Algebraic Theories and Quantum Communication

Théo Chengkai Wang

Balliol College

University of Oxford

Submitted in partial completion of the

MSc Advanced Computer Science

Michaelmas 2024

Acknowledgements

To my supervisor, Sam: thank you for your passion, enthusiasm and endless streams of ideas in our weekly meetings, but also your kindness, patience, support and mentorship throughout these last four months. This thesis would not have happened without you.

To Sam's group: thank you for the fun we had together, especially because I learnt so much from every one of you. Also thank you for all the helpful comments, ideas and discussions. Specifically to Paolo, thank you for teaching me coends and the Day convolution.

To the Quantum group and the friends I made thanks to the joy of quantum courses (and trauma of the mini-projects): thank you for having me at the MSc seminars, thank you for all your helpful comments, and thank you for the fun we had together.

To Bartek: thank you for your help and support throughout the other half of the year. This year definitely would not have gone this smoothly without your presence.

To Gabriele Tedeschi: thank you for the helpful discussions we had about quantum process calculi.

To PLDI, thank you for making me realise that I prefer theory.

To my friends, my family, and Lily, thank you for everything.

Abstract

With the advent of quantum computing and communication comes the need for corresponding programming language primitives. In this dissertation, we study a notion of classically controlled quantum communication (sending and receiving qubits) using Staton's language of parameterised algebraic theories (PAT) [67]. More specifically, we start by showing the well-formedness of the standard combinations of algebraic theories (sums and tensors) for PATs. Then, we define a PAT for quantum communication, $\mathsf{I/O} \otimes \mathsf{QUANTUM}$, as the tensor of a standard I/O theory [31] and Staton's complete axiomatisation for qubit quantum computing $\mathsf{QUANTUM}$ [67]. Finally, we give two complementary models to this algebraic theory: one operational and one denotational. The first model is based on an operational semantics where the program sends and receives qubits to and from an environment modelled by a quantum stream. It concretely demonstrates the notion of quantum communication and ensures that our theory is non-degenerate (unlike the Eckmann-Hilton setting). The second model is based on an elegant denotational semantics, constructed as a free I/O monad on $\mathbf{CP}^{\infty}$, the category of matrix algebras and CP maps completed with infinite biproducts [52]. We show that these two models correspond to the same notion of communication in the sense that the denotational model is adequate and fully abstract with respect to the operational model.

Contents

List of Figures

List of Tables

1
Introduction

Quantum computing is a promising model of computation which exploits quantum mechanical phenomena. It has demonstrated provable complexity-theoretic computational advantages to classical computing. Shor's seminal algorithm [65], for instance, solves the factoring problem in polynomial time on a quantum computer, whereas the best classical algorithms are, at best, sub-exponential. Relatedly, quantum communication emerged both as a way of transmitting quantum information and as a way to enable distributed quantum computing and alleviate the current engineering constraints of quantum computers (e.g. [12]). Optical quantum computing, for instance, studies the use of photons as the information carriers for computing and communication. Quantum communication protocols such as teleportation [10] as well as cryptographic protocols such as the BB84 protocol for key distribution [9] have also been developed.

Accompanying the emergence of quantum computing and communication is the field of quantum programming languages, which can be visualised in fig. 1.1.

On the one hand, for quantum computing without communication, a large proportion of existing languages are based on the quantum circuit model (labelled as QC in fig. 1.1), a standard model of quantum computation allowing state preparation, unitary evolution, measurements and classical control. This includes languages like QPL [63] or Silq [11], as well as quantum DSLs designed to be embedded in classical host languages, like the quantum I/O monad [1] [5], Quipper [26] or QWire [53].

Quantum communication, on the other hand, has mostly been studied in the context of quantum concurrency and quantum process calculi (drawn as 'QPC' in fig. 1.1). Quantum process calculi

[1] The reader should note that the quantum I/O monad is completely different from what we mean by quantum I/O: it is a monadic interface for quantum computing in Haskell.

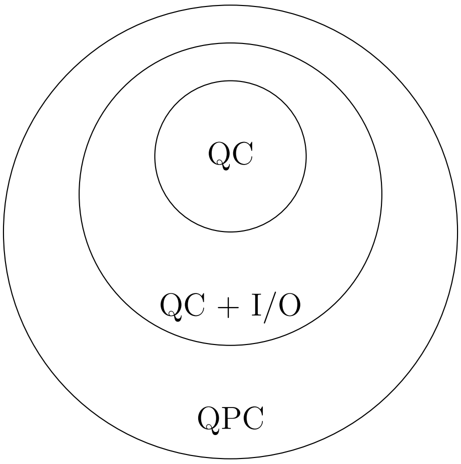
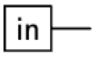


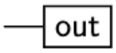
Figure 1.1: Venn diagram of expressivity of the different quantum programming models. QC denotes quantum circuits, QC+I/O denotes quantum I/O, and QPC denotes quantum process calculi.

[36, 24, 73, 17] extend classical process calculi like CCS [39], CSP [29] or the π calculus [40] with quantum computing primitives and the ability of sending and receiving qubits via qubit channels. However, it is difficult to isolate the semantics of communication itself from the inherent complexity of concurrency. We discuss this in more detail in section 1.2.

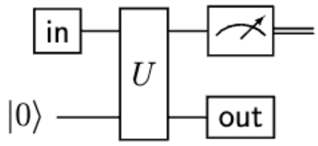
This work aims to study the relatively unexplored 'QC + I/O' space in fig. 1.1: instead of studying quantum concurrency in its full complexity, we study the programming language primitives for quantum communication *in isolation*, as simple I/O operations extending the circuit model. Concretely, this means having 'gates' like



which receives a qubit from an unknown environment, and



which sends a qubit to an unknown environment. Note that by environment, we mean an unknown quantum system that interacts with our quantum computer via a fixed communication channel which can transmit qubits. These 'gates' could combine into circuits like



which receives a qubit from the environment, prepares another qubit, applies a unitary on both, sends the second qubit to the environment and measures the first. In a standard programming language, this would be written as

```
q_1: qbit = in()
q_2: qbit = new_{|0⟩}()
(q_1, q_2) = apply_U(q_1, q_2)
out(q_2)
measurement_outcome: bit = measure(q_1)
```

Naturally, for any reasonable programming model, we also want to be able to send or receive qubits *based on a measurement outcome*. For instance, we could send q_2 to the environment only when the measurement outcome is 1. It is rather awkward to express this notion of *classically controlled* quantum I/O in quantum circuit diagrams. This is because the hypothetical controlled out box will have varying numbers of outputs according to the control bit: if the control bit is 1, then the qubit is sent, and the out box should have no output qubit, whereas if the control bit is 0, then the qubit is not sent, and the out box should act like the identity wire. In contrast, the same notion is very easy to express in an imperative programming language:

```
q_1: qbit = in()
q_2: qbit = new_{|0⟩}()
(q_1, q_2) = apply_U(q_1, q_2)
measurement_outcome: bool = measure(q_1)
if (measurement_outcome = 1):
    out(q_2)
else: # do nothing
```

In fact, classically controlled quantum I/O seems so natural as a programming primitive that it is worth studying more rigorously.

In this work, we choose to study the classically controlled quantum communication primitives presented above as operations in an algebraic theory. Algebraic theories are no more than a ‘signature’ of operations and equations between terms formed by these operations. Originally developed as an abstract notion of algebra, they were first repurposed and generalised by Plotkin and Power [56] to give semantics to notions of computational effects treated as the combined effect of individual operations. Their respective program equivalences can then be axiomatised as equations over the terms formed by those operations. Instances of so-called ‘algebraic’ theories of effects include non-determinism (with a binary choice operation), interactive I/O [31] (with an input and an output operation), and cooperative threading [1]. Intuitively, one can think of an algebraic theory as the interface of the computational effect it models and its equations as the assumptions we make about that interface. This naturally leads us to the notion of a *model* of an algebraic theory, which corresponds to the ‘implementation’ of such an interface as an object in a category.

Studying effects in algebraic theories has numerous advantages. Firstly, it gives us a mathematically principled way of studying the semantics of computational effects. In fact, doing so with algebraic theories directly gives us a well-behaved mathematical object to study and potentially 'implement' with different models, which could give different insights. Moreover, algebraic effects are modular and compositional: we could, for instance, combine notions of computational effects simply by combining their respective algebraic theories. Finally, there are also practical advantages: giving semantics of effects using algebraic theories enables equational reasoning on program equivalence either by hand or by a compiler.

Recently, Staton [67] developed a framework for algebraic theories with linear parameters and gave the first complete algebraic theory of qubit quantum computing ($\mathsf{QUANTUM}$) (informally corresponding to the QC circle in fig. 1.1). The signature includes an operation for state preparation $\mathsf{new}(q.c)$, an operation for applying unitaries $\mathsf{apply}_U(\vec{p}, \vec{q}.c)$, and an operation for computational basis measurement $\mathsf{measure}(q, c_0, c_1)$, where p, q denote qubits and c, c_0, c_1 denote continutations. The semantics of these operations are then encapsulated into equations such as

$$\mathsf{apply}_U(a, a.\mathsf{discard}(a, x)) = \mathsf{discard}(a, x)$$

where $\mathsf{discard}(a, x) = \mathsf{measure}(a, x, x)$. This intuitively says that discarding a qubit after applying a unitary on it is equivalent to simply discarding[2].

In this dissertation, we base ourselves on Staton's framework for parameterised algebraic theories. The goal is to formulate a semantics for quantum I/O by combining Staton's theory of quantum computing and a theory for I/O with an operation $\mathsf{in}(q.c)$ for receiving qubits and an operation $\mathsf{out}(q, c)$ for sending qubits and studying the resulting algebraic theory. Note that such a combination does indeed give a notion of classical control for quantum I/O operations because we can write programs like $\mathsf{measure}(p, \mathsf{out}(q.c), c')$, where we decide whether to send a qubit based on the result of a measurement.

