7.4), while the one below it does by (7.5). The corresponding diagrams on the right commute by similar reasons. The region at the bottom commutes by definition of p. The commutativity of the outer square is immediate. This and the fact that all the outer regions commute, and the top left morphism $\lambda_I^{-1} \otimes \mathrm{id}$ is an isomorphism imply that the inner region composed of the two inner squares commutes as well. An elementary diagram chase shows that the morphisms

$$(m_{X,Y\times Z} \circ (\mathrm{id} \otimes m_{Y,Z}) \circ \alpha_{p(X),p(Y),p(Z)}) \circ ((\iota_x \otimes \iota_y) \otimes \iota_z)$$

and

$$(p(\alpha_{X,Y,Z}) \circ m_{X\times Y,Z} \circ (m_{X,Y} \otimes \mathrm{id})) \circ ((\iota_x \otimes \iota_y) \otimes \iota_z),$$

are equal, and so by the unique mapping property of coproducts, the region above the bottom one commutes, that is, (7.3) commutes.

- **left unitor:** We will show that the diagram

$$\begin{tikzcd}
I \otimes p(X) \arrow[r, "\lambda_{p(X)}"] \arrow[d, "m_1\otimes\mathrm{id}"'] & p(X) \\
p(1) \otimes p(X) \arrow[r, "m_{1,X}"'] & p(1 \times X) \arrow[u, "p(\pi_X)"']
\end{tikzcd} \tag{7.6}$$

commutes. Since $I \otimes p(X) = p(1) \otimes p(X) = \coprod_{*\in 1} I_* \otimes \coprod_{x\in X} I_x$ is a coproduct, it suffices to show that the top and bottom paths of (7.6) yield the same morphism when precomposed with an injection $\iota_* \otimes \iota_x : I_* \otimes I_x \to \coprod_{*\in 1} I_* \otimes \coprod_{x\in X} I_x$. Consider the following diagram:

$$\begin{tikzcd}
\coprod_{*\in 1} I_* \otimes \coprod_{x\in X} I_x \arrow[rr, "m_{1,X}"] & & \coprod_{(*,x)\in 1\times X} I_{(*,x)} \arrow[dd, "\mathrm{id}"] \\
 & \coprod_{(*,x)\in 1\times X}(I_* \otimes I_x) \arrow[ur, "{[\iota_{(*,x)} \circ \lambda_I]_{(*,x)}}"] & \\
I_* \otimes I_x \arrow[uu, "\iota_* \otimes \iota_x"] \arrow[ur, "\bar{\iota}_{(*,x)}"] \arrow[r, "\lambda_I"] \arrow[d, "\iota_* \otimes \iota_x"'] \arrow[dr, "\mathrm{id}\otimes \iota_x"] & I_{(*,x)} = I_x \arrow[r, "\iota_{(*,x)} = \iota_x"] & \coprod_{(*,x)\in 1\times X} I_{(*,x)} = \coprod_{x\in X} I_x \arrow[d, "p(\pi_X) = \mathrm{id}"] \\
\coprod_{*\in 1} I_* \otimes \coprod_{x\in X} I_x \arrow[r, "(m_1 \otimes \mathrm{id})^{-1} = \mathrm{id}"'] & I_* \otimes \coprod_{x\in X} I_x \arrow[ur, "\lambda_{\coprod_{x\in X} I_x}"] \arrow[r, "\lambda_{\coprod_{x\in X} I_x}"'] & \coprod_{x\in X} I_x
\end{tikzcd} \tag{7.7}$$

The top triangle commute by definition of $m_{1,X}$, while the one below it does so by definition of $[\iota_{(*,x)} \circ \lambda_I]_{(*,x)}$. The middle triangle commutes by naturality of λ. The commutativity of the bottom two triangles is immediate. Hence, the top and bottom paths of (7.7) are equal, that is,

$$(p(\pi_X) \circ m_{1,X}) \circ (\iota_* \otimes \iota_x) = (\lambda_{\coprod_{x\in X} I_x} \circ (m_1 \otimes \mathrm{id})^{-1}) \circ (\iota_* \otimes \iota_x),$$

and so, by the unique mapping property of coproducts,

$$p(\pi_X) \circ m_{1,X} = \lambda_{\coprod_{x\in X} I_x} \circ (m_I \otimes \mathrm{id})^{-1}.$$

This implies that (7.6) commutes.

- **right unitor:** To show that the diagram

$$\begin{tikzcd}
p(X) \otimes I \arrow[r, "\rho_{p(X)}"] \arrow[d, "\mathrm{id}\otimes m_1"'] & p(X) \\
p(X) \otimes p(1) \arrow[r, "m_{X,1}"'] & p(X \times 1) \arrow[u, "p(\pi_X)"']
\end{tikzcd}$$

commutes, one proceeds as in the previous case.

- **symmetry:** We will show that the diagram

$$\begin{tikzcd}
p(X) \otimes p(Y) \arrow[r, "\sigma_{p(X),p(Y)}"] \arrow[d, "m_{X,Y}"'] & p(Y) \otimes p(X) \arrow[d, "m_{Y,X}"] \\
p(X \times Y) \arrow[r, "p(\sigma_{X,Y})"'] & p(Y \times X)
\end{tikzcd} \tag{7.8}$$

commutes. Since $p(X) \otimes p(Y) = \coprod_{x\in X} I_x \otimes \coprod_{y\in Y} I_y$ is a coproduct, it suffices to show that the top and bottom paths of (7.8) yield the same morphism when precomposed with an injection $\iota_x \otimes \iota_y : I_x \otimes I_y \to \coprod_{x\in X} I_x \otimes \coprod_{y\in Y} I_y$. Consider the following diagram:

$$\begin{tikzcd}
I_x \otimes I_y \arrow[rrr, "\sigma_{I_x,I_y}"] \arrow[dr, "\iota_x\otimes\iota_y"] \arrow[ddd, "\lambda_I"'] & & & I_y \otimes I_x \arrow[dl, "\iota_y\otimes\iota_x"'] \arrow[ddd, "\lambda_I"] \\
& \coprod_{x\in X} I_x \otimes \coprod_{y\in Y} I_y \arrow[r, "\sigma_{p(X),p(Y)}"] \arrow[d, "m_{X,Y}"'] & \coprod_{y\in Y} I_y \otimes \coprod_{x\in X} I_x \arrow[d, "m_{Y,X}"] & \\
& \coprod_{(x,y)\in X\times Y} I_{(x,y)} \arrow[r, "p(\sigma_{X,Y})"'] & \coprod_{(y,x)\in Y\times X} I_{(y,x)} & \\
I_{(x,y)} \arrow[ur, "\iota_{(x,y)}"] \arrow[rrr, "\mathrm{id}"'] & & & I_{(y,x)}. \arrow[ul, "\iota_{(y,x)}"']
\end{tikzcd} \tag{7.9}$$

The top diagram commutes by naturality of σ, while the left and right diagrams do by definition of $m_{X,Y}$ and $m_{Y,X}$, respectively. The bottom diagram commutes by definition of p and the outermost square commutes because $\mathbf{L}$ is a symmetric monoidal category. A elementary diagram chase shows that

$$(m_{Y,X} \circ \sigma_{p(X),p(Y)}) \circ (\iota_x \otimes \iota_y) = (p(\sigma_{X,Y}) \circ m_{X,Y}) \circ (\iota_x \otimes \iota_y),$$

and so, by the unique mapping property of coproducts, the middle diagram commutes, that is, (7.8) commutes.

Hence, p is a strong monoidal functor and the result follows.

□

Remark 7.1.6. The category $\overline{\overline{\mathbf{M}}}$ of Chapter 4 is a model of Proto-Quipper-M because it satisfies Definition 7.1.1. In this case, the functors p and $\flat$ are given by

$p(X) = (X, (I)_{x \in X})$ and $\flat((X, (A_x)_{x \in X})) = \overline{\overline{\mathbf{M}}}((1, (I)), (X, (A_x)_{x \in X}))$, and they yield an instance of an LNL model of Proto-Quipper-M.

7.1.3 The Boxing Comonad

Naturally, the LNL model of the previous section induces a comonad on $\mathbf{L}$. We will use this additional structure to give meaning to some terms of Proto-Quipper-M. Moreover, we will see that this model also yields a model of intuitionistic linear logic.

Definition 7.1.7. Writing ! for the endofunctor $p \circ \flat : \mathbf{L} \to \mathbf{L}$, we define the *boxing comonad* to be the comonad on $\mathbf{L}$ given by $(!, \varepsilon : ! \Rightarrow 1_{\mathbf{L}}, \delta : ! \Rightarrow !!)$ where η and ε are the unit and counit of the adjunction $p \dashv \flat$, respectively, and δ is the natural transformation with components $\delta_A : !A \to !!A$ given by $\delta_A = p(\eta_{\flat(A)})$ for every object A in $\mathbf{L}$. ◇

The counit will be used in the denotational semantics of terms involving "force." In particular, for every object A in $\mathbf{L}$, we let $\mathbf{force}_A$ be $\varepsilon_A : !A \to A$. As usual, we may drop the subscripts when no confusion arises. The full meaning of terms of the form "force M" will be given in Section 7.3.4.

Our LNL model has a rich structure; in particular, the boxing comonad is symmetric monoidal:

Lemma 7.1.8 (Benton [17])**.** *In any LNL model, the comonad* $(!, \varepsilon, \delta)$ *is symmetric monoidal.* ◇

This result is rather relevant since a symmetric monoidal comonad is an appropriate categorical structure to model the ! modality of linear logic. More is true.

Theorem 7.1.9 (Benton [17])**.** *Any LNL model yields a linear category.* ◇

Theorem 7.1.10 (Benton, Bierman, de Paiva, Hyland [20]. Bierman [23])**.** *A linear category is a model of intuitionistic linear logic.* ◇

And thus, a model of Proto-Quipper-M yields a model of intuitionistic linear logic as well. In particular, the category $\overline{\overline{\mathbf{M}}}$ yields a concrete instance of such a model.

7.2 Interpreting Types

In general, a categorical semantics associates to each type A an object $[\![A]\!]$ of an appropriate category. In this section, we introduce several interpretations of the types and terms of Proto-Quipper-M along with the maps relating them.

As in Section 5.1.1, we assume that a locally small symmetric monoidal category $\mathbf{M}$ of generalized circuits has been given along with a set $\mathcal{W}$ of wire types and an interpretation function $[\![-]\!]_{\mathcal{W}} : \mathcal{W} \to \mathbf{M}_0$, assigning an object of $\mathbf{M}$ to every wire type. We denote wire types by Greek letters such as α, β, and γ.

7.2.1 Simple M-Types and Inductive Types

Before we present the semantics in $\mathbf{L}$ of all the types of Proto-Quipper-M, we give the interpretation in $\mathbf{M}$ of the simple M-types and the interpretation in $\mathbf{L}$ of the inductive types.

Definition 7.2.1. The *semantics of the simple M-types* of Proto-Quipper-M in the category $\mathbf{M}$ is given by

$$\begin{array}{rcl} [\![\alpha]\!]_s & = & [\![\alpha]\!]_{\mathcal{W}} \\ [\![I]\!]_s & = & I_{\mathbf{M}} \\ [\![T \otimes U]\!]_s & = & [\![T]\!]_s \otimes [\![U]\!]_s. \end{array}$$

◇

Remark 7.2.2. For each object A in $\mathbf{L}$, define $\mathbf{list}(A) = I + A + A \otimes A + \cdots$. Note that this is the initial algebra of the endofunctor $F : \mathbf{L} \to \mathbf{L}$ defined by $F(X) = I + A \otimes X$. This initial algebra exists because $\mathbf{L}$ is a symmetric monoidal closed category with coproducts, hence distributive. We also define $\mathbf{nat}$ to be the initial algebra $\mathbf{list}(I)$. These objects give the *semantics of the inductive types* $\mathbf{list}\,A$ and $\mathbf{nat}$, respectively.

Note that the observations in this remark apply in particular when $\mathbf{L} = \mathbf{Set}$ and $\otimes$ is the cartesian product of sets. We will use this in Definition 7.2.5 below.

7.2.2 Types in L

Let $\mathcal{Y} : \mathbf{M} \to \mathbf{L}$ be a full strong symmetric monoidal embedding with monoidal structure $r_{A,B} : \mathcal{Y}(A) \otimes \mathcal{Y}(B) \to \mathcal{Y}(A \otimes B)$ and $r_{I_{\mathbf{M}}} : I_{\mathbf{L}} \to \mathcal{Y}(I_{\mathbf{M}})$. The types of Proto-Quipper-M can now be interpreted as follows.

Definition 7.2.3. The *semantics of the types* of Proto-Quipper-M is given by

$$\begin{array}{rcl}
[\![\alpha]\!] & = & \mathcal{Y}([\![\alpha]\!]_s) \\
[\![0]\!] & = & 0_{\mathbf{L}} \\
[\![A+B]\!] & = & [\![A]\!]+[\![B]\!] \\
[\![I]\!] & = & I_{\mathbf{L}} \\
[\![A\otimes B]\!] & = & [\![A]\!]\otimes[\![B]\!] \\
[\![A\multimap B]\!] & = & [\![A]\!]\multimap[\![B]\!] \\
[\![!A]\!] & = & ![\![A]\!] \\
[\![\mathbf{nat}]\!] & = & \mathbf{nat} \\
[\![\mathbf{list}\ A]\!] & = & \mathbf{list}([\![A]\!]) \\
[\![\mathrm{Circ}(T,U)]\!] & = & p(\mathbf{M}([\![T]\!]_s,[\![U]\!]_s))
\end{array}$$

◇

Note that all the objects in this definition are indeed objects of $\mathbf{L}$. An object of $\mathbf{L}$ is called a *simple M-object* if it is of form $\mathcal{Y}(A)$ for some object A of $\mathbf{M}$. The following definition tells us that the interpretation of a simple M-type is essentially a simple M-object.

Definition 7.2.4. For every simple M-type T, the isomorphism $\mathcal{Y}_T : [\![T]\!] \to \mathcal{Y}([\![T]\!]_s)$ is given inductively by

$$\begin{array}{rcl}
\mathcal{Y}_\alpha & = & [\![\alpha]\!] \xrightarrow{\mathrm{id}_{[\![\alpha]\!]}} \mathcal{Y}([\![\alpha]\!]_s) \\
\mathcal{Y}_I & = & [\![I]\!] \xrightarrow{r_I} \mathcal{Y}([\![I]\!]_s) \\
\mathcal{Y}_{T\otimes U} & = & [\![T\otimes U]\!] \xrightarrow{\mathrm{id}} [\![T]\!]\otimes[\![U]\!] \xrightarrow{\mathcal{Y}_T\otimes\mathcal{Y}_U} \mathcal{Y}([\![T]\!]_s)\otimes\mathcal{Y}([\![U]\!]_s) \\
 & & \xrightarrow{r_{T,U}} \mathcal{Y}([\![T]\!]_s\otimes[\![U]\!]_s) \xrightarrow{\mathrm{id}} \mathcal{Y}([\![T\otimes U]\!]_s).
\end{array}$$

Here r_I and $r_{T,U}$ are the appropriate components of the monoidal structure of $\mathcal{Y}$. Also, for every label context $Q = \ell_1 : \alpha_1, \ldots, \ell_n : \alpha_n$, note that $[\![Q]\!]_s = [\![\alpha_1\otimes\ldots\otimes\alpha_n]\!]_s$, where $[\![Q]\!]_s$ was defined in Section 5.1.2 and $\alpha_1\otimes\cdots\otimes\alpha_n$ is assumed to have the same bracketing as $[\![Q]\!]_s$. The morphism $\mathcal{Y}_Q$ is defined to be the isomorphism $\mathcal{Y}_{\alpha_1\otimes\cdots\otimes\alpha_n}$. ◇

7.2.3 Morphisms box and apply

Note that for any simple M-types T and U, we have the following chain of isomorphisms:

$$\begin{array}{rcll}
\flat(\llbracket T\rrbracket \multimap \llbracket U\rrbracket) & = & \mathbf{L}(I, \llbracket T\rrbracket \multimap \llbracket U\rrbracket) & \text{by def. of } \flat \\
& \overset{i_1}{\cong} & \mathbf{L}(I \otimes \llbracket T\rrbracket, \llbracket U\rrbracket) & \text{since } \mathbf{L} \text{ is closed} \\
& \overset{i_2}{\cong} & \mathbf{L}(\llbracket T\rrbracket, \llbracket U\rrbracket) & \text{since } \lambda_T \text{ is an iso} \\
& \overset{i_3}{\cong} & \mathbf{L}(\mathcal{Y}(\llbracket T\rrbracket_s), \mathcal{Y}(\llbracket U\rrbracket_s)) & \text{since } \mathcal{Y}_T \text{ is an iso} \\
& \overset{i_4}{\cong} & \mathbf{M}(\llbracket T\rrbracket_s, \llbracket U\rrbracket_s) & \text{since } \mathcal{Y} \text{ is a full embedding.}
\end{array}$$

We take $i_{T,U} : \flat(\llbracket T\rrbracket \multimap \llbracket U\rrbracket) \to \mathbf{M}(\llbracket T\rrbracket_s, \llbracket U\rrbracket_s)$ to be the isomorphism

$$i_{T,U} = i_4 \circ i_3 \circ i_2 \circ i_1 \tag{7.10}$$

and take **box** : $!(\llbracket T\rrbracket \multimap \llbracket U\rrbracket) \to \llbracket \mathrm{Circ}(T, U)\rrbracket$ to be the isomorphism $p(i_{T,U})$; we denote its inverse by **unbox**.

Using **unbox**, **force** and the closed structure of **L**, we define the morphism **apply** : $\llbracket \mathrm{Circ}(T, U)\rrbracket \otimes \llbracket T\rrbracket \to \llbracket U\rrbracket$ as follows:

$$\mathbf{apply} = \llbracket \mathrm{Circ}(T, U)\rrbracket \otimes \llbracket T\rrbracket \xrightarrow{\mathbf{unbox}\otimes T} !(\llbracket T\rrbracket \multimap \llbracket U\rrbracket) \otimes \llbracket T\rrbracket \xrightarrow{\mathbf{force}\otimes T} (\llbracket T\rrbracket \multimap \llbracket U\rrbracket) \otimes \llbracket T\rrbracket \xrightarrow{\varepsilon_U} \llbracket U\rrbracket. \tag{7.11}$$

These morphisms will play a central role in the interpretation of the corresponding terms of Proto-Quipper-M.

7.2.4 Parameter Types

Now we give semantics to the parameter types in the category of sets and functions. In the following sections, we will see how the various type interpretations of Proto-Quipper-M interact.

Definition 7.2.5. The *semantics of the parameter types* of Proto-Quipper-M in the category **Set** is given by

$$\begin{array}{rcl}
[\![0]\!]_p & = & \emptyset \\
[\![P+R]\!]_p & = & [\![P]\!]_p + [\![R]\!]_p \\
[\![I]\!]_p & = & 1 \\
[\![P \otimes R]\!]_p & = & [\![P]\!]_p \times [\![R]\!]_p \\
[\![!A]\!]_p & = & \flat([\![A]\!]) \\
[\![\mathbf{nat}]\!]_p & = & \mathbb{N} \\
[\![\mathbf{list}\ P]\!]_p & = & \mathbf{list}([\![P]\!]_p) \\
[\![\mathrm{Circ}(T,U)]\!]_p & = & \mathbf{M}([\![T]\!]_s, [\![U]\!]_s).
\end{array}$$

◇

An object of **L** is called a *parameter object* if it is of form $p(X)$ for some set X. The following definition tells us that the interpretation of a parameter type is essentially a parameter object.

Definition 7.2.6. For every parameter type P, the isomorphism $\tau_P : [\![P]\!] \to p([\![P]\!]_p)$ is defined inductively as follows:

$$\begin{array}{rcl}
\tau_0 & = & [\![0]\!] \xrightarrow{\mathrm{id}_{[\![0]\!]}} p([\![0]\!]_p) \\
\tau_{P+R} & = & [\![P+R]\!] \xrightarrow{\mathrm{id}} [\![P]\!] + [\![R]\!] \xrightarrow{\tau_P + \tau_R} p([\![P]\!]_p) + p([\![R]\!]_p) \\
 & & \quad \xrightarrow{\cong} p([\![P]\!]_p + [\![R]\!]_p) \xrightarrow{\mathrm{id}} p([\![P+R]\!]_p) \\
\tau_I & = & [\![I]\!] \xrightarrow{m_I} p([\![I]\!]_p) \\
\tau_{P \otimes R} & = & [\![P \otimes R]\!] \xrightarrow{\mathrm{id}} [\![P]\!] \otimes [\![R]\!] \xrightarrow{\tau_P \otimes \tau_R} p([\![P]\!]_p) \otimes p([\![R]\!]_p) \\
 & & \quad \xrightarrow{m_{P,R}} p([\![P]\!]_p \times [\![R]\!]_p) \xrightarrow{\mathrm{id}} p([\![P \otimes R]\!]_p) \\
\tau_{!A} & = & [\![!A]\!] \xrightarrow{\mathrm{id}_{[\![!A]\!]}} p([\![!A]\!]_p) \\
\tau_{\mathbf{nat}} & = & [\![\mathbf{nat}]\!] \xrightarrow{\mathrm{id}_{[\![\mathbf{nat}]\!]}} p([\![\mathbf{nat}]\!]_p) \\
\tau_{\mathbf{list}\,P} & = & [\![\mathbf{list}\,P]\!] \xrightarrow{\mathrm{id}} \mathbf{list}([\![P]\!]) \xrightarrow{\mathrm{id}} [\![I]\!] + [\![P]\!] + [\![P]\!] \otimes [\![P]\!] + \cdots \\
 & & \quad \xrightarrow{\coprod_{S \in \{I\} \cup \{\otimes_{i=1}^{n} P : n \in \mathbb{N}_{>0}\}} \tau_S} p([\![I]\!]_p) + p([\![P]\!]_p) + p([\![P]\!]_p \times [\![P]\!]_p) + \cdots \\
 & & \quad \xrightarrow{\cong} p([\![I]\!]_p + [\![P]\!]_p + [\![P]\!]_p \times [\![P]\!]_p + \cdots) \\
 & & \quad \xrightarrow{\mathrm{id}} p(\mathbf{list}([\![P]\!]_p)) \xrightarrow{\mathrm{id}} p([\![\mathbf{list}\ P]\!]_p) \\
\tau_{\mathrm{Circ}(T,U)} & = & [\![\mathrm{Circ}(T,U)]\!] \xrightarrow{\mathrm{id}} p(\mathbf{M}([\![T]\!]_s, [\![U]\!]_s)) \xrightarrow{\mathrm{id}} p([\![\mathrm{Circ}(T,U)]\!]_p).
\end{array}$$

Here, $\cong$ denotes the appropriate isomorphisms witnessing the fact that p, being a left adjoint to $\flat$, preserves coproducts. Moreover, m_I and $m_{P,R}$ are the appropriate components of the monoidal structure of the functor p. ◇

7.3 Interpreting Terms

As usual, our categorical semantics will associate an appropriate morphism to each valid term of Proto-Quipper-M. In this section, we carefully introduce the interpretations of the various components of our language and auxiliary structures. How all these pieces interact with each other will be further explored in later chapters.

7.3.1 Labeling Structure

In Section 5.1.2, we equipped **M** with an additional labeling structure to enable our programming language to manipulate morphisms of **M**. We now extend this structure to include label tuple judgements and provide a semantics in each of the categories **M** and **L**.

Definition 7.3.1. To every valid label tuple judgement $Q \vdash_{\mathcal{L}} \vec{\ell} : T$, we inductively associate a canonical isomorphism $[\![Q \vdash_{\mathcal{L}} \vec{\ell} : T]\!]_s : [\![Q]\!]_s \to [\![T]\!]_s$ of **M** as follows:

$$\begin{aligned}
[\![\emptyset \vdash_{\mathcal{L}} * : I]\!]_s &= [\![\emptyset]\!]_s \xrightarrow{\mathrm{id}_{[\![\emptyset]\!]_s}} [\![I]\!]_s \\
[\![\ell : \alpha \vdash_{\mathcal{L}} \ell : \alpha]\!]_s &= [\![\alpha]\!]_s \xrightarrow{\mathrm{id}_{[\![\alpha]\!]_s}} [\![\alpha]\!]_s \\
[\![Q, Q' \vdash_{\mathcal{L}} \langle \vec{\ell}, \vec{\ell'} \rangle : T \otimes U]\!]_s &= [\![Q, Q']\!]_s \xrightarrow{\cong} [\![Q]\!]_s \otimes [\![Q']\!]_s \xrightarrow{[\![Q \vdash_{\mathcal{L}} \vec{\ell} : T]\!]_s \otimes [\![Q' \vdash_{\mathcal{L}} \vec{\ell'} : U]\!]_s} [\![T]\!]_s \otimes [\![U]\!]_s \\
&\quad \xrightarrow{\mathrm{id}} [\![T \otimes U]\!]_s .
\end{aligned}$$

Here, $\cong$ is the natural isomorphism obtained from the monoidal structure of **M**. We often write $[\![\vec{\ell}]\!]_s : [\![Q]\!]_s \to [\![T]\!]_s$ instead of $[\![Q \vdash_{\mathcal{L}} \vec{\ell} : T]\!]_s : [\![Q]\!]_s \to [\![T]\!]_s$ to simplify our notation. Moreover, for every such isomorphism $[\![\vec{\ell}]\!]_s : [\![Q]\!]_s \to [\![T]\!]_s$, we define the isomorphism $[\![\vec{\ell}]\!] : [\![Q]\!] \to [\![T]\!]$ of the category **L** as follows:

$$[\![\vec{\ell}]\!] = [\![Q]\!] \xrightarrow{\mathcal{Y}_Q} \mathcal{Y}([\![Q]\!]_s) \xrightarrow{\mathcal{Y}([\![\vec{\ell}]\!]_s)} \mathcal{Y}([\![T]\!]_s) \xrightarrow{\mathcal{Y}_T^{-1}} [\![T]\!]. \tag{7.12}$$

This morphism is the interpretation in **L** of the valid label tuple judgement $Q \vdash_{\mathcal{L}} \vec{\ell} : T$ and is also denoted by $[\![Q \vdash_{\mathcal{L}} \vec{\ell} : T]\!]$. ◇

7.3.2 Morphisms circ and lift

We now introduce a number of auxiliary morphisms that will be used in the definition of the categorical semantics of some Proto-Quipper-M terms.

Since $\mathbf{L}$ is a symmetric monoidal closed category, $-\otimes T \dashv T \multimap - : \mathbf{L} \to \mathbf{L}$ for every object T of $\mathbf{L}$. In the sequel, we will denote the transpose of a morphism $m : A \otimes T \to B$ in $\mathbf{L}$ under this adjunction by $m^* : A \to (T \multimap B)$, and the transpose of a morphism $n : A \to (T \multimap B)$ by $n_* : A \otimes T \to B$. Then, $(m^*)_* = m$ and $(n_*)^* = n$. Similarly, we will denote the transpose of a morphism $u : p(X) \to A$ in $\mathbf{L}$ under the adjunction $p \dashv \flat$ by $u^\circ : X \to \flat(A)$, and the transpose of a function $v : X \to \flat(A)$ by $v_\circ : p(X) \to A$. Then, $(u^\circ)_\circ = u$ and $(v_\circ)^\circ = v$.

Recall from Definition 5.1.3 that a labeled circuit, that is, a morphism $C : Q \to Q'$ of $\mathbf{M}_{\mathcal{L}}$, is a morphism $\llbracket C \rrbracket_s : \llbracket Q \rrbracket_s \to \llbracket Q' \rrbracket_s$ in $\mathbf{M}$, and so given a labeled circuit $C : Q \to Q'$ and valid label tuples judgements $Q \vdash_{\mathcal{L}} \vec{\ell} : T$ and $Q' \vdash_{\mathcal{L}} \vec{\ell'} : U$, we can construct the morphisms $\llbracket C \rrbracket$ and $[\vec{\ell}, C, \vec{\ell'}]$ of $\mathbf{L}$ as follows:

$$\llbracket C \rrbracket = \llbracket Q \rrbracket \xrightarrow{\mathcal{Y}_Q} \mathcal{Y}(\llbracket Q \rrbracket_s) \xrightarrow{\mathcal{Y}(\llbracket C \rrbracket_s)} \mathcal{Y}(\llbracket Q' \rrbracket_s) \xrightarrow{\mathcal{Y}_{Q'}^{-1}} \llbracket Q' \rrbracket, \tag{7.13}$$

and

$$[\vec{\ell}, C, \vec{\ell'}] = \llbracket T \rrbracket \xrightarrow{\mathcal{Y}_T} \mathcal{Y}(\llbracket T \rrbracket_s) \xrightarrow{\mathcal{Y}(\llbracket \vec{\ell} \rrbracket_s^{-1})} \mathcal{Y}(\llbracket Q \rrbracket_s) \xrightarrow{\mathcal{Y}(\llbracket C \rrbracket_s)} \mathcal{Y}(\llbracket Q' \rrbracket_s) \xrightarrow{\mathcal{Y}(\llbracket \vec{\ell'} \rrbracket_s)} \mathcal{Y}(\llbracket U \rrbracket_s) \xrightarrow{\mathcal{Y}_U^{-1}} \llbracket U \rrbracket.$$

That is,

$$[\vec{\ell}, C, \vec{\ell'}] = \llbracket T \rrbracket \xrightarrow{\mathcal{Y}_T} \mathcal{Y}(\llbracket T \rrbracket_s) \xrightarrow{\mathcal{Y}(\llbracket \vec{\ell'} \rrbracket_s \circ \llbracket C \rrbracket_s \circ \llbracket \vec{\ell} \rrbracket_s^{-1})} \mathcal{Y}(\llbracket U \rrbracket_s) \xrightarrow{\mathcal{Y}_U^{-1}} \llbracket U \rrbracket. \tag{7.14}$$

To simplify our notation, let

$$\underline{C} = \llbracket I \rrbracket \otimes \llbracket T \rrbracket \xrightarrow{\lambda_T} \llbracket T \rrbracket \xrightarrow{[\vec{\ell}, C, \vec{\ell'}]} \llbracket U \rrbracket. \tag{7.15}$$

We now define the morphism $\mathbf{circ}\,(\vec{\ell}, C, \vec{\ell'}) : \llbracket I \rrbracket \to \llbracket \mathrm{Circ}(T, U) \rrbracket$ as follows:

$$\mathbf{circ}\,(\vec{\ell}, C, \vec{\ell'}) = \llbracket I \rrbracket \xrightarrow{\tau_I} p(\llbracket I \rrbracket_p) \xrightarrow{p((\underline{C}^* \circ \tau_I^{-1})^\circ)} p(\flat(\llbracket T \rrbracket \multimap \llbracket U \rrbracket)) \xrightarrow{\mathbf{box}} \llbracket \mathrm{Circ}(T, U) \rrbracket. \tag{7.16}$$

To endow the terms of Proto-Quipper-M with meaning, we also need to interpret the contexts in which they occur. We already did that for label contexts in Definition 5.1.2. We now do it for variable contexts in the following definition.

Definition 7.3.2. To each variable context $\Gamma = x_1 : A_1, \ldots, x_n : A_n$, we associate an object $\llbracket \Gamma \rrbracket = \llbracket A_1 \rrbracket \otimes \ldots \otimes \llbracket A_n \rrbracket$ of $\mathbf{L}$, and to each parameter context

$\Phi = y_1 : P_1, \ldots, y_n : P_n$, we associate an object $[\![\Phi]\!]_p = [\![P_1]\!]_p \times \ldots \times [\![P_n]\!]_p$ of **Set**. The objects $[\![\Gamma]\!]$ and $[\![\Phi]\!]_p$ are called the *semantics of contexts* Γ and Φ, respectively. When $\Gamma = \emptyset$, we set $[\![\Gamma]\!] = I_{\mathbf{L}}$ and when $\Phi = \emptyset$, we set $[\![\Phi]\!]_p = 1$. As in the case of label contexts, it will be convenient to think of these objects as fixed bracketings of the tensor products of their corresponding factors. Without loss of generality, we will assume that $[\![\Phi]\!]$ and $[\![\Phi]\!]_p$ have the same bracketing. In the sequel, we will let $\tau_\Phi : [\![\Phi]\!] \to p([\![\Phi]\!]_p)$ be the isomorphism

$$\tau_\Phi = \tau_{P_1 \otimes \cdots \otimes P_n}$$

where $P_1 \otimes \cdots \otimes P_n$ is assumed to have the same bracketing as $[\![\Phi]\!]$. ◇

For every parameter context Φ, type A, and morphism $f : [\![\Phi]\!] \otimes [\![I]\!] \to [\![A]\!]$ in $\mathbf{L}$, the composition $p([\![\Phi]\!]_p) \xrightarrow{\tau_\Phi^{-1}} [\![\Phi]\!] \xrightarrow{\rho_\Phi^{-1}} [\![\Phi]\!] \otimes [\![I]\!] \xrightarrow{f} [\![A]\!]$ is a morphism in $\mathbf{L}$ whose transpose $(f \circ \rho_\Phi^{-1} \circ \tau_\Phi^{-1})^\circ$ is a function from $[\![\Phi]\!]_p$ to $\flat([\![A]\!])$, and so we can define the morphism

$$\mathbf{lift}\ f = p([\![\Phi]\!]_p) \xrightarrow{p((f\circ\rho_\Phi^{-1}\circ\tau_\Phi^{-1})^\circ)} p(\flat([\![A]\!])) \xrightarrow{\text{id}} p([\![!A]\!]_p). \qquad (7.17)$$

Morphisms of the form $\mathbf{circ}\,(\vec{\ell}, C, \vec{\ell'})$ and $\mathbf{lift}\ f$ will play a central role in the interpretation of the corresponding Proto-Quipper-M terms.

7.3.3 Duplicating and Discarding

Our categorical model has morphisms that duplicate parameter contexts. We begin with the case in which all the contexts involved are of parameter type.