1.1 Contributions

The contributions of this thesis are twofold.

Firstly, we prove the existence of the sum and the tensor for parameterised algebraic theories (theorem 3.3.11, theorem 3.3.15), the standard ways of combining algebraic theories developed by Hyland, Plotkin and Power [31]. This provides evidence that Staton's PAT framework is well-formed and well-behaved and, therefore, provides a solid foundation for working with parameterised algebraic theories.

In more detail, we start by explicitly characterising the relations between Staton's syntactic notion of PAT [69, 67] and the corresponding semantic object of enriched Lawvere theories. Given

[2] This is a simplified version of axiom (C) (fig. 4.1)

a PAT P, we show that its syntactic category $\mathbb{L}_P$ is a $\mathsf{FinProd}$ doctrine, $[\mathbf{Ctx_0}, \mathbf{Set}]$ enriched Lawvere theory (theorem 3.2.21), and that the models of P and $\mathbb{L}_P$ coincide (corollary 3.2.22). Then, we generalise the enrichment structure of the Lawvere theories to a symmetric premonoidal structure and relate it to the syntactic sequencing of terms in the PATs (theorem 3.3.8). We then leverage this construction to show the existence of sums and tensors for the enriched Lawvere theories generated by PATs by explicitly constructing them in terms of those syntactic PATs (theorem 3.3.11, theorem 3.3.15).

Secondly, in chapter 4 and chapter 5, we systematically study classically controlled quantum I/O. We give the algebraic theory as a tensor $\mathsf{I/O} \otimes \mathsf{QUANTUM}$, a sound operational model, and a sound monadic denotational model, which we prove is adequate and fully abstract to the operational model. To the best of our knowledge, this is the first such programming-language-theoretic development for quantum communication.

In chapter 4 we present our novel notion of classically controlled quantum communication as a tensor of linear parameterised algebraic theories $\mathsf{I/O} \otimes \mathsf{QUANTUM}$ (section 4.1.2) and argue for its sensibility. We give a novel operational semantics based on an ad hoc notion of quantum streams (definition 4.2.5). We then define a precise notion of program equivalence based on the operational semantics and show it is compatible with quantum theory by showing that it is indeed a model of $\mathsf{I/O} \otimes \mathsf{QUANTUM}$ (theorem 4.3.25). Giving such a non-trivial model not only exemplifies the notion of quantum communication we had in mind but also shows that the theory itself is not degenerate. Indeed, tensoring two theories together is known to have unexpected effects. A well-known example is the Eckmann-Hilton result (proposition 4.2.1, [47]), where, for example, two monoids tensored together collapse into a single commutative monoid (as opposed to two distinct monoids). Giving a non-trivial operational model as such shows that this sort of phenomenon does not happen in $\mathsf{I/O} \otimes \mathsf{QUANTUM}$.

Finally, in chapter 5, we leverage the insights obtained from the proofs in the previous chapter to give a novel and elegant monadic denotational semantics for our algebraic theory based on the free I/O monad constructed on the category $\mathbf{CP}^\infty$ of matrix algebras and CP maps completed with infinite biproducts following [52]. We then show that this denotational semantics is adequate (theorem 5.3.4) and fully abstract (theorem 5.3.6) for our operational semantics, meaning that they correspond to exactly the same computational model. As a corollary, the denotational semantics is also a model of our algebraic theory (theorem 5.4.1, corollary 5.4.2).

1.2 Related Work

Our work is related to several lines of work in both classical and quantum programming languages.

Algebraic theories of effects The use of universal algebra to model computational effects has been pioneered by Moggi [41] for the use of monads, and Plotkin and Power [56] for the use of algebraic theories. Hyland, Plotkin and Power [31] further studied the different standard combinations of algebraic theories and their respective computational interpretations. In this work, we transfer their insights to Staton's framework for parameterised algebraic theories [67, 69] and show the existence of sums and tensors of presented theories.

Also, both quantum computing [67] and I/O [31] have individually been studied using the power of algebraic theories. In fact, the novelty of our work lies in their combination. As previously mentioned, quantum computing has mainly been studied by Staton [67], whose work we base our dissertation on. On the other hand, I/O was studied mostly for classical computing, appearing in the context of interactive I/O [31] as well as message passing in distributed computing, for example, as a primitive of the π calculus [66, 69]. When treated as interactive I/O, e.g. when used as part of the resumptions monad [31], the input and output operators are usually combined with other algebraic theories in a *sum*, meaning that they are set not to commute with the other operations. When treated as communication, I/O operations are often made to commute with independent local operations (e.g. name creation), like in the case of Stark's free model for the π calculus [66]. In a similar vein to this latter work, in our theory, we make I/O operations commute with quantum operations, and we will further argue in section 4.1.2 that such commutation is necessary to obtain a notion of communication that is consistent with the peculiarities of quantum theory.

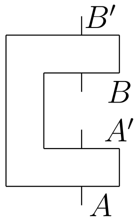
Categorical Semantics and Quantum Programming Giving categorical semantics for quantum computing is not a new idea. Selinger, in the seminal work about the Quantum Programming Language (QPL) [63], already gave a categorical semantics based on the density operator formalism. This was subsequently generalised to operator algebras (e.g. C* algebras) [19] and taken as the standard approach for giving denotational semantics for quantum programming languages. A different line of work is that of Categorical Quantum Mechanics (CQM) [3], which leverages the graphical tools derived from dagger compact closed categories to reason about general quantum theory. Much of the mathematical tools used in this thesis were, in fact, developed in this programme.

Axiomatisations of quantum program equivalence We base our work on Staton's fully complete axiomatisation of qubit quantum computing as an algebraic theory. Recently, Carette et al. complemented Staton's work by giving a sound and complete equational theory for unitary quantum circuits taken from various gate sets [14]. They do so by extending a model of classical reversible computing, Π, with two maps and three equations suffices to achieve such results. An alternative line of work bases itself on the insights of CQM to produce graphical calculi like the ZX calculus [20, 21] and related calculi (e.g. ZH calculus [7], ZW calculus [27] etc.). However, such calculi are distinguished from the aforementioned axiomatisations, for they model general quantum theory, and quantum circuits are only a subset thereof.

Notions of (classically controlled) Quantum I/O in Programming In the context of programming languages, quantum communication has mostly been studied as qubit message passing in quantum process calculi. Quantum process calculi generalise classical process calculi to model a notion of message-passing concurrent quantum programming. Noteable lines of work include CQP [24], QPAlg [36], qCCS [73] and the most recent lqCCS [17]. These works usually give an operational account of their calculi using a labelled transition system (LTS), where labels are effects such as sending or receiving a qubit. In particular, message-passing is done with a synchronisation primitive where $P||Q \to P'||Q'$ if $P \overset{c?v}{\to} P'$ and $Q \overset{c!v}{\to} Q'$ (P, P', Q, Q' are processes, $||$ means parallel composition, c is a channel, v is a qubit name and !, ? mean send and receive respectively). Then, they give a notion of program equivalence based on bisimilarity. Informally, if the LTS is defined on configurations of the form of state-program pairs, a bisimilarity is the largest relation on configurations defined according to the following informal pattern: two configurations are bisimilar if whenever one reduces, the other reduces accordingly, and the resulting configurations are equivalent. However, as pointed out by the authors [17], giving a notion of program equivalence for quantum process calculi is difficult because of the subtle interactions of concurrency, synchronisation and local non-determinism with quantum effects. Indeed, they demonstrate that each of the notions of bisimilarity proposed by the various lines of work differ in slight and subtle ways. In a later work, the same authors go on to develop a categorical, coalgebraic account of 'effect' LTSs, generalising probabilistic LTSs with effect algebras to better reason about the subtleties of what they call quantum bisimulation [16].

Our work differs from this line of work in three ways. Firstly, we focus on (possibly classically controlled) communication instead of concurrency and, by doing so, restrict our problem to a much more tractable setting. In a way, our work could be seen as studying the 'local' sublanguage of quantum process calculi, with only quantum computing primitives and communication. The second difference lies between bisimilarities and the (operational) notion of program equivalence we defined. Bisimilarities reason about every state in the transition sequences of the programs we want to equate, because the programs in question are not assumed to terminate – this is the case in general quantum process calculi. In contrast, our quantum equivalence is based on operational semantics where every program terminates, and therefore, we need only consider the final state of the programs. With that said, rather surprisingly, the subtleties we encountered when defining a program equivalence compatible with quantum theory are similar to some of those encountered by various authors in defining bisimilarity relations for quantum process calculi. Finally, our approach is more denotational, following the line of work of algebraic effects. Bisimilarities fundamentally are operational notions because because they reason about the transitions of the program state. In contrast, our work additionally has a categorical denotational semantics that is adequate and fully abstract with respect to the operational semantics, as well as a sound equational theory $\mathsf{I/O} \otimes \mathsf{QUANTUM}$ axiomatising its equality.

Higher-order quantum computing The notion of quantum communication presented in our work very much seems to relate to the recent trends of higher-order quantum computing, where instead of reasoning about simple quantum channels, we reason, for example, about maps from quantum channels to quantum channels. A particularly interesting graphical notation introduced by [18] is the Comb notation:



which corresponds to a morphism $\mathcal{C}(I, A^* \otimes A' \otimes B^* \otimes B')$ according to compact closure.

Further along this line of research, Kissinger and Uijlen [34] introduce the Caus construction of higher order 'causal' processes on a special class of compact closed categories called 'precausal', including **CP** as introduced in section 2.3.3. This, in particular, allows for reasoning about higher-order quantum channels with causality and signalling constraints. Intuitively, our denotational semantics of quantum communication is akin to a quantum program with open input and output wires with causality constraints: the input and output wires should be used in the order specified by the I/O trace $\sigma \in \{\mathsf{in}, \mathsf{out}\}^*$. As future work, it would be interesting to understand whether this relates to higher-order causal processes in the Caus construction.