Definition 7.3.3. Let Φ_1 and Φ_2 be parameter contexts for which there exist mutually disjoint parameter contexts Φ'_1, Φ'_2, and Φ' such that $\Phi_1 = \Phi'_1, \Phi'$ and $\Phi_2 = \Phi'_2, \Phi'$. Recall that $\Phi_1 \cup \Phi_2 = \Phi'_1, \Phi', \Phi'_2$. Let $\Delta_{\Phi'} = \langle \text{id}_{[\![\Phi']\!]_p}, \text{id}_{[\![\Phi']\!]_p} \rangle : [\![\Phi']\!]_p \to [\![\Phi']\!]_p \times [\![\Phi']\!]_p$, the diagonal function on the set $[\![\Phi']\!]_p$. We define $\triangle^p_{\Phi_1 \cup \Phi_2} : [\![\Phi_1 \cup \Phi_2]\!]_p \to [\![\Phi_1]\!]_p \times [\![\Phi_2]\!]_p$ to be the function that makes the following diagram commute:

$$\begin{tikzcd}
{[\![\Phi_1 \cup \Phi_2]\!]_p = [\![\Phi'_1, \Phi', \Phi'_2]\!]_p} \arrow[r, "\triangle^p_{\Phi_1\cup\Phi_2}"] \arrow[d, "\cong"'] & {[\![\Phi_1]\!]_p \times [\![\Phi_2]\!]_p} \\
{[\![\Phi'_1]\!]_p \times [\![\Phi']\!]_p \times [\![\Phi'_2]\!]_p} \arrow[r, "\text{id}\times\Delta_{\Phi'}\times\text{id}"'] & {[\![\Phi'_1]\!]_p \times [\![\Phi']\!]_p \times [\![\Phi']\!]_p \times [\![\Phi'_2]\!]_p.} \arrow[u, "\cong"']
\end{tikzcd}$$

Similarly, we define the morphisms $\vartriangle_{\Phi_1 \cup \Phi_2} : [\![\Phi_1 \cup \Phi_2]\!] \to [\![\Phi_1]\!] \otimes [\![\Phi_2]\!]$ and $\vartriangle_{\Phi_1 \cup \Phi_2;\emptyset} : [\![\Phi_1 \cup \Phi_2;\emptyset]\!] \to [\![\Phi_1;\emptyset]\!] \otimes [\![\Phi_2;\emptyset]\!]$ in $\mathbf{L}$ to be the ones making the following diagrams commute, respectively:

$$\begin{tikzcd}
{[\![\Phi_1 \cup \Phi_2]\!]} \arrow[r, "\vartriangle_{\Phi_1 \cup \Phi_2}"] \arrow[dd, "\tau_{\Phi_1 \cup \Phi_2}"'] & {[\![\Phi_1]\!] \otimes [\![\Phi_2]\!].} \arrow[d, "\tau_{\Phi_1} \otimes \tau_{\Phi_2}"] \\
& p([\![\Phi_1]\!]_p) \otimes p([\![\Phi_2]\!]_p) \arrow[d, "m_{\Phi_1,\Phi_2}"] \\
p([\![\Phi_1 \cup \Phi_2]\!]_p) \arrow[r, "p(\vartriangle^p_{\Phi_1 \cup \Phi_2})"'] & p([\![\Phi_1]\!]_p \times [\![\Phi_2]\!]_p)
\end{tikzcd}$$

$$\begin{tikzcd}
{[\![\Phi_1 \cup \Phi_2;\emptyset]\!]} \arrow[r, "\vartriangle_{\Phi_1 \cup \Phi_2;\emptyset}"] \arrow[d, "\rho_{\Phi_1 \cup \Phi_2}"'] & {[\![\Phi_1;\emptyset]\!] \otimes [\![\Phi_2;\emptyset]\!]} \arrow[d, "\rho_{\Phi_1} \otimes \rho_{\Phi_2}"] \\
{[\![\Phi_1 \cup \Phi_2]\!]} \arrow[r, "\vartriangle_{\Phi_1 \cup \Phi_2}"'] & {[\![\Phi_1]\!] \otimes [\![\Phi_2]\!].}
\end{tikzcd}$$

When $\Phi = \Phi_1 = \Phi_2$, the morphism $\vartriangle_{\Phi_1 \cup \Phi_2} = \vartriangle_{\Phi} : [\![\Phi]\!] \to [\![\Phi]\!] \otimes [\![\Phi]\!]$ is called the *duplication morphism on* $[\![\Phi]\!]$ and is also denoted by $\vartriangle_\Phi$. ◇

We note that the union $\Phi_1 \cup \Phi_2$ is associative and commutative and the expected coherence conditions hold by the coherence theorem for symmetric monoidal categories. This observation extends to unions of arbitrary contexts $\Gamma_1 \cup \Gamma_2$.

Definition 7.3.4. Let Γ_1 and Γ_2 be variable contexts for which there exist mutually disjoint contexts Γ_1', Γ_2', and Φ such that $\Gamma_1 = \Gamma_1', \Phi$ and $\Gamma_2 = \Gamma_2', \Phi$ and Φ is a parameter context. We define the morphism $\vartriangle_{\Gamma_1 \cup \Gamma_2} : [\![\Gamma_1 \cup \Gamma_2]\!] \to [\![\Gamma_1]\!] \otimes [\![\Gamma_2]\!]$ in $\mathbf{L}$ to be the one that makes the following diagram commute:

$$\begin{tikzcd}
{[\![\Gamma_1 \cup \Gamma_2]\!] = [\![\Gamma_1', \Phi, \Gamma_2']\!]} \arrow[r, "\vartriangle_{\Gamma_1 \cup \Gamma_2}"] \arrow[d, "\cong"'] & {[\![\Gamma_1]\!] \otimes [\![\Gamma_2]\!]} \\
{[\![\Gamma_1']\!] \otimes [\![\Phi]\!] \otimes [\![\Gamma_2']\!]} \arrow[r, "\mathrm{id} \otimes \vartriangle_\Phi \otimes \mathrm{id}"'] & {[\![\Gamma_1']\!] \otimes [\![\Phi]\!] \otimes [\![\Phi]\!] \otimes [\![\Gamma_2']\!].} \arrow[u, "\cong"']
\end{tikzcd}$$

Similarly, we define $\vartriangle_{\Gamma_1 \cup \Gamma_2;Q_1,Q_2} : [\![\Gamma_1 \cup \Gamma_2; Q_1, Q_2]\!] \to [\![\Gamma_1; Q_1]\!] \otimes [\![\Gamma_2; Q_2]\!]$ to be the morphism in $\mathbf{L}$ that makes the following diagram commute:

$$\begin{tikzcd}
\llbracket \Gamma_1 \cup \Gamma_2; Q_1, Q_2 \rrbracket \arrow[r, "\triangle_{\Gamma_1 \cup \Gamma_2; Q_1, Q_2}"] \arrow[d, "\cong"'] & \llbracket \Gamma_1; Q_1 \rrbracket \otimes \llbracket \Gamma_2; Q_2 \rrbracket \\
\llbracket \Gamma_1 \cup \Gamma_2 \rrbracket \otimes \llbracket Q_1 \rrbracket \otimes \llbracket Q_2 \rrbracket \arrow[r, "\triangle_{\Gamma_1 \cup \Gamma_2} \otimes \mathrm{id}"'] & \llbracket \Gamma_1 \rrbracket \otimes \llbracket \Gamma_2 \rrbracket \otimes \llbracket Q_1 \rrbracket \otimes \llbracket Q_2 \rrbracket \arrow[u, "\cong"']
\end{tikzcd}$$

The morphisms $\triangle_{\Gamma_1 \cup \Gamma_2}$ and $\triangle_{\Gamma_1 \cup \Gamma_2; Q_1, Q_2}$ are called *generalized duplication morphisms.* ◇

Our categorical model also has morphisms that discard parameter contexts.

Definition 7.3.5. Given a parameter context Φ, we define the *discarding morphism* $\diamond_\Phi : \llbracket \Phi \rrbracket \to \llbracket I \rrbracket$ to be the one that makes the following diagram commute:

$$\begin{tikzcd}
\llbracket \Phi \rrbracket \arrow[r, "\diamond_\Phi"] \arrow[d, "\tau_\Phi"'] & \llbracket I \rrbracket \arrow[d, "\tau_I"] \\
p(\llbracket \Phi \rrbracket_p) \arrow[r, "p(\diamond^p_\Phi)"'] & p(\llbracket I \rrbracket_p).
\end{tikzcd}$$

Here $\diamond^p_\Phi : \llbracket \Phi \rrbracket_p \to \llbracket I \rrbracket_p$ denotes the unique function from the set $\llbracket \Phi \rrbracket_p$ to the terminal set $\llbracket I \rrbracket_p = 1$. ◇

7.3.4 Typing Derivations

In the sequel we use the following notation for the morphisms arising from the coproduct structure. For objects A, B, and C, $\Box_C : 0 \to C$ denotes the unique morphism from the initial object to C, $\mathrm{left}_{A,B} : A \to A+B$ and $\mathrm{right}_{A,B} : B \to A+B$ denote the left and right coprojections, respectively, and $[f, g] : A + B \to C$ the copairing morphism of $f : A \to C$ and $g : B \to C$. In what follows, we may omit the superscripts and subscripts of objects and morphisms when no confusion arises.

Now that we have introduced all the necessary structures for the interpretation of our language, we proceed to give the formal definition of its meaning. As expected, the semantics of Proto-Quipper-M associates to each typing derivation $\mathcal{D}$ with conclusion sequent $\Gamma; Q \vdash M : A$ a morphism $\llbracket \mathcal{D} \rrbracket : \llbracket \Gamma \rrbracket \otimes \llbracket Q \rrbracket \to \llbracket A \rrbracket$ of $\mathbf{L}$. When no confusion arises, we also denote $\llbracket \mathcal{D} \rrbracket$ with $\llbracket \Gamma; Q \vdash M : A \rrbracket$, or even $\llbracket M \rrbracket$. More precisely, we have:

Definition 7.3.6. The *semantics of the typing derivations* of Proto-Quipper-M is defined inductively by the typing rules and is shown in Tables 7.1 and 7.2. ◇

$$
\begin{aligned}
\llbracket \Phi, x : A; \emptyset \vdash x : A \rrbracket &= \llbracket \Phi, x : A \rrbracket \otimes \llbracket \emptyset \rrbracket \xrightarrow{\rho_{\Phi, x:A}} \llbracket \Phi, x : A \rrbracket \xrightarrow{\cong} \llbracket \Phi \rrbracket \otimes \llbracket A \rrbracket \\
&\qquad \xrightarrow{\Diamond_\Phi \otimes \mathrm{id}} \llbracket I \rrbracket \otimes \llbracket A \rrbracket \xrightarrow{\lambda_A} \llbracket A \rrbracket \\
\llbracket \Phi; \ell : \alpha \vdash \ell : \alpha \rrbracket &= \llbracket \Phi \rrbracket \otimes \llbracket \alpha \rrbracket \xrightarrow{\Diamond_\Phi \otimes \mathrm{id}} \llbracket I \rrbracket \otimes \llbracket \alpha \rrbracket \xrightarrow{\lambda_\alpha} \llbracket \alpha \rrbracket \\
\llbracket \Phi; \emptyset \vdash c : A_c \rrbracket &= \llbracket \Phi \rrbracket \otimes \llbracket \emptyset \rrbracket \xrightarrow{\rho_\Phi} \llbracket \Phi \rrbracket \xrightarrow{\Diamond_\Phi} \llbracket I \rrbracket \xrightarrow{\llbracket c \rrbracket} \llbracket A_c \rrbracket \\
\llbracket \Gamma_1 \cup \Gamma_2; Q_1, Q_2 \vdash \text{let } x = M \text{ in } N : B \rrbracket &= \llbracket \Gamma_1 \cup \Gamma_2 \rrbracket \otimes \llbracket Q_1, Q_2 \rrbracket \xrightarrow{\triangle} (\llbracket \Gamma_1 \rrbracket \otimes \llbracket Q_1 \rrbracket) \otimes (\llbracket \Gamma_2 \rrbracket \otimes \llbracket Q_2 \rrbracket) \\
&\qquad \xrightarrow{\llbracket M \rrbracket \otimes \mathrm{id}} \llbracket A \rrbracket \otimes (\llbracket \Gamma_2 \rrbracket \otimes \llbracket Q_2 \rrbracket) \\
&\qquad \xrightarrow{\cong} \llbracket \Gamma_2, x : A; Q_2 \rrbracket \xrightarrow{\llbracket N \rrbracket} \llbracket B \rrbracket \\
\llbracket \Gamma; Q \vdash \Box_C M : C \rrbracket &= \llbracket \Gamma \rrbracket \otimes \llbracket Q \rrbracket \xrightarrow{\llbracket M \rrbracket} \llbracket 0 \rrbracket \xrightarrow{\Box_C} \llbracket C \rrbracket \\
\llbracket \Gamma; Q \vdash \mathrm{left}_{A,B}\, M : A + B \rrbracket &= \llbracket \Gamma \rrbracket \otimes \llbracket Q \rrbracket \xrightarrow{\llbracket M \rrbracket} \llbracket A \rrbracket \xrightarrow{\mathrm{left}_{A,B}} \llbracket A \rrbracket + \llbracket B \rrbracket \\
\llbracket \Gamma; Q \vdash \mathrm{right}_{A,B}\, M : A + B \rrbracket &= \llbracket \Gamma \rrbracket \otimes \llbracket Q \rrbracket \xrightarrow{\llbracket M \rrbracket} \llbracket B \rrbracket \xrightarrow{\mathrm{right}_{A,B}} \llbracket A \rrbracket + \llbracket B \rrbracket \\
\left\llbracket \begin{array}{l} \Gamma_1 \cup \Gamma_2; Q_1, Q_2 \vdash \text{case } M \text{ of} \\ \quad \{\mathrm{left}\, x \to N \mid \mathrm{right}\, y \to P\} : C \end{array} \right\rrbracket &= \llbracket \Gamma_1 \cup \Gamma_2 \rrbracket \otimes \llbracket Q_1, Q_2 \rrbracket \xrightarrow{\triangle} (\llbracket \Gamma_1 \rrbracket \otimes \llbracket Q_1 \rrbracket) \otimes (\llbracket \Gamma_2 \rrbracket \otimes \llbracket Q_2 \rrbracket) \\
&\qquad \xrightarrow{\llbracket M \rrbracket \otimes \mathrm{id}} (\llbracket A \rrbracket + \llbracket B \rrbracket) \otimes (\llbracket \Gamma_2 \rrbracket \otimes \llbracket Q_2 \rrbracket) \\
&\qquad \xrightarrow{\cong} (\llbracket A \rrbracket \otimes \llbracket \Gamma_2 \rrbracket \otimes \llbracket Q_2 \rrbracket) + (\llbracket B \rrbracket \otimes \llbracket \Gamma_2 \rrbracket \otimes \llbracket Q_2 \rrbracket) \\
&\qquad \xrightarrow{\cong} \llbracket \Gamma_2, x : A; Q_2 \rrbracket + \llbracket \Gamma_2, y : B; Q_2 \rrbracket \\
&\qquad \xrightarrow{[\llbracket N \rrbracket, \llbracket P \rrbracket]} \llbracket C \rrbracket \\
\llbracket \Phi; \emptyset \vdash * : I \rrbracket &= \llbracket \Phi \rrbracket \otimes \llbracket \emptyset \rrbracket \xrightarrow{\rho_\Phi} \llbracket \Phi \rrbracket \xrightarrow{\Diamond_\Phi} \llbracket I \rrbracket \\
\llbracket \Gamma_1 \cup \Gamma_2; Q_1, Q_2 \vdash M; N : C \rrbracket &= \llbracket \Gamma_1 \cup \Gamma_2 \rrbracket \otimes \llbracket Q_1, Q_2 \rrbracket \xrightarrow{\triangle} (\llbracket \Gamma_1 \rrbracket \otimes \llbracket Q_1 \rrbracket) \otimes (\llbracket \Gamma_2 \rrbracket \otimes \llbracket Q_2 \rrbracket) \\
&\qquad \xrightarrow{\llbracket M \rrbracket \otimes \llbracket N \rrbracket} \llbracket I \rrbracket \otimes \llbracket C \rrbracket \xrightarrow{\lambda_C} \llbracket C \rrbracket
\end{aligned}
$$

Table 7.1: The Interpretation of Typing Derivations.

$$
\begin{aligned}
[\![\Gamma_1 \cup \Gamma_2; Q_1, Q_2 \vdash \langle M, N\rangle : A \otimes B]\!] &= [\![\Gamma_1 \cup \Gamma_2]\!] \otimes [\![Q_1, Q_2]\!] \xrightarrow{\triangle} ([\![\Gamma_1]\!] \otimes [\![Q_1]\!]) \otimes ([\![\Gamma_2]\!] \otimes [\![Q_2]\!]) \\
&\quad \xrightarrow{[\![M]\!] \otimes [\![N]\!]} [\![A]\!] \otimes [\![B]\!] \\
[\![\Gamma_1 \cup \Gamma_2; Q_1, Q_2 \vdash \text{let } \langle x, y\rangle = M \text{ in } N : C]\!] &= [\![\Gamma_1 \cup \Gamma_2]\!] \otimes [\![Q_1, Q_2]\!] \xrightarrow{\triangle} ([\![\Gamma_1]\!] \otimes [\![Q_1]\!]) \otimes ([\![\Gamma_2]\!] \otimes [\![Q_2]\!]) \\
&\quad \xrightarrow{[\![M]\!] \otimes \text{id}} [\![A \otimes B]\!] \otimes ([\![\Gamma_2]\!] \otimes [\![Q_2]\!]) \\
&\quad \xrightarrow{\cong} [\![\Gamma_2, x : A, y : B; Q_2]\!] \xrightarrow{[\![N]\!]} [\![C]\!] \\
[\![\Gamma; Q \vdash \lambda x^A.M : A \multimap B]\!] &= [\![\Gamma]\!] \otimes [\![Q]\!] \xrightarrow{f^*} [\![A \multimap B]\!], \text{ where} \\
&\quad f = ([\![\Gamma]\!] \otimes [\![Q]\!]) \otimes [\![A]\!] \xrightarrow{\cong} [\![\Gamma, x : A]\!] \otimes [\![Q]\!] \xrightarrow{[\![M]\!]} [\![B]\!] \\
[\![\Gamma_1 \cup \Gamma_2; Q_1, Q_2 \vdash MN : B]\!] &= [\![\Gamma_1 \cup \Gamma_2]\!] \otimes [\![Q_1, Q_2]\!] \xrightarrow{\triangle} ([\![\Gamma_1]\!] \otimes [\![Q_1]\!]) \otimes ([\![\Gamma_2]\!] \otimes [\![Q_2]\!]) \\
&\quad \xrightarrow{[\![M]\!] \otimes [\![N]\!]} [\![A \multimap B]\!] \otimes [\![A]\!] \xrightarrow{\varepsilon_B} [\![B]\!] \\
[\![\Phi; \emptyset \vdash \text{lift } M : !A]\!] &= [\![\Phi; \emptyset]\!] \xrightarrow{\rho_\Phi} [\![\Phi]\!] \xrightarrow{\tau_\Phi} p([\![\Phi]\!]_p) \xrightarrow{\textbf{lift } [\![M]\!]} p([\![!A]\!]_p) \xrightarrow{\tau_{!A}^{-1}} [\![!A]\!] \\
[\![\Gamma; Q \vdash \text{force } M : A]\!] &= [\![\Gamma]\!] \otimes [\![Q]\!] \xrightarrow{[\![M]\!]} ![\![A]\!] \xrightarrow{\textbf{force}} [\![A]\!] \\
[\![\Gamma; Q \vdash \text{box}_T\, M : \text{Circ}(T, U)]\!] &= [\![\Gamma]\!] \otimes [\![Q]\!] \xrightarrow{[\![M]\!]} !([\![T]\!] \multimap [\![U]\!]) \xrightarrow{\textbf{box}} [\![\text{Circ}(T, U)]\!] \\
[\![\Gamma_1 \cup \Gamma_2; Q_1, Q_2 \vdash \text{apply}(M, N) : U]\!] &= [\![\Gamma_1 \cup \Gamma_2]\!] \otimes [\![Q_1, Q_2]\!] \xrightarrow{\triangle} ([\![\Gamma_1]\!] \otimes [\![Q_1]\!]) \otimes ([\![\Gamma_2]\!] \otimes [\![Q_2]\!]) \\
&\quad \xrightarrow{[\![M]\!] \otimes [\![N]\!]} [\![\text{Circ}(T, U)]\!] \otimes [\![T]\!] \xrightarrow{\textbf{apply}} [\![U]\!] \\
[\![\Phi; \emptyset \vdash (\vec{\ell}, C, \vec{\ell'}) : \text{Circ}(T, U)]\!] &= [\![\Phi; \emptyset]\!] \xrightarrow{\rho_\Phi} [\![\Phi]\!] \xrightarrow{\diamond_\Phi} [\![I]\!] \xrightarrow{\textbf{circ}\,(\vec{\ell}, C, \vec{\ell'})} [\![\text{Circ}(T, U)]\!]
\end{aligned}
$$

Table 7.2: The Interpretation of Typing Derivations (Continuation).

Chapter 8

Semantic Properties

In this chapter, we present semantic versions of several of the syntactic results obtained in Chapter 6. These semantic results will be used in the proof of the main lemma of this chapter, namely, the *Semantic Substitution Lemma*.

8.1 Semantic Parameter Value Lemma

Before we introduce the semantic version of the Parameter Value Lemma 6.1.4, we give meaning in the category of sets to the well-typed values of parameter type.

Definition 8.1.1. Let $\Phi; \emptyset \vdash V : P$ be a valid typing judgement with derivation $\mathcal{D}$ where Φ is a parameter context, V a value, and P a parameter type. The *set-theoretic semantics of the parameter value* V is given by the function $[\![V]\!]_p : [\![\Phi]\!]_p \to [\![P]\!]_p$ defined inductively as follows:

- If the last (and only) rule of $\mathcal{D}$ is (*var*) with $V = x$, then $\mathcal{D}$ is

  $$\frac{}{\Phi', x : P; \emptyset \vdash x : P}\ (var)$$

  where $\Phi = \Phi', x : P$. In this case, let

  $$[\![V]\!]_p = [\![\Phi', x : P]\!]_p \xrightarrow{\cong} [\![\Phi']\!]_p \times [\![P]\!]_p \xrightarrow{\diamond^p_{\Phi'} \times \mathrm{id}} [\![I]\!]_p \times [\![P]\!]_p \xrightarrow{\pi_P} [\![P]\!]_p.$$

- If the last (and only) rule of $\mathcal{D}$ is (*const*) with $V = c$, then $\mathcal{D}$ is

  $$\frac{}{\Phi; \emptyset \vdash c : P_c}\ (const)$$

  where $P = P_c$. In this case, let

  $$[\![V]\!]_p = [\![\Phi]\!]_p \xrightarrow{\diamond^p_{\Phi}} [\![I]\!]_p \xrightarrow{[\![c]\!]_p} [\![P_c]\!]_p.$$

- If the last rule of $\mathcal{D}$ is (*left*) with $V = \mathrm{left}_{R,S}\, W$, then $\mathcal{D}$ is

$$\dfrac{\dfrac{\vdots}{\Phi; \emptyset \vdash W : R}}{\Phi; \emptyset \vdash \mathrm{left}_{R,S}\, W : R + S}\ (\mathit{left})$$

where W is a value term and $P = R + S$ for some parameter types R and S. In this case, let

$$[\![V]\!]_p = [\![\Phi]\!]_p \xrightarrow{[\![W]\!]_p} [\![R]\!]_p \xrightarrow{\mathrm{left}_{R,S}} [\![R + S]\!]_p.$$

- If the last rule of $\mathcal{D}$ is (*right*) with $V = \mathrm{right}_{R,S}\, W$, then $\mathcal{D}$ is

$$\dfrac{\dfrac{\vdots}{\Phi; \emptyset \vdash W : S}}{\Phi; \emptyset \vdash \mathrm{right}_{R,S}\, W : R + S}\ (\mathit{right})$$

where W is a value term and $P = R + S$ for some parameter types R and S. In this case, let

$$[\![V]\!]_p = [\![\Phi]\!]_p \xrightarrow{[\![W]\!]_p} [\![S]\!]_p \xrightarrow{\mathrm{right}_{R,S}} [\![R + S]\!]_p.$$

- If the last (and only) rule of $\mathcal{D}$ is $(*)$ with $V = *$, then $\mathcal{D}$ is

$$\dfrac{}{\Phi; \emptyset \vdash * : I}\ (*)$$

where $P = I$. In this case, let

$$[\![V]\!]_p = [\![\Phi]\!]_p \xrightarrow{\diamond^p_\Phi} [\![I]\!]_p.$$

- If the last rule of $\mathcal{D}$ is (*pair*) with $V = \langle W_1, W_2 \rangle$ for some value terms W_1 and W_2, then $\mathcal{D}$ is

$$\dfrac{\dfrac{\vdots}{\Phi_1; \emptyset \vdash W_1 : R} \qquad \dfrac{\vdots}{\Phi_2; \emptyset \vdash W_2 : S}}{\Phi_1 \cup \Phi_2; \emptyset \vdash \langle W_1, W_2 \rangle : R \otimes S}\ (\mathit{pair})$$

where $\Phi = \Phi_1 \cup \Phi_2$ for some parameter contexts Φ_1 and Φ_2, and $P = R \otimes S$ for some parameter types R and S. In this case, let

$$[\![V]\!]_p = [\![\Phi_1 \cup \Phi_2]\!]_p \xrightarrow{\triangle^p_{\Phi_1 \cup \Phi_2}} [\![\Phi_1]\!]_p \times [\![\Phi_2]\!]_p \xrightarrow{[\![W_1]\!]_p \times [\![W_2]\!]_p} [\![R]\!]_p \times [\![S]\!]_p.$$

- If the last rule of $\mathcal{D}$ is (*lift*) with $V = \text{lift } M$, then $\mathcal{D}$ is

$$\dfrac{\dfrac{\vdots}{\Phi; \emptyset \vdash M : A}}{\Phi; \emptyset \vdash \text{lift } M : !A} \ (\textit{lift})$$

where $P = !A$. In this case, let

$$[\![V]\!]_p = [\![\Phi]\!]_p \xrightarrow{([\![M]\!] \circ \rho_\Phi^{-1} \circ \tau_\Phi^{-1})^\circ} [\![!A]\!]_p.$$

- If the last (and only) rule of $\mathcal{D}$ is (*circ*) with $V = (\vec{\ell}, C, \vec{\ell}')$, then $\mathcal{D}$ is

$$\dfrac{Q \vdash_{\mathcal{L}} \vec{\ell} : T \quad Q' \vdash_{\mathcal{L}} \vec{\ell}' : U \quad C \in \mathbf{M}_{\mathcal{L}}(Q, Q')}{\Phi; \emptyset \vdash (\vec{\ell}, C, \vec{\ell}') : \text{Circ}(T, U)} \ (\textit{circ})$$

where $P = \text{Circ}(T, U)$ for some simple M-types T and U. In this case, let

$$[\![V]\!]_p = [\![\Phi]\!]_p \xrightarrow{\diamond_\Phi^p} [\![I]\!]_p \xrightarrow{(\underline{C}^* \circ \tau_I^{-1})^\circ} \flat([\![T]\!] \multimap [\![U]\!]) \xrightarrow{i_{T,U}} [\![\text{Circ}(T, U)]\!]_p$$

where $i_{T,U}$ is as in (7.10) and $\underline{C}$ as in (7.15). $\diamond$

We now show that the interpretation in $\mathbf{L}$ of a value of parameter type is essentially the same as its interpretation in **Set**.

Lemma 8.1.2 (Semantic Parameter Value Lemma). *If $\Phi; \emptyset \vdash V : P$ is a valid typing judgement where Φ is a parameter context, V a value term, and P a parameter type, then the following diagram commutes:*

$$\begin{tikzcd}
{[\![\Phi; \emptyset]\!]} \arrow[r, "{[\![\Phi;\emptyset \vdash V : P]\!]}"] \arrow[d, "\rho_\Phi"'] & {[\![P]\!]} \arrow[dd, "\tau_P"] \\
{[\![\Phi]\!]} \arrow[d, "\tau_\Phi"'] & \\
p([\![\Phi]\!]_p) \arrow[r, "p([\![V]\!]_p)"'] & p([\![P]\!]_p).
\end{tikzcd}$$

Proof. By special rule induction on the set of valid typing judgements of the form $\Phi; \emptyset \vdash V : P$ where Φ is a parameter context, V a value term, and P a parameter type. We treat each case in turn.

- If the typing rule is (var) with $V = x$, then the rule is

$$\frac{}{\Phi', x : P; \emptyset \vdash x : P}\ (var)$$

where $\Phi = \Phi', x : P$. Suppose $\Phi' = x_1 : P_1, \ldots, x_{n-1} : P_{n-1}$ and $x_n : P_n = x : P$. Consider the following diagram:

$$\begin{tikzcd}
\llbracket \Phi', x : P \rrbracket \arrow[rr, "\cong"] \arrow[ddd, "\tau_\Phi"'] \arrow[rdd, "\bigotimes_{i=1}^n \tau_i"] & & \llbracket \Phi' \rrbracket \otimes \llbracket P \rrbracket \arrow[d, "\tau_{\Phi'} \otimes \mathrm{id}"] \arrow[dd, bend right=60, "\tau_{\Phi'} \otimes \tau_P"'] \\
& & p(\llbracket \Phi' \rrbracket_p) \otimes \llbracket P \rrbracket \arrow[d, "\mathrm{id} \otimes \tau_P"] \\
& \bigotimes_{i=1}^n p(\llbracket P_i \rrbracket_p) \arrow[r, "\cong"] \arrow[ld, "m_{P_1, \ldots, P_n}"'] & p(\llbracket \Phi' \rrbracket_p) \otimes p(\llbracket P \rrbracket_p) \arrow[d, "m_{\Phi', P}"] \\
p(\llbracket \Phi', x : P \rrbracket_p) \arrow[rr, "p(\cong)"'] & & p(\llbracket \Phi' \rrbracket_p \times \llbracket P \rrbracket_p)
\end{tikzcd} \tag{8.1}$$

The left diagram commutes by definition of τ_Φ, while the top diagram does by naturality of the monoidal structure. The top right diagram commutes by functoriality of $\otimes$, while the bottom diagram does by monoidality of p.

We now turn our attention to the following diagram:

$$\begin{tikzcd}
\llbracket \Phi' \rrbracket \otimes \llbracket P \rrbracket \arrow[rr, "\diamond_{\Phi'} \otimes \mathrm{id}"] \arrow[d, "\tau_{\Phi'} \otimes \mathrm{id}"'] & & \llbracket I \rrbracket \otimes \llbracket P \rrbracket \arrow[dd, "\mathrm{id} \otimes \tau_P"] \\
p(\llbracket \Phi' \rrbracket_p) \otimes \llbracket P \rrbracket \arrow[r, "p(\diamond^p_{\Phi'}) \otimes \mathrm{id}"] \arrow[d, "\mathrm{id} \otimes \tau_P"'] & p(\llbracket I \rrbracket_p) \otimes \llbracket P \rrbracket \arrow[ru, "\tau_I^{-1} \otimes \mathrm{id}"] \arrow[d, "\mathrm{id} \otimes \tau_P"] & \\
p(\llbracket \Phi' \rrbracket_p) \otimes p(\llbracket P \rrbracket_p) \arrow[r, "p(\diamond^p_{\Phi'}) \otimes \mathrm{id}"] \arrow[d, "m_{\Phi', P}"'] & p(\llbracket I \rrbracket_p) \otimes p(\llbracket P \rrbracket_p) \arrow[r, "\tau_I^{-1} \otimes \mathrm{id}"] \arrow[d, "m_{I, P}"] & \llbracket I \rrbracket \otimes p(\llbracket P \rrbracket_p) \arrow[d, "\lambda_P"] \\
p(\llbracket \Phi' \rrbracket_p \times \llbracket P \rrbracket_p) \arrow[r, "p(\diamond^p_{\Phi'} \times \mathrm{id})"'] & p(\llbracket I \rrbracket_p \times \llbracket P \rrbracket_p) \arrow[r, "p(\pi_P)"'] & p(\llbracket P \rrbracket_p)
\end{tikzcd} \tag{8.2}$$

Reading from top to bottom and left to right, the first diagram commutes by definition of $\Diamond_{\Phi'}$, while the second and third diagrams do by functoriality of $\otimes$. The fourth and fifth diagrams commute by naturality of m and monoidality of p, respectively.

Finally, we consider the following diagram:

$$\begin{tikzcd}
\llbracket \Phi', x : P; \emptyset \rrbracket \arrow[rrr, "\llbracket \Phi', x:P; \emptyset \vdash x:P \rrbracket"] \arrow[d, "\rho_{\Phi}"'] & & & \llbracket P \rrbracket \arrow[ddddd, "\tau_P"] \\
\llbracket \Phi', x : P \rrbracket \arrow[r, "\cong"] \arrow[dddd, "\tau_{\Phi}"'] & \llbracket \Phi' \rrbracket \otimes \llbracket P \rrbracket \arrow[r, "\Diamond_{\Phi'} \otimes \mathrm{id}"] \arrow[d, "\tau_{\Phi'} \otimes \mathrm{id}"'] & \llbracket I \rrbracket \otimes \llbracket P \rrbracket \arrow[ur, "\lambda_P"] \arrow[dd, "\mathrm{id} \otimes \tau_P"] & \\
& p(\llbracket \Phi' \rrbracket_p) \otimes \llbracket P \rrbracket \arrow[d, "\mathrm{id} \otimes \tau_P"'] & & \\
& p(\llbracket \Phi' \rrbracket_p) \otimes p(\llbracket P \rrbracket_p) \arrow[d, "m_{\Phi', P}"'] & \llbracket I \rrbracket \otimes p(\llbracket P \rrbracket_p) \arrow[ddr, bend left, "\lambda_P"] & \\
& p(\llbracket \Phi' \rrbracket_p \times \llbracket P \rrbracket_p) \arrow[r, "p(\Diamond^p_{\Phi'} \times \mathrm{id})"] & p(\llbracket I \rrbracket_p \times \llbracket P \rrbracket_p) \arrow[dr, "p(\pi_P)"] & \\
p(\llbracket \Phi', x : P \rrbracket_p) \arrow[ur, "p(\cong)"] \arrow[rrr, "p(\pi_P \circ (\Diamond^p_{\Phi'} \times \mathrm{id}) \circ \cong) \; = \; p(\llbracket V \rrbracket_p)"'] & & & p(\llbracket P \rrbracket_p)
\end{tikzcd}$$

Note that top diagram commutes by definition of $\llbracket \Phi', x : P; \emptyset \vdash x : P \rrbracket$, while the leftmost diagram does by (8.1). The center diagram commutes by (8.2), while the one to its right does by naturality of λ. Since p is a functor, the bottom diagram commutes as well. By definition, $\llbracket V \rrbracket_p = \pi_P \circ (\Diamond^p_{\Phi'} \times \mathrm{id}) \circ \cong : \llbracket \Phi \rrbracket_p \to \llbracket P \rrbracket_p$, and so the result follows.

- If the typing rule is (*const*) with $V = c$, then the rule is

$$\frac{}{\Phi; \emptyset \vdash c : P_c} \ (\mathit{const})$$

where $P = P_c$. Consider the following diagram:

$$\begin{tikzcd}
\llbracket \Phi; \emptyset \rrbracket \arrow[rr, "\llbracket \Phi;\emptyset \vdash c:P_c \rrbracket"] \arrow[d, "\rho_\Phi"'] & & \llbracket P_c \rrbracket \arrow[ddd, "\tau_{P_c}"] \\
\llbracket \Phi \rrbracket \arrow[r, "\Diamond_\Phi"'] \arrow[dd, "\tau_\Phi"'] & \llbracket I \rrbracket \arrow[ur, "\llbracket c \rrbracket"'] \arrow[d, "\tau_I"] & \\
& p(\llbracket I \rrbracket_p) \arrow[dr, "p(\llbracket c \rrbracket_p)"] & \\
p(\llbracket \Phi \rrbracket_p) \arrow[ur, "p(\Diamond^p_\Phi)"] \arrow[rr, "p(\llbracket c \rrbracket_p \circ \Diamond^p_\Phi) \; = \; p(\llbracket V \rrbracket_p)"'] & & p(\llbracket P_c \rrbracket_p)
\end{tikzcd}$$

Note that the top diagram commutes by definition of $\llbracket \Phi; \emptyset \vdash c : P_c \rrbracket$. The left diagram commutes by definition of $\Diamond_\Phi$, while the right one does by definition of $\llbracket c \rrbracket$. Since p is a functor, the bottom triangle commutes as well. By definition, $\llbracket V \rrbracket_p = \llbracket c \rrbracket_p \circ \Diamond^p_\Phi : \llbracket \Phi \rrbracket_p \to \llbracket P_c \rrbracket_p$, and so the result follows.