Quantum Controlled Communication Relatedly to the questions of higher order processes and causality, quantum information theorists have developed higher order quantum channels which allow for quantum (coherent) control [2, 35]. This includes channels with indefinite causal order (where the order of applying channels can be controlled by a qubit in superposition) and, more interestingly, channel multiplexing (with a qubit controlling which channel to send other qubits to). In this work, for simplicity, we have only considered classically controlled quantum I/O without a notion of deferred measurement (briefly discussed in section 4.1.2). However, as future work, it would be interesting to explore how to potentially formulate such a notion of deferred measurement if we allow for this kind of coherently controlled communication.

Monads in quantum computing Finally, adapting classical monads to the quantum case is not a new idea. For instance, Sati and Schreiber [62] show that classical monads such as the reader monad $(X \mapsto (W \to X))$, the writer monad $(X \mapsto W \times X)$ and the state monad $(X \mapsto W \to (W \times X))$ can be adapted to the quantum setting. They further show that, for instance, the quantum reader monad can be seen as equivalent to Coecke's classical structures [23] and that the quantum state monad can be used to reason about a certain class of quantum channels. However, they do not consider the free I/O monad that we are considering for this work.

1.3 Outline

Chapter 2 introduces the relevant background in category theory, algebraic effects, quantum computing and string diagrams to understand this dissertation. Chapter 3 presents the framework for parameterised algebraic theories à la Staton [67], explicitly constructs the Lawvere theory generated by a PAT, and develops the notion of sum and tensor. Chapter 4 presents the algebraic theory of interest, $\mathsf{I/O} \otimes \mathsf{QUANTUM}$, and gives it a non-trivial, operational-semantics-based model using a program equivalence compatible with quantum behaviour. Chapter 5 leverages the insights from the proofs of the previous chapter to give a monadic denotational semantics, which is a model of $\mathsf{I/O} \otimes \mathsf{QUANTUM}$ and is adequate and fully abstract with respect to the operational semantics.

2
Background

This chapter provides an informal overview of the required background in category theory (section 2.1), algebraic effects (section 2.2), and quantum computing (section 2.3).

2.1 Basic Category Theory

Category theory is the go-to mathematical tool for studying (and building) abstractions. We briefly present several basic definitions and facts in category theory. For further reading, the reader is referred to [38, 61, 54].

2.1.1 Categories

2.1.1.1 Basic facts about categories

We start by fixing the notation while presenting basic definitions.
Definition 2.1.1 (Category)**.** *We say that $\mathcal{C}$ is a category if it has*

- *A collection of objects* $\mathbf{Ob}\ \mathcal{C}$*,*
- *For each* $X, Y \in \mathbf{Ob}\ \mathcal{C}$*, a collection of morphisms* $\mathcal{C}(X, Y)$*,*
- *For every* $X \in \mathbf{Ob}\ \mathcal{C}$ *there is a morphism* $\mathbf{id}_X : \mathcal{C}(X, X)$*, and for every* $X, Y, Z \in \mathbf{Ob}\ \mathcal{C}$ *and every morphism* $f \in \mathcal{C}(X, Y)$ *and* $g \in \mathcal{C}(Y, Z)$*, there is a* $g \circ f \in \mathcal{C}(X, Z)$*,*

such that $\mathbf{id} \circ f = f \circ \mathbf{id} = f$ *and* $(h \circ g) \circ f\ = h \circ (g \circ f)$*.*

We say that $\mathcal{C}$ is locally small if all the morphism collections are sets, and $\mathcal{C}$ is small if the object collection is also a set.

Definition 2.1.2 (Functor). *A functor $F : \mathcal{C} \to \mathcal{D}$ is an object map which associates every $X \in \mathcal{C}$ with an object $F(X) \in \mathcal{D}$, and a morphism map which associates every $f : X \to Y$ to a morphism $F(f) : F(X) \to F(Y)$, preserving identity and composition.*

Definition 2.1.3 (Natural Transformation). *A natural transformation $\alpha : F \to G$ between two functors $F, G : \mathcal{C} \to \mathcal{D}$ is a family of morphisms α_X for $X \in \mathcal{C}$ such that the following commutes*

$$\begin{tikzcd}
F(X) \arrow[r, "F(f)"] \arrow[d, "\alpha_X"'] & F(Y) \arrow[d, "\alpha_Y"] \\
G(X) \arrow[r, "G(f)"'] & G(Y)
\end{tikzcd}$$

for all $X, Y \in \mathcal{C}$ and $f : X \to Y$.

2.1.1.2 Presheaves and free cocompletions

It is often useful to take a category $\mathcal{C}$ and 'freely add' constructions like all (small) coproducts or all (small) colimits. For instance, a free cocompletion of a category $\mathcal{C}$, is a category $\mathcal{C}'$ with all small colimits which embeds $\mathcal{C}$ along $E : \mathcal{C} \to \mathcal{C}'$, such that for any $\mathcal{D}$ with all small colimits and any functor $F : \mathcal{C} \to \mathcal{D}$, there is a unique small colimit preserving functor $G : \mathcal{C}' \to \mathcal{D}$ such that $G \circ E = F$. The dual notion of completions (freely adding small limits) can be defined similarly. Similar notions can for instance be defined for finite coproducts, and dually, for finite products.

For explicitly constructing free cocompletions, we need some basic facts about presheaf categories, which we present in the style of [51]. For any locally small $\mathcal{C}, \mathcal{D}$, we write $[\mathcal{C}, \mathcal{D}]$ to denote the category of functors from $\mathcal{C}$ to $\mathcal{D}$ with natural transformations as morphisms.

Definition 2.1.4 (Functor underlying the Yoneda Embedding). *Let $\mathcal{C}$ be locally small. There is a functor $\text{よ} : \mathcal{C} \to [\mathcal{C}^{op}, \mathbf{Set}]$ defined by $X \mapsto \mathcal{C}(-, X)$.*

Note that we call $[\mathcal{C}^{op}, \mathbf{Set}]$ the category of presheaves on $\mathcal{C}$.

Proposition 2.1.5 (Yoneda's Lemma). *For any $F \in [\mathcal{C}^{op}, \mathbf{Set}]$, there is a canonical isomorphism*

$$[\mathcal{C}^{op}, \mathbf{Set}](\text{よ}(X), F) \simeq F(X)$$

Corollary 2.1.6 (Yoneda's embedding). *よ is a full and faithful functor (injective and surjective on morphism sets), hence an embedding.*

Definition 2.1.7 (Representable presheaf/functor). *A presheaf $F : \mathcal{C}^{op} \to \mathbf{Set}$ is representable if it is naturally isomorphic to $\mathcal{C}(-, X)$ for some $X \in \mathcal{C}$.*

The Yoneda embedding is useful because it defines (along with its codomain $[\mathcal{C}^{op}, \mathbf{Set}]$) the free (small-)cocompletion of $\mathcal{C}$ (cocompletion meaning completing with all colimits) [48]. Using this fact, we can nicely define the free finite coproduct cocompletion as follows:

Example 2.1.8 (Free finite coproduct cocompletion)**.** *Let $\mathcal{C}$ be a small category. The free finite coproduct cocompletion of $\mathcal{C}$ is the full subcategory $[\mathcal{C}^{op}, \mathbf{Set}]_+$ of $[\mathcal{C}^{op}, \mathbf{Set}]$ of the finite coproducts of representables. Then, clearly the Yoneda embedding factors through it: よ $: \mathcal{C}^{op} \hookrightarrow [\mathcal{C}^{op}, \mathbf{Set}]_+ \hookrightarrow [\mathcal{C}^{op}, \mathbf{Set}]$.*

By duality, we can analogously define the free product completion:

Example 2.1.9 (Free finite product completion)**.** *Let $\mathcal{C}$ be a small category. The free finite product completion of $\mathcal{C}$ is exactly $[\mathcal{C}, \mathbf{Set}]_+^{op}$.*

2.1.2 Monoidal categories

Now we move on to monoidal categories, which are categories with an additional notion of 'monoidal' product $\otimes$, which some authors (e.g. [22]) refer to as 'parallel composition'.

2.1.2.1 Basics facts about monoidal categories

Definition 2.1.10 (Monoidal category)**.** *A monoidal category $(\mathcal{C}, \otimes, I, \alpha, \lambda, \rho)$ (or just $\mathcal{C}$ or $(\mathcal{C}, \otimes, I)$) is a category $\mathcal{C}$ equipped with:*

- *A functor $\otimes : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ (sometimes referred to as a 'bifunctor' as it's a functor with two inputs), called the* monoidal product
- *A distinguished object $I \in \mathcal{C}$ called the* monoidal unit
- *And natural isomorphisms*
  - $\alpha_{A,B,C} : (A \otimes B) \otimes C \to A \otimes (B \otimes C)$*, the* associator*, natural in A, B, C;*
  - $\lambda_A : I \otimes A \to A$ *the* left unitor*, natural in A*
  - *and $\rho_A : A \otimes I \to A$ the* right unitor*, natural in A*

*such that the following diagrams (*Coherence relations*) commute for any $A, B, C, D \in \mathcal{C}$:*

- *The* Triangle equation*:*

$$\begin{tikzcd}
(A \otimes I) \otimes B \arrow[r, "\alpha_{A,I,B}"] \arrow[dr, "\rho_A \otimes B"'] & A \otimes (I \otimes B) \arrow[d, "A \otimes \lambda_B"] \\
& A \otimes B
\end{tikzcd}$$

- *The* Pentagon equation

$$\begin{tikzcd}
((A \otimes B) \otimes C) \otimes D \arrow[r, "\alpha_{A \otimes B,C,D}"] \arrow[d, "\alpha_{A,B,C} \otimes D"'] & (A \otimes B) \otimes (C \otimes D) \arrow[r, "\alpha_{A,B,C \otimes D}"] & A \otimes (B \otimes (C \otimes D)) \\
(A \otimes (B \otimes C)) \otimes D \arrow[rr, "\alpha_{A,B \otimes C,D}"'] & & A \otimes ((B \otimes C) \otimes D) \arrow[u, "A \otimes \alpha_{B,C,D}"']
\end{tikzcd}$$