- If the typing rule is (*left*) with $V = \mathrm{left}_{R,S}\, W$, then the rule is

$$\frac{\Phi; \emptyset \vdash W : R}{\Phi; \emptyset \vdash \mathrm{left}_{R,S}\, W : R + S} \; (\textit{left})$$

where W is a value term and $P = R + S$ for some parameter types R and S. Consider the following diagram:

$$\begin{tikzcd}
\llbracket \Phi; \emptyset \rrbracket \arrow[rr, "\llbracket \Phi;\emptyset \vdash \mathrm{left}_{R,S}\, W : R+S \rrbracket"] \arrow[d, "\rho_\Phi"'] \arrow[dr, "\llbracket W \rrbracket"'] & & \llbracket R+S \rrbracket \arrow[ddd, "\tau_{R+S}"] \\
\llbracket \Phi \rrbracket \arrow[dd, "\tau_\Phi"'] & \llbracket R \rrbracket \arrow[ur, "\mathrm{left}_{R,S}"'] \arrow[d, "\tau_R"] & \\
& p(\llbracket R \rrbracket_p) \arrow[dr, "p(\mathrm{left}_{R,S})"] & \\
p(\llbracket \Phi \rrbracket_p) \arrow[ur, "p(\llbracket W \rrbracket_p)"] \arrow[rr, "p(\mathrm{left}_{R,S} \circ \llbracket W \rrbracket_p) \; = \; p(\llbracket V \rrbracket_p)"'] & & p(\llbracket R+S \rrbracket_p)
\end{tikzcd}$$

Note that the top triangle commutes by definition of $\llbracket \Phi; \emptyset \vdash \mathrm{left}_{R,S}\, W : R + S \rrbracket$, while the diagram on the right commutes by definition of τ_{R+S}. The validity of

$\Phi; \emptyset \vdash W : R$ implies that the diagram on the left commutes by the induction hypothesis. Since p is a functor, the bottom triangle commutes as well. By definition, $[\![V]\!]_p = \mathrm{left}_{R,S} \circ [\![W]\!]_p : [\![\Phi]\!]_p \to [\![R+S]\!]_p$, and so the result follows.

- If the typing rule is (*right*), then one proceeds as in the (*left*) case.

- If the tying rule is $(*)$ with $V = *$, then the rule is

$$\frac{}{\Phi; \emptyset \vdash * : I}\ (*)$$

where $P = I$. Consider the following diagram:

$$\begin{tikzcd}
{[\![\Phi;\emptyset]\!]} \arrow[r, "{[\![\Phi;\emptyset\vdash *:I]\!]}"] \arrow[d, "\rho_\Phi"'] & {[\![I]\!]} \arrow[dd, "\tau_I"] \\
{[\![\Phi]\!]} \arrow[ur, "\diamond_\Phi"'] \arrow[d, "\tau_\Phi"'] & \\
p([\![\Phi]\!]_p) \arrow[r, "p(\diamond^p_\Phi)\ =\ p([\![V]\!]_p)"'] & p([\![I]\!]_p)
\end{tikzcd}$$

Note that the top diagram commutes by definition of $[\![\Phi; \emptyset \vdash * : I]\!]$, while the one at the bottom does by definition of $\diamond_\Phi$. By definition, $[\![V]\!]_p = \diamond^p_\Phi : [\![\Phi]\!]_p \to [\![I]\!]_p$, and so the result follows.

- If the typing rule is (*pair*) with $V = \langle W_1, W_2\rangle$ for some value terms W_1 and W_2, then the rule is

$$\frac{\Phi_1; \emptyset \vdash W_1 : R \quad \Phi_2; \emptyset \vdash W_2 : S}{\Phi_1 \cup \Phi_2; \emptyset \vdash \langle W_1, W_2\rangle : R \otimes S}\ (\mathit{pair})$$

where $\Phi = \Phi_1 \cup \Phi_2$ for some parameter contexts Φ_1 and Φ_2, and $P = R \otimes S$ for some parameter types R and S. Consider the following diagram:

$$\begin{tikzcd}
\llbracket \Phi_1 \cup \Phi_2 ; \emptyset \rrbracket \arrow[rrr, "\llbracket \Phi_1\cup\Phi_2;\emptyset\vdash\langle W_1,W_2\rangle:R\otimes S\rrbracket"] \arrow[d, "\rho_{\Phi_1\cup\Phi_2}"'] \arrow[dr, "\triangle_{\Phi_1\cup\Phi_2;\emptyset}"] & & & \llbracket R\otimes S\rrbracket \arrow[ddddd, "\tau_{R\otimes S}"] \\
\llbracket \Phi \rrbracket \arrow[dr, "\triangle_{\Phi_1\cup\Phi_2}"] \arrow[dddd, "\tau_{\Phi_1\cup\Phi_2}"'] & \llbracket \Phi_1 ; \emptyset \rrbracket \otimes \llbracket \Phi_2 ; \emptyset \rrbracket \arrow[r, "\llbracket W_1\rrbracket\otimes\llbracket W_2\rrbracket"] \arrow[d, "\rho_{\Phi_1}\otimes\rho_{\Phi_2}"'] & \llbracket R\rrbracket \otimes \llbracket S\rrbracket \arrow[ur, "\mathrm{id}"] \arrow[dd, "\tau_R\otimes\tau_S"] & \\
 & \llbracket \Phi_1\rrbracket \otimes \llbracket \Phi_2\rrbracket \arrow[d, "\tau_{\Phi_1}\otimes\tau_{\Phi_2}"'] & & \\
 & p(\llbracket \Phi_1\rrbracket_p) \otimes p(\llbracket \Phi_2\rrbracket_p) \arrow[r, "p(\llbracket W_1\rrbracket_p)\otimes p(\llbracket W_2\rrbracket_p)"] \arrow[d, "m_{\Phi_1,\Phi_2}"'] & p(\llbracket R\rrbracket_p) \otimes p(\llbracket S\rrbracket_p) \arrow[d, "m_{R,S}"] & \\
 & p(\llbracket \Phi_1\rrbracket_p \times \llbracket \Phi_2\rrbracket_p) \arrow[r, "p(\llbracket W_1\rrbracket_p\times\llbracket W_2\rrbracket_p)"] & p(\llbracket R\rrbracket_p \times \llbracket S\rrbracket_p) \arrow[dr, "\mathrm{id}"] & \\
p(\llbracket \Phi_1\cup\Phi_2\rrbracket_p) \arrow[ur, "p(\triangle^p_{\Phi_1\cup\Phi_2})"] \arrow[rrr, "p((\llbracket W_1\rrbracket_p\times\llbracket W_2\rrbracket_p)\circ\triangle^p_{\Phi_1\cup\Phi_2}) \;=\; p(\llbracket V\rrbracket_p)"'] & & & p(\llbracket R\otimes S\rrbracket_p)
\end{tikzcd}$$

The top diagram commutes by definition of $\llbracket\Phi_1\cup\Phi_2;\emptyset \vdash \langle W_1, W_2\rangle : R\otimes S\rrbracket$, while the diagram on the right commutes by definition of $\tau_{R\otimes S}$. The leftmost diagrams commute by Definition 7.3.3, while the pentagon in the middle commutes by the induction hypothesis applied to the valid judgements $\Phi_1;\emptyset \vdash W_1 : R$ and $\Phi_2;\emptyset \vdash W_2 : S$. The square below it commutes by naturality of m, and since p is a functor, the bottom diagram commutes as well. By definition, $\llbracket V\rrbracket_p = (\llbracket W_1\rrbracket_p \times \llbracket W_2\rrbracket_p) \circ \triangle^p_{\Phi_1\cup\Phi_2} : \llbracket\Phi_1 \cup \Phi_2\rrbracket_p \to \llbracket R \otimes S\rrbracket_p$, and so the result follows.

- If the typing rule is (*lift*) with $V = \text{lift } M$, then the rule is

$$\frac{\Phi;\emptyset \vdash M : A}{\Phi;\emptyset \vdash \text{lift } M : !A}\ (\textit{lift})$$

where $P = !A$. By definition of $\llbracket\Phi;\emptyset \vdash \text{lift } M : !A\rrbracket$, the following diagram commutes:

$$\begin{tikzcd}
\llbracket \Phi;\emptyset\rrbracket \arrow[r, "\llbracket\Phi;\emptyset\vdash\text{lift } M:!A\rrbracket"] \arrow[d, "\rho_\Phi"'] & \llbracket !A\rrbracket \arrow[dd, "\tau_{!A}"] \\
\llbracket \Phi\rrbracket \arrow[d, "\tau_\Phi"'] & \\
p(\llbracket \Phi\rrbracket_p) \arrow[r, "\textbf{lift } \llbracket M\rrbracket"'] & p(\llbracket !A\rrbracket_p)
\end{tikzcd}$$

But **lift** $\llbracket M \rrbracket = p((\llbracket \Phi; \emptyset \vdash M : A \rrbracket \circ \rho_\Phi^{-1} \circ \tau_\Phi^{-1})^\circ)$ as seen in Section 7.3.2, and since $\llbracket V \rrbracket_p = (\llbracket \Phi; \emptyset \vdash M : A \rrbracket \circ \rho_\Phi^{-1} \circ \tau_\Phi^{-1})^\circ : \llbracket \Phi \rrbracket_p \to \llbracket !A \rrbracket_p$ by definition, the result follows.

- If the typing rule is $(circ)$ with $V = (\vec{\ell}, C, \vec{\ell'})$, then the rule is

$$\frac{Q \vdash_{\mathcal{L}} \vec{\ell} : T \quad Q' \vdash_{\mathcal{L}} \vec{\ell'} : U \quad C \in \mathbf{M}_{\mathcal{L}}(Q, Q')}{\Phi; \emptyset \vdash (\vec{\ell}, C, \vec{\ell'}) : \mathrm{Circ}(T, U)}\ (circ)$$

where $P = \mathrm{Circ}(T, U)$ for some simple M-types T and U. Consider the following diagram:

$$\begin{tikzcd}
\llbracket \Phi; \emptyset \rrbracket \arrow[rrr, "\llbracket \Phi;\emptyset \vdash (\vec{\ell}, C, \vec{\ell'}) : \mathrm{Circ}(T,U) \rrbracket"] \arrow[d, "\rho_\Phi"'] & & & \llbracket \mathrm{Circ}(T, U) \rrbracket \arrow[ddd, "\tau_{\mathrm{Circ}(T,U)}"] \\
\llbracket \Phi \rrbracket \arrow[r, "\diamond_\Phi"] \arrow[dd, "\tau_\Phi"'] & \llbracket I \rrbracket \arrow[urr, bend right=10, "\mathbf{circ}\ (\vec{\ell}, C, \vec{\ell'})"] \arrow[d, "\tau_I"] & & \\
& p(\llbracket I \rrbracket_p) \arrow[r, "p((\underline{C}^* \circ \tau_I^{-1})^\circ)"] & p(\flat(\llbracket T \rrbracket \multimap \llbracket U \rrbracket)) \arrow[uur, bend right=10, "\mathbf{box}"] \arrow[dr, "p(i_{T,U})"] & \\
p(\llbracket \Phi \rrbracket_p) \arrow[ur, "p(\diamond^p_\Phi)"] \arrow[rrr, "p(i_{T,U} \circ (\underline{C}^* \circ \tau_I^{-1})^\circ \circ \diamond^p_\Phi)\ =\ p(\llbracket V \rrbracket_p)"'] & & & p(\llbracket \mathrm{Circ}(T, U) \rrbracket_p)
\end{tikzcd}$$

The top region commutes by definition of $\llbracket \Phi; \emptyset \vdash (\vec{\ell}, C, \vec{\ell'}) : \mathrm{Circ}(T, U) \rrbracket$ while the one on the right does by definition of **box** and $\tau_{\mathrm{Circ}(T,U)}$. The leftmost diagram commutes by definition of $\diamond_\Phi$, while the one in the middle does by definition of **circ** $(\vec{\ell}, C, \vec{\ell'})$. Since p is a functor, the bottom region commutes as well. By definition, $\llbracket V \rrbracket_p = i_{T,U} \circ (\underline{C}^* \circ \tau_I^{-1})^\circ \circ \diamond^p_\Phi : \llbracket \Phi \rrbracket_p \to \llbracket \mathrm{Circ}(T, U) \rrbracket_p$, and so the result follows.

□

8.2 Discarding Simultaneously vs Discarding in Stages

We can discard parameter contexts simultaneously or in stages.

Lemma 8.2.1. *If Φ and Φ' are disjoint parameter contexts, then the following diagram commutes:*

$$\begin{tikzcd}
\llbracket \Phi, \Phi' \rrbracket \arrow[r, "\diamond_{\Phi,\Phi'}"] \arrow[dd, "\cong"'] & \llbracket I \rrbracket \\
 & \llbracket \Phi' \rrbracket \arrow[u, "\diamond_{\Phi'}"'] \\
\llbracket \Phi \rrbracket \otimes \llbracket \Phi' \rrbracket \arrow[r, "\diamond_\Phi \otimes \mathrm{id}"'] & \llbracket I \rrbracket \otimes \llbracket \Phi' \rrbracket . \arrow[u, "\lambda_{\Phi'}"']
\end{tikzcd}$$

Proof. Let Φ and Φ' be disjoint parameter contexts. Consider the following diagram:

$$\begin{tikzcd}
\llbracket \Phi, \Phi' \rrbracket \arrow[rrrr, "\diamond_{\Phi,\Phi'}"] \arrow[dr, "\tau_{\Phi,\Phi'}"] \arrow[ddddd, "\cong"'] & & & & \llbracket I \rrbracket \\
 & p(\llbracket \Phi, \Phi' \rrbracket_p) \arrow[rr, "p(\diamond^p_{\Phi,\Phi'})"] \arrow[d, "p(\cong)"'] & & p(\llbracket I \rrbracket_p) \arrow[ur, "\tau_I^{-1}"] & \\
 & p(\llbracket \Phi \rrbracket_p \times \llbracket \Phi' \rrbracket_p) \arrow[r, "p(\diamond^p_\Phi \times \mathrm{id})"] \arrow[d, "m^{-1}_{\Phi,\Phi'}"'] & p(\llbracket I \rrbracket_p \times \llbracket \Phi' \rrbracket_p) \arrow[r, "p(\pi_{\Phi'})"] & p(\llbracket \Phi' \rrbracket_p) \arrow[u, "p(\diamond^p_{\Phi'})"'] \arrow[r, "\tau_{\Phi'}^{-1}"] & \llbracket \Phi' \rrbracket \arrow[uu, "\diamond_{\Phi'}"'] \\
 & p(\llbracket \Phi \rrbracket_p) \otimes p(\llbracket \Phi' \rrbracket_p) \arrow[r, "p(\diamond^p_\Phi)\otimes \mathrm{id}"] \arrow[d, "\tau_\Phi^{-1}\otimes \mathrm{id}"'] & p(\llbracket I \rrbracket_p) \otimes p(\llbracket \Phi' \rrbracket_p) \arrow[u, "m_{I,\Phi'}"'] \arrow[dr, "\tau_I^{-1}\otimes \mathrm{id}"] & & \\
 & \llbracket \Phi \rrbracket \otimes p(\llbracket \Phi' \rrbracket_p) \arrow[rr, "\diamond_\Phi \otimes \mathrm{id}"] \arrow[dl, "\mathrm{id}\otimes \tau_{\Phi'}^{-1}"'] & & \llbracket I \rrbracket \otimes p(\llbracket \Phi' \rrbracket_p) \arrow[uu, "\lambda_{p(\Phi')}"'] \arrow[dr, "\mathrm{id}\otimes \tau_{\Phi'}^{-1}"] & \\
\llbracket \Phi \rrbracket \otimes \llbracket \Phi' \rrbracket \arrow[rrrr, "\diamond_\Phi \otimes \mathrm{id}"'] & & & & \llbracket I \rrbracket \otimes \llbracket \Phi' \rrbracket \arrow[uuu, "\lambda_{\Phi'}"']
\end{tikzcd}$$

The top region commutes by definition of $\diamond_{\Phi,\Phi'}$, while the one below it does because $\llbracket I \rrbracket_p$ is terminal in **Set** and p is a functor. The square in the center commutes by naturality of m, while the region to its right does by monoidality of p. The region below it commutes by definition of $\diamond_\Phi$ and the functoriality of $- \otimes \mathrm{id}$, while the bottom one does by functoriality of $\otimes$. The leftmost diagram commutes because the isomorphism $\cong\colon \llbracket \Phi, \Phi' \rrbracket \to \llbracket \Phi \rrbracket \otimes \llbracket \Phi' \rrbracket$ can be decomposed precisely as the right-hand side path of that diagram, see diagram (8.1). The top rightmost diagram commutes by definition of $\diamond_{\Phi'}$, while the one below it does by naturality of λ. Hence, the outer square commutes as required. $\square$

In what follows, we may omit the subscripts and superscripts of the discarding morphisms $\diamond$ when no confusion arises.

8.3 Special Semantic Weakening Lemma

In this section, we prove the semantic version of the Special Weakening Lemma 6.2.1.

Definition 8.3.1 (Weakening of a Derivation)**.** Let $\mathcal{D}$ be a derivation of $\Gamma; Q \vdash N : B$ and Φ a parameter context. We say that Φ *is disjoint from* $\mathcal{D}$, and denote it by $\Phi \cap \mathcal{D} = \emptyset$, if for every variable context Γ' of $\mathcal{D}$, $\Phi \cap \Gamma' = \emptyset$. If Φ is disjoint from $\mathcal{D}$, we define $\Phi, \mathcal{D}$ to be the derivation of $\Phi, \Gamma; Q \vdash N : B$ obtained by extending every variable context of $\mathcal{D}$ by Φ. We call $\Phi, \mathcal{D}$ the *weakening of* $\mathcal{D}$ *by* Φ. If $\Phi' = \Phi \backslash \Gamma$ is disjoint from $\mathcal{D}$, we define $\Phi \cup \mathcal{D}$ to be the weakening of $\mathcal{D}$ by Φ'. $\diamond$

Note that the weakening of a derivation is indeed a valid derivation by the syntactic Special Weakening Lemma 6.2.1.

Lemma 8.3.2 (Special Semantic Weakening Lemma)**.** *If* $\Gamma; Q \vdash N : B$ *is a valid typing judgement with derivation* $\mathcal{D}$*, and* Φ *a parameter context disjoint from* $\mathcal{D}$*, then the valid typing judgment* $\Phi, \Gamma; Q \vdash N : B$ *with derivation* $\Phi, \mathcal{D}$ *makes the following diagram commute:*

$$\begin{tikzcd}
\llbracket \Phi, \Gamma; Q \rrbracket \arrow[r, "\llbracket \Phi, \Gamma; Q \vdash N : B \rrbracket"] \arrow[dd, "\cong"'] & \llbracket B \rrbracket \\
 & \llbracket \Gamma; Q \rrbracket \arrow[u, "\llbracket \Gamma; Q \vdash N : B \rrbracket"'] \\
\llbracket \Phi \rrbracket \otimes \llbracket \Gamma; Q \rrbracket \arrow[r, "\diamond_\Phi \otimes \mathrm{id}"'] & \llbracket I \rrbracket \otimes \llbracket \Gamma; Q \rrbracket, \arrow[u, "\lambda_{\Gamma; Q}"']
\end{tikzcd}$$

where $\llbracket \Gamma; Q \vdash N : B \rrbracket = \llbracket \mathcal{D} \rrbracket$ *and* $\llbracket \Phi, \Gamma; Q \vdash N : B \rrbracket = \llbracket \Phi, \mathcal{D} \rrbracket$*.*

Proof. By induction on the typing derivation $\mathcal{D}$ of $\Gamma; Q \vdash N : B$. Let Φ be a parameter context disjoint from $\mathcal{D}$, and $\Phi, \Gamma; Q \vdash N : B$ the valid typing judgment with derivation $\Phi, \mathcal{D}$. We now proceed by case distinction:

- If the last (and only) rule of $\mathcal{D}$ is (*var*) with $N = x$, then $\mathcal{D}$ is

$$\frac{}{\Phi', x : B; \emptyset \vdash x : B}\ (\mathit{var})$$

where $\Gamma = (\Phi', x : B)$ with Φ' a parameter context and $Q = \emptyset$. Since Φ is disjoint from $\mathcal{D}$, $\Phi \cap (\Phi', x : B) = \emptyset$, and so $\Phi, \Phi', x : B$ is a well-defined context. By the (*var*) rule, $\Phi, \Phi', x : B; \emptyset \vdash x : B$ is valid, and so, $\llbracket \Phi, \Phi', x : B; \emptyset \vdash x : B \rrbracket$ is defined. Consider the following diagram:

$$\begin{tikzcd}
\llbracket \Phi, \Phi', x : B; \emptyset \rrbracket \arrow[rrr, "\llbracket \Phi, \Phi', x:B; \emptyset \vdash x:B \rrbracket"] \arrow[ddddd, "\cong"'] \arrow[ddr, "\cong"] & & & \llbracket B \rrbracket \\
& & \llbracket x : B; \emptyset \rrbracket \arrow[ur, bend right=10, "\rho_{x:B}"] & \\
& \llbracket \Phi, \Phi' \rrbracket \otimes \llbracket x : B; \emptyset \rrbracket \arrow[r, "\diamond_{\Phi,\Phi'} \otimes \mathrm{id}"] \arrow[dd, "\cong \otimes \mathrm{id}"'] & \llbracket I \rrbracket \otimes \llbracket x : B; \emptyset \rrbracket \arrow[u, "\lambda_{x:B;\emptyset}"] & \\
& & \llbracket \Phi' \rrbracket \otimes \llbracket x : B; \emptyset \rrbracket \arrow[u, "\diamond_{\Phi'} \otimes \mathrm{id}"] & \llbracket \Phi', x : B; \emptyset \rrbracket \arrow[l, "\cong"'] \arrow[uuu, "\llbracket \Phi', x:B; \emptyset \vdash x:B \rrbracket"'] \\
& \llbracket \Phi \rrbracket \otimes \llbracket \Phi' \rrbracket \otimes \llbracket x : B; \emptyset \rrbracket \arrow[r, "\diamond_{\Phi} \otimes \mathrm{id} \otimes \mathrm{id}"] & \llbracket I \rrbracket \otimes \llbracket \Phi' \rrbracket \otimes \llbracket x : B; \emptyset \rrbracket \arrow[u, "\lambda_{\Phi'} \otimes \mathrm{id}"] & \\
\llbracket \Phi \rrbracket \otimes \llbracket \Phi'; x : B; \emptyset \rrbracket \arrow[rrr, "\diamond_{\Phi} \otimes \mathrm{id}"'] \arrow[ur, "\cong"] & & & \llbracket I \rrbracket \otimes \llbracket \Phi', x : B; \emptyset \rrbracket \arrow[ul, "\cong"'] \arrow[uu, "\lambda_{\Phi', x:B;\emptyset}"']
\end{tikzcd}$$

Note that the top diagram commutes by definition of $\llbracket \Phi, \Phi', x : B; \emptyset \vdash x : B \rrbracket$, while the one to its right commutes by definition of $\llbracket \Phi', x : B; \emptyset \vdash x : B \rrbracket$. The diagram below the latter commutes by coherence for symmetric monoidal categories as does the leftmost diagram. The center diagram commutes by Lemma 8.2.1 and the functoriality of $- \otimes \mathrm{id}$, while the one at the bottom does by naturality of the monoidal structure. The result now follows.

- If the last (and only) rule of $\mathcal{D}$ is (*label*) with $N = \ell$, then $\mathcal{D}$ is

$$\frac{}{\Phi'; \ell : \alpha \vdash \ell : \alpha}\ (\mathit{label})$$

where $\Gamma = \Phi'$, a parameter context, $Q = \ell : \alpha$, and $B = \alpha$, a wire type. Since Φ is disjoint from $\mathcal{D}$, $\Phi \cap \Phi' = \emptyset$, and so Φ, Φ' is a well-defined parameter context. By the (*label*) rule, $\Phi, \Phi'; \ell : \alpha \vdash \ell : \alpha$ is valid, and so $\llbracket \Phi, \Phi'; \ell : \alpha \vdash \ell : \alpha \rrbracket$ is defined. Consider the following diagram:

$$\begin{tikzcd}
\llbracket \Phi, \Phi'; \ell : \alpha \rrbracket \arrow[rrrr, "\llbracket \Phi,\Phi';\ell:\alpha\vdash\ell:\alpha \rrbracket"] \arrow[dr, "\mathrm{id}"] \arrow[dddd, "\cong"'] & & & & \llbracket \alpha \rrbracket \\
& \llbracket \Phi, \Phi' \rrbracket \otimes \llbracket \alpha \rrbracket \arrow[rr, "\diamond_{\Phi,\Phi'}\otimes\mathrm{id}"] \arrow[dd, "\cong\otimes\mathrm{id}"'] & & \llbracket I \rrbracket \otimes \llbracket \alpha \rrbracket \arrow[ur, "\lambda_\alpha"'] & \\
& & & \llbracket \Phi' \rrbracket \otimes \llbracket \alpha \rrbracket \arrow[u, "\diamond_{\Phi'}\otimes\mathrm{id}"] & \llbracket \Phi'; \ell : \alpha \rrbracket \arrow[l, "\mathrm{id}"'] \arrow[uu, "\llbracket \Phi';\ell:\alpha\vdash\ell:\alpha \rrbracket"'] \\
& \llbracket \Phi \rrbracket \otimes \llbracket \Phi' \rrbracket \otimes \llbracket \alpha \rrbracket \arrow[rr, "\diamond_\Phi\otimes\mathrm{id}\otimes\mathrm{id}"] \arrow[dl, "\cong"'] & & \llbracket I \rrbracket \otimes \llbracket \Phi' \rrbracket \otimes \llbracket \alpha \rrbracket \arrow[u, "\lambda_{\Phi'}\otimes\mathrm{id}"] & \\
\llbracket \Phi \rrbracket \otimes \llbracket \Phi'; \ell : \alpha \rrbracket \arrow[rrrr, "\diamond_\Phi\otimes\mathrm{id}"'] & & & & \llbracket I \rrbracket \otimes \llbracket \Phi'; \ell : \alpha \rrbracket \arrow[ul, "\cong"'] \arrow[uu, "\lambda_{\Phi';\ell:\alpha}"']
\end{tikzcd}$$

Note that the top diagram commutes by definition of $\llbracket \Phi, \Phi'; \ell : \alpha \vdash \ell : \alpha \rrbracket$, while the one on the top right commutes by definition of $\llbracket \Phi'; \ell : \alpha \vdash \ell : \alpha \rrbracket$. The diagram below the latter commutes by coherence for symmetric monoidal categories as does the leftmost diagram. The center diagram commutes by Lemma 8.2.1 and the functoriality of $- \otimes \mathrm{id}$, while the one at the bottom does by naturality of the monoidal structure. The result now follows.

- If the last (and only) rule of $\mathcal{D}$ is $(*)$, then we proceed as in the (*label*) case.

- If the last (and only) rule of $\mathcal{D}$ is (*const*) with $N = c$, then $\mathcal{D}$ is

$$\frac{}{\Phi'; \emptyset \vdash c : P_c}\ (\mathit{const})$$

where $\Gamma = \Phi'$, a parameter context, $Q = \emptyset$, and $B = P_c$, a parameter type. Since Φ is disjoint from $\mathcal{D}$, $\Phi \cap \Phi' = \emptyset$, and so Φ, Φ' is a well-defined parameter context. By the (*const*) rule, $\Phi, \Phi'; \emptyset \vdash c : P_c$ is valid, and so $\llbracket \Phi, \Phi'; \emptyset \vdash c : P_c \rrbracket$ is defined. Observe that in the following diagram, the top triangle commutes by definition of $\llbracket \Phi'; \emptyset \vdash c : P_c \rrbracket$, while the other one does by naturality of ρ:

$$\begin{tikzcd}
\llbracket \Phi'; \emptyset \rrbracket \arrow[rr, "\llbracket \Phi';\emptyset\vdash c:P_c \rrbracket"] \arrow[dr, "\rho_{\Phi'}"] \arrow[dd, "\diamond_{\Phi'}\otimes \mathrm{id}"'] & & \llbracket P_c \rrbracket \\
& \llbracket \Phi' \rrbracket \arrow[dr, "\diamond_{\Phi'}"] & \\
\llbracket I \rrbracket \otimes \llbracket I \rrbracket \arrow[rr, "\rho_I"'] & & \llbracket I \rrbracket \arrow[uu, "\llbracket c \rrbracket"']
\end{tikzcd} \tag{8.3}$$

Now, consider the following diagram:

$$\begin{tikzcd}
\llbracket \Phi, \Phi'; \emptyset \rrbracket \arrow[rrr, "\llbracket \Phi,\Phi';\emptyset\vdash c:P_c \rrbracket"] \arrow[dr, "\rho_{\Phi,\Phi'}"'] \arrow[ddddd, "\cong"] & & & \llbracket P_c \rrbracket \\
& \llbracket \Phi, \Phi' \rrbracket \arrow[r, "\diamond_{\Phi,\Phi'}"] \arrow[d, "\rho^{-1}_{\Phi,\Phi'}"'] & \llbracket I \rrbracket \arrow[ur, "\llbracket c \rrbracket"'] & \\
& \llbracket \Phi, \Phi' \rrbracket \otimes \llbracket I \rrbracket \arrow[r, "\diamond_{\Phi,\Phi'}\otimes \mathrm{id}"] \arrow[dd, "\cong\otimes \mathrm{id}"'] & \llbracket I \rrbracket \otimes \llbracket I \rrbracket \arrow[u, "\rho_I"'] & \\
& & \llbracket \Phi' \rrbracket \otimes \llbracket I \rrbracket \arrow[u, "\diamond_{\Phi'}\otimes \mathrm{id}"'] & \llbracket \Phi'; \emptyset \rrbracket \arrow[l, "\mathrm{id}"'] \arrow[uuu, "\llbracket \Phi';\emptyset\vdash c:P_c \rrbracket"'] \\
& \llbracket \Phi \rrbracket \otimes \llbracket \Phi' \rrbracket \otimes \llbracket I \rrbracket \arrow[r, "\diamond_{\Phi}\otimes \mathrm{id}\otimes \mathrm{id}"] & \llbracket I \rrbracket \otimes \llbracket \Phi' \rrbracket \otimes \llbracket I \rrbracket \arrow[u, "\lambda_{\Phi'}\otimes \mathrm{id}"'] & \\
\llbracket \Phi \rrbracket \otimes \llbracket \Phi'; \emptyset \rrbracket \arrow[ur, "\cong"] \arrow[rrr, "\diamond_{\Phi}\otimes \mathrm{id}"'] & & & \llbracket I \rrbracket \otimes \llbracket \Phi'; \emptyset \rrbracket \arrow[ul, "\cong"'] \arrow[uu, "\lambda_{\Phi';\emptyset}"']
\end{tikzcd}$$

Note that the top diagram commutes by definition of $\llbracket \Phi, \Phi'; \emptyset \vdash c : P_c \rrbracket$, while the one on the right commutes by (8.3). The diagram below the latter commutes by coherence for symmetric monoidal categories as does the leftmost diagram. The top center diagram commutes by naturality of ρ, while the one below it does by Lemma 8.2.1 and the functoriality of $-\otimes \mathrm{id}$. The diagram at the bottom commutes by naturality of the monoidal structure. The result now follows.

- If the last (and only) rule of $\mathcal{D}$ is (*circ*), then we proceed as in the (*const*) case.
- If the last rule of $\mathcal{D}$ is (*initial*) with $N = \Box_B M$, then $\mathcal{D}$ is

$$\frac{\begin{array}{c}\vdots \\ \Gamma; Q \vdash M : 0\end{array}}{\Gamma; Q \vdash \Box_B M : B}\ (\textit{initial}),$$

where $\Gamma; Q \vdash M : 0$ is a valid typing judgment with derivation $\mathcal{D}'$. Since Φ is disjoint from $\mathcal{D}$, Φ is disjoint from $\mathcal{D}'$, and so $\Phi, \Gamma; Q \vdash M : 0$ and $\Phi, \Gamma; Q \vdash \Box_B M : B$ are valid. Hence, $\llbracket \Phi, \Gamma; Q \vdash M : 0 \rrbracket$ and $\llbracket \Phi, \Gamma; Q \vdash \Box_B M : B \rrbracket$ are defined as well. Consider the following diagram:

$$\begin{tikzcd}
\llbracket \Phi, \Gamma; Q \rrbracket \arrow[rrrr, "{\llbracket \Phi,\Gamma;Q\vdash \Box_B M:B \rrbracket}"] \arrow[rrd, "{\llbracket \Phi,\Gamma;Q\vdash M:0 \rrbracket}"'] \arrow[ddd, "\cong"'] & & & & \llbracket B \rrbracket \\
& & \llbracket 0 \rrbracket \arrow[rru, "\Box_B"'] & & \\
& & & & \llbracket \Gamma; Q \rrbracket \arrow[llu, "{\llbracket \Gamma;Q\vdash M:0 \rrbracket}"] \arrow[uu, "{\llbracket \Gamma;Q\vdash \Box_B M:B \rrbracket}"'] \\
\llbracket \Phi \rrbracket \otimes \llbracket \Gamma; Q \rrbracket \arrow[rrrr, "\Diamond_\Phi \otimes \mathrm{id}"'] & & & & \llbracket I \rrbracket \otimes \llbracket \Gamma; Q \rrbracket \arrow[u, "\lambda_{\Gamma;Q}"']
\end{tikzcd}$$

Note that the top triangle commutes by definition of $\llbracket \Phi, \Gamma; Q \vdash \Box_B M : B \rrbracket$, while the one on the right commutes by definition of $\llbracket \Gamma; Q \vdash \Box_B M : B \rrbracket$. The bottom diagram commutes by applying the induction hypothesis to the subderivation $\mathcal{D}'$ of $\mathcal{D}$. The result now follows.

- If the last rule of $\mathcal{D}$ is (*left*), (*right*), (*force*), or (*box*), then one proceeds as in the (*initial*) case.

- If the last rule of $\mathcal{D}$ is (*lift*) with $N = \text{lift } M$, then $\mathcal{D}$ is

$$\dfrac{\dfrac{\vdots}{\Phi'; \emptyset \vdash M : A}}{\Phi'; \emptyset \vdash \text{lift } M : !A}\ (\textit{lift})$$

where $\Gamma = \Phi'$, a parameter context, $Q = \emptyset$, $B = !A$, and $\Phi'; \emptyset \vdash M : A$ is a valid typing judgement with derivation $\mathcal{D}'$. Since Φ is disjoint from $\mathcal{D}$, Φ is disjoint from $\mathcal{D}'$, and so $\Phi, \Phi'; \emptyset \vdash M : A$ and $\Phi, \Phi'; \emptyset \vdash \text{lift } M : !A$ are valid. Hence, $\llbracket \Phi, \Phi'; \emptyset \vdash M : A \rrbracket$ and $\llbracket \Phi, \Phi'; \emptyset \vdash \text{lift } M : !A \rrbracket$ are defined as well. To simplify notation, we may denote $\llbracket \Phi'; \emptyset \vdash M : A \rrbracket$ by $\llbracket M_{\Phi'} \rrbracket$ and $\llbracket \Phi, \Phi'; \emptyset \vdash M : A \rrbracket$ by

$\llbracket M_{\Phi,\Phi'} \rrbracket$. We want to show that the following diagram commutes:

$$\begin{tikzcd}
\llbracket \Phi, \Phi'; \emptyset \rrbracket \arrow[r, "\llbracket \Phi,\Phi';\emptyset \vdash \mathrm{lift}\ M : !A \rrbracket"] \arrow[dd, "\cong"'] & \llbracket !A \rrbracket \\
 & \llbracket \Phi'; \emptyset \rrbracket \arrow[u, "\llbracket \Phi';\emptyset \vdash \mathrm{lift}\ M : !A \rrbracket"'] \\
\llbracket \Phi \rrbracket \otimes \llbracket \Phi'; \emptyset \rrbracket \arrow[r, "\diamond_\Phi \otimes \mathrm{id}"'] & \llbracket I \rrbracket \otimes \llbracket \Phi'; \emptyset \rrbracket . \arrow[u, "\lambda_{\Phi';\emptyset}"']
\end{tikzcd} \tag{8.4}$$

This will be accomplished by proving several coherence conditions. First, we show that the following diagram commutes:

$$\begin{tikzcd}
\llbracket \Phi, \Phi'; \emptyset \rrbracket \arrow[r, "\rho_{\Phi,\Phi'}"] \arrow[d, "\cong"'] & \llbracket \Phi, \Phi' \rrbracket \arrow[r, "\tau_{\Phi,\Phi'}"] & p(\llbracket \Phi, \Phi' \rrbracket_p) \arrow[dd, "\mathrm{id}"] \\
\llbracket \Phi \rrbracket \otimes \llbracket \Phi'; \emptyset \rrbracket \arrow[d, "\mathrm{id} \otimes \rho_{\Phi'}"'] & & \\
\llbracket \Phi \rrbracket \otimes \llbracket \Phi' \rrbracket \arrow[uur, "\cong"] \arrow[r, "\tau_\Phi \otimes \tau_{\Phi'}"] \arrow[d, "\diamond_\Phi \otimes \mathrm{id}"'] & p(\llbracket \Phi \rrbracket_p) \otimes p(\llbracket \Phi' \rrbracket_p) \arrow[r, "m_{\Phi,\Phi'}"] \arrow[d, "p(\diamond^p_\Phi) \otimes p(\mathrm{id})"] & p(\llbracket \Phi \rrbracket_p \times \llbracket \Phi' \rrbracket_p) \arrow[d, "p(\diamond^p_\Phi \times \mathrm{id})"] \\
\llbracket I \rrbracket \otimes \llbracket \Phi' \rrbracket \arrow[r, "\tau_I \otimes \tau_{\Phi'}"] \arrow[d, "\mathrm{id} \otimes \rho^{-1}_{\Phi'}"'] \arrow[dr, "\mathrm{id} \otimes \tau_{\Phi'}"] \arrow[ddr, "\lambda_{\Phi'}"'] & p(\llbracket I \rrbracket_p) \otimes p(\llbracket \Phi' \rrbracket_p) \arrow[r, "m_{I,\Phi'}"] \arrow[d, "\tau_I^{-1} \otimes \mathrm{id} = m_I^{-1} \otimes \mathrm{id}"] & p(\llbracket I \rrbracket_p \times \llbracket \Phi' \rrbracket_p) \arrow[dd, "p(\overline{\pi}_{\Phi'})"] \\
\llbracket I \rrbracket \otimes \llbracket \Phi'; \emptyset \rrbracket \arrow[d, "\lambda_{\Phi';\emptyset}"'] & \llbracket I \rrbracket \otimes p(\llbracket \Phi' \rrbracket_p) \arrow[dr, "\lambda_{\Phi'_p}"] & \\
\llbracket \Phi' \rrbracket \otimes \llbracket I \rrbracket \arrow[r, "\rho_{\Phi'}"'] & \llbracket \Phi' \rrbracket \arrow[r, "\tau_{\Phi'}"'] & p(\llbracket \Phi' \rrbracket_p).
\end{tikzcd} \tag{8.5}$$

The top left triangle commutes by coherence for symmetric monoidal categories; the region to its right does by definition of τ. The square on the left commutes by definition of $\diamond_\Phi$ and functoriality of $\otimes$. The commutativity of the triangle below it is immediate. The square on the right and the region below it commute by monoidality of p. The bottom two triangles commute by naturality of the left unitor λ. The commutativity of the outer square now follows.

Since the compositions on the left- and right-hand sides of the previous diagram yield the morphisms

$$[\![\Phi, \Phi'; \emptyset]\!] \xrightarrow{\cong} [\![\Phi]\!] \otimes [\![\Phi'; \emptyset]\!] \xrightarrow{\diamond_\Phi \otimes \mathrm{id}} [\![I]\!] \otimes [\![\Phi'; \emptyset]\!] \xrightarrow{\lambda_{\Phi';\emptyset}} [\![\Phi']\!] \otimes [\![I]\!]$$

and

$$p([\![\Phi, \Phi']\!]_p) \xrightarrow{p(\pi_{\Phi'})} p([\![\Phi']\!]_p),$$

respectively, diagram (8.5) can be written as

$$\begin{tikzcd}
[\![\Phi, \Phi'; \emptyset]\!] \arrow[r, "\rho_{\Phi,\Phi'}"] \arrow[d, "\cong"'] & [\![\Phi, \Phi']\!] \arrow[r, "\tau_{\Phi,\Phi'}"] & p([\![\Phi, \Phi']\!]_p) \arrow[ddd, "p(\pi_{\Phi'})"] \\
[\![\Phi]\!] \otimes [\![\Phi'; \emptyset]\!] \arrow[d, "\diamond_\Phi \otimes \mathrm{id}"'] & & \\
[\![I]\!] \otimes [\![\Phi'; \emptyset]\!] \arrow[d, "\lambda_{\Phi';\emptyset}"'] & & \\
[\![\Phi']\!] \otimes [\![I]\!] \arrow[r, "\rho_{\Phi'}"'] & [\![\Phi']\!] \arrow[r, "\tau_{\Phi'}"'] & p([\![\Phi']\!]_p).
\end{tikzcd} \tag{8.6}$$

Applying the functor $! = p \circ \flat$ to this square yields the following commutative diagram:

$$\begin{tikzcd}
{!p(\llbracket \Phi, \Phi' \rrbracket_p)} \arrow[r, "{!\tau^{-1}_{\Phi,\Phi'}}"] \arrow[ddd, "{!p(\pi_{\Phi'})}"'] & {!\llbracket \Phi, \Phi' \rrbracket} \arrow[r, "{!\rho^{-1}_{\Phi,\Phi'}}"] & {!\llbracket \Phi, \Phi'; \emptyset \rrbracket} \arrow[d, "!\cong"] \\
 & & {!(\llbracket \Phi \rrbracket \otimes \llbracket \Phi'; \emptyset \rrbracket)} \arrow[d, "{!(\diamond_\Phi \otimes \mathrm{id})}"] \\
 & & {!(\llbracket I \rrbracket \otimes \llbracket \Phi'; \emptyset \rrbracket)} \arrow[d, "{!\lambda_{\Phi';\emptyset}}"] \\
{!p(\llbracket \Phi' \rrbracket_p)} \arrow[r, "{!\tau^{-1}_{\Phi'}}"'] & {!\llbracket \Phi' \rrbracket} \arrow[r, "{!\rho^{-1}_{\Phi'}}"'] & {!(\llbracket \Phi' \rrbracket \otimes \llbracket I \rrbracket).}
\end{tikzcd} \tag{8.7}$$

Now the morphism $p(\llbracket \Phi, \Phi' \rrbracket_p) \xrightarrow{\tau^{-1}_{\Phi,\Phi'}} \llbracket \Phi, \Phi' \rrbracket \xrightarrow{\rho^{-1}_{\Phi,\Phi'}} \llbracket \Phi, \Phi'; \emptyset \rrbracket \xrightarrow{\llbracket M_{\Phi,\Phi'} \rrbracket} \llbracket A \rrbracket$ has as its transpose is the unique morphism $(M_{\Phi,\Phi'} \circ \rho^{-1}_{\Phi,\Phi'} \circ \tau^{-1}_{\Phi,\Phi'})^\circ$ making the following diagram commute:

$$\begin{tikzcd}
{\llbracket \Phi, \Phi' \rrbracket_p} \arrow[rr, "{(\llbracket M_{\Phi,\Phi'} \rrbracket \circ \rho^{-1}_{\Phi,\Phi'} \circ \tau^{-1}_{\Phi,\Phi'})^\circ}"] \arrow[d, "{\eta_{\Phi,\Phi'}}"'] & & {\flat(\llbracket A \rrbracket)} \\
{\flat p(\llbracket \Phi, \Phi' \rrbracket_p)} \arrow[r, "{\flat(\tau^{-1}_{\Phi,\Phi'})}"'] & {\flat(\llbracket \Phi, \Phi' \rrbracket)} \arrow[r, "{\flat(\rho^{-1}_{\Phi,\Phi'})}"'] & {\flat(\llbracket \Phi, \Phi'; \emptyset \rrbracket)} \arrow[u, "{\flat(\llbracket M_{\Phi,\Phi'} \rrbracket)}"']
\end{tikzcd}$$

Applying p to this diagram yields:

$$\begin{tikzcd}
{p(\llbracket \Phi, \Phi' \rrbracket_p)} \arrow[rr, "{p((\llbracket M_{\Phi,\Phi'} \rrbracket \circ \rho^{-1}_{\Phi,\Phi'} \circ \tau^{-1}_{\Phi,\Phi'})^\circ)}"] \arrow[d, "{p(\eta_{\Phi,\Phi'})}"'] & & {!\llbracket A \rrbracket} \\
{!p(\llbracket \Phi, \Phi' \rrbracket_p)} \arrow[r, "{!\tau^{-1}_{\Phi,\Phi'}}"'] & {!\llbracket \Phi, \Phi' \rrbracket} \arrow[r, "{!\rho^{-1}_{\Phi,\Phi'}}"'] & {!\llbracket \Phi, \Phi'; \emptyset \rrbracket} \arrow[u, "{!\llbracket M_{\Phi,\Phi'} \rrbracket}"']
\end{tikzcd} \tag{8.8}$$

By the induction hypothesis applied to the subderivation $\mathcal{D}'$ of $\Phi'; \emptyset \vdash M : A$, we have that the following diagram commutes:

$$\begin{tikzcd}
\llbracket \Phi, \Phi'; \emptyset \rrbracket \arrow[r, "\llbracket M_{\Phi,\Phi'} \rrbracket"] \arrow[dd, "\cong"'] & \llbracket A \rrbracket \\
 & \llbracket \Phi'; \emptyset \rrbracket \arrow[u, "\llbracket M_{\Phi'} \rrbracket"'] \\
\llbracket \Phi \rrbracket \otimes \llbracket \Phi'; \emptyset \rrbracket \arrow[r, "\diamond_{\Phi} \otimes \mathrm{id}"'] & \llbracket I \rrbracket \otimes \llbracket \Phi'; \emptyset \rrbracket. \arrow[u, "\lambda_{\Phi';\emptyset}"']
\end{tikzcd}$$

Thus, applying ! to this diagram implies that

$$\begin{tikzcd}
!\llbracket \Phi, \Phi'; \emptyset \rrbracket \arrow[r, "!\llbracket M_{\Phi,\Phi'} \rrbracket"] \arrow[dr, "!(\lambda_{\Phi';\emptyset} \circ (\diamond_{\Phi} \circ \mathrm{id}) \circ \cong)"'] & !\llbracket A \rrbracket \\
 & !\llbracket \Phi'; \emptyset \rrbracket \arrow[u, "!\llbracket M_{\Phi'} \rrbracket"']
\end{tikzcd} \tag{8.9}$$

commutes.

Similarly, the morphism $p(\llbracket \Phi' \rrbracket_p) \xrightarrow{\tau_{\Phi'}^{-1}} \llbracket \Phi' \rrbracket \xrightarrow{\rho_{\Phi'}^{-1}} \llbracket \Phi'; \emptyset \rrbracket \xrightarrow{\llbracket M_{\Phi'} \rrbracket} \llbracket A \rrbracket$ has as its transpose is the unique morphism $(M_{\Phi'} \circ \rho_{\Phi'}^{-1} \circ \tau_{\Phi'}^{-1})^{\circ}$ making the following diagram commute:

$$\begin{tikzcd}
\llbracket \Phi' \rrbracket_p \arrow[rr, "(\llbracket M_{\Phi'} \rrbracket \circ \rho_{\Phi'}^{-1} \circ \tau_{\Phi'}^{-1})^{\circ}"] \arrow[d, "\eta_{\Phi'}"'] & & \flat(\llbracket A \rrbracket) \\
\flat p(\llbracket \Phi' \rrbracket_p) \arrow[r, "\flat(\tau_{\Phi'}^{-1})"'] & \flat(\llbracket \Phi' \rrbracket) \arrow[r, "\flat(\rho_{\Phi'}^{-1})"'] & \flat(\llbracket \Phi'; \emptyset \rrbracket) \arrow[u, "\flat(\llbracket M_{\Phi'} \rrbracket)"']
\end{tikzcd}$$

Applying p to this diagram yields:

$$\begin{tikzcd}
p(\llbracket \Phi' \rrbracket_p) \arrow[rr, "p((\llbracket M_{\Phi'} \rrbracket \circ \rho_{\Phi'}^{-1} \circ \tau_{\Phi'}^{-1})^{\circ})"] \arrow[d, "p(\eta_{\Phi'})"'] & & !\llbracket A \rrbracket \\
!p(\llbracket \Phi' \rrbracket_p) \arrow[r, "!\tau_{\Phi'}^{-1}"'] & !\llbracket \Phi' \rrbracket \arrow[r, "!\rho_{\Phi'}^{-1}"'] & !\llbracket \Phi'; \emptyset \rrbracket \arrow[u, "!\llbracket M_{\Phi'} \rrbracket"']
\end{tikzcd} \tag{8.10}$$

Recall from Section 7.3.2 that $\textbf{lift}\,\llbracket M_{\Phi,\Phi'}\rrbracket = p((\llbracket M_{\Phi,\Phi'}\rrbracket \circ \rho^{-1}_{\Phi,\Phi'} \circ \tau^{-1}_{\Phi,\Phi'})^{\circ})$ and $\textbf{lift}\,\llbracket M_{\Phi'}\rrbracket = p((\llbracket M_{\Phi'}\rrbracket \circ \rho^{-1}_{\Phi'} \circ \tau^{-1}_{\Phi'})^{\circ})$. Consider the following diagram:

$$\begin{tikzcd}
p(\llbracket \Phi, \Phi' \rrbracket_p) \arrow[rrr, "\textbf{lift}\,\llbracket M_{\Phi,\Phi'}\rrbracket"] \arrow[dr, "p(\eta_{\Phi,\Phi'})"] \arrow[ddd, "p(\pi_{\Phi'})"'] & & & !\llbracket A \rrbracket \arrow[ddd, "\mathrm{id}"] \\
& !p(\llbracket \Phi, \Phi' \rrbracket_p) \arrow[r, "!\rho^{-1}_{\Phi,\Phi'} \circ !\tau^{-1}_{\Phi,\Phi'}"] \arrow[d, "!p(\pi_{\Phi'})"'] & !\llbracket \Phi, \Phi'; \emptyset \rrbracket \arrow[ur, "!\llbracket M_{\Phi,\Phi'}\rrbracket"] \arrow[d, "!(\lambda_{\Phi';\emptyset} \circ (\diamond_{\Phi} \otimes \mathrm{id}) \circ \cong)"] & \\
& !p(\llbracket \Phi' \rrbracket_p) \arrow[r, "!\rho^{-1}_{\Phi'} \circ !\tau^{-1}_{\Phi'}"'] & !(\llbracket \Phi' \rrbracket \otimes \llbracket I \rrbracket) \arrow[dr, "!\llbracket M_{\Phi}\rrbracket"] & \\
p(\llbracket \Phi' \rrbracket_p) \arrow[ur, "p(\eta_{\Phi'})"] \arrow[rrr, "\textbf{lift}\,\llbracket M_{\Phi'}\rrbracket"'] & & & !\llbracket A \rrbracket.
\end{tikzcd}$$

(8.11)

The top diagram commutes by (8.8), while the leftmost diagram does by naturality of the unit η of the adjunction $p \dashv \flat$ and the functoriality of p. The center diagram commutes by (8.7), while the one to its right does by (8.9). The bottom diagram commutes by (8.10). Thus, the outer square commutes as well.

Finally, consider the following diagram:

$$\begin{tikzcd}
\llbracket \Phi, \Phi'; \emptyset \rrbracket \arrow[rrr, "\llbracket \Phi,\Phi';\emptyset \vdash \mathrm{lift}\ M : !A \rrbracket"] \arrow[dr, "\tau_{\Phi,\Phi'} \circ \rho_{\Phi,\Phi'}"'] \arrow[ddd, "\cong"'] & & & \llbracket !A \rrbracket \\
& p(\llbracket \Phi, \Phi' \rrbracket_p) \arrow[urr, "\textbf{lift}\,\llbracket M_{\Phi,\Phi'}\rrbracket"] \arrow[d, "p(\pi_{\Phi'})"'] & & \\
& p(\llbracket \Phi' \rrbracket_p) \arrow[uurr, "\textbf{lift}\,\llbracket M_{\Phi'}\rrbracket"'] & & \llbracket \Phi'; \emptyset \rrbracket \arrow[ll, "\tau_{\Phi'} \circ \rho_{\Phi'}"] \arrow[uu, "\llbracket \Phi';\emptyset \vdash \mathrm{lift}\ M : !A \rrbracket"'] \\
\llbracket \Phi \rrbracket \otimes \llbracket \Phi'; \emptyset \rrbracket \arrow[rrr, "\diamond_{\Phi} \otimes \mathrm{id}"'] & & & \llbracket I \rrbracket \otimes \llbracket \Phi'; \emptyset \rrbracket. \arrow[u, "\lambda_{\Phi';\emptyset}"']
\end{tikzcd}$$

The top triangle commutes by definition of $\llbracket \Phi, \Phi'; \emptyset \vdash \text{lift } M : !A \rrbracket$, while the one to its right does by (8.11). Similarly, the rightmost triangle commutes by definition of $\llbracket \Phi'; \emptyset \vdash \text{lift } M : !A \rrbracket$, while the bottom region does by (8.6). The commutativity of the outer square now follows.

- The remaining cases are similar.

□

8.4 Duplicating and Discarding Contexts

Lemma 8.4.1. *If Φ is a parameter context and Γ is a variable context for which there exist parameter contexts Φ' and Φ'' and variable context Γ', all mutually disjoint, such that $\Phi = (\Phi', \Phi'')$, while $\Gamma = (\Phi'', \Gamma')$ and $\Phi \cup \Gamma = (\Phi', \Phi'', \Gamma') = (\Phi', \Gamma)$, then the following diagram commutes:*

$$\begin{tikzcd}
\llbracket \Phi \cup \Gamma; Q \rrbracket \arrow[r, "\triangle_{\Phi\cup\Gamma;Q}"] \arrow[dd, "\cong"'] & \llbracket \Phi; \emptyset \rrbracket \otimes \llbracket \Gamma; Q \rrbracket \arrow[d, "\rho_\Phi \otimes \mathrm{id}"] \\
& \llbracket \Phi \rrbracket \otimes \llbracket \Gamma; Q \rrbracket \arrow[d, "\diamond_\Phi \otimes \mathrm{id}"] \\
\llbracket \Phi' \rrbracket \otimes \llbracket \Gamma; Q \rrbracket \arrow[r, "\diamond_{\Phi'} \otimes \mathrm{id}"'] & \llbracket I \rrbracket \otimes \llbracket \Gamma; Q \rrbracket .
\end{tikzcd}$$

Proof. First, note that in **Set** the following diagram commutes:

$$\begin{tikzcd}
& \llbracket \Phi'' \rrbracket_p \times \llbracket \Phi'' \rrbracket_p \arrow[r, "\sigma_{\Phi'',\Phi''}"] \arrow[d, "\pi_2"] & \llbracket \Phi'' \rrbracket_p \times \llbracket \Phi'' \rrbracket_p \arrow[r, "\mathrm{id} \times \diamond_{\Phi''}"] \arrow[d, "\pi_1"] & \llbracket \Phi'' \rrbracket_p \times \llbracket I' \rrbracket_p \arrow[d, "\overline{\pi}_1"] \\
\llbracket \Phi'' \rrbracket_p \arrow[ur, "\triangle^p_{\Phi''} = \langle \mathrm{id}, \mathrm{id} \rangle"] \arrow[r, "\mathrm{id}"'] & \llbracket \Phi'' \rrbracket_p \arrow[r, "\mathrm{id}"'] & \llbracket \Phi'' \rrbracket_p \arrow[r, "\mathrm{id}"'] & \llbracket \Phi'' \rrbracket_p
\end{tikzcd} \tag{8.12}$$

Now consider the following diagram:

$$\begin{tikzcd}
\llbracket \Phi'' \rrbracket \otimes \llbracket \Phi'' \rrbracket \arrow[rr, "\diamond_{\Phi''} \otimes \mathrm{id}"] & & \llbracket I \rrbracket \otimes \llbracket \Phi'' \rrbracket \arrow[d, "\tau_I \otimes \tau_{\Phi''}"] \\
p(\llbracket \Phi'' \rrbracket_p) \otimes p(\llbracket \Phi'' \rrbracket_p) \arrow[u, "\tau^{-1}_{\Phi''} \otimes \tau^{-1}_{\Phi''}"] \arrow[rr, "p(\diamond^p_{\Phi''}) \otimes \mathrm{id}"] & & p(\llbracket I \rrbracket_p) \otimes p(\llbracket \Phi'' \rrbracket_p) \arrow[dl, "m_{I,\Phi''}"'] \arrow[d, "\sigma_{I,\Phi''}"] \\
& p(\llbracket I \rrbracket_p \times \llbracket \Phi'' \rrbracket_p) \arrow[dr, "p(\sigma_{I,\Phi''})"] & p(\llbracket \Phi'' \rrbracket_p) \otimes p(\llbracket I \rrbracket_p) \arrow[d, "m_{\Phi'',I}"] \\
p(\llbracket \Phi'' \rrbracket_p \times \llbracket \Phi'' \rrbracket_p) \arrow[uu, "m^{-1}_{\Phi'',\Phi''}"] \arrow[ur, "p(\diamond^p_{\Phi''} \times \mathrm{id})"] \arrow[r, "p(\sigma_{\Phi'',\Phi''})"'] & p(\llbracket \Phi'' \rrbracket_p \times \llbracket \Phi'' \rrbracket_p) \arrow[r, "p(\mathrm{id} \times \diamond^p_{\Phi''})"'] & p(\llbracket \Phi'' \rrbracket_p \times \llbracket I \rrbracket_p) \arrow[d, "p(\overline{\pi}_1)"] \\
p(\llbracket \Phi'' \rrbracket_p) \arrow[u, "p(\triangle^p_{\Phi''})"] \arrow[rr, "p(\mathrm{id}) = \mathrm{id}"'] & & p(\llbracket \Phi'' \rrbracket_p)
\end{tikzcd} \tag{8.13}$$

Note that the top diagram commutes by definition of $\diamond_{\Phi''}$ and functoriality of $- \otimes \mathrm{id}$, while the one below it does by naturality of m. The rightmost diagram commutes because p is a monoidal functor. By (8.12), the bottom diagram commutes, while the one above it does by naturality of σ and functoriality of p.

The commutativity of the following diagram will be useful:

$$\begin{tikzcd}
p(\llbracket I \rrbracket_p) \otimes p(\llbracket \Phi'' \rrbracket_p) \arrow[d, "\sigma_{I,\Phi''}"'] \arrow[rrr, "\tau_I^{-1} \otimes \tau^{-1}_{\Phi''}"] & & & \llbracket I \rrbracket \otimes \llbracket \Phi'' \rrbracket \arrow[dl, "\sigma_{I,\Phi''}"] \arrow[dd, "\lambda_{\Phi''}"] \\
p(\llbracket \Phi'' \rrbracket_p) \otimes p(\llbracket I \rrbracket_p) \arrow[d, "m_{\Phi'',I}"'] \arrow[r, "\mathrm{id} \otimes \tau_I^{-1}"] & p(\llbracket \Phi'' \rrbracket_p) \otimes \llbracket I \rrbracket \arrow[d, "\rho_{\Phi''}"] \arrow[r, "\tau^{-1}_{\Phi''} \otimes \mathrm{id}"] & \llbracket \Phi'' \rrbracket \otimes \llbracket I \rrbracket \arrow[dr, "\rho_{\Phi''}"] & \\
p(\llbracket \Phi'' \rrbracket_p \times \llbracket I \rrbracket_p) \arrow[r, "p(\overline{\pi}_1)"'] & p(\llbracket \Phi'' \rrbracket_p) \arrow[rr, "\tau^{-1}_{\Phi''}"'] & & \llbracket \Phi'' \rrbracket
\end{tikzcd} \tag{8.14}$$

The top region commutes by naturality of σ, while the one on the right does by coherence for symmetric monoidal categories. The bottom left diagram commutes by monoidality of p, while the one to its right does by naturality of ρ.

Now we can show that the following diagram commutes as well:

$$\begin{tikzcd}
\llbracket \Phi'' \rrbracket \arrow[r, "\vartriangle_{\Phi''}"] \arrow[dddd, "\tau_{\Phi''}"'] & \llbracket \Phi'' \rrbracket \otimes \llbracket \Phi'' \rrbracket \arrow[rr, "\diamond_{\Phi''} \otimes \mathrm{id}"] & & \llbracket I \rrbracket \otimes \llbracket \Phi'' \rrbracket \arrow[dl, "\tau_I \otimes \tau_{\Phi''}"] \arrow[dddd, "\lambda_{\Phi''}"] \\
& p(\llbracket \Phi'' \rrbracket_p) \otimes p(\llbracket \Phi'' \rrbracket_p) \arrow[u, "\tau_{\Phi''}^{-1} \otimes \tau_{\Phi''}^{-1}"] & p(\llbracket I \rrbracket_p) \otimes p(\llbracket \Phi'' \rrbracket_p) \arrow[d, "\sigma_{I,\Phi''}"] & \\
& & p(\llbracket \Phi'' \rrbracket_p) \otimes p(\llbracket I \rrbracket_p) \arrow[d, "m_{\Phi'',I}"] & \\
& p(\llbracket \Phi'' \rrbracket_p \times \llbracket \Phi'' \rrbracket_p) \arrow[uu, "m_{\Phi'',\Phi''}^{-1}"] & p(\llbracket \Phi'' \rrbracket_p \times \llbracket I \rrbracket_p) \arrow[d, "p(\overline{\pi}_1)"] & \\
p(\llbracket \Phi'' \rrbracket_p) \arrow[ur, "p(\vartriangle_{\Phi''}^p)"] \arrow[rr, "\mathrm{id}"'] & & p(\llbracket \Phi'' \rrbracket_p) \arrow[r, "\tau_{\Phi''}^{-1}"'] & \llbracket \Phi'' \rrbracket
\end{tikzcd}$$

(8.15)

The left diagram commutes by definition of $\vartriangle_{\Phi''}$, while the middle one does by (8.13). The right diagram commutes by (8.14).

Similarly, consider the following:

$$\begin{tikzcd}
\llbracket \Phi \cup \Gamma \rrbracket \arrow[rrr, "\vartriangle_{\Phi \cup \Gamma}"] \arrow[dr, "\cong"] \arrow[ddd, "\cong"'] & & & \llbracket \Phi \rrbracket \otimes \llbracket \Gamma \rrbracket \arrow[dl, "\cong"] \arrow[ddd, "\diamond_{\Phi} \otimes \mathrm{id}"] \\
& \llbracket \Phi' \rrbracket \otimes \llbracket \Phi'' \rrbracket \otimes \llbracket \Gamma' \rrbracket \arrow[r, "\mathrm{id} \otimes \vartriangle_{\Phi''} \otimes \mathrm{id}"] \arrow[d, "\mathrm{id}"'] & \llbracket \Phi' \rrbracket \otimes \llbracket \Phi'' \rrbracket \otimes \llbracket \Phi'' \rrbracket \otimes \llbracket \Gamma' \rrbracket \arrow[d, "\mathrm{id} \otimes \diamond_{\Phi''} \otimes \mathrm{id} \otimes \mathrm{id}"] & \\
& \llbracket \Phi' \rrbracket \otimes \llbracket \Phi'' \rrbracket \otimes \llbracket \Gamma' \rrbracket \arrow[dl, "\cong"] & \llbracket \Phi' \rrbracket \otimes \llbracket I \rrbracket \otimes \llbracket \Phi'' \rrbracket \otimes \llbracket \Gamma' \rrbracket \arrow[l, "\mathrm{id} \otimes \lambda_{\Phi''} \otimes \mathrm{id}"] & \\
\llbracket \Phi' \rrbracket \otimes \llbracket \Gamma \rrbracket \arrow[rrr, "\diamond_{\Phi'} \otimes \mathrm{id}"'] & & & \llbracket I \rrbracket \otimes \llbracket \Gamma \rrbracket
\end{tikzcd}$$

(8.16)

The top diagram commutes by definition of $\vartriangle_{\Phi \cup \Gamma}$, while the one on the right does by Lemma 8.2.1 and the functoriality of $- \otimes \mathrm{id}$ and $\mathrm{id} \otimes -$. The left diagram commutes by coherence for symmetric monoidal categories, while the one in the center does by (8.15) and the functoriality of $- \otimes \mathrm{id}$ and $\mathrm{id} \otimes -$.

Our desired result can now be easily established:

$$\begin{tikzcd}
\llbracket \Phi \cup \Gamma ; Q \rrbracket \arrow[rr, "\vartriangle_{\Phi\cup\Gamma;Q}"] \arrow[dr, bend right, "\vartriangle_{\Phi\cup\Gamma}\otimes \mathrm{id}"] \arrow[dd, "\cong"'] & & \llbracket \Phi ; \emptyset \rrbracket \otimes \llbracket \Gamma ; Q \rrbracket \arrow[d, "\rho_\Phi \otimes \mathrm{id}"] \\
& \llbracket \Phi \rrbracket \otimes \llbracket \Gamma \rrbracket \otimes \llbracket Q \rrbracket \arrow[ur, bend right, "\rho_\Phi^{-1}\otimes \mathrm{id}"] \arrow[dr, bend right, "\diamond_\Phi \otimes \mathrm{id}"] & \llbracket \Phi \rrbracket \otimes \llbracket \Gamma ; Q \rrbracket \arrow[d, "\diamond_\Phi \otimes \mathrm{id}"] \\
\llbracket \Phi' \rrbracket \otimes \llbracket \Gamma ; Q \rrbracket \arrow[rr, "\diamond_{\Phi'}\otimes \mathrm{id}"'] & & \llbracket I \rrbracket \otimes \llbracket \Gamma ; Q \rrbracket
\end{tikzcd} \tag{8.17}$$

Note that the top diagram commutes by definition of $\vartriangle_{\Phi\cup\Gamma;Q}$, while the one at the bottom does by (8.16) and the functoriality of $-\otimes \mathrm{id}$. The commutativity of the right diagram is immediate. The result now follows. □

8.5 Semantic Weakening Lemma

The meaning of a term remains essentially unchanged when its variable context is extended with a parameter context:

Lemma 8.5.1 (Semantic Weakening Lemma). *If $\Gamma ; Q \vdash N : B$ is a valid typing judgement with derivation $\mathcal{D}$, and Φ is a parameter context such that $\Phi \cup \mathcal{D}$ is defined, then the valid typing judgement $\Phi \cup \Gamma ; Q \vdash N : B$ with derivation $\Phi \cup \mathcal{D}$ makes the following diagram commute:*

$$\begin{tikzcd}
\llbracket \Phi \cup \Gamma ; Q \rrbracket \arrow[r, "\llbracket \Phi\cup\Gamma;Q\vdash N:B \rrbracket"] \arrow[d, "\vartriangle_{\Phi\cup\Gamma;Q}"'] & \llbracket B \rrbracket \\
\llbracket \Phi ; \emptyset \rrbracket \otimes \llbracket \Gamma ; Q \rrbracket \arrow[d, "\rho_\Phi \otimes \mathrm{id}"'] & \llbracket \Gamma ; Q \rrbracket \arrow[u, "\llbracket \Gamma;Q\vdash N:B \rrbracket"'] \\
\llbracket \Phi \rrbracket \otimes \llbracket \Gamma ; Q \rrbracket \arrow[r, "\diamond_\Phi \otimes \mathrm{id}"'] & \llbracket I \rrbracket \otimes \llbracket \Gamma ; Q \rrbracket , \arrow[u, "\lambda_{\Gamma;Q}"']
\end{tikzcd}$$

where $\llbracket \Gamma ; Q \vdash N : B \rrbracket = \llbracket \mathcal{D} \rrbracket$ and $\llbracket \Phi \cup \Gamma ; Q \vdash N : B \rrbracket = \llbracket \Phi \cup \mathcal{D} \rrbracket$.

Proof. Let $\Gamma ; Q \vdash N : B$ be a valid typing judgement with derivation $\mathcal{D}$, and Φ a parameter context such that $\Phi \cup \mathcal{D}$ is defined. This implies that $\Phi \cup \Gamma ; Q \vdash N : B$ is valid typing judgement with derivation $\Phi', \mathcal{D}$ where $\Phi' = \Phi \backslash \Gamma$. Consider the following diagram:

$$\begin{tikzcd}
\llbracket \Phi \cup \Gamma; Q \rrbracket \arrow[rr, "\llbracket \Phi\cup\Gamma;Q\vdash N:B \rrbracket"] \arrow[d, "\triangle_{\Phi\cup\Gamma;Q}"'] \arrow[dr, "\cong"] & & \llbracket B \rrbracket \\
\llbracket \Phi; \emptyset \rrbracket \otimes \llbracket \Gamma; Q \rrbracket \arrow[d, "\rho_\Phi \otimes \mathrm{id}"'] & \llbracket \Phi' \rrbracket \otimes \llbracket \Gamma; Q \rrbracket \arrow[dr, "\diamond_{\Phi'} \otimes \mathrm{id}"] & \llbracket \Gamma; Q \rrbracket \arrow[u, "\llbracket \Gamma;Q\vdash N:B \rrbracket"'] \\
\llbracket \Phi \rrbracket \otimes \llbracket \Gamma; Q \rrbracket \arrow[rr, "\diamond_\Phi \otimes \mathrm{id}"'] & & \llbracket I \rrbracket \otimes \llbracket \Gamma; Q \rrbracket \arrow[u, "\lambda_{\Gamma;Q}"']
\end{tikzcd}$$

The top triangle commutes by the Special Semantic Weakening Lemma 8.3.2, while the one at the bottom does by Lemma 8.4.1. Thus, the outer square commutes as well.