We say that $\mathcal{C}$ is strict *monoidal if all the coherence isomorphisms are identity.*

Definition 2.1.11 (Symmetric monoidal category). *A monoidal category $(\mathcal{C}, \otimes, I)$ is symmetric if there is a natural isomorphism $\sigma_{X,Y} : X \otimes Y \cong Y \otimes X$ such that $\sigma_{Y,X} \circ \sigma_{X,Y} = \mathbf{id}_{X \otimes Y}$, and the following additional diagram is commutative:*

$$\begin{tikzcd}
(X \otimes Y) \otimes Z \arrow[r, "\alpha"] \arrow[d, "\sigma \otimes \mathbf{id}"'] & X \otimes (Y \otimes Z) \arrow[r, "\sigma"] & (Y \otimes Z) \otimes X \arrow[d, "\alpha"] \\
(Y \otimes X) \otimes Z \arrow[r, "\alpha"'] & Y \otimes (X \otimes Z) \arrow[r, "\mathbf{id} \otimes \sigma"'] & Y \otimes (Z \otimes X)
\end{tikzcd}$$

Definition 2.1.12 (Monoidal closure (e.g. [43])). *A closed monoidal category is a monoidal category $(\mathcal{C}, \otimes, I)$ such that for every $Y \in \mathcal{C}$, the functor $(-) \otimes Y : \mathcal{C} \to \mathcal{C}$ has a left adjoint $[Y, -] : \mathcal{C} \to \mathcal{C}$ such that for all $X, Y, Z \in \mathcal{C}$ there is a natural isomorphism $\mathcal{C}(X \otimes Y, Z) \simeq \mathcal{C}(X, [Y, Z])$.*

Proposition 2.1.13 (Mac Lane's coherence theorem (e.g. [38])). *Every monoidal category is monoidally equivalent to a strict monoidal category. In other words, the associators and unitors 'do not matter' because every 'formal' diagram made up of associators and unitors commutes.*

Definition 2.1.14 (Monoidal functor (e.g. [49])). *A (lax) monoidal functor $F : (\mathcal{C}, \otimes, I) \to (\mathcal{D}, \otimes, I)$ is a functor $F : \mathcal{C} \to \mathcal{D}$ together with a coherent map $\epsilon : I_{\mathcal{D}} \to F(I_{\mathcal{C}})$ and a natural transformation $\mu_{X,Y} : FX \otimes_{\mathcal{D}} FY \to F(X \otimes_{\mathcal{C}} Y)$ which commute with the associators and unitors of $\mathcal{C}$ and $\mathcal{D}$. It is strong (resp. strict) if these morphisms are isomorphisms (resp. identity).*

Definition 2.1.15 (Symmetric monoidal functor). *A symmetric monoidal functor is just a monoidal functor such that the natural transformation μ commutes with the symmetry.*

Now, we define some additional useful mathematical tools based on monoidal categories.

2.1.2.2 Actions of monoidal categories

As presented by Capucci and Gavranović [13], monoid actions are a way of providing an 'implementation' of the laws of a monoid as 'operations' on another object.

Definition 2.1.16 (Action). *Let $(M, \cdot, 1)$ be a monoid. Then, an* action *of M on a set X is an operation $\bullet : M \times X \to X$ such that*

- $1 \bullet X = X$
- $(a \cdot b) \bullet X = a \bullet (b \bullet X)$

Example 2.1.17 (Scalar multiplication (e.g. [13])). *The scalar multiplication $K \times V \to V$ of a vector space V over the field of scalars K is an action of K on V.*

As we can see, not only do actions provide an implementation of the monoid M, they also provide additional structure to the set X and allow us to e.g. relate collinear vectors by their scalar factors. In category theory, we can define a similar notion of 'action' for monoidal categories.

Definition 2.1.18 (Actions of monoidal categories). *A (left)-action of a monoidal category $(\mathcal{C}, \otimes, I)$ on a category $\mathcal{D}$ is a functor $\bullet : \mathcal{C} \times \mathcal{D} \to \mathcal{D}$ along with natural isomorphisms $\eta_X : X \to I \bullet X$ and $\mu_{A,B,X} : A \bullet (B \bullet X) \to (A \otimes B) \bullet X$ satisfying some coherence equations shown in definition 3.1.1 of [13].*

This definition will allow us to define a notion of parameterised computation: we will define an action $\bullet : \mathbf{Ctx_0} \times \mathcal{C} \to \mathcal{C}$, where $\mathbf{Ctx_0}$ is the monoidal category exhibiting the structure of parameterised computations and $\mathcal{C}$ is the 'implementation' category instantiating that structure. Then, $p \bullet X$ may be interpreted as 'the implementation of computations depending on p parameters in X', or simply 'a computation in X depending on p parameters'.

2.1.2.3 Making the free cocompletion compatible with the monoidal structure

We've seen in section 2.1.1.2 how to construct free cocompletions; we now want to make it compatible with monoidal categories. For simplicity of definitions, we only consider cocompleting symmetric monoidal categories, but the construction works for general monoidal categories.

Let $(\mathcal{C}, \otimes, I)$ be a symmetric monoidal category. Then, it suffices to define a symmetric monoidal structure on its free cocompletion $[\mathcal{C}^{op}, \mathbf{Set}]$ compatible with $\otimes$ (i.e. such that よ preserves $\otimes$). This can be achieved using the Day convolution structure:

Definition 2.1.19 (Day convolution). *Let $F, G \in [\mathcal{C}^{op}, \mathbf{Set}]$. Their Day convolution is:*

$$F \otimes_{Day} G : X \mapsto \int^{A,B \in \mathcal{C}} F(A) \times G(B) \times \mathcal{C}(X, A \otimes B)$$

While this definition may seem obscure[1], it is fully (and universally) determined by the very nice properties listed below [33]:

Proposition 2.1.20 (Monoidal closed structure (e.g. [44])). *Let $(\mathcal{C}, \otimes, I)$ be a symmetric monoidal category. Then $([\mathcal{C}^{op}, \mathbf{Set}], \otimes_{Day}, よ(I))$ is a symmetric monoidal closed category.*

Proposition 2.1.21 (Yoneda embedding is strong monoidal (e.g. [45])). *The Yoneda embedding よ $: \mathcal{C} \to [\mathcal{C}^{op}, \mathbf{Set}]$ is a strong monoidal functor (i.e. preserves the monoidal structure up to isomorphism) with respect to the $\otimes_{Day}$ monoidal structure. If $\mathcal{C}$ is symmetric monoidal, then よ is additionally a symmetric monoidal functor.*

Proposition 2.1.22 (Day convolution preserves colimits (e.g. [44])). *Day convolution $\otimes_{Day}$ preserves colimits separately on each of its arguments.*

Proof. Follows from the symmetry and closure of $\otimes_{\text{Day}}$. □

As a corollary, it means that the free coproduct cocompletion and the free product completion can also be made compatible with $\mathcal{C}$'s monoidal structure with the Day convolution.

[1] Not unlike most of category theory

2.1.3 Premonoidal categories

In a monoidal category $\mathcal{C}$, the monoidal product $\otimes$ is a bifunctor, meaning that it is a functor separately on both arguments, which additionally satisfies that the following diagram commutes.

$$\begin{tikzcd}
X \otimes A \arrow[r, "X\otimes g"] \arrow[d, "f\otimes A"'] & X \otimes B \arrow[d, "f\otimes B"] \\
Y \otimes A \arrow[r, "Y\otimes g"'] & Y \otimes B
\end{tikzcd}$$

for any $f : X \to Y$ and $g : A \to B$, with $X, Y, A, B \in \mathcal{C}$.

This intuitively means that f and g can be run 'in parallel', i.e. their order does not matter. However, if we interpret f and g as impure computations acting on different resources, then the order will matter. In fact, the programs $f; g$ and $g; f$ are usually not equivalent. This motivates the notion of premonoidal category, introduced by Power and others in e.g. [59, 6], where bifunctoriality of $\otimes$ (i.e. the equation above) does not hold in general.

Definition 2.1.23 (Binoidal category)**.** *A binoidal category is a category $\mathcal{C}$ with*

- *$X \otimes Y \in \mathcal{C}$ for each $X, Y \in \mathcal{C}$*
- *For each X a functor $X \rtimes -$ such that $X \rtimes Y = X \otimes Y$*
- *For each X a functor $- \ltimes X$ such that $Y \ltimes X = Y \otimes X$.*

In a binoidal category $\mathcal{C}$, we can define *central* morphisms, which we think intuitively of as a special subset of computations which commute with *every* other computation.

Definition 2.1.24 (Central morphisms)**.** *A morphism $f : X \to Y$ is* central *if for any $f' : X' \to Y'$ the following two diagrams commute:*

$$\begin{tikzcd}
X \otimes X' \arrow[r, "X\rtimes f'"] \arrow[d, "f\ltimes X'"'] & X \otimes Y' \arrow[d, "f\ltimes Y'"] \\
Y \otimes X' \arrow[r, "Y\rtimes f'"'] & Y \otimes Y'
\end{tikzcd}$$

and

$$\begin{tikzcd}
X' \otimes X \arrow[r, "X'\rtimes f"] \arrow[d, "f'\ltimes X"'] & X' \otimes Y \arrow[d, "f'\ltimes Y"] \\
Y' \otimes X \arrow[r, "Y'\rtimes f"'] & Y' \otimes Y
\end{tikzcd}$$

We then denote these composites as $f \otimes f'$ and $f' \otimes f$ respectively.