□

8.6 Semantic Substitution Lemma

We now have all the ingredients to prove the semantic version of the syntactic Substitution Lemma 6.3.1

Lemma 8.6.1 (Semantic Substitution Lemma). *If* $\Gamma, x : A; Q \vdash N : B$ *and* $\Gamma'; Q' \vdash V : A$ *are valid typing judgements with derivations* $\mathcal{D}$ *and* $\mathcal{D}'$*, respectively, such that* $\Gamma' \cup \mathcal{D}$ *and* $\Gamma \cup \mathcal{D}'$ *are defined,* $Q \cap Q' = \emptyset$*, and* V *is a value term, then*

$$\llbracket \Gamma \cup \Gamma'; Q, Q' \vdash \text{let } x = V \text{ in } N : B \rrbracket = \llbracket \Gamma \cup \Gamma'; Q, Q' \vdash N[V/x] : B \rrbracket.$$

Proof. By induction on the typing derivation $\mathcal{D}$ of $\Gamma, x : A; Q \vdash N : B$. Since $\Gamma' \cup \mathcal{D}$ is defined, so is the union of contexts $\Gamma' \cup \Gamma$, and since $Q \cap Q' = \emptyset$, the typing judgment $\Gamma \cup \Gamma'; Q, Q' \vdash \text{let } x = V \text{ in } N : B$ is valid by the (*let*) rule. Thus, $\llbracket \Gamma \cup \Gamma'; Q, Q' \vdash \text{let } x = V \text{ in } N : B \rrbracket$ is defined. Similarly, by the syntactic Substitution Lemma 6.3.1, $\Gamma \cup \Gamma'; Q, Q' \vdash N[V/x] : B$ is valid, and so $\llbracket \Gamma \cup \Gamma'; Q, Q' \vdash N[V/x] : B \rrbracket$ is defined. We want to show that

$$\llbracket \Gamma \cup \Gamma'; Q, Q' \vdash \text{let } x = V \text{ in } N : B \rrbracket = \llbracket \Gamma \cup \Gamma'; Q, Q' \vdash N[V/x] : B \rrbracket.$$

We now proceed by case distinction:

- If the last (and only) rule of $\mathcal{D}$ is (var) with $N = x$, then $\mathcal{D}$ is

$$\frac{}{\Phi, x : A; \emptyset \vdash x : A}\ (var)$$

with $\Gamma = \Phi$, a parameter context, $Q = \emptyset$, and $B = A$. Since $N[V/x] = x[V/x] = V$, we want to show that

$$\llbracket \Phi \cup \Gamma'; Q' \vdash \text{let } x = V \text{ in } x : A \rrbracket = \llbracket \Phi \cup \Gamma'; Q' \vdash V : A \rrbracket.$$

Consider the following diagram:

$$\begin{tikzcd}
\llbracket \Phi \cup \Gamma'; Q' \rrbracket \arrow[rrr, "\llbracket \Phi\cup\Gamma';Q'\vdash \text{let } x=V \text{in} x:A \rrbracket"] \arrow[d, "\triangle_{\Phi\cup\Gamma';Q'}"'] & & & \llbracket A \rrbracket \\
\llbracket \Phi; \emptyset \rrbracket \otimes \llbracket \Gamma'; Q' \rrbracket \arrow[r, "\mathrm{id}\otimes\llbracket V \rrbracket"] \arrow[dd, "\rho_\Phi\otimes\mathrm{id}"'] & \llbracket \Phi; \emptyset \rrbracket \otimes \llbracket A \rrbracket \arrow[r, "\cong"] \arrow[d, "\rho_\Phi\otimes\mathrm{id}"] & \llbracket \Phi, x : A; \emptyset \rrbracket \arrow[ur, bend right, "\llbracket \Phi,x:A;\emptyset\vdash x:A \rrbracket"'] & \\
& \llbracket \Phi \rrbracket \otimes \llbracket A \rrbracket \arrow[r, "\diamond_\Phi\otimes\mathrm{id}"] & \llbracket I \rrbracket \otimes \llbracket A \rrbracket \arrow[uur, bend right, "\lambda_A"] & \llbracket \Gamma'; Q' \rrbracket \arrow[uu, "\llbracket \Gamma';Q'\vdash V:A \rrbracket"'] \\
\llbracket \Phi \rrbracket \otimes \llbracket \Gamma'; Q' \rrbracket \arrow[rrr, "\diamond_\Phi\otimes\mathrm{id}"'] & & & \llbracket I \rrbracket \otimes \llbracket \Gamma'; Q' \rrbracket \arrow[u, "\lambda_{\Gamma';Q'}"'] \arrow[ul, "\mathrm{id}\otimes\llbracket V \rrbracket"]
\end{tikzcd} \tag{8.18}$$

The top region commutes by definition of $\llbracket \Phi \cup \Gamma'; Q' \vdash \text{let } x = V \text{ in } x : A \rrbracket$, while the one to its right does by definition of $\llbracket \Phi, x : A; \emptyset \vdash x : A \rrbracket$. The bottom diagram commutes by functoriality of $\otimes$, while the rightmost diagram does by naturality of λ. Hence, the outer square commutes as well.

Since $\Phi \cup \mathcal{D}'$ is defined, the Semantic Weakening Lemma 8.5.1 implies that the following diagram commutes:

$$\begin{tikzcd}
\llbracket \Phi \cup \Gamma'; Q' \rrbracket \arrow[rr, "\llbracket \Phi\cup\Gamma';Q'\vdash V:A \rrbracket"] \arrow[d, "\dot{\triangle}_{\Phi\cup\Gamma';Q'}"'] & & \llbracket A \rrbracket \\
\llbracket \Phi; \emptyset \rrbracket \otimes \llbracket \Gamma'; Q' \rrbracket \arrow[d, "\rho_{\Phi'}\otimes\mathrm{id}"'] & & \llbracket \Gamma'; Q' \rrbracket \arrow[u, "\llbracket \Gamma';Q'\vdash V:A \rrbracket"'] \\
\llbracket \Phi \rrbracket \otimes \llbracket \Gamma'; Q' \rrbracket \arrow[rr, "\diamond_{\Phi}\otimes\mathrm{id}"'] & & \llbracket I \rrbracket \otimes \llbracket \Gamma'; Q' \rrbracket. \arrow[u, "\lambda_{\Gamma';Q'}"']
\end{tikzcd} \tag{8.19}$$

Since the bottom paths of (8.18) and (8.19) are equal, it follows that the top morphisms of those diagrams are equal as well. Hence,

$$\llbracket \Phi \cup \Gamma'; Q' \vdash \text{let } x = V \text{ in } x : A \rrbracket = \llbracket \Phi \cup \Gamma'; Q' \vdash V : A \rrbracket,$$

as required.

- If the last (and only) rule of $\mathcal{D}$ is (var) with $N = y \neq x$, then $\mathcal{D}$ is

$$\frac{}{\Phi, x : A, y : B; \emptyset \vdash y : B}\ (var)$$

with $\Gamma = (\Phi, y : B)$ and $Q = \emptyset$, where $\Phi, x : A$ is a parameter context. Thus, V is a value of parameter type, and so the syntactic Parameter Value Lemma 6.1.4 implies that Γ' is a parameter context and $Q' = \emptyset$. Since $N[V/x] = y[V/x] = y$, we want to show that

$$\llbracket (\Phi, y : B) \cup \Gamma'; \emptyset \vdash \text{let } x = V \text{ in } y : B \rrbracket = \llbracket (\Phi, y : B) \cup \Gamma'; \emptyset \vdash y : B \rrbracket.$$

Consider the following diagram:

$$\begin{tikzcd}
\llbracket (\Phi, y : B) \cup \Gamma'; \emptyset \rrbracket \arrow[rrr, "\llbracket (\Phi,y:B)\cup\Gamma';\emptyset\vdash \mathrm{let}\ x=V \mathrm{in} y:B \rrbracket"] \arrow[d, "\triangle_{(\Phi,y:B)\cup\Gamma';\emptyset}"'] & & & \llbracket B \rrbracket \\
\llbracket \Phi, y : B; \emptyset \rrbracket \otimes \llbracket \Gamma'; \emptyset \rrbracket \arrow[r, "\mathrm{id}\otimes\llbracket V \rrbracket"] \arrow[d, "\mathrm{id}\otimes\rho_{\Gamma'}"'] & \llbracket \Phi, y : B; \emptyset \rrbracket \otimes \llbracket A \rrbracket \arrow[r, "\cong"] \arrow[d, "\mathrm{id}\otimes\tau_A"] \arrow[rrd, "\mathrm{id}\otimes\Diamond_A"] & \llbracket \Phi, x : A, y : B; \emptyset \rrbracket \arrow[ru, bend right, "\llbracket \Phi,x:A;y:B;\emptyset\vdash y:B \rrbracket"] & \llbracket \Phi, y : B; \emptyset \rrbracket \arrow[u, "\llbracket \Phi,y:B;\emptyset\vdash y:B \rrbracket"'] \\
\llbracket \Phi, y : B; \emptyset \rrbracket \otimes \llbracket \Gamma' \rrbracket \arrow[d, "\mathrm{id}\otimes\tau_{\Gamma'}"'] & \llbracket \Phi, y : B; \emptyset \rrbracket \otimes p(\llbracket A \rrbracket_p) \arrow[rrd, "\mathrm{id}\otimes p(\Diamond^p_A)"] & & \llbracket \Phi, y : B; \emptyset \rrbracket \otimes \llbracket I \rrbracket \arrow[u, "\rho_{\Gamma;\emptyset}"'] \\
\llbracket \Phi, y : B; \emptyset \rrbracket \otimes p(\llbracket \Gamma' \rrbracket_p) \arrow[ru, bend right, "\mathrm{id}\otimes p(\llbracket V \rrbracket_p)"] \arrow[rrr, "\mathrm{id}\otimes p(\Diamond^p_{\Gamma'})"'] & & & \llbracket \Phi, y : B; \emptyset \rrbracket \otimes p(\llbracket I \rrbracket_p) \arrow[u, "\mathrm{id}\otimes\tau_I"']
\end{tikzcd}$$

(8.20)

The top region commutes by definition of $\llbracket (\Phi, y : B) \cup \Gamma'; \emptyset \vdash \mathrm{let}\ x = V\ \mathrm{in}\ y : B \rrbracket$, while the one to its right does by the Special Semantic Weakening Lemma 8.3.2. The diagram below the latter commutes by definition $\Diamond_A$ and the functoriality of id $\otimes$ $-$, while the leftmost diagram does by the Semantic Parameter Value Lemma 8.1.2 and the functoriality of id $\otimes$ $-$. The bottom diagram commutes by definition of $\Diamond^p_{\Gamma'}$ and the functoriality of p and id $\otimes$ $-$. Hence, the outer square commutes as well.

Now consider the diagram:

$$\begin{tikzcd}
\llbracket (\Phi, y : B) \cup \Gamma'; \emptyset \rrbracket \arrow[r, "\llbracket (\Phi,y:B)\cup\Gamma';\emptyset\vdash y:B \rrbracket"] \arrow[d, "\triangle_{(\Phi,y:B)\cup\Gamma';\emptyset}"'] & \llbracket B \rrbracket \\
\llbracket \Phi, y : B; \emptyset \rrbracket \otimes \llbracket \Gamma'; \emptyset \rrbracket \arrow[d, "\mathrm{id}\otimes\rho_{\Gamma'}"'] & \llbracket \Phi, y : B; \emptyset \rrbracket \arrow[u, "\llbracket \Phi,y:B;\emptyset\vdash y:B \rrbracket"'] \\
\llbracket \Phi, y : B; \emptyset \rrbracket \otimes \llbracket \Gamma' \rrbracket \arrow[r, "\mathrm{id}\otimes\Diamond_{\Gamma'}"'] \arrow[d, "\mathrm{id}\otimes\tau_{\Gamma'}"'] & \llbracket \Phi, y : B; \emptyset \rrbracket \otimes \llbracket I \rrbracket, \arrow[u, "\rho_{\Phi,y:B;\emptyset}"'] \\
\llbracket \Phi, y : B; \emptyset \rrbracket \otimes p(\llbracket \Gamma' \rrbracket_p) \arrow[r, "\mathrm{id}\otimes p(\Diamond^p_{\Gamma'})"'] & \llbracket \Phi, y : B; \emptyset \rrbracket \otimes p(\llbracket I \rrbracket_p), \arrow[u, "\mathrm{id}\otimes\tau_I"']
\end{tikzcd}$$

(8.21)

The top diagram commutes by the Semantic Weakening Lemma 8.5.1, while the one at the bottom does by definition of $\Diamond_{\Gamma'}$ and the functoriality of id $\otimes$ $-$.

Hence, the outer square commutes as well.

Since the bottom paths of (8.20) and (8.21) are equal, it follows that the top morphisms of those diagrams are equal as well. Hence,

$$\llbracket(\Phi, y : B) \cup \Gamma'; \emptyset \vdash \text{let } x = V \text{ in } y : B\rrbracket = \llbracket(\Phi, y : B) \cup \Gamma'; \emptyset \vdash y : B\rrbracket,$$

as required.

- The cases where the last rule of $\mathcal{D}$ is any of (*label*), (*const*), $(*)$, or (*circ*) are similar. We present the (*label*) case:

  If the last (and only) rule of $\mathcal{D}$ is (*label*) with $N = \ell$, then $\mathcal{D}$ is

$$\frac{}{\Phi; \ell : \alpha \vdash \ell : \alpha}\ (\textit{label})$$

  where $\Gamma, x : A = \Phi$, a parameter context, $Q = \ell : \alpha$, and $B = \alpha$, a wire type. Thus, V is a value of parameter type, and so the syntactic Parameter Value Lemma 6.1.4 implies that Γ' is a parameter context and $Q' = \emptyset$. Since $N[V/x] = \ell[V/x] = \ell$, we want to show that

$$\llbracket\Gamma \cup \Gamma'; \ell : \alpha \vdash \text{let } x = V \text{ in } \ell : \alpha\rrbracket = \llbracket\Gamma \cup \Gamma'; \ell : \alpha \vdash \ell : \alpha\rrbracket.$$

  Consider the following diagram:

$$\begin{tikzcd}
\llbracket\Gamma \cup \Gamma'; \ell : \alpha\rrbracket \arrow[rrr, "\llbracket\Gamma\cup\Gamma';\ell:\alpha\vdash\text{let } x=V\text{in}\ell:\alpha\rrbracket"] \arrow[d, "\triangle_{\Gamma\cup\Gamma';\ell:\alpha}"'] & & & \llbracket\alpha\rrbracket \\
\llbracket\Gamma; \ell : \alpha\rrbracket \otimes \llbracket\Gamma'; \emptyset\rrbracket \arrow[r, "\mathrm{id}\otimes\llbracket V\rrbracket"] \arrow[d, "\mathrm{id}\otimes\rho_{\Gamma'}"'] & \llbracket\Gamma; \ell : \alpha\rrbracket \otimes \llbracket A\rrbracket \arrow[r, "\cong"] \arrow[d, "\mathrm{id}\otimes\tau_A"] \arrow[rrd, "\mathrm{id}\otimes\diamond_A"] & \llbracket\Gamma, x : A; \ell : \alpha\rrbracket \arrow[ur, bend right, "\llbracket\Gamma,x:A;\ell:\alpha\vdash\ell:\alpha\rrbracket"] & \llbracket\Gamma; \ell : \alpha\rrbracket \arrow[u, "\llbracket\Gamma;\ell:\alpha\vdash\ell:\alpha\rrbracket"'] \\
\llbracket\Gamma; \ell : \alpha\rrbracket \otimes \llbracket\Gamma'\rrbracket \arrow[d, "\mathrm{id}\otimes\tau_{\Gamma'}"'] & \llbracket\Gamma; \ell : \alpha\rrbracket \otimes p(\llbracket A\rrbracket_p) \arrow[rrd, "\mathrm{id}\otimes p(\diamond^p_A)"] & & \llbracket\Gamma; \ell : \alpha\rrbracket \otimes \llbracket I\rrbracket \arrow[u, "\rho_{\Gamma;\ell:\alpha}"'] \\
\llbracket\Gamma; \ell : \alpha\rrbracket \otimes p(\llbracket\Gamma'\rrbracket_p) \arrow[ur, bend right, "\mathrm{id}\otimes p(\llbracket V\rrbracket_p)"] \arrow[rrr, "\mathrm{id}\otimes p(\diamond^p_{\Gamma'})"'] & & & \llbracket\Gamma; \ell : \alpha\rrbracket \otimes p(\llbracket I\rrbracket_p) \arrow[u, "\mathrm{id}\otimes\tau_I"']
\end{tikzcd}$$

(8.22)

  The top region commutes by definition of $\llbracket\Gamma \cup \Gamma'; \ell : \alpha \vdash \text{let } x = V \text{ in } \ell : \alpha\rrbracket$, while the one to its right does by the Special Semantic Weakening Lemma 8.3.2.

The diagram below the latter commutes by definition $\diamond_A$ and the functoriality of $\mathrm{id} \otimes -$, while the leftmost diagram does by the Semantic Parameter Value Lemma 8.1.2 and the functoriality of $\mathrm{id} \otimes -$. The bottom diagram commutes by definition of $\diamond^p_{\Gamma'}$ and the functoriality of p and $\mathrm{id} \otimes -$. Hence, the outer square commutes as well.

Now consider the diagram:

$$\begin{tikzcd}
\llbracket \Gamma \cup \Gamma'; \ell : \alpha \rrbracket \arrow[rr, "\llbracket \Gamma \cup \Gamma'; \ell:\alpha \vdash \ell:\alpha \rrbracket"] \arrow[d, "\vartriangle_{\Gamma \cup \Gamma'; \ell:\alpha}"'] & & \llbracket \alpha \rrbracket \\
\llbracket \Gamma; \ell : \alpha \rrbracket \otimes \llbracket \Gamma'; \emptyset \rrbracket \arrow[d, "\mathrm{id} \otimes \rho_{\Gamma'}"'] & & \llbracket \Gamma; \ell : \alpha \rrbracket \arrow[u, "\llbracket \Gamma; \ell:\alpha \vdash \ell:\alpha \rrbracket"'] \\
\llbracket \Gamma; \ell : \alpha \rrbracket \otimes \llbracket \Gamma' \rrbracket \arrow[rr, "\mathrm{id} \otimes \diamond_{\Gamma'}"'] \arrow[d, "\mathrm{id} \otimes \tau_{\Gamma'}"'] & & \llbracket \Gamma; \ell : \alpha \rrbracket \otimes \llbracket I \rrbracket, \arrow[u, "\rho_{\Gamma; \ell:\alpha}"'] \\
\llbracket \Gamma; \ell : \alpha \rrbracket \otimes p(\llbracket \Gamma' \rrbracket_p) \arrow[rr, "\mathrm{id} \otimes p(\diamond^p_{\Gamma'})"'] & & \llbracket \Gamma; \ell : \alpha \rrbracket \otimes p(\llbracket I \rrbracket_p), \arrow[u, "\mathrm{id} \otimes \tau_I"']
\end{tikzcd} \tag{8.23}$$

The top diagram commutes by the Semantic Weakening Lemma 8.5.1, while the one at the bottom does by definition of $\diamond_{\Gamma'}$ and the functoriality of $\mathrm{id} \otimes -$. Hence, the outer square commutes as well.

Since the bottom paths of (8.22) and (8.23) are equal, it follows that the top morphisms of those diagrams are equal as well. Hence,

$$\llbracket \Gamma \cup \Gamma'; \ell : \alpha \vdash \text{let } x = V \text{ in } \ell : \alpha \rrbracket = \llbracket \Gamma \cup \Gamma'; \ell : \alpha \vdash \ell : \alpha \rrbracket,$$

as required.

- The cases where the last rule of $\mathcal{D}$ is any of (*initial*), (*left*), (*right*), (*force*), or (*box*) are similar. We present the (*left*) case:

  If the last rule of $\mathcal{D}$ is (*left*) with $N = \mathrm{left}_{C,D}\, M$, then $\mathcal{D}$ is

$$\dfrac{\begin{array}{c}\vdots\\ \Gamma, x : A; Q \vdash M : C\end{array}}{\Gamma, x : A; Q \vdash \mathrm{left}_{C,D}\, M : C + D}\ (\mathit{left})$$

where $B = C + D$. Since $N[V/x] = (\text{left}_{C,D}\, M)[V/x] = \text{left}_{C,D}\, M[V/x]$, we want to show that the morphisms

$$\llbracket \Gamma \cup \Gamma'; Q, Q' \vdash \text{left}_{C,D}\, M[V/x] : C + D \rrbracket$$

and

$$\llbracket \Gamma \cup \Gamma'; Q, Q' \vdash \text{let } x = V \text{ in left}_{C,D}\, M : C + D \rrbracket$$

are equal.

Consider the following diagram:

$$\begin{tikzcd}
\llbracket \Gamma \cup \Gamma'; Q, Q' \rrbracket \arrow[rr, "\llbracket \Gamma\cup\Gamma'; Q,Q' \vdash \text{left}_{C,D}\, M[V/x] : C+D \rrbracket"] \arrow[d, swap, "\text{id}"] \arrow[dr, bend left, "\llbracket M[V/x] \rrbracket"] & & \llbracket C + D \rrbracket \\
\llbracket \Gamma \cup \Gamma'; Q, Q' \rrbracket \arrow[r, "\llbracket \text{let } x = V \text{in} M \rrbracket"] \arrow[d, swap, "\vartriangle_{\Gamma\cup\Gamma'; Q,Q'}"] & \llbracket C \rrbracket \arrow[ur, bend right, "\text{left}_{C,D}"] & \\
\llbracket \Gamma; Q \rrbracket \otimes \llbracket \Gamma'; Q' \rrbracket \arrow[r, swap, "\text{id} \otimes \llbracket V \rrbracket"] & \llbracket \Gamma; Q \rrbracket \otimes \llbracket A \rrbracket \arrow[r, swap, "\cong"] & \llbracket \Gamma, x : A; Q \rrbracket. \arrow[ul, swap, "\llbracket M \rrbracket"] \arrow[uu, swap, "\llbracket \Gamma, x:A; Q \vdash \text{left}_{C,D}\, M \rrbracket"]
\end{tikzcd}$$

(8.24)

The top left diagram commutes by the induction hypothesis, while the top diagram does by definition of $\llbracket \Gamma \cup \Gamma'; Q, Q' \vdash \text{left}_{C,D}\, M[V/x] : C + D \rrbracket$. The rightmost diagram commutes by definition of $\llbracket \Gamma, x : A; Q \vdash \text{left}_{C,D}\, M : C + D \rrbracket$, while the one at the bottom does by definition of $\llbracket \Gamma \cup \Gamma'; Q, Q' \vdash \text{let } x = V \text{ in } M : C \rrbracket$. Hence, the outer square commutes as well. By definition, the bottom path of (8.24) is equal to $\llbracket \Gamma \cup \Gamma'; Q, Q' \vdash \text{let } x = V \text{ in left}_{C,D}\, M : C + D \rrbracket$, and so

$$\llbracket \Gamma \cup \Gamma'; Q, Q' \vdash \text{left}_{C,D}\, M[V/x] : C + D \rrbracket$$

and

$$\llbracket \Gamma \cup \Gamma'; Q, Q' \vdash \text{let } x = V \text{ in left}_{C,D}\, M : C + D \rrbracket$$

are equal, as required.

- The remaining cases are similar.

□

Chapter 9

Soundness

In this chapter, we prove the last main result of this thesis, namely, the *Soundness Theorem* for Proto-Quipper-M. As we saw before, the safety properties of the language relate its operational semantics to its typing system. Here, we will see how the soundness theorem extends this relationship to include the categorical semantics. Indeed, we can think of the soundness theorem as a *correctness* criterion for the operational semantics with respect to the categorical semantics in the sense that, if a well-typed configuration evaluates to a value configuration, then their meanings must be the same.

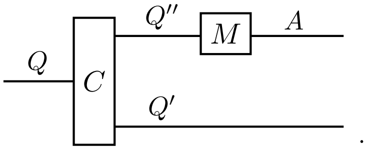
9.1 Categorical Semantics of Configurations

Since the operational semantics of Proto-Quipper-M is defined on configurations, while the categorical semantics is defined on well-typed terms, we first extend the categorical semantics of Proto-Quipper-M from well-typed terms to well-typed configurations:

Definition 9.1.1. Let $Q \vdash (C, M) : A; Q'$ be a well-typed configuration. We define its semantics to be a morphism $[\![(C, M)]\!] : [\![Q]\!] \to [\![A]\!] \otimes [\![Q']\!]$ of the category $\mathbf{L}$ as follows. Since (C, M) is well-typed, there exists a unique label context Q'' such that $C : Q \to Q'', Q'$ is a morphism in $\mathbf{M}_{\mathcal{L}}$ and $\emptyset; Q'' \vdash M : A$ is a valid typing judgement, and so we can define

$$[\![(C, M)]\!] = [\![Q]\!] \xrightarrow{[\![C]\!]} [\![Q'', Q']\!] \xrightarrow{\cong} [\![\emptyset; Q'']\!] \otimes [\![Q']\!] \xrightarrow{[\![M]\!] \otimes \mathrm{id}} [\![A]\!] \otimes [\![Q']\!],$$

where the morphism $[\![C]\!] : [\![Q]\!] \to [\![Q'', Q']\!]$ is the semantics of $C : Q \to Q'', Q'$ in $\mathbf{L}$ as defined in (7.13). Using string diagram notation,

$[\![(C, M)]\!] =$ 

.

9.2 The Soundness Theorem for Proto-Quipper-M

As is usual in soundness proofs, we verify for each rule of the operational semantics that an appropriate diagram commutes in the category of denotations. However, as the proof progresses, we emphasize the use of string diagrams, as opposed to commutative diagrams, not only to illustrate the circuit description nature of Proto-Quipper-M but also to make the arguments more transparent. The proof of the soundness theorem for Proto-Quipper-M is indeed lengthy due to the numerous rules involved. However, these rules naturally cluster in four main groups according to their structure: axiom rules, purely inductive rules, substitution rules, and circuit rules. The proofs for the cases within a group are similar. However, for the sake of completeness, we treat each case in turn.

Theorem 9.2.1 (Soundness). *If* $Q \vdash (C, M) : A; Q'$ *is a well-typed configuration and* $(C, M) \Downarrow (C', V)$*, then* $[\![(C, M)]\!] = [\![(C', V)]\!] : [\![Q]\!] \to [\![A]\!] \otimes [\![Q']\!]$.

Proof. We proceed by rule induction on the evaluation relation $\Downarrow$. See Table 5.3.

Axiom rules: The result is immediate for the axiom cases, namely: $(\mathit{label})_\Downarrow$, $(\mathit{const})_\Downarrow$, $(*)_\Downarrow$, $(\mathit{abs})_\Downarrow$, $(\mathit{lift})_\Downarrow$, and $(\mathit{circ})_\Downarrow$, because in these cases, $(C, M) = (C', V)$.

Purely inductive rules: These rules are all treated in a similar fashion and include $(\mathit{left})_\Downarrow$, $(\mathit{right})_\Downarrow$, $(\mathit{seq})_\Downarrow$, $(\mathit{pair})_\Downarrow$, and $(\mathit{force})_\Downarrow$. In these cases, the result follows by applications of the induction hypothesis and the Subject Reduction Theorem 6.4.2. We treat each case in turn:

- $(\mathit{left})_\Downarrow$: The evaluation rule is

$$\frac{(C, M) \Downarrow (C', V)}{(C, \mathrm{left}_{A,B}\, M) \Downarrow (C', \mathrm{left}_{A,B}\, V)}\ .$$

  Assume that $Q \vdash (C, \mathrm{left}_{A,B}\, M) : A + B; Q'$ is valid. We want to show that

$$[\![(C, \mathrm{left}_{A,B}\, M)]\!] = [\![(C', \mathrm{left}_{A,B}\, V)]\!] : [\![Q]\!] \to [\![A + B]\!] \otimes [\![Q']\!].$$

  By assumption, there is a label context Q'' such that $C : Q \to Q'', Q'$ and $\emptyset; Q'' \vdash \mathrm{left}_{A,B}\, M : A + B;$ is valid. By the typing rule (*left*), $\emptyset; Q'' \vdash M : A$ is valid. It follows that $Q \vdash (C, M) : A; Q'$ is valid as well.

By the Subject Reduction Theorem 6.4.2, $(C, M) \Downarrow (C', V)$ yields that $Q \vdash (C', V) : A; Q'$ is valid. Thus, there exists a label context Q''' such that $C' : Q \to Q''', Q'$ and $\emptyset; Q''' \vdash V : A$ is valid. Since $Q \vdash (C, M) : A; Q'$ is valid and $(C, M) \Downarrow (C', V)$, the induction hypothesis implies that the equation $\llbracket (C, M) \rrbracket = \llbracket (C', V) \rrbracket : \llbracket Q \rrbracket \to \llbracket A \rrbracket \otimes \llbracket Q' \rrbracket$ holds, and so the following diagram commutes:

$$\begin{tikzcd}
\llbracket Q'', Q' \rrbracket \arrow[r, "\cong"] & \llbracket \emptyset; Q'' \rrbracket \otimes \llbracket Q' \rrbracket \arrow[d, "\llbracket M \rrbracket \otimes \mathrm{id}"] \\
\llbracket Q \rrbracket \arrow[u, "C"] \arrow[d, "C'"'] & \llbracket A \rrbracket \otimes \llbracket Q' \rrbracket \\
\llbracket Q''', Q' \rrbracket \arrow[r, "\cong"'] & \llbracket \emptyset; Q''' \rrbracket \otimes \llbracket Q' \rrbracket \arrow[u, "\llbracket V \rrbracket \otimes \mathrm{id}"']
\end{tikzcd} \tag{9.1}$$

Since $\emptyset; Q'' \vdash M : A$ is valid, the definition of $\llbracket \mathrm{left}_{A,B}\, M \rrbracket : \llbracket \emptyset; Q'' \rrbracket \to \llbracket A \rrbracket$ implies that the following diagram commutes:

$$\begin{tikzcd}
\llbracket \emptyset; Q'' \rrbracket \otimes \llbracket Q' \rrbracket \arrow[d, "\llbracket M \rrbracket \otimes \mathrm{id}"'] \arrow[dr, "\llbracket \mathrm{left}_{A,B}\, M \rrbracket \otimes \mathrm{id}"] & \\
\llbracket A \rrbracket \otimes \llbracket Q' \rrbracket \arrow[r, "\mathrm{left}_{A,B} \otimes \mathrm{id}"'] & \llbracket A + B \rrbracket \otimes \llbracket Q' \rrbracket
\end{tikzcd} \tag{9.2}$$

Similarly, since $\emptyset; Q''' \vdash V : A$ is valid, the following diagram commutes by the definition of $\llbracket \mathrm{left}_{A,B}\, V \rrbracket : \llbracket \emptyset; Q''' \rrbracket \to \llbracket A \rrbracket$:

$$\begin{tikzcd}
\llbracket A \rrbracket \otimes \llbracket Q' \rrbracket \arrow[r, "\mathrm{left}_{A,B} \otimes \mathrm{id}"] & \llbracket A + B \rrbracket \otimes \llbracket Q' \rrbracket \\
\llbracket \emptyset; Q''' \rrbracket \otimes \llbracket Q' \rrbracket \arrow[u, "\llbracket V \rrbracket \otimes \mathrm{id}"] \arrow[ur, "\llbracket \mathrm{left}_{A,B}\, V \rrbracket \otimes \mathrm{id}"'] &
\end{tikzcd} \tag{9.3}$$

Consider

$$\begin{tikzcd}
\llbracket Q'', Q' \rrbracket \arrow[r, "\cong"] & \llbracket \emptyset; Q'' \rrbracket \otimes \llbracket Q' \rrbracket \arrow[d, "\llbracket M \rrbracket \otimes \mathrm{id}"'] \arrow[dr, "\llbracket \mathrm{left}_{A,B}\, M \rrbracket \otimes \mathrm{id}"] & \\
\llbracket Q \rrbracket \arrow[u, "C"] \arrow[d, "C'"'] & \llbracket A \rrbracket \otimes \llbracket Q' \rrbracket \arrow[r, "\mathrm{left}_{A,B} \otimes \mathrm{id}"] & \llbracket A + B \rrbracket \otimes \llbracket Q' \rrbracket. \\
\llbracket Q''', Q' \rrbracket \arrow[r, "\cong"'] & \llbracket \emptyset; Q''' \rrbracket \otimes \llbracket Q' \rrbracket \arrow[u, "\llbracket V \rrbracket \otimes \mathrm{id}"] \arrow[ur, "\llbracket \mathrm{left}_{A,B}\, V \rrbracket \otimes \mathrm{id}"'] &
\end{tikzcd}$$

The square commutes by (9.1), while the top and bottom triangles commute by (9.2) and (9.3), respectively. Hence the top and bottom paths are equal, and so

$$\llbracket (C, \mathrm{left}_{A,B}\, M) \rrbracket = \llbracket (C', \mathrm{left}_{A,B}\, V) \rrbracket : \llbracket Q \rrbracket \to \llbracket A + B \rrbracket \otimes \llbracket Q' \rrbracket,$$

as required.

- $(\mathit{right})_{\Downarrow}$: This case is similar to the previous one.
- $(\mathit{seq})_{\Downarrow}$: The evaluation rule is

$$\frac{(C, M) \Downarrow (C', *) \quad (C', N) \Downarrow (C'', W)}{(C, M; N) \Downarrow (C'', W)}\ .$$

Assume that $Q \vdash (C, M; N) : A; Q'$ is valid. We want to show that

$$\llbracket (C, M; N) \rrbracket = \llbracket (C'', W) \rrbracket : \llbracket Q \rrbracket \to \llbracket A \rrbracket \otimes \llbracket Q' \rrbracket.$$

By assumption, there exists a label context Q'' such that $C : Q \to Q'', Q'$ and $\emptyset; Q'' \vdash M; N : A$ is valid. By the typing rule (seq), there are label contexts Q''_1 and Q''_2 such that $Q'' = Q''_1, Q''_2$, and both $\emptyset; Q''_1 \vdash M : I$ and $\emptyset; Q''_2 \vdash N : A$ are valid.