Definition 2.1.25 (Premonoidal category)**.** *A premonoidal category $\mathcal{C}$ is a binoidal category with an object $I \in \mathcal{C}$ and central isomorphisms $\alpha_{A,B,C} : (A \otimes B) \otimes C \overset{\cong}{\to} A \otimes (B \otimes C)$, $\lambda_A : I \otimes A \overset{\cong}{\to} A$ and $\rho_A : A \otimes I \overset{\cong}{\to} A$ such that all the possible naturality squares for α, λ and ρ commute (i.e. they are natural in each index* separately*), and satisfying the pentagon and triangle laws.*

Definition 2.1.26 (Symmetric premonoidal category). *A symmetric premonoidal category is a premonoidal category* $(\mathcal{C}, \otimes, I)$ *such that there is a central isomorphism* $\sigma_{X,Y} : X \otimes Y \overset{\cong}{\to} Y \otimes X$ *natural separately in* X *and* Y*, satisfying the same axioms of as a symmetric monoidal category.*

An example will be the syntactic category constructed in chapter 3, which we will come to later.

Finally, we define a premonoidal functor similarly to monoidal functors, as in [58]:
Definition 2.1.27 (Premonoidal functor ([58] Definition 20)). *A premonoidal functor* $F : \mathcal{C} \to \mathcal{D}$ *is a functor that sends central maps to central maps, together with central natural transformations* $\mu_{X,Y} : FX \otimes_{\mathcal{D}} FY \to F(X \otimes_{\mathcal{C}} Y)$ *and* $\epsilon : I_{\mathcal{D}} \to F(I_{\mathcal{C}})$ *commuting with the associator and unitors.*

The functor is strong (resp. strict) if they are isomorphisms (resp. identity morphisms).

One can similarly generalise the definition for symmetric monoidal functors, but we will not detail it here.

2.2 Universal Algebra and the algebraic theory of effects

For programming language theorists, an important subfield of category theory is universal algebra. Algebra refers to the study of sets which have extra structure denoted by operations and equations. Universal algebra abstracts away from particular instances (*models*) of algebras and studies *algebraic theories*, which are generic algebraic structures defined by those operations and equations. In the language of computer science, algebraic theories specify the abstract interface, whereas models (or 'algebras') provide an implementation to that interface satisfying all the specified assumptions.

This section aims to provide an informal (but standard) overview of the basics of universal algebra by introducing a notion of simple (non-parameterised) algebraic theory and its corresponding semantic objects like Lawvere theories and Monads. Then, we show how algebraic theories give rise to notions of computational effects. This presentation follows various tutorial papers ([8, 60, 32]) and textbooks ([61, 38]), which we invite keen readers to browse through.

2.2.1 Syntactic notion of algebraic theories

Consider the definition of monoids. A monoid is defined as a set M with operation $\cdot : M^2 \to M$ and $\epsilon : 1 \to M$ (equivalently $\epsilon \in M$) such that

$$(x \cdot y) \cdot z = (x \cdot y) \cdot z$$

$$x \cdot \epsilon = x$$

$$\epsilon \cdot x = x$$

for any $x, y, z \in M$. Similarly, we could also define a 'monoid' in a cartesian category[2] $\mathcal{C}$ other than **Set**, i.e. an object $M \in \mathcal{C}$ along with morphisms $M^2 \to M$ and $1 \to M$ satisfying the equations.

Universal algebra allows us to abstract away from concrete instances (*models*) of monoids and consider their shared algebraic structure with the *theory* of monoids. For this purpose, we need to define a 'language' which can express the definitions of algebraic structures.

Definition 2.2.1 (Signature of simple algebraic theories). *A signature Σ is a set of operations with arities. We write* $\mathbf{op} : n$ *to express that* $\mathbf{op}$ *is n-ary.*

Now, we proceed to define the terms. A *context* is a list of distinct variables $\Gamma = x_1, x_2, x_3 ...$, and we write $\Gamma \vdash t$ to mean that t is a valid term in the context Γ. The relation $\cdot \vdash \cdot$ is the least one generated by the following rules

$$\frac{}{\Gamma, x, \Gamma' \vdash x}\ (\text{var}) \qquad\qquad \frac{(\Gamma \vdash t_i)_{i=1...n} \qquad \mathbf{op} : n}{\Gamma \vdash \mathbf{op}((x_i)_{i=1...n})}\ (\text{op})$$

Definition 2.2.2 (Axioms of simple algebraic theories). *An axiom, written as $\Gamma \vdash t = t'$, indicates that t and t' are equal under context Γ. We take the equality relation to be the least relation closed under reflexivity, transitivity, congruence and substitution.*

Definition 2.2.3 (Simple algebraic theory). *A simple algebraic theory is a signature Σ along with a set of axioms $\mathcal{E}$.*

For example, the theory of monoids, Mon, will be defined by the signature $\mu : 2, \eta : 0$ and axioms

$$\begin{aligned} x, y, z \vdash \mu(\mu(x, y), z) &= \mu(x, \mu(y, z)) \\ x \vdash \mu(x, \eta) &= x \\ x \vdash \mu(\eta, x) &= x \end{aligned}$$

Definition 2.2.4 (Models of algebraic theories). *A model is defined by an* carrier *object X in a cartesian category $\mathcal{C}$ such that*

- *Every operation* $\mathbf{op} : n$ *is interpreted as a morphism* $[\![\mathbf{op}]\!] : X^n \to X$
- *Every term $\Gamma \vdash t$ is interpreted as follows:*

$$\begin{aligned} [\![x_1...x_k \vdash x_i]\!] &= \pi_i : X^k \to X \\ [\![\Gamma \vdash \mathbf{op}(t_1...t_m)]\!] &= [\![\mathbf{op}]\!] \circ \langle [\![\Gamma \vdash t_i]\!] \rangle_{i=1...m} : X^{|\Gamma|} \to X \end{aligned}$$

- *And every equation holds, i.e. $\Gamma \vdash t = t'$ means $[\![t]\!] = [\![t']\!]$. In practice, it suffices to show that all the axioms hold.*

For example, models of Mon in **Set** correspond to the usual notion of monoid.

[2] i.e. has terminal objects and products.

2.2.2 Semantic notions of algebraic theories

While the syntactic notion of algebraic theories is convenient to work with, it has an important flaw: different signatures along with different axiomatisations could turn out to correspond to the same theory. Thus, we call a syntactic theory a *presentation* and define presentation-agnostic semantic notions of theory. A semantic theory is then said to be 'presented' if there exists a presentation which generates it.

2.2.2.1 Lawvere theory

The first construction is the Lawvere theory [37]. Informally, a Lawvere theory $\mathbb{L}$ is a category where all objects are a finite product of a special object $X \in \mathbb{L}$. Morphisms (which do not stem from the definition of products) then correspond to *operations*: for example, an n-ary operation is a morphism $X^n \to X$. Morphisms are equated according to the algebraic structure we wish to model.

Following the tradition of functorial semantics, the required predefined structure (finite products of a special object) is given by a category with only those structures, $\mathbb{F}^{op}$, along with a structure-preserving functor $K : \mathbb{F}^{op} \to \mathbb{L}$.

Let $\mathbb{F}$ be the skeleton category of **FinSet**: its objects are $n \in \mathbb{N}$ and $\mathbb{F}(n, m)$ is the set of all possible functions from $[n]$ to $[m]$. Further, let the coproducts in $\mathbb{F}$ be strict. We claim that $\mathbb{F}^{op}$ encodes exactly the required structure: the functions $[n] \to [m]$ are in bijective correspondence with the possible functions $X^m \to X^n$ using only 'structural' operations like projection and diagonal morphisms stemming from the products.

Definition 2.2.5 (Lawvere theory)**.** *A Lawvere theory is a category $\mathbb{L}$ along with an identity-on-objects and strictly finite product preserving functor $K : \mathbb{F}^{op} \to \mathbb{L}$.*

Then, in $\mathbb{L}$, the object 0 is the terminal object and any object n corresponds to the n-ary product of 1. By convention, we can then write $1 = X^1 = X$, $n = X^n$ and $0 = X^0 = I$.

A presentation generates a Lawvere theory by taking the terms modulo the equations as a syntactic category. On the other hand, a Lawvere theory is more general than a finite presentation: there could be algebraic theories which can't be axiomatised with a finite number of operations and equations.

Remark 2.2.6 (Combining algebraic theories)**.** *A big advantage of the Lawvere theory construction is that it provides natural semantic notions of the syntactic combination of algebraic theories [31]: the sum and the tensor. On the one hand, at the level of presentations, the sum $\mathbb{L} + \mathbb{L}'$ (which forms a coproduct) corresponds to taking the operations and axioms from both theories and not adding any additional equation. On the other hand, the tensor $\mathbb{L} \otimes \mathbb{L}'$ (which is a symmetric monoidal structure) corresponds to the same thing as the sum, except we add equations such that every operation from $\mathbb{L}$ commutes with every operation from $\mathbb{L}'$.*

Finally, a model can be defined naturally as a category along with another structure-preserving functor:

Definition 2.2.7 (Model of Lawvere theories)**.** *Let $(\mathbb{L}, K)$ be a Lawvere theory. A model to the theory is given by a product-preserving functor $M : \mathbb{L} \to \mathcal{C}$.*

Here, M simply points to a particular object X in $\mathcal{C}$, which acts as the carrier object.

2.2.2.2 Monad

Another semantic notion of algebraic theory is the notion of monads. Monads give a semantic notion of algebraic theories because each syntactic algebraic theory uniquely induces a monad in **Set** via its free model.