Since $C : Q \to Q''_1, Q''_2, Q'$ and $\emptyset; Q''_1 \vdash M : I$ is valid, it follows that $Q \vdash (C, M) : I; Q''_2, Q'$ is valid as well. By the Subject Reduction Theorem 6.4.2, the evaluation $(C, M) \Downarrow (C', *)$ yields that $Q \vdash (C', *) : I; Q''_2, Q'$ is valid. Thus, there exists a label context Q''' such that $C' : Q \to Q''', Q''_2, Q'$

and $\emptyset; Q''' \vdash * : I$ is valid. By the typing rule $(*)$, $Q''' = \emptyset$, and so, $C' : Q \to Q''_2, Q'$ and $\emptyset; \emptyset \vdash * : I$.

Since $Q \vdash (C, M) : I; Q''_2, Q'$ is valid and $(C, M) \Downarrow (C', *)$, the induction hypothesis implies that $\llbracket (C, M) \rrbracket = \llbracket (C', *) \rrbracket : \llbracket Q \rrbracket \to \llbracket I \rrbracket \otimes \llbracket Q''_2, Q' \rrbracket$, and so the following diagram commutes:

$$\begin{tikzcd}
\llbracket Q''_1, Q''_2, Q' \rrbracket \arrow[r, "\cong"] & \llbracket \emptyset; Q''_1 \rrbracket \otimes \llbracket Q''_2, Q' \rrbracket \arrow[d, "\llbracket M \rrbracket \otimes \mathrm{id}"] \\
\llbracket Q \rrbracket \arrow[u, "C"] \arrow[d, "C'"'] & \llbracket I \rrbracket \otimes \llbracket Q''_2, Q' \rrbracket \\
\llbracket Q''_2, Q' \rrbracket \arrow[r, "\cong"'] & \llbracket \emptyset; \emptyset \rrbracket \otimes \llbracket Q''_2, Q' \rrbracket \arrow[u, "\llbracket * \rrbracket \otimes \mathrm{id}"']
\end{tikzcd} \tag{9.4}$$

Since $C' : Q \to Q''_2, Q'$ and $\emptyset; Q''_2 \vdash N : A$, we have that $Q \vdash (C', N) : A; Q'$ is valid. By the Subject Reduction Theorem 6.4.2, $(C', N) \Downarrow (C'', W)$ implies that $Q \vdash (C'', W) : A; Q'$ is valid as well. Thus, there exists a label context $\overline{Q}$ such that $C'' : Q \to \overline{Q}, Q'$ and $\emptyset; \overline{Q} \vdash W : A$. Since $Q \vdash (C', N) : A; Q'$ is valid and $(C', N) \Downarrow (C'', W)$, the induction hypothesis implies that $\llbracket (C', N) \rrbracket = \llbracket (C'', W) \rrbracket : \llbracket Q \rrbracket \to \llbracket A \rrbracket \otimes \llbracket Q' \rrbracket$, and so the following diagram commutes:

$$\begin{tikzcd}
\llbracket Q''_2, Q' \rrbracket \arrow[r, "\cong"] & \llbracket \emptyset; Q''_2 \rrbracket \otimes \llbracket Q' \rrbracket \arrow[d, "\llbracket N \rrbracket \otimes \mathrm{id}"] \\
\llbracket Q \rrbracket \arrow[u, "C'"] \arrow[d, "C''"'] & \llbracket A \rrbracket \otimes \llbracket Q' \rrbracket \\
\llbracket \overline{Q}, Q' \rrbracket \arrow[r, "\cong"'] & \llbracket \emptyset; \overline{Q} \rrbracket \otimes \llbracket Q' \rrbracket \arrow[u, "\llbracket W \rrbracket \otimes \mathrm{id}"']
\end{tikzcd} \tag{9.5}$$

The definition of $\llbracket M; N \rrbracket : \llbracket \emptyset; Q'' \rrbracket \to \llbracket A \rrbracket$ and the coherence of the monoidal structure guarantee the commutativity of the following diagram as well:

$$\begin{tikzcd}
\llbracket \emptyset; Q'' \rrbracket \otimes \llbracket Q' \rrbracket \arrow[r, "\cong"] & \llbracket \emptyset, Q''_1 \rrbracket \otimes \llbracket \emptyset, Q''_2 \rrbracket \otimes \llbracket Q' \rrbracket \arrow[rr, "\llbracket M \rrbracket \otimes \llbracket N \rrbracket \otimes \mathrm{id}"] \arrow[d, "\llbracket M \rrbracket \otimes \mathrm{id} \otimes \mathrm{id}"'] & & \llbracket I \rrbracket \otimes \llbracket A \rrbracket \otimes \llbracket Q' \rrbracket \arrow[d, "\cong"] \\
& \llbracket I \rrbracket \otimes \llbracket \emptyset, Q''_2 \rrbracket \otimes \llbracket Q' \rrbracket \arrow[r, "\cong"'] & \llbracket \emptyset, Q''_2 \rrbracket \otimes \llbracket Q' \rrbracket \arrow[r, "\llbracket N \rrbracket \otimes \mathrm{id}"'] & \llbracket A \rrbracket \otimes \llbracket Q' \rrbracket
\end{tikzcd} \tag{9.6}$$

Now consider

$$\begin{tikzcd}
\llbracket Q'', Q' \rrbracket \arrow[rrrr, "\cong"] \arrow[rrd, "\cong"] & & & & \llbracket \emptyset; Q'' \rrbracket \otimes \llbracket Q' \rrbracket \arrow[ld, "\cong"'] \arrow[ddd, "\llbracket M;N \rrbracket \otimes \mathrm{id}"] \\
& & \llbracket \emptyset; Q''_1 \rrbracket \otimes \llbracket Q''_2, Q' \rrbracket \arrow[r, "\cong"] \arrow[d, "\llbracket M \rrbracket \otimes \mathrm{id}"] & \llbracket \emptyset; Q''_1 \rrbracket \otimes \llbracket \emptyset; Q''_2 \rrbracket \otimes \llbracket Q' \rrbracket \arrow[d, "\llbracket M \rrbracket \otimes \mathrm{id} \otimes \mathrm{id}"] & \\
& & \llbracket I \rrbracket \otimes \llbracket Q''_2, Q' \rrbracket \arrow[r, "\cong"] & \llbracket I \rrbracket \otimes \llbracket \emptyset; Q''_2 \rrbracket \otimes \llbracket Q' \rrbracket \arrow[d, "\cong"] & \\
\llbracket Q \rrbracket \arrow[uuu, "C"] \arrow[r, "C'"] \arrow[d, "C''"'] & \llbracket Q''_2, Q' \rrbracket \arrow[r, "\cong"] & \llbracket \emptyset; \emptyset \rrbracket \otimes \llbracket Q''_2, Q' \rrbracket \arrow[u, "\llbracket * \rrbracket \otimes \mathrm{id}"'] \arrow[r, "\cong"] & \llbracket \emptyset; Q''_2 \rrbracket \otimes \llbracket Q' \rrbracket \arrow[r, "\llbracket N \rrbracket \otimes \mathrm{id}"] & \llbracket A \rrbracket \otimes \llbracket Q' \rrbracket \\
\llbracket \overline{Q}, Q' \rrbracket \arrow[rrrr, "\cong"'] & & & & \llbracket \emptyset; \overline{Q} \rrbracket \otimes \llbracket Q' \rrbracket \arrow[u, "\llbracket W \rrbracket \otimes \mathrm{id}"]
\end{tikzcd}$$

The leftmost diagram commutes by (9.4), and the rightmost by (9.6). The bottom diagram commutes by (9.5). The remaining regions commute by coherence for symmetric monoidal categories. Hence, the outer square commutes as well. This implies that

$$\llbracket (C, M; N) \rrbracket = \llbracket (C'', W) \rrbracket : \llbracket Q \rrbracket \to \llbracket A \rrbracket \otimes \llbracket Q' \rrbracket,$$

as required.

- $(pair)_\Downarrow$: The evaluation rule is

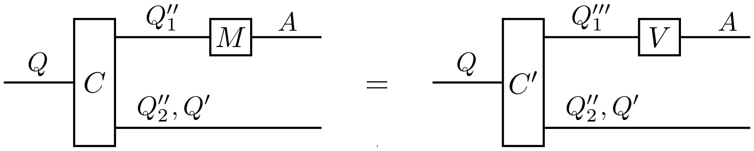
$$\frac{(C, M) \Downarrow (C', V) \quad (C', N) \Downarrow (C'', V')}{(C, \langle M, N \rangle) \Downarrow (C'', \langle V, V' \rangle)} .$$

Assume that $Q \vdash (C, \langle M, N \rangle) : A \otimes B; Q'$ is valid. We want to show that

$$\llbracket (C, \langle M, N \rangle) \rrbracket = \llbracket (C'', \langle V, V' \rangle) \rrbracket : \llbracket Q \rrbracket \to \llbracket A \otimes B \rrbracket \otimes \llbracket Q' \rrbracket.$$

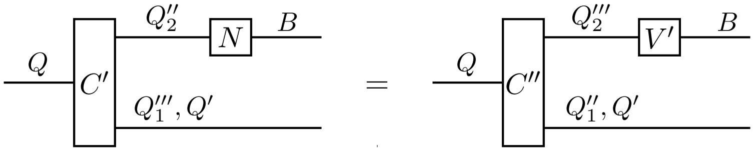
By assumption, there exists a label context Q'' such that $C : Q \to Q'', Q'$ and $\emptyset; Q'' \vdash \langle M, N \rangle : A \otimes B$ is valid. By the typing rule (*pair*), there are label contexts Q''_1 and Q''_2 such that $Q'' = Q''_1, Q''_2$, and both $\emptyset; Q''_1 \vdash M : A$ and $\emptyset; Q''_2 \vdash N : B$ are valid.

Since $C : Q \to Q_1'', Q_2'', Q'$ and $\emptyset; Q_1'' \vdash M : A$ is valid, it follows that $Q \vdash (C, M) : A; Q_2'', Q'$ is valid as well. By the Subject Reduction Theorem 6.4.2, $(C, M) \Downarrow (C', V)$ yields that $Q \vdash (C', V) : A; Q_2'', Q'$ is valid. Thus, there exists a label context Q_1''' such that $C' : Q \to Q_1''', Q_2'', Q'$ and $\emptyset; Q_1''' \vdash V : A$ is valid. Since $Q \vdash (C, M) : A; Q_2'', Q'$ is valid and $(C, M) \Downarrow (C', V)$, the induction hypothesis implies that the equation $[\![(C, M)]\!] = [\![(C', V)]\!] : [\![Q]\!] \to [\![A]\!] \otimes [\![Q_2'', Q']\!]$ holds. Diagrammatically,



(9.7)

Since $\emptyset; Q_2'' \vdash N : B$ is valid, it follows that $Q \vdash (C', N) : B; Q_1''', Q'$ is valid as well. By the Subject Reduction Theorem 6.4.2, $(C', N) \Downarrow (C'', V')$ implies that $Q \vdash (C'', V') : B; Q_1''', Q'$ is valid too. Thus, there exists a label context Q_2''' such that $C'' : Q \to Q_2''', Q_1''', Q'$ and $\emptyset, Q_2''' \vdash V' : B$ is valid. Since $Q \vdash (C', N) : B; Q_1''', Q'$ is valid and $(C', N) \Downarrow (C'', V')$, the induction hypothesis implies that $[\![(C', N)]\!] = [\![(C'', V')]\!] : [\![Q]\!] \to [\![A]\!] \otimes [\![Q_1'', Q']\!]$ holds. Diagrammatically,



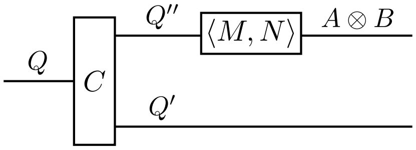
(9.8)

We can now prove that

$$[\![(C, \langle M, N \rangle)]\!] = [\![(C'', \langle V, V' \rangle)]\!] : [\![Q]\!] \to [\![A \otimes B]\!] \otimes [\![Q']\!],$$

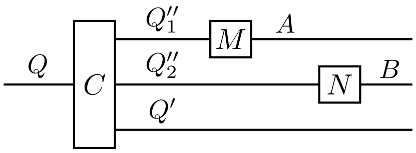
as follows:

$[\![(C, \langle M, N\rangle)]\!] =$



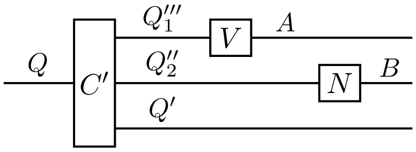
by def.

$=$



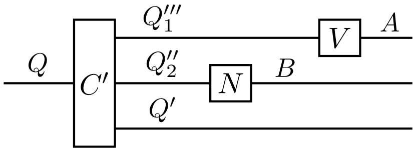
by def.

$=$



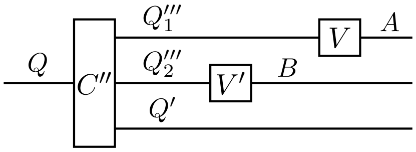
by (9.7)

$=$



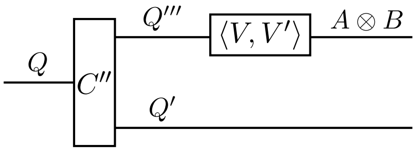
by functor.

$=$



by (9.8)

$=$



by def.

$= \quad [\![(C, \langle V, V'\rangle)]\!]$ by def.

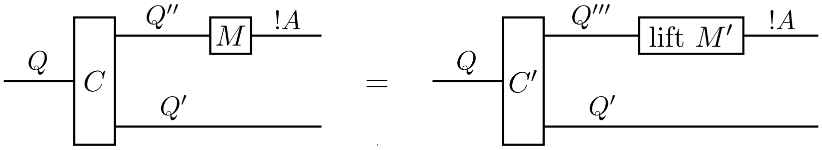
- $(\mathit{force})_\Downarrow$: The evaluation rule is

$$\frac{(C, M) \Downarrow (C', \text{lift } M') \quad (C', M') \Downarrow (C'', V)}{(C, \text{force } M) \Downarrow (C'', V)}\ .$$

Assume that $Q \vdash (C, \text{force } M) : A; Q'$ is valid. We want to show that

$$[\![(C, \text{force } M)]\!] = [\![(C'', V)]\!] : [\![Q]\!] \to [\![A]\!] \otimes [\![Q']\!].$$

By assumption, there exists a label context Q'' such that $C : Q \to Q'', Q'$ and $\emptyset; Q'' \vdash \text{force } M : A$ is valid. By the typing rule (*force*), we have that $\emptyset; Q'' \vdash M : !A$ is valid, and so, $Q \vdash (C, M) : !A; Q'$ is valid as well. By the Subject Reduction Theorem 6.4.2, $(C, M) \Downarrow (C', \text{lift } M')$ yields that $Q \vdash (C', \text{lift } M') : !A, Q'$ is valid. Thus, there exists a label context Q''' such that $C' : Q \to Q''', Q'$ and $\emptyset; Q''' \vdash \text{lift } M' : !A$ is valid. Since $Q \vdash (C, M) : !A; Q'$ is valid and $(C, M) \Downarrow (C', \text{lift } M')$, the induction hypothesis implies that $\llbracket (C, M) \rrbracket = \llbracket (C', \text{lift } M') \rrbracket : \llbracket Q \rrbracket \to \llbracket !A \rrbracket \otimes \llbracket Q' \rrbracket$. Diagrammatically,

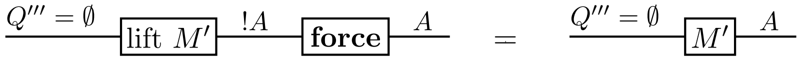


(9.9)

By the typing rule (*lift*), we have that $Q''' = \emptyset$ and $\emptyset; \emptyset \vdash M' : A$ is valid. Since $(\llbracket M' \rrbracket \circ \rho_I^{-1} \circ \tau_I^{-1})^\circ$ is the transpose of the morphism $\llbracket M' \rrbracket \circ \rho_I^{-1} \circ \tau_I^{-1} : p(\llbracket I \rrbracket_p) \to \llbracket A \rrbracket$ under the adjunction $p \dashv \flat$, the following diagram commutes:

$$\begin{tikzcd}
 & p(\llbracket I \rrbracket_p) \arrow[d, "\tau_I^{-1}"] \arrow[ddl, "p((\llbracket M' \rrbracket \circ \rho_I^{-1} \tau_I^{-1})^\circ)"'] \\
 & \llbracket I \rrbracket \arrow[d, "\rho^{-1}"] \\
 & \llbracket I \rrbracket \otimes \llbracket I \rrbracket \arrow[d, "\llbracket M' \rrbracket"] \\
!\llbracket A \rrbracket \arrow[r, "\mathbf{force}"'] & \llbracket A \rrbracket.
\end{tikzcd}$$

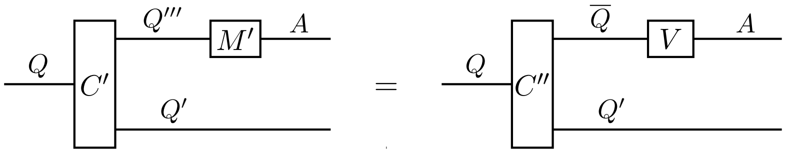
That is, $\mathbf{force} \circ p((\llbracket M' \rrbracket \circ \rho_I^{-1} \circ \tau_I^{-1})^\circ) \circ \tau_I \circ \rho_I = \llbracket M' \rrbracket$, and so we have that $\mathbf{force} \circ \llbracket \emptyset; \emptyset \vdash \text{lift } M' : A \rrbracket = \llbracket \emptyset; \emptyset \vdash M' : A \rrbracket$. Diagrammatically,



(9.10)

Since $C' : Q \to Q''', Q'$ and $\emptyset; \emptyset \vdash M' : A$ is valid, $Q \vdash (C', M') : A, Q'$ is valid as well. By the Subject Reduction Theorem 6.4.2, $(C', M') \Downarrow (C'', V)$ yields that $Q \vdash (C'', V) : A, Q'$ is valid. Thus, there exists a label context $\overline{Q}$ such that $C'' : Q \to \overline{Q}, Q'$ and $\emptyset; \overline{Q} \vdash V : A$ is valid. By the induction hypothesis, $(C', M') \Downarrow (C'', V)$ implies that the equation

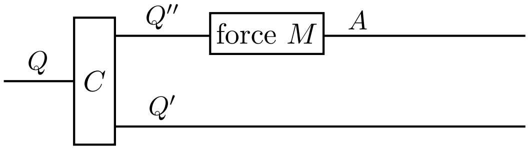
$[\![(C', M')]\!] = [\![(C'', V)]\!] : [\![Q]\!] \to [\![A]\!] \otimes [\![Q']\!]$ holds. Diagrammatically,



(9.11)

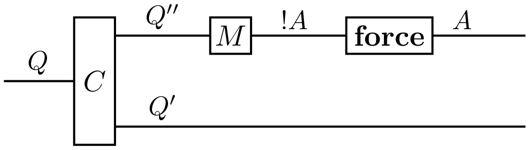
We can now prove that $[\![(C, \text{force } M)]\!] = [\![(C'', V)]\!] : [\![Q]\!] \to [\![A]\!] \otimes [\![Q']\!]$ as follows:

$[\![(C, \text{force } M)]\!] =$



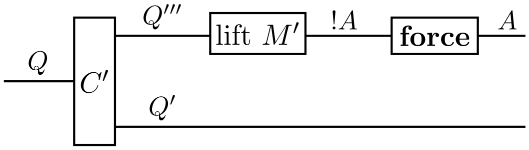
by def.

$=$



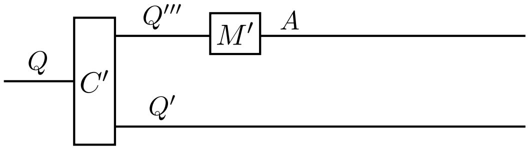
by def.

$=$



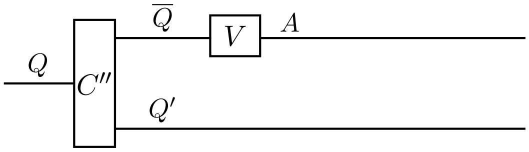
by (9.9)

$=$



by (9.10)

$=$



by (9.11)

$= \quad [\![(C'', V)]\!]$ by def.

Substitution rules: These are rules with a configuration containing a term in which a substitution has occurred, namely, $(\mathit{let})_{\Downarrow}$, $(\mathit{let\text{-}pair})_{\Downarrow}$, $(\mathit{case\text{-}left})_{\Downarrow}$, $(\mathit{case\text{-}right})_{\Downarrow}$, and $(\mathit{app})_{\Downarrow}$. In these cases, the result follows by applications of the induction hypothesis, the Semantic Substitution Lemma 8.6.1, and the Subject Reduction Theorem 6.4.2. We treat each case in turn:

- $(\mathit{let})_{\Downarrow}$: The evaluation rule is

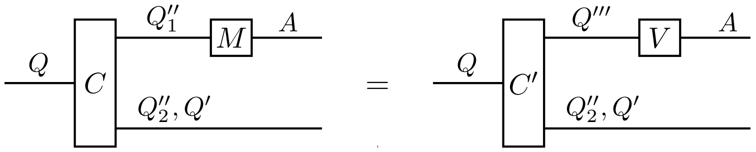
$$\frac{(C, M) \Downarrow (C', V) \quad (C', N[V/x]) \Downarrow (C'', W)}{(C, \text{let } x = M \text{ in } N) \Downarrow (C'', W)}\ .$$

Assume that $Q \vdash (C, \text{let } x = M \text{ in } N) : B; Q'$ is valid. We want to show that

$$[\![(C, \text{let } x = M \text{ in } N)]\!] = [\![(C'', W)]\!] : [\![Q]\!] \to [\![B]\!] \otimes [\![Q']\!].$$

By assumption, there exists a label context Q'' such that $C : Q \to Q'', Q'$ and $\emptyset; Q'' \vdash \text{let } x = M \text{ in } N : B$ is valid. By the typing rule (*let*), there are label contexts Q''_1 and Q''_2 such that $Q'' = Q''_1, Q''_2$, and both $\emptyset; Q''_1 \vdash M : A$ and $x : A; Q''_2 \vdash N : B$ are valid.

Since $C : Q \to Q''_1, Q''_2, Q'$ and $\emptyset; Q''_1 \vdash M : A$ is valid, it follows that $Q \vdash (C, M) : A; Q''_2, Q'$ is valid as well. By the Subject Reduction Theorem 6.4.2, $(C, M) \Downarrow (C', V)$ yields that $Q \vdash (C', V) : A; Q''_2, Q'$ is also valid. Thus, there exists a label context Q''' such that $C' : Q \to Q''', Q''_2, Q'$ and $\emptyset; Q''' \vdash V : A$ is valid. Since $Q \vdash (C, M) : A; Q''_2, Q'$ is valid and $(C, M) \Downarrow (C', V)$, the induction hypothesis implies that the equation $[\![(C, M)]\!] = [\![(C', V)]\!] : [\![Q]\!] \to [\![A]\!] \otimes [\![Q''_2, Q']\!]$ holds. Diagrammatically,

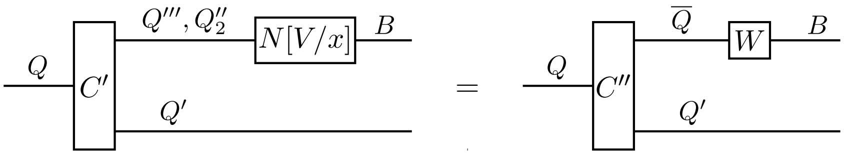


(9.12)

Now $x : A; Q''_2 \vdash N : B$ is valid, and so $\emptyset; Q''', Q_2 \vdash N[V/x] : B$ is also valid by the syntactic Substitution Lemma 6.3.1. This together with $C' : Q \to Q''', Q''_2, Q'$ implies that $Q \vdash (C', N[V/x]) : B; Q'$ is valid. By the Subject Reduction Theorem 6.4.2, $(C', N[V/x]) \Downarrow (C'', W)$ implies that $Q \vdash (C'', W) : B; Q'$ is valid as well. Thus, there exists a label context $\overline{Q}$ such that $C'' : Q \to \overline{Q}, Q'$ and $\emptyset; \overline{Q} \vdash W : B$ is valid. Since $Q \vdash (C', N[V/x]) : B; Q'$ is valid and $(C', N[V/x]) \Downarrow (C'', W)$, the induction hypothesis implies that

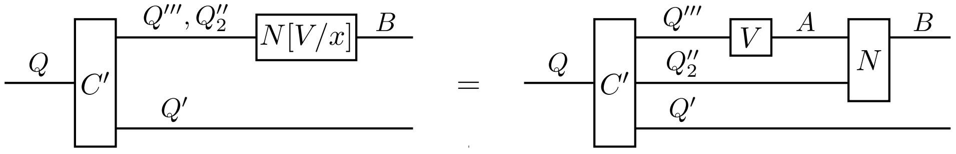
$$[\![((C', N[V/x])]\!] = [\![(C'', W)]\!] : [\![Q]\!] \to [\![B]\!] \otimes [\![Q']\!].$$

Diagrammatically,



(9.13)

Since both $x : A; Q''_2 \vdash N : B$ and $\emptyset; Q''' \vdash V : A$ are valid typing judgements and $C' : Q \to Q''', Q''_2, Q'$, the Semantic Substitution Lemma 8.6.1 implies that $[\![((C', N[V/x]))]\!] = [\![(C', \text{let } x = V \text{ in } N)]\!] : [\![Q]\!] \to [\![B]\!] \otimes [\![Q']\!]$. Diagrammatically,



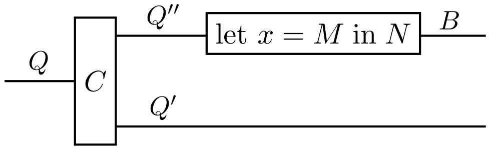
(9.14)

We can now show that

$$[\![(C, \text{let } x = M \text{ in } N)]\!] = [\![(C'', W)]\!] : [\![Q]\!] \to [\![B]\!] \otimes [\![Q']\!]$$

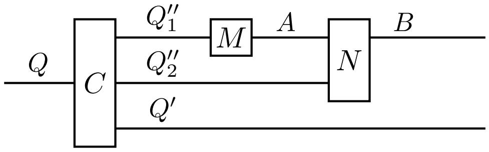
as follows:

$[\![(C, \text{let } x = M \text{ in } N)]\!] =$



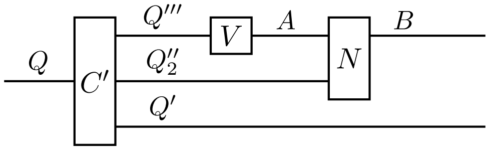
by def.

$=$



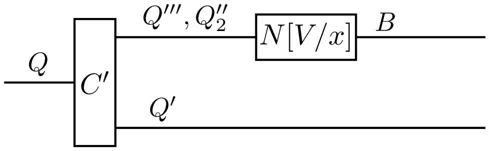
by def.

$=$



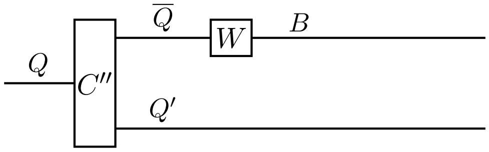
by (9.12)

$=$



by (9.14)

$=$



by (9.13)

$= \ [\![(C'', W)]\!]$ by def.

- (*let-pair*)$_\Downarrow$: This case is similar to the (*let*)$_\Downarrow$ case.

- $(\textit{case-left})_{\Downarrow}$: The evaluation rule is

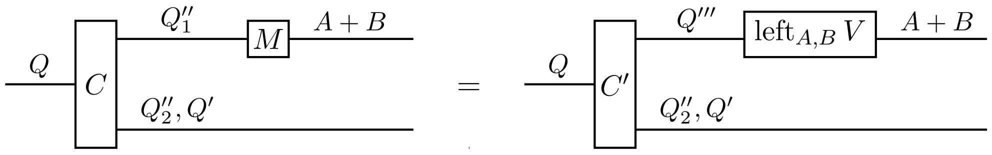
$$\frac{(C, M) \Downarrow (C', \text{left}\, V) \quad (C', N[V/x]) \Downarrow (C'', W)}{(C, \text{case } M \text{ of } \{\text{left}\, x \to N \mid \text{right}\, y \to P\}) \Downarrow (C'', W)}.$$

Assume that $Q \vdash (C, \text{case } M \text{ of } \{\text{left}\, x \to N \mid \text{right}\, y \to P\}) : D; Q'$ is valid. We want to show that

$$[\![(C, \text{case } M \text{ of } \{\text{left}\, x \to N \mid \text{right}\, y \to P\})]\!] = [\![(C'', W)]\!] : [\![Q]\!] \to [\![D]\!] \otimes [\![Q']\!].$$

By assumption, there exists a label context Q'' such that $C : Q \to Q'', Q'$ and $\emptyset; Q'' \vdash \text{case } M \text{ of } \{\text{left}\, x \to N \mid \text{right}\, y \to P\} : D$ is valid. By the typing rule (*case*), there are label contexts Q''_1 and Q''_2 such that $Q'' = Q''_1, Q''_2$, and $\emptyset; Q''_1 \vdash M : A + B$ as well as $x : A; Q''_2 \vdash N : D$ and $y : B; Q''_2 \vdash P : D$ are valid.

Since $C : Q \to Q''_1, Q''_2, Q'$ and $\emptyset; Q''_1 \vdash M : A + B$ is valid, it follows that $Q \vdash (C, M) : A + B; Q''_2, Q'$ is valid as well. By the Subject Reduction Theorem 6.4.2, $(C, M) \Downarrow (C', \text{left}_{A,B}\, V)$ implies that the judgement $Q \vdash (C', \text{left}_{A,B}\, V) : A + B; Q''_2, Q'$ is valid. Thus, there exists a label context Q''' such that $C' : Q \to Q''', Q''_2, Q'$ and $\emptyset; Q''' \vdash \text{left}_{A,B}\, V : A + B$ is valid. By the induction hypothesis, $(C, M) \Downarrow (C', \text{left}_{A,B}\, V)$ implies that $[\![(C, M)]\!] = [\![(C', \text{left}_{A,B}\, V)]\!] : [\![Q]\!] \to [\![A + B]\!] \otimes [\![Q''_2, Q']\!]$. Diagrammatically,



(9.15)

By definition of $[\![\emptyset; Q''' \vdash \text{left}_{A,B}\, V : A + B]\!]$, the following diagram commutes:

$$\begin{tikzcd}
{[\![\emptyset; Q''']\!] \otimes [\![Q''_2]\!]} \arrow[r, "{[\![\text{left}_{A,B}\, V]\!] \otimes \text{id}}"] \arrow[dr, "{[\![V]\!] \otimes \text{id}}"'] & {([\![A]\!] + [\![B]\!]) \otimes [\![Q''_2]\!]} \\
 & {[\![A]\!] \otimes [\![Q''_2]\!]} \arrow[u, "\text{left} \otimes \text{id}"']
\end{tikzcd}$$

Similarly, by definition of $[\llbracket N\rrbracket, \llbracket P\rrbracket]$, the following diagram commutes:

$$\begin{tikzcd}
(\llbracket A\rrbracket \otimes \llbracket Q_2''\rrbracket) + (\llbracket B\rrbracket \otimes \llbracket Q_2''\rrbracket) \arrow[r, "{[\llbracket N\rrbracket, \llbracket P\rrbracket]}"] & \llbracket D\rrbracket \\
\llbracket A\rrbracket \otimes \llbracket Q_2''\rrbracket \arrow[u, "\text{left}"] \arrow[ur, "\llbracket N\rrbracket"'] &
\end{tikzcd}$$

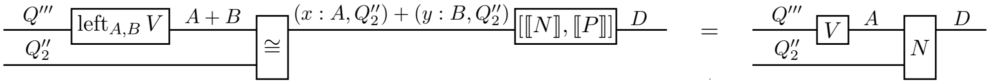
Moreover, since $\mathbf{L}$ is a distributive category, the following diagram commutes as well:

$$\begin{tikzcd}
(\llbracket A\rrbracket + \llbracket B\rrbracket) \otimes \llbracket Q_2''\rrbracket \arrow[r, "\cong"] & (\llbracket A\rrbracket \otimes \llbracket Q_2''\rrbracket) + (\llbracket B\rrbracket \otimes \llbracket Q_2''\rrbracket) \\
\llbracket A\rrbracket \otimes \llbracket Q_2''\rrbracket \arrow[u, "\text{left} \otimes \text{id}"] \arrow[r, "\text{id}"'] & \llbracket A\rrbracket \otimes \llbracket Q_2''\rrbracket \arrow[u, "\text{left}"']
\end{tikzcd}$$

The previous three diagrams imply that

$$\begin{tikzcd}
\llbracket \emptyset; Q'''\rrbracket \otimes \llbracket Q_2''\rrbracket \arrow[r, "\llbracket \text{left}_{A,B}\, V\rrbracket \otimes \text{id}"] \arrow[dr, "\llbracket V\rrbracket \otimes \text{id}"'] & (\llbracket A\rrbracket + \llbracket B\rrbracket) \otimes \llbracket Q_2''\rrbracket \arrow[r, "\cong"] & (\llbracket A\rrbracket \otimes \llbracket Q_2''\rrbracket) + (\llbracket B\rrbracket \otimes \llbracket Q_2''\rrbracket) \arrow[r, "{[\llbracket N\rrbracket, \llbracket P\rrbracket]}"] & \llbracket D\rrbracket \\
& \llbracket A\rrbracket \otimes \llbracket Q_2''\rrbracket \arrow[u, "\text{left} \otimes \text{id}"'] \arrow[r, "\text{id}"'] & \llbracket A\rrbracket \otimes \llbracket Q_2''\rrbracket \arrow[u, "\text{left}"'] \arrow[ur, "\llbracket N\rrbracket"'] &
\end{tikzcd}$$

commutes, and so the resulting morphisms from the top and the bottom paths are equal. Diagrammatically,

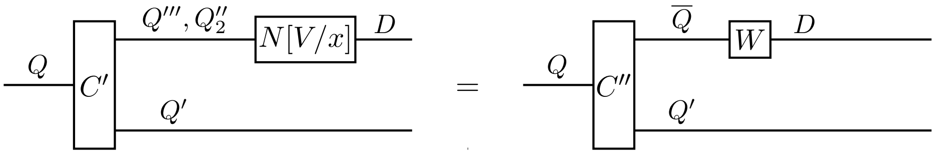

Q'''
Q_2''
$\text{left}_{A,B}\, V$
$A + B$
$\cong$
$(x : A, Q_2'') + (y : B, Q_2'')$
$[\llbracket N\rrbracket, \llbracket P\rrbracket]$
D
$=$
Q'''
Q_2''
V
A
N
D

(9.16)

Since $\emptyset; Q''' \vdash \text{left}_{A,B}\, V : A + B$ is valid, the (*left*) rule implies that $\emptyset; Q''' \vdash V : A$ is also valid. Since $x : A; Q_2'' \vdash N : D$ is valid, we have that $\emptyset; Q''', Q_2 \vdash N[V/x] : D$ is valid by the syntactic Substitution Lemma 6.3.1. By the Subject Reduction Theorem 6.4.2, $(C', N[V/x]) \Downarrow (C'', W)$ implies that $Q \vdash (C'', W) : D; Q'$ is valid. Thus, there exists a label context $\overline{Q}$ such that $C'' : Q \to \overline{Q}, Q'$ and $\emptyset; \overline{Q} \vdash W : D$ is valid. By the induction hypothesis, $(C', N[V/x]) \Downarrow (C'', W)$ implies that

$$\llbracket ((C', N[V/x])\rrbracket = \llbracket (C'', W)\rrbracket : \llbracket Q\rrbracket \to \llbracket D\rrbracket \otimes \llbracket Q'\rrbracket.$$

Diagrammatically,

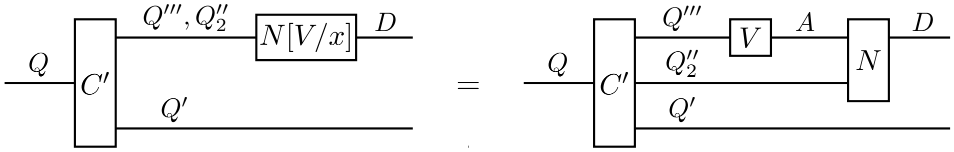


(9.17)

Similarly, since $\emptyset; Q''' \vdash V : A$ and $x : A; Q''_2 \vdash N : D$ are valid and $C' : Q \to Q''', Q''_2, Q'$, the Semantic Substitution Lemma 8.6.1 implies that

$$[\![((C', N[V/x])]\!] = [\![(C', \text{let } x = V \text{ in } N)]\!] : [\![Q]\!] \to [\![D]\!] \otimes [\![Q']\!].$$

Diagrammatically,



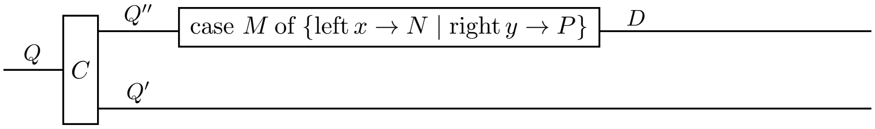
(9.18)

We now have all the ingredients to prove that

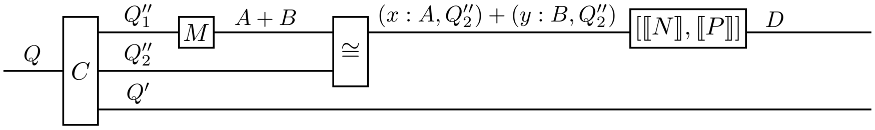
$$[\![(C, \text{case } M \text{ of } \{\text{left}\, x \to N \mid \text{right}\, y \to P\})]\!] = [\![(C'', W)]\!] : [\![Q]\!] \to [\![D]\!] \otimes [\![Q']\!],$$

as shown below:

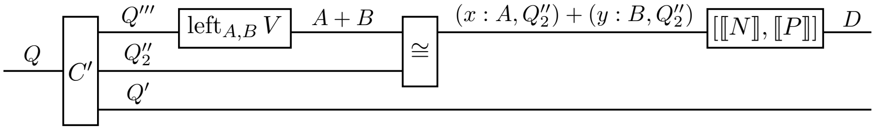
$$[\![(C, \text{case } M \text{ of } \{\text{left}\, x \to N \mid \text{right}\, y \to P\})]\!] =$$

=


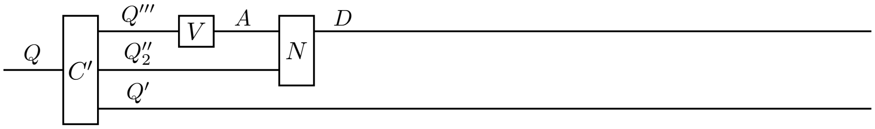
by def.