Definition 2.2.8 (Monad)**.** *A monad on category $\mathcal{C}$ is an endofunctor $T : \mathcal{C} \to \mathcal{C}$ along with natural transformations $\mu : TT \to T$ and $\eta : \mathbf{Id} \to T$ such that the following diagrams commute:*

$$\begin{tikzcd}
T \arrow[r, "\eta T"] \arrow[dr, "\mathbf{id}"'] & TT \arrow[d, "\mu"'] & T \arrow[l, "T\eta"'] \arrow[dl, "\mathbf{id}"] \\
& T &
\end{tikzcd}$$

$$\begin{tikzcd}
TTT \arrow[r, "T\mu"] \arrow[d, "\mu T"'] & TT \arrow[d, "\mu"] \\
TT \arrow[r, "\mu"] & T
\end{tikzcd}$$

In computer science, it is often useful to consider the alternative equivalent definition [41] of monads as Kleisli triples:

Definition 2.2.9 (Monad as a Kleisli triple)**.** *A monad is a triple $(T, \eta, [\cdot]^\dagger)$ where T is an object map from $\mathcal{C}$ to $\mathcal{C}$, $\eta_X : X \to TX$ is a morphism for every $X \in \mathcal{C}$, and, for every $f : X \to T(Y)$ in $\mathcal{C}$, a morphism $TX \to TY$ such that:*

- $(\eta_X)^\dagger = \mathbf{id}_{TX}$
- *for every $f : X \to TY$, $f^\dagger \circ \eta_X = f$*
- *for every $f : X \to TY$ and $g : W \to TX$, $f^\dagger \circ g^\dagger = (f^\dagger \circ g)^\dagger$.*

Remark 2.2.10 (Equivalence of the two definitions)**.** *T is a functor with $Tf = (\eta_B \circ f)^T$ for $f : A \to B$. We can then further derive that η is a natural transformation $\mathbf{Id} \to T$. Then, we can define the multiplication simply as $\mu_A : TTA \to TA \triangleq [\mathbf{id}_{TA}]^\dagger$.*

Conversely, given $f : A \to TB$, we can obtain $[\cdot]^\dagger$ from $\mu : TT \to T$ by defining $f^\dagger \triangleq TA \overset{Tf}{\to} TTB \overset{\mu_B}{\to} TB$.

Let $P = (\Sigma, \mathcal{E})$ be a presentation and let A be an arbitrary set. We define the free model $F_P(A)$.

We first define free terms:

$$\begin{aligned} t \triangleq\ & \mathbf{return}\ a && (a \in A) \\ \mid\ & \mathbf{op}(t_1 ... t_n) && (\mathbf{op} : n \in \Sigma) \end{aligned}$$

We denote the set of all such free terms $\mathsf{Tree}_\Sigma(A)$.

Furthermore, let $=_\mathcal{E}$ be the equality our axioms induce as defined in definition 2.2.2. Then we can define an equivalence relation $\approx$ over $\mathsf{Tree}_\Sigma(A)$ simply by taking every instance of the relation $\Gamma \vdash t =_\mathcal{E} t'$ and including every possible way of closing t and t' by substituting each $x \in \Gamma$ with **return** a_x (for $a_x \in A$).

Definition 2.2.11 (Free model)**.** *For P a simple presentation, its free model over set A is defined by $F_P(A) \triangleq \mathsf{Tree}_\Sigma(A)/\approx$.*

Example 2.2.12. *The free model of* Mon *over a set A is exactly the free monoid over A, i.e. the set of finite lists with elements from A, which forms a monoid with list concatenation and the empty list.*

Proposition 2.2.13 (Free model monad (e.g. [8]))**.** *The free model F_P forms a monad over* **Set***.*

Proof. The object map is given by F_P, η is given by $\eta_A(a) \triangleq [\mathbf{return}\ a]$, and for any $\phi : A \to TB$, $[\phi]^\dagger : TA \to TB$ is given by induction as follows:

- $\phi^\dagger([\mathbf{return}\ a]) \triangleq \phi(a)$
- $\phi^\dagger([\mathbf{op}(t_1 ... t_n)]) \triangleq \mathbf{op}(\phi^\dagger([t_1]) ... \phi^\dagger([t_n]))$.

□

Example 2.2.14 (Free monoid monad)**.** *The free model of* Mon *induces exactly the free monoid monad, i.e. the List monad for Haskell users. The unit η_A applied to a just returns singleton list $[a]$, and for any $\phi : A \to TB$ the $\dagger$ operation gives $\phi^\dagger = \mathsf{flatmap}\ \phi$, where* $\mathsf{flatmap}\ \phi$ `a_list` $= \mathsf{flatten}(\mathsf{map}\ \phi$ `a_list`$)$ *in functional programming notation.*

Finally, we could further define the notion of *algebras* of monads, which correspond to a particular object A in the category $\mathcal{C}$ on which the monad T is defined, along with a morphism $TA \to A$ which intuitively gives an 'implementation' to the structure defined by the monad. When a monad on **Set** is generated by an algebraic theory, its algebras turn out to correspond to the models of that algebraic theory in **Set**. We leave details to the reader.

2.2.2.3 Correspondence

But are monads and Lawvere theories related? It turns out that (simple) Lawvere theories, as we present them here, correspond exactly to a specific class of monads over **Set** called 'finitary', which intuitively means that the underlying functor is fully determined by its definition on the 'finitary' elements of **Set** (i.e. the finite sets).

While we will not show the full proof, let us attempt to provide some intuition.

If $T : \mathcal{C} \to \mathcal{C}$ is a monad, it determines a special category called the *Kleisli Category* Kl_T, where objects are the same as $\mathcal{C}$ and morphisms are defined as $\mathsf{Kl}_T(A, B) \triangleq \mathcal{C}(A, TB)$. We now claim that the Lawvere theory $\mathbb{L}_P$ of a presentation P is exactly the opposite of the Kleisli category of the corresponding monad of **Set**, restricted to finite sets, $\mathsf{Kl}_{T_P^{\mathbf{FinSet}}}$.

To see this, take our running example, Mon, and call its corresponding Lawvere theory $\mathbb{L}$ and its corresponding monad T. Then, consider $\mathbb{L}(2, 1)$, the set of all distinct 'operations' from X^2 to X. Because $\mathbb{L}$ is presented by Mon, this set is exactly the set of distinct operations of the form $x, y \vdash t$ modulo $=$, the equivalence relation generated by Mon's axioms. Writing Mon's multiplication infix with $\cdot$ and its unit with ϵ we can see that the possible terms are exactly

$$\{x, y \vdash \epsilon; x, y \vdash x; x, y \vdash y; x, y \vdash x \cdot x; x, y \vdash x \cdot y; x, y \vdash y \cdot y; ...\}$$

which is exactly the set of distinct finite lists we can form out of 2 distinct elements. In general, then, $\mathbb{L}(m, 1)$ is in bijective correspondence with all finite lists with elements from an m-element set, and this generalises to $\mathbb{L}(m, n)$ easily as $\mathbb{L}(m, n) \cong \mathbb{L}(m, 1)^n$.

On the other hand, let's write n to denote an n-element set for any $n \in \mathbb{N}$. Then $\mathsf{Kl}_T(1, m) = \mathbf{Set}(1, T(m)) \cong T(m)$ corresponds to the underlying set of the free monoid over m, i.e. exactly the number of possible finite lists over a set of m elements, and this similarly generalises to $\mathsf{Kl}_T(n, m) = \mathbf{Set}(1, T(m)) \cong T(m)^n$.

Thus

$$\mathbb{L}(m, n) \cong \mathsf{Kl}_T(n, m)$$

which gives

$$\mathbb{L} \simeq \mathsf{Kl}^{op}_{T^{\mathbf{FinSet}}}.$$

2.2.3 Algebraic Effects

Why, as computer scientists, do we care about these mathematical constructions? In short, because they capture notions of computation.

Plotkin and Power, in their seminal paper in 2002 [56], show how notions of computation can be presented in the form of algebraic theories, i.e. operations along with equations. For pedagogical reasons, we follows Bauer's tutorial [8] take the opposite perspective and see how an algebraic theory can be interpreted computationally.

Consider the term

$$x, y \vdash \mu(x, y)$$

in Mon. Here, x, y are traditionally interpreted as elements of the monoid, and μ as the monoid multiplication. Instead, we interpret x and y as *computations* and μ as a computational operation. Recalling that a pair is the same thing as a function from booleans ($X^2 \cong 2 \to X$), we could equivalently write

$$x, y \vdash \mu(\lambda b.\ \textbf{if}\ b\ \textbf{then}\ x\ \textbf{else}\ y).$$

The meaning of μ now becomes *generating a boolean, and then running the continuation x or y accordingly.* In other words, an algebraic theory corresponds to a notion of effectful computation in *continuation-passing* style (or **algebraic** style), and models of such algebraic theories are their implementations: the carrier object X becomes the set of computations, and n-ary operations, interpreted as morphisms of the form $X^n \to X$, are ways of *forming a new computation from n other computations.*

In the case of Mon, the binary operation μ can be interpreted as a notion of non-determinism, where each non-deterministic branch is run with the results collected together in a list. Here, μ is an associative operation which corresponds to list concatenation, and the unit ϵ is interpreted as the empty list, i.e. a way of marking that a particular branch does not give results.

On the other hand, Moggi [41] presented monads as a way of representing effects. To date, this is the most widely adopted method, partly thanks to Haskell's popularity. Let $(T, \eta, [\cdot]^\dagger)$ be a monad on **Set**, its Kleisli category naturally represents gives a notion of computation. We take $T(A)$ to mean 'computations returning a value of type A', $\eta_A(x)$ to mean the 'do nothing' computation returning the value $x : A$, and for any computations $x : T(A)$ and $\phi : A \to T(B)$, we can sequence them by taking $\phi^\dagger(x) : T(B)$.

Working in the Kleisli category of a monad gives rise to **direct** style programming. For instance, consider the List monad $T_{\mathsf{List}} \triangleq \mathsf{List}(A)$, generated by the monoid theory (e.g. §2.7 in [71], defined in example 2.2.14). It is interpreted as the same notion of non-deterministism presented above. Using the correspondence between algebraic theories and monads presented in section 2.2.2.3, we could interpret the monoid operations μ and η in the Kleisli category of T_{List}, with $[\![\mu]\!] : 1 \to T_{\mathsf{List}}(2)$ and $[\![\eta]\!] : 1 \to T_{\mathsf{List}}(0)$ (not to mix up with the monad's multiplication and unit). Here, $[\![\eta]\!]$'s type forces it to give the empty list, while $[\![\mu]\!]$ has the type of a function 'generating' a boolean (instead of a function choosing between two continuations). This style of programming is closer to what one would write in normal imperative programming, hence why it is qualified as 'direct'.

Remark 2.2.15 (Parameterised theories)**.** *In this section, we only considered non-parameterised algebraic theories, which model only a very limited number of effects. We usually use a more generalised notion of algebraic theory that is parameterised, with operations of the form* $\mathbf{op}(p, \lambda a.\ t)$ *where* $p : P$ *corresponds to the parameter, and* $a : A$ *the generalised arity, where* A *is no longer assumed to be finite. We can then write* $\mathbf{op} : (P \mid A)$*. For instance, we can imagine a theory modelling states with an operation* $\mathsf{read} : (L \mid V)$ *and* $\mathsf{write} : (L \times V \mid 1)$ *where* L *is the collection of locations and* V *the type of values those locations can store. Later in this dissertation, we also consider operations which can send and receive qubits ('quantum' bits):* $\mathsf{in} : (1 \mid \mathbf{qbit})$ *and* $\mathsf{out} : (\mathbf{qbit} \mid 1)$*.*

2.3 Quantum Computing and String Diagrams

Quantum computing is notoriously tedious to describe using the traditional linear-algebraic notations. In this work, we instead opt for string-diagrammatic tools developed in the Categorical Quantum Mechanics (CQM) programme (e.g.[3, 28]).

This section starts with an overview of basic quantum information theory, followed by a description of the required string diagrammatic tools based on monoidal categories. Then, we use knowledge from both sides to describe the category **CP**, which we make use of extensively in this work.

2.3.1 Quantum Computing

Quantum computing is a computational model which relies on quantum phenomena like superposition and entanglement to perform computation. In this section, we give a brief introduction to basic quantum computing and quantum information theory. We assume knowledge of basic linear algebra and Hilbert spaces. For more detail, we invite the reader to textbooks like [42, 72].

2.3.1.1 Basics

Bra-ket notation We note column vectors $\psi \in \mathbb{C}^n$ as $|\psi\rangle$ and their complex conjugates $\psi^\dagger$ as $\langle\psi|$. Then, inner product between two vectors $\phi, \psi \in \mathbb{C}^n$ can be noted as $\langle\phi|\psi\rangle$, and the outer product as $|\phi\rangle\langle\psi|$. Finally, their tensor product $\phi \otimes \psi$ can be noted as $|\phi\rangle|\psi\rangle$ or sometimes $|\phi\psi\rangle$.

States The space of possible (pure) quantum states consists of the normalised vectors of some Hilbert space $\mathcal{H}$. The basic unit of computation is the *qubit*, which corresponds to the space $\mathbb{C}^2$. We can then define $|0\rangle \triangleq (1, 0)^T$ and $|1\rangle \triangleq (0, 1)^T$.

Superposition The power of quantum computing comes from the fact that qubits are normalised vectors: instead of only having two possible values, 0 and 1, like a normal bit, a qubit can take any value in the unit ball. For example, it can take value $\alpha|0\rangle + \beta|1\rangle$ (where $|\alpha|^2 + |\beta|^2 = 1$). When this is the case, we say that the qubit is in *superposition*, intuitively meaning that it is 'simultaneously' in both states.

Entanglement Qubit states can be combined together using the tensor product. For example, $|0\rangle \otimes |1\rangle = |01\rangle = (0, 1, 0, 0)^T \in \mathbb{C}^4 = \mathbb{C}^2 \otimes \mathbb{C}^2$. However, not all states in $\mathbb{C}^2 \otimes \mathbb{C}^2$ can be separated into the tensor product of two individual states in $\mathbb{C}^2$. For example, consider the state $\psi = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$: the reader can check that it is not separable. When this is the case, we say that the state *entangled*. Entanglement can be the vector of non-local effects. Where two physically separated qubits are entangled, observing one determines the state of the other one.

Unitary Evolution A unitary is a matrix U such that $U^\dagger = U^{-1}$. According to the Schrödinger equation, closed quantum systems evolve according to a unitary operator. In quantum computing, we usually talk about 'applying' a unitary on a quantum state, meaning evolving it accordingly. We additionally use a set of unitary 'gates' as the basic building blocks of the unitary. For example, Pauli X gate, defined as $X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, is similar to the quantum equivalent of a not gate: $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$.

No-go: no-cloning One can show that there is no unitary that allows one to clone arbitrary quantum states, i.e. there is no U such that $U|\psi\rangle|0\rangle = |\psi\rangle|\psi\rangle$.

Measurement In quantum computing, measurement is the way to observe a quantum state. We can use it to obtain classical information, as well as to distinguish between different states. A measurement is defined as a set of operators $\{M_m\}_i$ such that $\sum_m M_m^\dagger M_m = \mathbf{id}$, where the index m corresponds to the classical outcome of the measurement. Given a state ψ, performing such a measurement on it yields outcome m with probability

$$p(m) = \left\langle \psi \middle| M_m^\dagger M_m \middle| \psi \right\rangle$$

(this is the *Born rule*) and the state evolves as follows

$$|\psi\rangle = \frac{M_m|\psi\rangle}{\sqrt{p(m)}}$$

Note that because we are 'opening' the quantum system, its evolution is no longer a unitary.

Classical control This refers to using the classical outcomes of a prior measurement to control the application of later gates.

Demolition and non-demolition measurement Informally, for some instances of measurements, we can distinguish two variants: demolition and non-demolition. One nice example is the computational basis measurement. The non-demolition variant would be the set $\{|0\rangle\langle 0|, |1\rangle\langle 1|\}$: given a state of the form $\alpha|0\rangle + \beta|1\rangle$, doing a computational basis measurement gives the classical outcome 0 (resp. 1) and evolves the state to $|0\rangle$ (resp. $|1\rangle$) with probability $|\alpha|^2$ (resp. $|\beta|^2$). The demolition variant would be the set $\{\langle 0|, \langle 1|\}$: performing it demolishes the quantum state, but the classical outcomes still occur, and so with the same probabilities as above.

No-go: no-signalling When two physically separated qubits are entangled, measuring one collapses the state of the other qubit instantaneously. Thus, it might seem like this allows simultaneous communication of information. However, this is not true according to the no-signalling theorem: it is not possible for the observer of one qubit to communicate any information in the information-theoretic sense to the observers of the other qubit via measurement, and so despite the collapsing of the other qubit's state.

Global and Local Phase The astutious reader may have noticed that the formulation of the non-demolition computational basis measurement has omitted a factor of $\alpha/|\alpha|$ (resp. $\beta/|\beta|$) on the post-measurement state. This is allowed because it is a global phase. A *global phase* is a factor $e^{i\theta}$ applied globally to the quantum state. For example, a state $e^{i\theta}|\psi\rangle$ is said to be equal up to a global phase to $|\psi\rangle$. Importantly, the measurement statistics of these two states are completely equal, which is why we can often work up to the global phase. On the other hand, *local phases* are factors which apply locally to a part of the state. For example, the states $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ differ by a local phase on $|1\rangle$. Local phases cannot be ignored; indeed, $|+\rangle$ and $|-\rangle$ have very different measurement statistics. For instance, we can distinguish them using the measurement $\{|+\rangle\langle +|, |-\rangle\langle -|\}$.

Quantum Circuit Model In quantum computing, algorithms are often described in terms of the quantum circuit model. It is a computational model in which we are allowed to create new qubits, apply unitaries, perform computational basis measurements, and use measurement results to control later gates classically.

2.3.1.2 Quantum Computing with Mixed States

Intuition As Nielson and Chuang present it, the language of density operators is used to describe a system whose state is not completely known ([42] §2.4). In particular, it allows us to describe quantum systems which are in one of a number of states $|\psi_i\rangle$, with probability p_i each. We call the set $\{(p_i, |\psi_i\rangle)\}$ an ensemble of pure states or a *mixed state*, with a density operator of the form

$$\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i|.$$

One can already begin to notice an advantage of using this formalism: it allows us to ignore the global phase. If we have a pure state $e^{i\theta}|\psi\rangle$, its density operator would be $e^{i\theta}|\psi\rangle\langle\psi|e^{-i\theta} = |\psi\rangle\langle\psi|$.

Formal definition We now proceed to the formal characterisation of density operators. We will write $\mathcal{M}_n(\mathbb{C})$ as the set of complex matrices. Then, the qubit type will be $\mathcal{M}_2(\mathbb{C})$.

Definition 2.3.1 (Positive Semidefinite Matrix)**.** *A matrix $M \in \mathcal{M}_n(\mathbb{C})$ is said to be positive semidefinite if for all $|\psi\rangle \in \mathbb{C}^n$, $\langle\psi|M|\psi\rangle \geq 0$.*

Definition 2.3.2 (Trace)**.** *Let $\rho \in \mathcal{M}_n(\mathbb{C})$. The trace is defined as $\mathbf{tr}\rho = \sum_{i=1}^{n}\langle e_i|\rho|e_i\rangle$, where the $|e_i\rangle$'s form an orthonormal basis of $\mathbb{C}^n$.*

Definition 2.3.3 (Mixed state, Density operator)**.** *A density operator is a map $\rho \in \mathcal{M}_n(\mathbb{C})$, which is positive semi-definite and has trace 1.*

The trace 1 condition generalises the sum of probabilities being 1. The positive semidefiniteness is equivalent to having the general form of $\rho = \sum_i p_i|\psi_i\rangle\langle\psi_i|$ (where $|\psi_i\rangle$ is normalised). Note that the decomposition is not unique. For example, $\frac{1}{2}(|+\rangle\langle+| + |-\rangle\langle-|) = \frac{1}{2}(|0\rangle\langle 0| + |1\rangle\langle 1|)$. Thus, as we shall see later, summing over the possibilities 'discards' the classical information of the branch we are on.