=


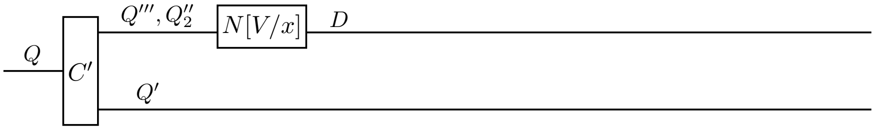
by def.

=


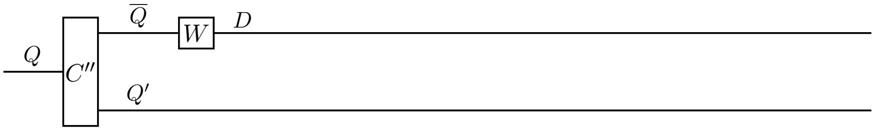
by (9.15)

=


by (9.16)

=


by (9.18)

=


by (9.17)

$= \; [\![(C'', W)]\!]$ by def.

- $(\textit{case-right})_{\Downarrow}$: This case is similar to the $(\textit{case-left})_{\Downarrow}$ case.
- $(\textit{app})_{\Downarrow}$: The evaluation rule is

$$\frac{(C, M) \Downarrow (C', \lambda x^A.M') \quad (C', N) \Downarrow (C'', V) \quad (C'', M'[V/x]) \Downarrow (C''', W)}{(C, MN) \Downarrow (C''', W)} \; .$$

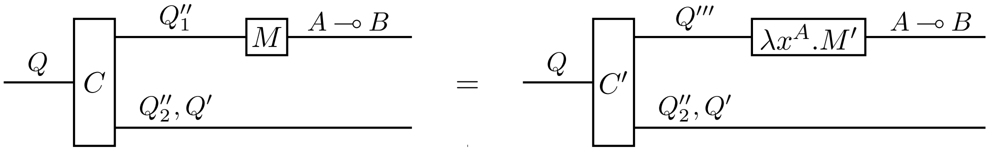
Assume that $Q \vdash (C, MN) : B; Q'$ is valid. We want to show that

$$[\![(C, MN)]\!] = [\![(C''', W)]\!] : [\![Q]\!] \to [\![B]\!] \otimes [\![Q']\!].$$

By assumption, there exists a label context Q'' such that $C : Q \to Q'', Q'$ and $\emptyset; Q'' \vdash MN : B$ is valid. By the typing rule (*app*), there are label contexts Q_1'' and Q_2'' such that $Q'' = Q_1'', Q_2''$, and both $\emptyset; Q_1'' \vdash M : A \multimap B$ and $\emptyset; Q_2'' \vdash N : A$ are valid.

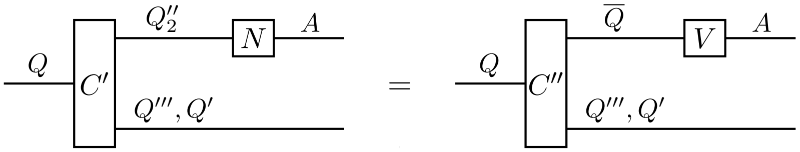
Since $C : Q \to Q_1'', Q_2'', Q'$ and $\emptyset; Q_1'' \vdash M : A \multimap B$ is valid, it follows that $Q \vdash (C, M) : A \multimap B; Q_2'', Q'$ is valid as well. By the Subject Reduction Theorem 6.4.2, $(C, M) \Downarrow (C', \lambda x^A.M')$ implies that the judgement

$Q \vdash (C', \lambda x^A.M') : A \multimap B; Q_2'', Q'$ is also valid. Thus, there exists a label context Q''' such that $C' : Q \to Q''', Q_2'', Q'$ and $\emptyset; Q''' \vdash \lambda x^A.M' : A \multimap B$ is valid. By the induction hypothesis, $(C, M) \Downarrow (C', \lambda x^A.M')$ implies that $\llbracket (C, M) \rrbracket = \llbracket (C', \lambda x^A.M') \rrbracket : \llbracket Q \rrbracket \to \llbracket A \multimap B \rrbracket \otimes \llbracket Q_2'', Q' \rrbracket$. Diagrammatically,



(9.19)

Since $C' : Q \to Q''', Q_2'', Q'$ and $\emptyset; Q_2'' \vdash N : A$ is valid, we have that $Q \vdash (C', N) : A; Q''', Q'$ is also valid. By the Subject Reduction Theorem 6.4.2, $(C', N) \Downarrow (C'', V)$ implies that $Q \vdash (C'', V) : A; Q''', Q'$ is valid too. Thus, there exists a label context $\overline{Q}$ such that $C'' : Q \to \overline{Q}, Q''', Q'$ and $\emptyset; \overline{Q} \vdash V : A$ is valid. By the induction hypothesis, $(C', N) \Downarrow (C'', V)$ implies that $\llbracket (C', N) \rrbracket = \llbracket (C'', V) \rrbracket : \llbracket Q \rrbracket \to \llbracket A \rrbracket \otimes \llbracket Q''', Q' \rrbracket$. Diagrammatically,



(9.20)

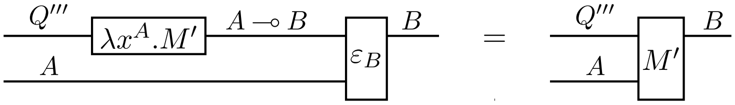
Now, since $\emptyset; Q''' \vdash \lambda x^A.M' : A \multimap B$ is valid, the typing rule (*abs*) implies that judgement $x : A; Q''' \vdash M' : B$ is also valid. By definition, $\llbracket \emptyset; Q''' \vdash \lambda x^A.M' : A \multimap B \rrbracket$ is the transpose of

$$\llbracket x : A; Q''' \vdash M' : B \rrbracket \circ \cong : \llbracket \emptyset; Q''' \rrbracket \otimes \llbracket A \rrbracket \to \llbracket x : A; Q''' \rrbracket \to \llbracket B \rrbracket$$

under the adjunction $- \otimes A \dashv A \multimap -$, and so makes the following diagram commute:

$$\begin{tikzcd}
 & \llbracket \emptyset; Q''' \rrbracket \otimes \llbracket A \rrbracket \arrow[d, "\cong"] \arrow[ddl, swap, "\llbracket \lambda^A.M' \rrbracket \otimes A"] \\
 & \llbracket x : A; Q''' \rrbracket \arrow[d, "\llbracket M' \rrbracket"] \\
\llbracket A \multimap B \rrbracket \otimes \llbracket A \rrbracket \arrow[r, swap, "\varepsilon_B"] & \llbracket B \rrbracket.
\end{tikzcd}$$

Diagrammatically,

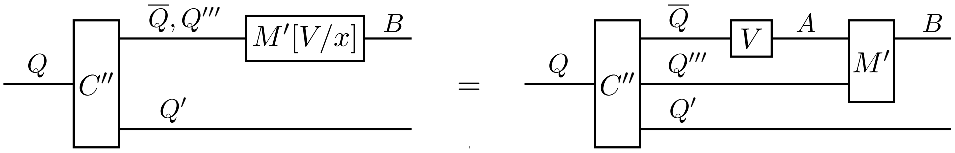


(9.21)

Also, since both typing judgements $x : A; Q''' \vdash M' : B$ and $\emptyset; \overline{Q} \vdash V : A$ are valid and $C'' : Q \to \overline{Q}, Q''', Q'$, the Semantic Substitution Lemma 8.6.1 implies that

$$[\![(C'', M'[V/x])]\!] = [\![(C'', \text{let } x = V \text{ in } M)]\!] : [\![Q]\!] \to [\![B]\!] \otimes [\![Q''', Q']\!].$$

Diagrammatically,

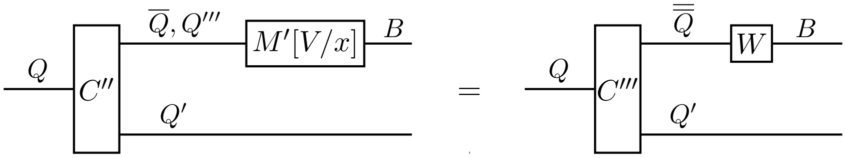


(9.22)

Similarly, $\emptyset; \overline{Q}, Q''' \vdash M'[V/x] : B$ is valid by the Substitution Lemma 6.3.1, and since $C'' : Q \to \overline{Q}, Q''', Q'$, we have that $Q \vdash (C'', M'[V/x]) : B; Q'$ is valid as well. By the Subject Reduction Theorem 6.4.2, we have that $(C'', M'[V/x]) \Downarrow (C''', W)$ implies that $Q \vdash (C''', W) : A; Q'$ is valid too. Thus, there exists a label context $\overline{\overline{Q}}$ such that $C'' : Q \to \overline{\overline{Q}}, Q'$ and $\emptyset; \overline{\overline{Q}} \vdash W : B$ is valid. Moreover, by the induction hypothesis, $(C'', M'[V/x]) \Downarrow (C''', W)$ implies that

$$[\![(C'', M'[V/x])]\!] = [\![(C''', W)]\!] : [\![Q]\!] \to [\![B]\!] \otimes [\![Q']\!].$$

Diagrammatically,



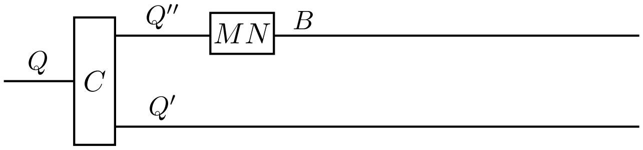
(9.23)

We can now prove that

$$[\![(C, MN)]\!] = [\![(C''', W)]\!] : [\![Q]\!] \to [\![B]\!] \otimes [\![Q']\!]$$

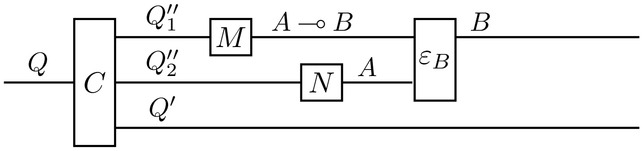
as follows:

$[\![(C, MN)]\!] =$



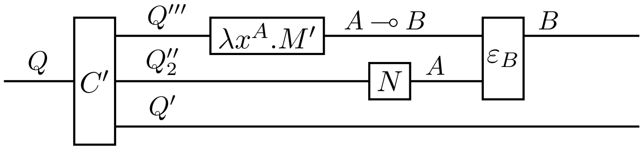
by def.

$=$



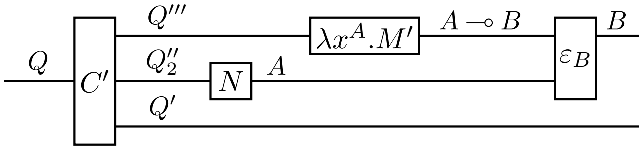
by def.

$=$



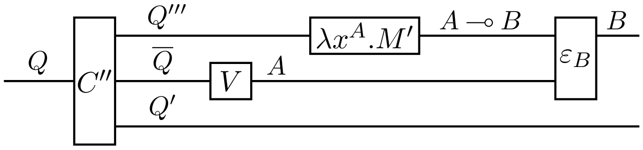
by (9.19)

$=$



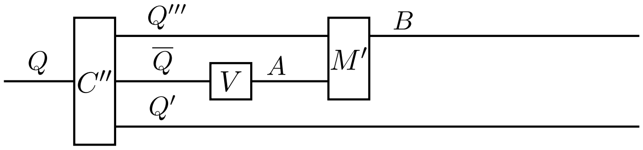
by funct.

$=$



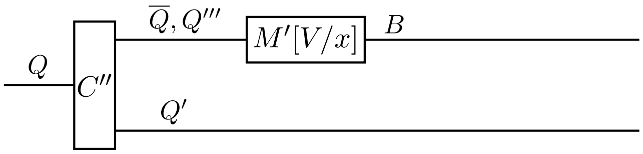
by (9.20)

$=$



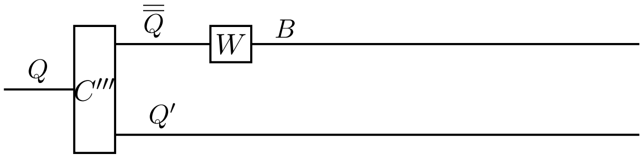
by (9.21)

$=$



by (9.22)

$=$



by (9.23)

$= \; [\![(C''', W)]\!]$ by def.

Circuit rules: These are the rules $(box)_\Downarrow$ and $(apply)_\Downarrow$, and they are the most interesting rules of Proto-Quipper-M as these are the ones that generate circuits. In these cases, the result follows by applications of the induction hypothesis, the Subject Reduction Theorem 6.4.2, and the adjunctions $- \otimes T \dashv T \multimap -$ and

$p \dashv \flat$. We treat each in turn:

- $(\mathit{box})_{\Downarrow}$: The evaluation rule is

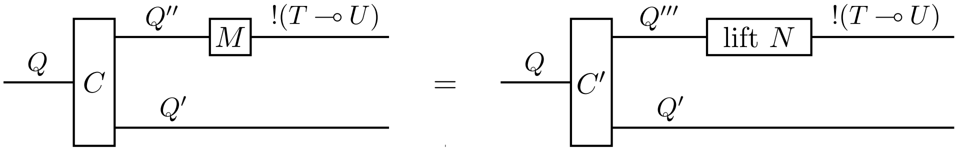
$$\frac{(C,M) \Downarrow (C',\text{lift } N) \quad \mathit{freshlabels}(T) = (\overline{Q},\vec{\ell}) \quad (\mathrm{id}_{\overline{Q}}, N\vec{\ell}) \Downarrow (D,\vec{\ell'})}{(C,\text{box}_T\, M) \Downarrow (C',(\vec{\ell},D,\vec{\ell'}))}.$$

Assume that $Q \vdash (C, \text{box}_T\, M) : A; Q'$ is valid. We want to show that

$$\llbracket (C,\text{box}_T\, M) \rrbracket = \llbracket (C',(\vec{\ell},D,\vec{\ell'})) \rrbracket : \llbracket Q \rrbracket \to \llbracket A \rrbracket \otimes \llbracket Q' \rrbracket.$$

By assumption, there exists a label context Q'' such that $C : Q \to Q'', Q'$ and $\emptyset; Q'' \vdash \text{box}_T\, M : A$ is valid. By the typing rule (box), $A = \mathrm{Circ}(T,U)$ for some simple M-type U, and $\emptyset; Q'' \vdash M : !(T \multimap U)$ is valid.

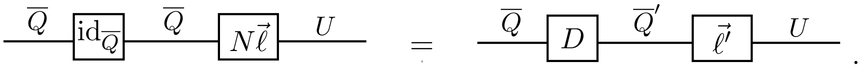
Since $C : Q \to Q'', Q'$, we have that $Q \vdash (C,M) : !(T \multimap U); Q'$ is valid as well. By the Subject Reduction Theorem 6.4.2, $(C,M) \Downarrow (C',\text{lift } N)$ implies that $Q \vdash (C',\text{lift } N) : !(T \multimap U); Q'$ is also valid. Thus, there exists a label context Q''' such that $C' : Q \to Q''', Q'$ and $\emptyset; Q''' \vdash \text{lift } N : !(T \multimap U)$ is valid. Since the judgement $Q \vdash (C,M) : !(T \multimap U); Q'$ is valid and $(C,M) \Downarrow (C',\text{lift } N)$, the induction hypothesis implies that the equation $\llbracket (C,M) \rrbracket = \llbracket (C',\text{lift } N) \rrbracket : \llbracket Q \rrbracket \to \llbracket !(T \multimap U) \rrbracket \otimes \llbracket Q' \rrbracket$ holds. Diagrammatically,



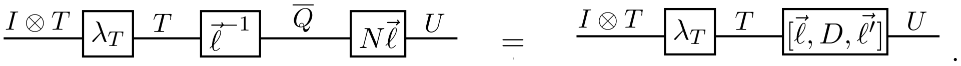
(9.24)

By the typing rule (lift), $Q''' = \emptyset$ and $\emptyset; \emptyset \vdash N : T \multimap U$ is valid. But $C' : Q \to \emptyset, Q'$, and so $Q \vdash (C',N) : T \multimap U; Q'$ is valid too. Since $\mathit{freshlabels}(T) = (\overline{Q},\vec{\ell})$, we have that $\overline{Q} \vdash_{\mathcal{L}} \vec{\ell} : T$ is valid, and by Lemma 6.2.3, $\emptyset; \overline{Q} \vdash \vec{\ell} : T$ is valid as well. Since $\emptyset; \emptyset \vdash N : T \multimap U$, the (app) rule implies that $\emptyset; \overline{Q} \vdash N\vec{\ell} : U$ is valid too.

Since $\mathrm{id}_{\overline{Q}} : \overline{Q} \to \overline{Q}, \emptyset$, the judgement $\overline{Q} \vdash (\mathrm{id}_{\overline{Q}}, N\vec{\ell}) : U; \emptyset$ is also valid. By the Subject Reduction Theorem 6.4.2, $(\mathrm{id}_{\overline{Q}}, N\vec{\ell}) \Downarrow (D,\vec{\ell'})$ implies that $\overline{Q} \vdash (D,\vec{\ell'}) : U; \emptyset$ is valid too. Thus, there exists a label context $\overline{Q}'$ such that $D : \overline{Q} \to \overline{Q}', \emptyset$ and $\emptyset; \overline{Q}' \vdash \vec{\ell}' : U$ is valid. Since $\overline{Q} \vdash (\mathrm{id}_{\overline{Q}}, N\vec{\ell}) : U; \emptyset$ is valid and $(\mathrm{id}_{\overline{Q}}, N\vec{\ell}) \Downarrow (D, \vec{\ell}')$, the induction hypothesis implies that the equation $\llbracket(\mathrm{id}_{\overline{Q}}, N\vec{\ell})\rrbracket = \llbracket(D, \vec{\ell}')\rrbracket : \llbracket\overline{Q}\rrbracket \to \llbracket U\rrbracket \otimes \llbracket\emptyset\rrbracket$ holds. Diagrammatically,



But this along with definitions (7.12), (7.13), and (7.14) imply that



To simply notation, let $\underline{N} = \llbracket\emptyset; \emptyset \vdash N : T \multimap U\rrbracket \circ \lambda_I^{-1}$, $\underline{D} = [\vec{\ell}, D, \vec{\ell}'] \circ \lambda_T$, and $\underline{N\vec{\ell}} = \llbracket\emptyset; \overline{Q} \vdash N\vec{\ell} : U\rrbracket \circ \llbracket\emptyset; \overline{Q} \vdash \vec{\ell} : T\rrbracket^{-1} \circ \lambda_T$. Thus,

$$\underline{N\vec{\ell}} = \underline{D}. \tag{9.25}$$

Since $- \otimes T \dashv T \multimap -$, the transpose $\underline{N}_*$ of $\underline{N} : \llbracket I\rrbracket \to \llbracket T\rrbracket \multimap \llbracket U\rrbracket$ makes the following diagram commute:

$$\begin{tikzcd}
 & \llbracket I\rrbracket \otimes \llbracket T\rrbracket \arrow[dl, "\underline{N}\otimes T"'] \arrow[d, "\underline{N}_*"] \\
(\llbracket T\rrbracket \multimap \llbracket U\rrbracket) \otimes \llbracket T\rrbracket \arrow[r, "\varepsilon_U"'] & \llbracket U\rrbracket.
\end{tikzcd} \tag{9.26}$$

By definition of $\llbracket\emptyset; \overline{Q} \vdash N\vec{\ell} : U\rrbracket$, the following diagram commutes as well:

$$\begin{tikzcd}
\llbracket\emptyset; \overline{Q}\rrbracket \arrow[rr, "\llbracket\emptyset;\overline{Q}\vdash N\vec{\ell}:U\rrbracket"] \arrow[d, "\triangle"'] & & \llbracket U\rrbracket \\
\llbracket\emptyset; \emptyset\rrbracket \otimes \llbracket\emptyset; \overline{Q}\rrbracket \arrow[r, "\mathrm{id}\otimes\llbracket\vec{\ell}\rrbracket"'] & \llbracket\emptyset; \emptyset\rrbracket \otimes \llbracket T\rrbracket \arrow[r, "\llbracket N\rrbracket\otimes\mathrm{id}"'] & \llbracket T \multimap U\rrbracket \otimes \llbracket T\rrbracket \arrow[u, "\varepsilon_U"']
\end{tikzcd} \tag{9.27}$$

and since the variable contexts of $\emptyset; \emptyset \vdash N : T \multimap U$ and $\emptyset; \overline{Q} \vdash \vec{\ell} : T$ are empty,

$$\begin{tikzcd}
\llbracket \emptyset ; \overline{Q} \rrbracket \arrow[rr, "◬"] \arrow[d, "\lambda_I^{-1} \otimes \mathrm{id}"'] & & \llbracket \emptyset ; \emptyset \rrbracket \otimes \llbracket \emptyset ; \overline{Q} \rrbracket \\
\llbracket \emptyset ; \emptyset \rrbracket \otimes \llbracket \overline{Q} \rrbracket \arrow[urr, "\mathrm{id} \otimes \lambda_{\overline{Q}}^{-1}"'] & &
\end{tikzcd} \tag{9.28}$$

commutes too.

By coherence for symmetric monoidal categories, the following diagram commutes:

$$\begin{tikzcd}
\llbracket \emptyset ; \overline{Q} \rrbracket \arrow[r, "\lambda_I^{-1} \otimes \mathrm{id}"] & \llbracket \emptyset ; \emptyset \rrbracket \otimes \llbracket \overline{Q} \rrbracket \arrow[r, "\mathrm{id} \otimes \lambda_{\overline{Q}}^{-1}"] & \llbracket \emptyset ; \emptyset \rrbracket \otimes \llbracket \emptyset ; \overline{Q} \rrbracket \arrow[d, "\mathrm{id} \otimes \llbracket \vec{\ell} \rrbracket"] \\
\llbracket T \rrbracket \arrow[u, "\llbracket \vec{\ell} \rrbracket^{-1}"] \arrow[r, "\lambda_T^{-1}"'] & \llbracket \emptyset \rrbracket \otimes \llbracket T \rrbracket \arrow[r, "\lambda_I^{-1} \otimes \mathrm{id}"'] & \llbracket \emptyset ; \emptyset \rrbracket \otimes \llbracket T \rrbracket .
\end{tikzcd} \tag{9.29}$$

Consider

$$\begin{tikzcd}
\llbracket \emptyset ; \overline{Q} \rrbracket \arrow[rr, "◬"] & & \llbracket \emptyset ; \emptyset \rrbracket \otimes \llbracket \emptyset ; \overline{Q} \rrbracket \arrow[d, "\mathrm{id}"] \\
\llbracket \emptyset ; \overline{Q} \rrbracket \arrow[u, "\mathrm{id}"] \arrow[r, "\lambda_I^{-1} \otimes \mathrm{id}"] & \llbracket \emptyset ; \emptyset \rrbracket \otimes \llbracket \overline{Q} \rrbracket \arrow[r, "\mathrm{id} \otimes \lambda_{\overline{Q}}^{-1}"] & \llbracket \emptyset ; \emptyset \rrbracket \otimes \llbracket \emptyset ; \overline{Q} \rrbracket \arrow[d, "\mathrm{id} \otimes \llbracket \vec{\ell} \rrbracket"] \\
\llbracket T \rrbracket \arrow[u, "\llbracket \vec{\ell} \rrbracket^{-1}"] \arrow[r, "\lambda_T^{-1}"] & \llbracket I \rrbracket \otimes \llbracket T \rrbracket \arrow[r, "\lambda_I^{-1} \otimes \mathrm{id}"] \arrow[dr, "\underline{N} \otimes T"'] & \llbracket \emptyset ; \emptyset \rrbracket \otimes \llbracket T \rrbracket \arrow[d, "\llbracket N \rrbracket \otimes \mathrm{id}"] \\
 & & (\llbracket T \rrbracket \multimap \llbracket U \rrbracket) \otimes \llbracket T \rrbracket \arrow[d, "\varepsilon_U"] \\
\llbracket I \rrbracket \otimes \llbracket T \rrbracket \arrow[uu, "\lambda_T"] \arrow[uur, "\mathrm{id}"'] \arrow[rr, "\underline{N}_*"'] & & \llbracket U \rrbracket
\end{tikzcd}$$

The top square commutes by (9.28), and the one below by (9.29). The commutativity of the left triangle is immediate. The right triangle commutes by definition of $\underline{N}$ and the functoriality of $- \otimes T$; the bottom region does by (9.26). Hence, the outer square commutes as well. By (9.27), the top path yields the morphism $[\![N\vec{\ell}]\!] \circ [\![\vec{\ell}]\!]^{-1} \circ \lambda_T = \underline{N\vec{\ell}}$, and the bottom $\underline{N}_*$. Hence, $\underline{N\vec{\ell}} = \underline{N}_*$, and so by (9.25), the transpose $\underline{D}^*$ of $\underline{D} : [\![I]\!] \otimes [\![T]\!] \to [\![U]\!]$ under the adjunction $- \otimes T \dashv T \multimap -$ satisfies

$$\underline{D}^* = (\underline{N\vec{\ell}})^* = (\underline{N}_*)^* = \underline{N} = [\![N]\!] \circ \lambda_I^{-1} : [\![I]\!] \to ([\![T]\!] \multimap [\![U]\!]).$$

Similarly, the transpose $(\underline{D}^* \circ \tau_I^{-1})^\circ$ of

$$\underline{D}^* \circ \tau_I^{-1} : p([\![I]\!]_p) \to [\![I]\!] \to ([\![T]\!] \multimap [\![U]\!])$$

under the adjunction $p \dashv \flat$ satisfies

$$(\underline{D}^* \circ \tau_I^{-1})^\circ = ([\![N]\!] \circ \lambda_I^{-1} \circ \tau_I^{-1})^\circ : [\![I]\!]_p \to \flat([\![T]\!] \multimap [\![U]\!]).$$

This in turn implies that the following diagram commutes:

$$\begin{tikzcd}
 & & p([\![I]\!]_p) \arrow[r, "p((\underline{D}^* \circ \tau_I^{-1})^\circ)"] & p(\flat(T \multimap U)) \arrow[d, "\mathbf{box}"] \\
{[\![\emptyset; \emptyset]\!]} \arrow[r, "\rho_\emptyset"] & {[\![\emptyset]\!]} \arrow[r, "\Diamond_\emptyset = \mathrm{id}"] \arrow[dr, "\tau_\emptyset"'] & {[\![\emptyset]\!] = [\![I]\!]} \arrow[u, "\tau_I"] \arrow[d, "\tau_I"'] & {[\![\mathrm{Circ}(T, U)]\!]} \\
 & & p([\![I]\!]_p) \arrow[r, "p(([\![N]\!] \circ \lambda_I^{-1} \circ \tau_I^{-1})^\circ)"'] & p(\flat(T \multimap U)). \arrow[u, "\mathbf{box}"']
\end{tikzcd}$$

By (7.16), the top path is the morphism

$$\mathbf{circ}\,(\vec{\ell}, D, \vec{\ell'}) \circ \Diamond_\emptyset \circ \rho_\emptyset = [\![\emptyset; \emptyset \vdash (\vec{\ell}, D, \vec{\ell'}) : \mathrm{Circ}(T, U)]\!],$$

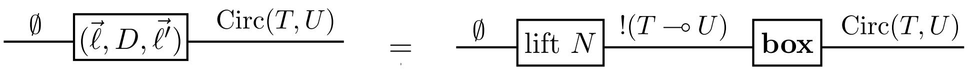
while the bottom path is

$$\mathbf{box} \circ \mathbf{lift}\, [\![N]\!] \circ \tau_\emptyset \circ \rho_\emptyset = \mathbf{box} \circ [\![\emptyset; \emptyset \vdash \text{lift } N : !(T \multimap U)]\!]$$

by (7.17). Thus,

$$[\![\emptyset; \emptyset \vdash (\vec{\ell}, D, \vec{\ell'}) : \mathrm{Circ}(T, U)]\!] = \mathbf{box} \circ [\![\emptyset; \emptyset \vdash \text{lift } N : !(T \multimap U)]\!].$$

Diagrammatically,



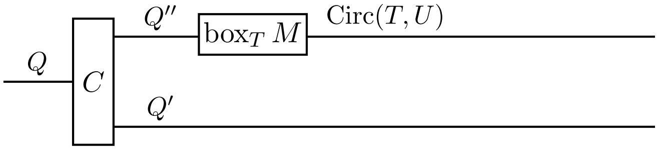
(9.30)

We can now show that

$$[\![(C, \mathrm{box}_T\, M)]\!] = [\![(C', (\vec{\ell}, D, \vec{\ell'}))]\!] : [\![Q]\!] \to [\![\mathrm{Circ}(T,U)]\!] \otimes [\![Q']\!]$$

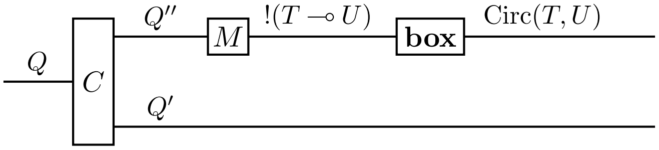
as follows:

$[\![(C, \mathrm{box}_T\, M)]\!] =$



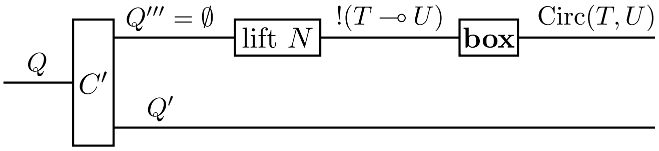
by def.

$=$



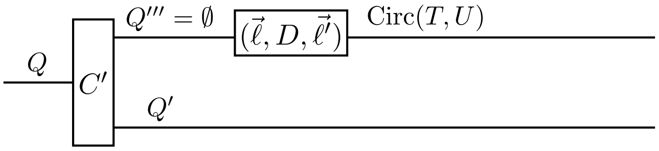
by def.

$=$



by (9.24)

$=$



by (9.30)

$= \ [\![(C', (\vec{\ell}, D, \vec{\ell'}))]\!]$ by def.

- $(\mathit{apply})_{\Downarrow}$: The evaluation rule is

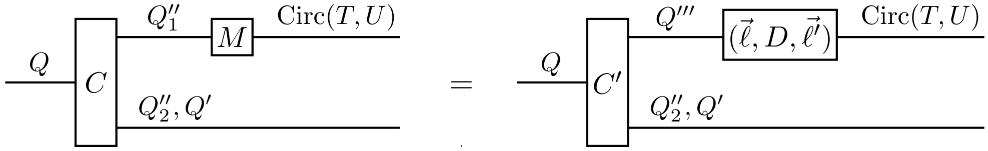
$$\frac{(C, M) \Downarrow (C', (\vec{\ell}, D, \vec{\ell'})) \quad (C', N) \Downarrow (C'', \vec{k}) \quad \mathit{append}(C'', \vec{k}, \vec{\ell}, D, \vec{\ell'}) = (C''', \vec{k'})}{(C, \mathrm{apply}(M, N)) \Downarrow (C''', \vec{k'})}$$

Assume that $Q \vdash (C, \mathrm{apply}(M,N)) : U; Q'$ is valid. We want to show

$$[\![(C, \mathrm{apply}(M,N))]\!] = [\![(C''', \vec{k'})]\!] : [\![Q]\!] \to [\![U]\!] \otimes [\![Q']\!].$$

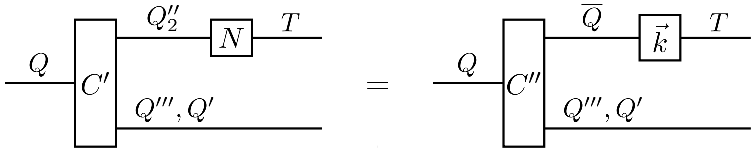
By assumption, there exists a label context Q'' such that $C : Q \to Q'', Q'$ and $\emptyset; Q'' \vdash \mathrm{apply}(M,N) : U$ is valid. By the (apply) rule, there exist label contexts Q''_1 and Q''_2 such that $Q'' = Q''_1, Q''_2$ and both $\emptyset; Q''_1 \vdash M : \mathrm{Circ}(T,U)$ and $\emptyset; Q''_2 \vdash N : T$ are valid.

Since $C : Q \to Q_1'', Q_2'', Q'$ and $\emptyset; Q_1'' \vdash M : \mathrm{Circ}(T, U)$, it follows that $Q \vdash (C, M) : \mathrm{Circ}(T, U); Q_2'', Q'$ is valid as well. By the Subject Reduction Theorem 6.4.2, $(C, M) \Downarrow (C', (\vec{\ell}, D, \vec{\ell'}))$ yields the validity of the judgement $Q \vdash (C', (\vec{\ell}, D, \vec{\ell'})) : \mathrm{Circ}(T, U); Q_2'', Q'$. Thus, there is a label context Q''' such that $C' : Q \to Q''', Q_2'', Q'$ and $\emptyset; Q''' \vdash (\vec{\ell}, D, \vec{\ell'}) : \mathrm{Circ}(T, U)$ is valid. By the induction hypothesis, $(C, M) \Downarrow (C', (\vec{\ell}, D, \vec{\ell'}))$ implies that $[\![(C, M)]\!] = [\![(C', (\vec{\ell}, D, \vec{\ell'}))]\!] : [\![Q]\!] \to [\![\mathrm{Circ}(T, U)]\!] \otimes [\![Q_2'', Q']\!]$. Diagrammatically,



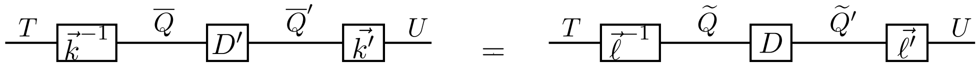
(9.31)

Since $C' : Q \to Q''', Q_2'', Q'$ and $\emptyset; Q_2'' \vdash N : T$ is valid, we have that $Q \vdash (C', N) : T; Q''', Q'$ is valid as well. By the Subject Reduction Theorem 6.4.2, $(C', N) \Downarrow (C'', \vec{k})$ yields that $Q \vdash (C'', \vec{k}) : T; Q''', Q'$ is valid too. Thus, there exists a label context $\overline{Q}$ such that $C'' : Q \to \overline{Q}, Q''', Q'$ and $\emptyset; \overline{Q} \vdash \vec{k} : T$ is valid. By the induction hypothesis, $(C', N) \Downarrow (C'', \vec{k})$ implies that $[\![(C', N)]\!] = [\![(C'', \vec{k})]\!] : [\![Q]\!] \to [\![T]\!] \otimes [\![Q''', Q']\!]$. Diagrammatically,



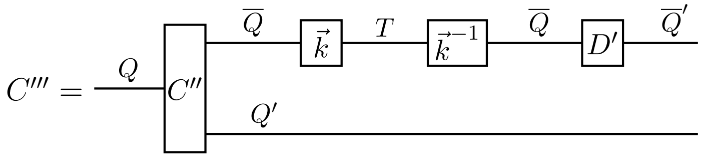
(9.32)

Since $\emptyset; Q''' \vdash (\vec{\ell}, D, \vec{\ell'}) : \mathrm{Circ}(T, U)$ is valid, the (*circ*) rule implies that $Q''' = \emptyset$, that there exist label contexts $\widetilde{Q}$ and $\widetilde{Q}'$ such that $D \in \mathbf{M}_{\mathcal{L}}(\widetilde{Q}, \widetilde{Q}')$, and that both $\widetilde{Q} \vdash_{\mathcal{L}} \vec{\ell} : T$ and $\widetilde{Q}' \vdash_{\mathcal{L}} \vec{\ell'} : U$ are valid. Also, since $\mathit{append}(C'', \vec{k}, \vec{\ell}, D, \vec{\ell'}) = (C''', \vec{k'})$, there is a boxed circuit $(\vec{k}, D', \vec{k'})$ equivalent to $(\vec{\ell}, D, \vec{\ell'})$, where $D' : \overline{Q} \to \overline{Q}'$ is the labeled circuit derived from $D : \widetilde{Q} \to \widetilde{Q}'$ with $\overline{Q}$ being the label context obtained by the relabeling of $\vec{\ell}$ by $\vec{k}$, and $\overline{Q}'$ being the label context obtained by the relabeling of $\vec{\ell'}$ by fresh $\vec{k'}$. Diagrammatically,

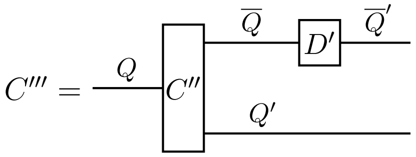


(9.33)

Also, by definition of *append*,



That is,



(9.34)

Let $\underline{D} = [\vec{\ell}, D, \vec{\ell'}] \circ \lambda_T$. Since $- \otimes T \dashv T \multimap -$, the transpose $\underline{D}_*$ of the morphism $\underline{D} : \llbracket I \rrbracket \otimes \llbracket T \rrbracket \to \llbracket U \rrbracket$ makes the following diagram commute:

$$\begin{tikzcd}
 & \llbracket I \rrbracket \otimes \llbracket T \rrbracket \arrow[dl, "\underline{D}_* \otimes T"'] \arrow[d, "\underline{D}"] \\
(\llbracket T \rrbracket \multimap \llbracket U \rrbracket) \otimes \llbracket T \rrbracket \arrow[r, "\varepsilon_U"'] & \llbracket U \rrbracket
\end{tikzcd} \tag{9.35}$$

Similarly, the transpose $(\underline{D}_* \circ \tau_I^{-1})^\circ$ of $\underline{D}_* \circ \tau_I^{-1}$ under the adjunction $p \dashv \flat$ makes the following diagram commute:

$$\begin{tikzcd}
 & p(\llbracket I \rrbracket_p) \arrow[ddl, "p((\underline{D}_* \circ \tau_I^{-1})^\circ)"'] \arrow[d, "\tau_I^{-1}"] \\
 & \llbracket I \rrbracket \arrow[d, "\underline{D}_*"] \\
!(\llbracket T \rrbracket \multimap \llbracket U \rrbracket) \arrow[r, "\mathbf{force}"'] & \llbracket T \rrbracket \multimap \llbracket U \rrbracket
\end{tikzcd}$$

(9.36)

Consider

$$\begin{tikzcd}
 & \llbracket \mathrm{Circ}(T,U) \rrbracket \otimes \llbracket T \rrbracket \arrow[r, "\mathbf{unbox} \otimes T"] & !(\llbracket T \rrbracket \multimap \llbracket U \rrbracket) \otimes \llbracket T \rrbracket \arrow[d, "\mathbf{force} \otimes T"] \\
p(\llbracket I \rrbracket_p) \otimes \llbracket T \rrbracket \arrow[r, "p((\underline{D}_* \circ \tau_I^{-1})^\circ) \otimes T"] & !(\llbracket T \rrbracket \multimap \llbracket U \rrbracket) \otimes \llbracket T \rrbracket \arrow[u, "\mathbf{box} \otimes T"] \arrow[ur, "\mathrm{id}"] & (\llbracket T \rrbracket \multimap \llbracket U \rrbracket) \otimes \llbracket T \rrbracket \arrow[ddd, "\varepsilon_U"] \\
\llbracket I \rrbracket \otimes \llbracket T \rrbracket \arrow[u, "\tau_I \otimes T = \mathrm{id}"] \arrow[r, "\tau_I^{-1} \otimes T = \mathrm{id}"] & \llbracket I \rrbracket \otimes \llbracket T \rrbracket \arrow[ur, "\underline{D}_* \otimes T"] \arrow[dd, "\lambda_T"'] \arrow[ddr, "\underline{D}"] & \\
\llbracket \emptyset \rrbracket \otimes \llbracket T \rrbracket \arrow[u, "\Diamond_\emptyset \otimes T = \mathrm{id}"] & & \\
\llbracket \emptyset ; \emptyset \rrbracket \otimes \llbracket T \rrbracket \arrow[u, "\rho_\emptyset \otimes T"] \arrow[r, "\lambda_T \circ (\rho_\emptyset \otimes T)"'] & \llbracket T \rrbracket \arrow[r, "{[\vec{\ell}, D, \vec{\ell'}]}"'] & \llbracket U \rrbracket
\end{tikzcd}$$

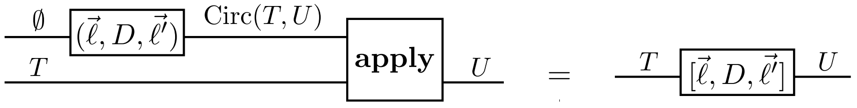
(9.37)

The top triangle commutes since **box** and **unbox** are inverses of each other. The middle region commutes by (9.36) and the functoriality of $-\otimes T$. The rightmost triangle commutes by (9.35), and the one below by definition of $\underline{D}$. The commutativity of the bottom square is immediate. Thus, the outer region commutes as well, and so the top and bottom paths are equal.

Recall that by (7.11), $\mathbf{apply} = \varepsilon_U \circ (\mathbf{force} \otimes T) \circ (\mathbf{unbox} \otimes T)$; by (7.16), $\mathbf{circ}\,(\vec{\ell}, D, \vec{\ell'}) = \mathbf{box} \circ p((\underline{D}_* \circ \tau_I^{-1})^\circ) \circ \tau_I$; and by definition,

$$\llbracket \emptyset ; \emptyset \vdash (\vec{\ell}, D, \vec{\ell'}) : \mathrm{Circ}(T,U) \rrbracket = \mathbf{circ}\,(\vec{\ell}, D, \vec{\ell'}) \circ \Diamond_\emptyset \circ \rho_\emptyset .$$

Hence, the top path of (9.37) is $\mathbf{apply} \circ (\llbracket \emptyset ; \emptyset \vdash (\vec{\ell}, D, \vec{\ell'}) : \mathrm{Circ}(T,U) \rrbracket \otimes T)$, which is equal to the bottom path, $[\vec{\ell}, D, \vec{\ell'}] \circ \lambda_T \circ (\rho_\emptyset \otimes T)$. Diagrammatically,

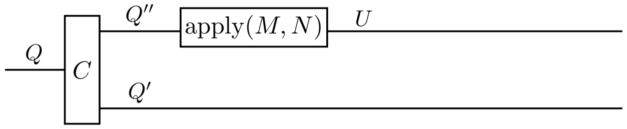

∅
$(\vec{\ell}, D, \vec{\ell'})$
Circ(T, U)
T
apply
U
=
T
$[\vec{\ell}, D, \vec{\ell'}]$
U

(9.38)

We can now prove that

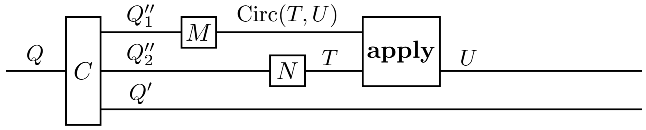
$$[\![(C,\mathrm{apply}(M,N))]\!] = [\![(C''',\vec{k'})]\!] : [\![Q]\!] \to [\![U]\!] \otimes [\![Q']\!] :$$

$[\![(C,\mathrm{apply}(M,N))]\!] =$



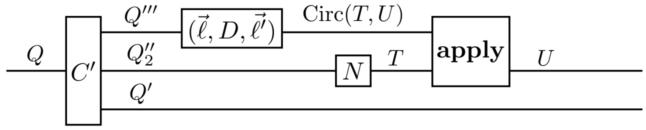
by def.

$=$



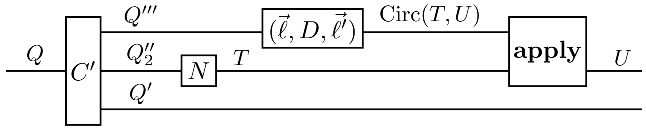
by def.

$=$



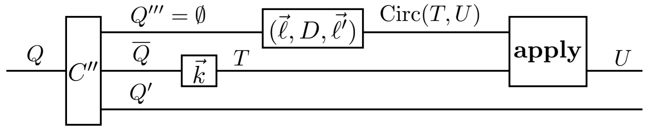
by (9.31)

$=$



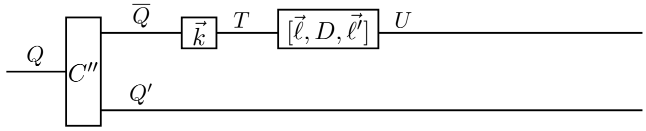
by funct.

$=$



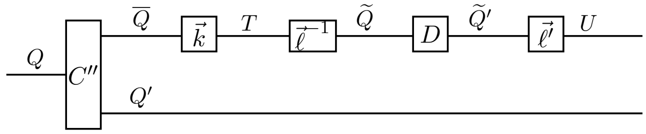
by (9.32)

$=$



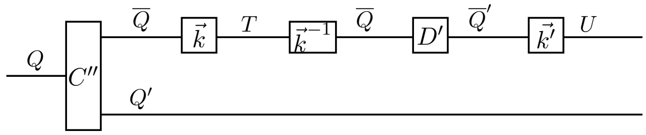
by (9.38)

$=$



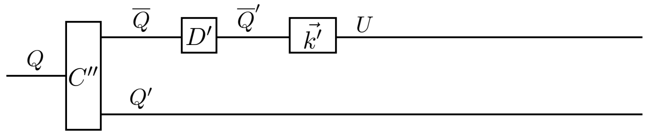
by def.

$=$



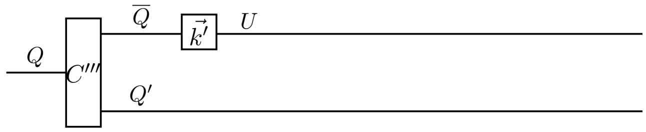
by (9.33)

$=$



by inv.

$=$



by (9.34)

$= \ [\![(C''',\vec{k'})]\!]$ by def.

□

Chapter 10

Conclusion

In this thesis, we developed a new quantum programming language and investigated its mathematical semantics with the goal of providing a foundation for formal reasoning about quantum programs. This is a nontrivial enterprise since a reliable passage from quantum algorithms to machine-readable code is rather intricate, and so are the languages designed to enable such transitions. This problem is moreover aggravated by the nature of quantum information, which renders some of the classical techniques used to analyze programs inadequate. We contributed to the solution of some of these problems by introducing Proto-Quipper-M, a type-safe, categorically sound quantum programming language for describing families of generalized circuits.

We devised the language as a simply-typed lambda calculus with a designated type for generalized circuits. To prevent violations of the no-cloning property, Proto-Quipper-M's strong type system imposes the linear use of quantum data. We gave the language computational meaning by equipping it with a big-step operational semantics and showed that it satisfies the subject-reduction and error-freeness properties, thus guaranteeing its type-safety. The categorical semantics of Proto-Quipper-M was given in an appropriate class of monoidal categories, which we showed yield linear-non-linear models, and so models of intuitionistic linear logic. Finally, we proved the soundness theorem for Proto-Quipper-M, thus solidifying the connection between the syntax and the semantics of the language.

Several possible directions for future work can be identified. Here, we just present a few. One possibility is exploring the relationship between Proto-Quipper-M and Proto-Quipper-S [90]. Since Proto-Quipper-M is the first version of Proto-Quipper with a categorical semantics rich enough to model the ! modality of linear logic, "lift" and "force" operations are required to transform terms of linear type into terms of non-linear type and vice versa. This makes programming in Proto-Quipper-M inconvenient. On the other hand, Proto-Quipper-S uses subtyping to determine the linearity of its programs, which makes programming in Proto-Quipper-S less cumbersome. Thus, it would be interesting to find translations between these two languages.

Another issue that is worthwhile addressing is the lack of a type-safe mechanism to handle invertible circuits in Proto-Quipper-M. This is relevant since quantum circuits are not necessarily invertible. For example, those composed exclusively of unitary gates are invertible while those containing measurements are not, and so trying to invert a non-invertible circuit would yield a run-time error. To avoid this type of error, the type system should be enhanced to ensure the proper use of an "invert" operation: The "invert" operation could be applied to a circuit only if the circuit is indeed invertible.

Yet another interesting question to consider in the effort to close the gap between Proto-Quipper-M and Quipper is that of dynamic lifting. Dynamic lifting is an operation in Quipper that turns a state into a parameter, thus enabling circuit generation to depend on results obtained during circuit execution. This is indeed useful since some quantum algorithms require the classical computer and the quantum device to interact in such a way that the measurements coming from the execution of a partially constructed circuit are necessary to determine how the construction of the rest of the circuit should proceed. How to extend the type system of Proto-Quipper-M to enable the dynamic lifting operation in a type-safe manner is an open question.

Bibliography

[1] Scott Aaronson. *Quantum Computing Since Democritus*. Cambridge University Press, 2013.

[2] Ali J. Abhari, Arvin Faruque, Mohammad J. Dousti, Lukas Svec, Oana Catu, Amlan Chakrabati, Chen-Fu Chiang, Seth Vanderwilt, John Black, and Fred Chong. Scaffold: Quantum programming language. Technical report, Princeton Univ NJ Dept of Computer Science, 2012.

[3] Samson Abramsky and Nikos Tzevelekos. Introduction to categories and categorical logic. In *New Structures for Physics*, pages 3–94. Springer, 2010.

[4] Dorit Aharonov. A simple proof that Toffoli and Hadamard are quantum universal, 2003. Preprint available from `arXiv:quant-ph/0301040`.

[5] Dorit Aharonov, Wim Van Dam, Julia Kempe, Zeph Landau, Seth Lloyd, and Oded Regev. Adiabatic quantum computation is equivalent to standard quantum computation. *SIAM review*, 50(4):755–787, 2008.

[6] D. S. Alexander, Neil J. Ross, P. Selinger, J.M. Smith, and B. Valiron. Programming the quantum future. *Communications of the ACM*, 2015.

[7] Thorsten Altenkirch and Jonathan Grattage. A functional quantum programming language. In *Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science (LICS 2005)*, pages 249–258. IEEE Computer Society Press, 2005.

[8] A. Ambainis, A. M. Childs, B. W. Reichardt, R. Špalek, and S. Zhang. Any AND-OR formula of size n can be evaluated in time $n^{\frac{1}{2}+o(1)}$ on a quantum computer. *SIAM J. Comput.*, 39:2513—-2530, 2010.

[9] Juan Miguel Arrazola, Timjan Kalajdzievski, Christian Weedbrook, and Seth Lloyd. Quantum algorithm for nonhomogeneous linear partial differential equations. *Physical Review A*, 100(3):032306, 2019.

[10] Peter J. Ashenden. *The Designer's Guide to VHDL*. Morgan Kaufmann, 2010.

[11] Steve Awodey. *Category Theory*. Oxford University Press, 2nd edition, 2010.

[12] Andrew G. Barber. Dual intuitionistic linear logic. Technical Report ECS-LFCS-96-347, Department of Computer Science, University of Edinburgh, 1996.

[13] Adriano Barenco, Charles H Bennett, Richard Cleve, David P DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52(5):3457, 1995.

[14] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* North-Holland Linguistic Series. North-Holland, 1984.

[15] Michael Barr and Charles Wells. *Category Theory for Computing Science.* Les Publications CRM, 3rd edition, 1999.

[16] 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, 1980.

[17] Nick Benton. A mixed linear and non-linear logic: Proofs, terms and models (extended abstract). In *Proceedings of the 8th Workshop on Computer Science Logic, CSL'94, Selected Papers*, Springer Lecture Notes in Computer Science 933, pages 121–135, 1995.

[18] Nick Benton, Gavin Bierman, Valeria de Paiva, and Martin Hyland. Linear λ-calculus and categorical models revisited. In E. Börger, G. Jäger, H. Kleine Büning, S. Martini, and M. M. Richter, editors, *Computer Science Logic*, pages 61–84, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[19] Nick Benton, Gavin Bierman, Valeria de Paiva, and Martin Hyland. A term calculus for intuitionistic linear logic. In Marc Bezem and Jan Friso Groote, editors, *Typed Lambda Calculi and Applications*, pages 75–90, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[20] Nick Benton, Gavin Bierman, Valeria Paiva, and Martin Hyland. Term assignment for intuitionistic linear logic (preliminary report). Technical report, University of Cambridge, Computer Laboratory, August 1992.

[21] Dominic W. Berry. High-order quantum algorithm for solving linear differential equations. *Journal of Physics A: Mathematical and Theoretical*, 47(10):105301, 2014.

[22] S. Bettelli, T. Calarco, and L. Serafini. Toward an architecture for quantum programming, 2001. Available from `arXiv:cs/0103009`.

[23] G.M. Bierman. On intuitionistic linear logic. Technical Report UCAM-CL-TR-346, University of Cambridge, Computer Laboratory, August 1994.

[24] Alex Bocharov, Yuri Gurevich, and Krysta M. Svore. Efficient decomposition of single-qubit gates into V basis circuits. *Phys. Rev. A*, 88:012313 (13 pages), 2013. Also available from `arXiv:1303.1411`.

[25] Francis Borceux. *Handbook of Categorical Algebra 1: Basic Category Theory*, volume 1. Cambridge University Press, 1994.

[26] Francis Borceux. *Handbook of Categorical Algebra 2: Categories and Structures*, volume 2. Cambridge University Press, 1994.

[27] Luca Cardelli. Type systems. In Allen B. Tucker, editor, *Handbook of Computer Science and Engineering*, chapter 97. CRC Press, 2nd edition, 1997.

[28] A. M. Childs, R. Cleve, E. Deotto, E. Farhi, S. Gutmann, and D. A. Spielman. Exponential algorithmic speedup by a quantum walk. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, pages 59–68, 2003.

[29] Andrew M. Childs and Tongyang Li. Efficient simulation of sparse Markovian quantum dynamics, 2016. Preprint available from `arXiv:1611.05543`.

[30] Frederic T. Chong, Diana Franklin, and Margaret Martonosi. Programming languages and compiler design for realistic quantum hardware. *Nature*, 549(7671):180–187, 2017.

[31] Dominique Clément, Thierry Despeyroux, Gilles Kahn, and Joëlle Despeyroux. A simple applicative language: Mini-ML. In *Proceedings of the 1986 ACM Conference on LISP and Functional Programming*, LFP '86, page 13–27, New York, NY, USA, 1986. Association for Computing Machinery.

[32] Richard Cleve. An introduction to quantum complexity theory. *Collected Papers on Quantum Computation and Quantum Information Theory*, pages 103–127, 2000.

[33] Roy L. Crole. *Categories for Types.* Cambridge University Press, 1993.

[34] Andrew W. Cross, Lev S. Bishop, John A Smolin, and Jay M Gambetta. Open quantum assembly language, 2017. Available from `arXiv:1707.03429`.

[35] David Deutsch. Quantum theory, the Church–Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 400(1818):97–117, 1985.

[36] David Elieser Deutsch. Quantum computational networks. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 425(1868):73–90, 1989.

[37] DGBJ Dieks. Communication by EPR devices. *Physics Letters A*, 92(6):271–272, 1982.

[38] Vedran Dunjko, Jacob M. Taylor, and Hans J. Briegel. Quantum-enhanced machine learning. *Physical Review Letters*, 117(13):130501, 2016.

[39] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. Quantum computation by adiabatic evolution, 2000. Available from `arXiv:quant-ph/0001106`.

[40] Richard P. Feynman. Simulating physics with computers. *Int. J. Theor. Phys*, 21(6/7), 1982.

[41] Michael H. Freedman, Alexei Kitaev, and Zhenghan Wang. Simulation of topological field theories by quantum computers. *Communications in Mathematical Physics*, 227(3):587–603, 2002.

[42] Stephen H. Friedberg, Arnold J. Insel, and Lawrence E. Spence. *Linear Algebra*. Prentice Hall, 3rd edition, 1997.

[43] Simon J. Gay. Quantum programming languages: Survey and bibliography. *Mathematical Structures in Computer Science*, 16(4):581, 2006.

[44] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987.

[45] Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and Types*, volume 7. Cambridge University Press, 1989.

[46] Alexander Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. An introduction to quantum programming in Quipper. In *Proceedings of the 5th International Conference on Reversible Computation*, volume 7948 of *Lecture Notes in Computer Science*, pages 110–124, 2013. Preprint available from `arXiv:1304.5485`.

[47] Alexander Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. Quipper: a scalable quantum programming language. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, pages 333–342, 2013. Preprint available from `arXiv:1304.3390`.

[48] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 212–219, New York, NY, USA, 1996. ACM.

[49] Carl Gunter. *Semantics of Programming Languages*. MIT Press, 1992.

[50] Sean Hallgren. Polynomial-time quantum algorithms for Pell's equation and the principal ideal problem. *J. ACM*, 54(1):4:1–4:19, 2007.

[51] Thomas Häner, Damian S. Steiger, Krysta Svore, and Matthias Troyer. A software methodology for compiling quantum programs. *Quantum Science and Technology*, 3(2):020501, 2018.

[52] Sarah L. Harris and David Money Harris. Hardware description languages. In Sarah L. Harris and David Money Harris, editors, *Digital Design and Computer Architecture*, pages 172–237. Morgan Kaufmann, Boston, 2016.

[53] A. W. Harrow, B. Recht, and I. L. Chuang. Efficient discrete approximations of quantum gates. *Journal of Mathematical Physics*, 43, 2002. Also available from `arXiv:quant-ph/0111031`.

[54] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Phys. Rev. Lett.*, 103(15):150502, 2009.

[55] Matthew Hennessy. *The Semantics of Programming Languages: An Elementary Introduction Using Operational Semantics.* John Wiley and Sons, 1990.

[56] Horst Herrlich and George E. Strecker. *Category Theory: An Introduction*, volume 1. Heldermann, 3rd edition, 2007.

[57] Chris Heunen and Jamie Vicary. *Categories for Quantum Theory: An Introduction.* Oxford University Press, USA, 2019.

[58] Graham Hutton. *Programming in Haskell.* Cambridge University Press, January 2007.

[59] Gilles Kahn. Natural semantics. In *Proceedings of the 4th Annual Symposium on Theoretical Aspects of Computer Science (STACS 1987)*, pages 22–39. Springer, 1987.

[60] Phillip Kaye, Raymond Laflamme, and Michele Mosca. *An Introduction to Quantum Computing.* Oxford University Press, 2007.

[61] A. Yu. Kitaev. Quantum computations: algorithms and error correction. *Russian Mathematical Surveys*, 52(6):1191, 1997.

[62] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. Fast and efficient exact synthesis of single qubit unitaries generated by Clifford and T gates. *Quantum Information and Computation*, 13(7–8):607–630, 2013. Also available from `arXiv:1206.5236v4`.

[63] Emmanuel Knill. Conventions for quantum pseudocode. Technical report, Los Alamos National Lab., NM (United States), 1996.

[64] Yves Lafont. The linear abstract machine. *Theor. Comput. Sci.*, 59:157–180, 1988.

[65] Joachim Lambek and Philip J. Scott. *Introduction to Higher-Order Categorical Logic*, volume 7. Cambridge University Press, 1988.

[66] S. Mac Lane. *Categories for the Working Mathematician.* Graduate Texts in Mathematics. Springer, 2nd edition, 1998.

[67] Tom Leinster. *Basic Category Theory.* Cambridge University Press, 2014.

[68] Sarah K. Leyton and Tobias J. Osborne. A quantum algorithm to solve nonlinear differential equations, 2008. Preprint available from `arXiv:0812.4423`.

[69] Seymour Lipschutz and Marc Lipson. *Linear Algebra.* Schaum's Outlines. McGraw-Hill, 4rd edition, 2009.

[70] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum algorithms for supervised and unsupervised machine learning, 2013. Preprint available from `arXiv:1307.0411`.

[71] Frédéric Magniez, Miklos Santha, and Mario Szegedy. Quantum algorithms for the triangle problem, 2003. Available from `arXiv:quant-ph/0310134`.

[72] Paul-André Melliès. Categorical semantics of linear logic. In *Interactive Models of Computation and Program Behaviour*, volume 27 of *Panoramas et Synthèses*, pages 1–196. Société Mathématique de France, 2009.

[73] Microsoft Corp. What are the Q# programming language and QDK? Available from `https://docs.microsoft.com/en-us/quantum/overview/what-is-qsharp-and-qdk`. Accessed: 2020-12-31.

[74] Jarosław Adam Miszczak. Models of quantum computation and quantum programming languages, 2011. Available from `arXiv:1012.6035`.

[75] Michele Mosca, Martin Roetteler, and Peter Selinger. Quantum Programming Languages (Dagstuhl seminar 18381). In *Dagstuhl Reports*, volume 8. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2019.

[76] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, 2002.

[77] Hanne Riis Nielson and Flemming Nielson. *Semantics with Applications: An Appetizer.* Springer, 2007.

[78] Bernhard Ömer. A procedural formalism for quantum computing. Master's thesis, Department of Theoretical Physics, Technical University of Vienna, July 1998. Available from `http://tph.tuwien.ac.at/~oemer/qcl.html`.

[79] Bernhard Ömer. *Structured Quantum Programming.* PhD thesis, Department of Theoretical Physics, Technical University of Vienna, May 2003. Available from `http://tph.tuwien.ac.at/~oemer/qcl.html`.

[80] Luca Paolini, Mauro Piccolo, and Margherita Zorzi. QPCF: Higher-order languages and quantum circuits. *Journal of Automated Reasoning*, 63(4):941–966, 2019.

[81] Arun Kumar Pati and Samuel L. Braunstein. Impossibility of deleting an unknown quantum state. *Nature*, 404(6774):164–165, 2000.

[82] Jennifer Paykin, Robert Rand, and Steve Zdancewic. $\mathcal{Q}$WIRE: a core language for quantum circuits. *ACM SIGPLAN Notices*, 52(1):846–858, 2017.

[83] Benjamin C. Pierce. *Types and Programming Languages.* MIT Press, 2002.

[84] Gordon D. Plotkin. A structural approach to operational semantics. Lecture notes, DAIMI FN-19, Computer Science Department, Aarhus University Aarhus, Denmark, 1981.

[85] Gordon D. Plotkin. The origins of structural operational semantics. *The Journal of Logic and Algebraic Programming*, 60–61:3–15, 2004.

[86] Robert Raussendorf and Hans J. Briegel. Computational model underlying the one-way quantum computer. *Quantum Info. Comput.*, 2(6):443–486, October 2002.

[87] Oded Regev. Quantum computation and lattice problems. *SIAM J. Comput.*, 33(3):738–760, 2004.

[88] Emily Riehl. *Category theory in context.* Courier Dover Publications, 2017.

[89] Francisco Rios and Peter Selinger. A categorical model for a quantum circuit description language. In Bob Coecke and Aleks Kissinger, editors, *Proceedings of the 14th International Conference on Quantum Physics and Logic, QPL 2017, Nijmegen, The Netherlands, 3-7 July 2017*, volume 266 of *EPTCS*, pages 164–178, 2017.

[90] Neil J. Ross. *Algebraic and Logical Methods in Quantum Computation.* PhD thesis, Department of Mathematics and Statistics, Dalhousie University, 2015. Available from `arXiv:1510.02198`.

[91] Roland Rüdiger. Quantum programming languages: An introductory overview. *The Computer Journal*, 50(2):134–150, 2007.

[92] J. W. Sanders and P. Zuliani. Quantum programming. In *Mathematics of Program Construction*, Springer LNCS 1837, pages 80–99, 2000.

[93] Robert Seely. Linear logic, *-autonomous categories and cofree coalgebras. In *Applications of Categories in Logic and Computer Science*, volume 92 of *Contemporary Mathematics*, pages 371–382. American Mathematical Society, 1989.

[94] Peter Selinger. A brief survey of quantum programming languages. In *International Symposium on Functional and Logic Programming*, pages 1–6. Springer, 2004.

[95] Peter Selinger. Towards a quantum programming language. *Mathematical Structures in Computer Science*, 14(4):527–586, 2004.

[96] Peter Selinger. Lecture notes on the lambda calculus. Expository course notes, 120 pages. Available from `arXiv:0804.3434`, 2013.

[97] Peter Selinger. Efficient Clifford+T approximation of single-qubit operators. *Quantum Information and Computation*, 15(1–2):159–180, 2015. Preprint available from `arXiv:1212.6253`.

[98] Peter Selinger and Benoît Valiron. A lambda calculus for quantum computation with classical control. *Mathematical Structures in Computer Science*, 16(3):527–552, 2006.

[99] Peter Selinger and Benoît Valiron. Quantum lambda calculus. In Simon Gay and Ian Mackie, editors, *Semantic Techniques in Quantum Computation*, chapter 4, pages 135–172. Cambridge University Press, 2009.

[100] Peter W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994. Also available from `arXiv:quant-ph/9508027`.

[101] Robert S. Smith, Michael J. Curtis, and William J. Zeng. A practical quantum instruction set architecture, 2016. Available from `arXiv:1608.03355`.

[102] Krysta Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. Q#: Enabling scalable quantum computing and development with a high-level DSL. In *Proceedings of the Real World Domain Specific Languages Workshop 2018*, pages 1–10, 2018.

[103] Donald Thomas and Philip Moorby. *The Verilog Hardware Description Language*. Springer Science & Business Media, 2008.

[104] Benoît Valiron. A functional programming language for quantum computation with classical control. Master's thesis, University of Ottawa, September 2004.

[105] André van Tonder. A lambda calculus for quantum computation. *SIAM Journal on Computing*, 33(5):1109–1135, 2004.

[106] Philip Wadler. A taste of linear logic. In Andrzej M. Borzyszkowski and Stefan Sokołowski, editors, *Mathematical Foundations of Computer Science 1993*, pages 185–210, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[107] James D. Whitfield, Jacob Biamonte, and Alán Aspuru-Guzik. Simulation of electronic structure Hamiltonians using quantum computers. *Molecular Physics*, 109(5):735–750, 2011.

[108] Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction.* MIT Press, 1993.

[109] William K. Wootters and Wojciech H. Zurek. A single quantum cannot be cloned. *Nature*, 299(5886):802–803, 1982.

[110] Noson S. Yanofsky and Mirco A. Mannucci. *Quantum Computing for Computer Scientists.* Cambridge University Press, New York, NY, USA, 2008.

[111] Christof Zalka. Simulating quantum systems on a quantum computer. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 454(1969):313–322, 1998.

[112] Zhikuan Zhao, Alejandro Pozas-Kerstjens, Patrick Rebentrost, and Peter Wittek. Bayesian deep learning on a quantum computer. *Quantum Machine Intelligence*, 1(1-2):41–51, 2019.