Now, we present the primitives of quantum computing in the language of density operators.

- Preparing a new qubit at state $|0\rangle$: $\rho \mapsto \rho \otimes |0\rangle\langle 0|$.
- Applying a unitary $U \in \mathcal{M}_n(\mathbb{C})$: $\rho \mapsto U\rho U^\dagger$
- Performing a measurement $\{M_m\}_m$: The probability of each outcome is given by the new Born rule, which can be derived from the previous one:

$$\begin{aligned} p(m) &= \sum_i p_i p(m|i) = \sum_i p_i \left\langle\psi_i\middle|M_i^\dagger M_i\middle|\psi_i\right\rangle \\ &= \sum_i p_i \mathbf{tr}\left[M_m^\dagger M_m|\psi_i\rangle\langle\psi_i|\right] = \mathbf{tr}\left[M_m^\dagger M_m\rho\right] \end{aligned}$$

  And when $p(m) > 0$, the state evolves as follows:

$$\rho \to \frac{M_m\rho M_m^\dagger}{p(m)}$$

Quantum Channel The notion of evolution of the quantum state is defined with the quantum channel formalism.

Definition 2.3.4 (Positive and Complete Positive Map)**.** *A map $\Phi : \mathcal{M}_n(\mathbb{C}) \to \mathcal{M}_k(\mathbb{C})$ is* positive *if it maps positive semi-definite matrices to positive semi-definite matrices. It is* complete positive *if for any auxiliary system R of any size (of the form $R = \mathcal{M}_k(\mathbb{C})$ for some $k \in \mathbb{N}$), $\mathbf{id}_R \otimes \Phi$ is positive.*

Definition 2.3.5 (Quantum Channel)**.** *A quantum channel between state spaces $\mathcal{M}_n(\mathbb{C})$ and $\mathcal{M}_k(\mathbb{C})$ is a linear map $\Phi : \mathcal{M}_n(\mathbb{C}) \to \mathcal{M}_k(\mathbb{C})$ which is complete positive and trace-preserving (i.e. $\mathbf{tr}[\Phi(\rho)] = \mathbf{tr}[\rho]$).*

The CP condition ensures that positive-semi-definiteness is preserved and that this is preserved even when parallel-composed with an arbitrary auxiliary system. The second is analogous to preservation of probability.

Note that this additional property holds. Informally, it says that if a density operator corresponds to a probability of 0, then it has to be the zero matrix.

Lemma 2.3.6. *If a density operator $\rho \in \mathcal{M}_n(\mathbb{C})$ has trace 0, then it is the 0 matrix.*

Proof. Let $|i\rangle$ be the ith computational basis vector for $i = 0...n-1$. Assume that $\sum_i \langle i|\rho|i\rangle = 0$. WLOG we can further assume that $\rho = \sum_j |\psi_j\rangle\langle\psi_j|$ where $|\psi_j\rangle$ is not necessarily normalised. Then, we can write

$$\sum_i \langle i|\rho|i\rangle = \sum_j \sum_i |\langle\psi_j|i\rangle|^2 = 0$$

so each of $\langle\psi_j|i\rangle$ is 0, for all i, and because the $|i\rangle$'s form a basis, the only possible vector that satisfies this is $|\psi_j\rangle = 0$ (the zero vector) for all j, which concludes the proof. □

Now, let's give some example channels.

Example 2.3.7. *State preparation can be taken as a CPTP map $\mathbb{C} \to \mathcal{M}_2(\mathbb{C})$, and unitary evolution is a CPTP map $\mathcal{M}_n(\mathbb{C}) \to \mathcal{M}_n(\mathbb{C})$.*

Measurement, in contrast, does not have a straightforward interpretation. One way is to completely discard the classical measurement outcome. Then, it corresponds to summing over all the possible outcomes:

Example 2.3.8 (Noisy measurement channel (§4.5.1 of [72]))**.** *Let $\{M_m\}$ be a measurement. If we measure and lose the measurement outcome, the corresponding CPTP map is:*

$$\Phi : \rho \mapsto \sum_{m:p(m)>0} p(m)\frac{M_m \rho M_m^\dagger}{p(m)} = \sum_m M_m \rho M_m^\dagger$$

(well-defined because of lemma 2.3.6).

Note that if we are considering a demolition variant of the channel, this map just corresponds to discarding both the classical and the quantum outcome. If we are considering a non-demolition variant, we get a map that corresponds to decoherence (measuring, then encoding the classical outcome as the corresponding quantum state and discarding the classical outcome). However, there is no way of giving a demolition measurement which destroys the quantum state and keeps the classical outcome.

Finally, quantum channels being linear maps, we could always work with their adjoints instead.

Remark 2.3.9 (Adjoint channel)**.** *For any linear map* $\Phi : \mathcal{M}_n(\mathbb{C}) \to \mathcal{M}_k(\mathbb{C})$*, we can define its adjoint map* $\Phi^\dagger : \mathcal{M}_k(\mathbb{C}) \to \mathcal{M}_n(\mathbb{C})$ *with respect to the Hilbert-Schmidt inner product* $\langle C, D\rangle \triangleq \mathbf{tr}[C^\dagger D]$ *for* $C, D \in \mathcal{M}_n(\mathbb{C})$*. We can then show that the adjoint of CP maps are CP maps and that the adjoint of CPTP maps are 'CPU', i.e. CP and Unital, meaning that the map preserves the identity matrix:* $\Phi(\mathbf{id}) = \mathbf{id}$*.*

Remark 2.3.10 (Schrödinger and Heisenberg pictures)**.** *The duality between CPTP and CPU maps gives rise to two ways of characterising quantum computing: the Schroedinger picture is where the state evolves, but the observables stay constant, and the quantum channels from operator spaces* A *to* B *there correspond to CPTP maps* $\Phi : A \to B$*; the Heisenberg picture is where we make the observables evolve instead of the state, and the corresponding quantum channel will be a CPU map* $\Phi^\dagger : B \to A$ *which evolves the observable space.*

2.3.1.3 Including classical information

To include classical information, what we need is a coproduct-like structure $\oplus$ used to represent classical branching. Then, we can represent the classical boolean type as $\mathbb{C} \oplus \mathbb{C}$. In this section, we sketch the more common way based on operator algebras and, in particular, C* Algebras:

Definition 2.3.11 (C* Algebra (e.g. [30, 50]))**.** *A complex algebra consists of a vector space* V *over* $\mathbb{C}$ *and an operation* $\cdot : V \times V \to V$ *linear in each argument. A C* algebra* A *is a complex algebra* A *along with an additional operation* $* : A \to A$ *such that*

$$\begin{aligned}
\mathbf{x}^{**} &= \mathbf{x}\\
(a\mathbf{x})^* &= \bar{a}\mathbf{x}^*\\
(\mathbf{x}+\mathbf{y})^* &= \mathbf{x}^* + \mathbf{y}^*\\
(\mathbf{x}\mathbf{y})^* &= \mathbf{y}^*\mathbf{x}^*\\
1^* &= 1
\end{aligned}$$

and A *has a norm which makes it a Banach algebra, that is:*

$$\begin{aligned}
||\lambda\mathbf{x}|| &= |\lambda|||\mathbf{x}||\\
||\mathbf{x}+\mathbf{y}|| &\leq ||\mathbf{x}|| + ||\mathbf{y}||\\
||\mathbf{x}\mathbf{y}|| &\leq ||\mathbf{x}|| \cdot ||\mathbf{y}||
\end{aligned}$$

for all $\mathbf{x}, \mathbf{y} \in A$ *and* $\lambda \in \mathbb{C}$*, and* A *is complete in the metric* $d(a, b) = ||a - b||$*. Moreover, the following identity is satisfied:*

$$||\mathbf{x}^*\mathbf{x}|| = ||\mathbf{x}||^2.$$

We can now write $\mathcal{M}_n(A)$ as the $n \times n$ matrices with A entries, for any C* algebra A.

Luckily, in this thesis, we only work with finite-dimensional C* algebras, which are exactly the finite direct sums of matrix algebras up to isomorphism. This saves us from dealing with the complex mathematical definition presented above.

Example 2.3.12 (Matrix Algebra)**.** *The set $\mathcal{M}_n(\mathbb{C})$ has a C* algebra structure, with $(-)^* \triangleq (-)^\dagger$, $\cdot$ defined as matrix multiplication and $+$ as addition.*

Definition 2.3.13 (Direct sum of C* algebras)**.** *Let A, B be C* algebras. Their direct sum, $A \oplus B$, is defined by the set $\{(a, b) \mid a \in A, b \in B\}$, the algebraic structure is defined component-wise, and the norm is given by $||(a, b)|| = \max\{||a||, ||b||\}$.*

Proposition 2.3.14 (Finite Dimensional C* Algebras)**.** *All finite dimensional C* algebras are a finite direct sum of matrix algebras, i.e. of the form $\bigoplus_i \mathcal{M}_{n_i}(\mathbb{C})$, up to isomorphism.*

We shall see later that the direct sum $\oplus$ is a *biproduct* (i.e. both a product and a coproduct) in the category we are interested in. We therefore use it to model classical branching.

Now, we proceed to define the type of maps between them that we are interested in.

Definition 2.3.15 (Positive and Complete Positive maps)**.** *Let A be a C* algebra. A positive element $a \in A$ is a sum of elements of the form $b^* \cdot b$. A linear map between C* algebras $A \to B$ is positive if it maps positive elements to positive elements; a complete positive map $\Phi : A \to B$ is one such that for all $k \in \mathbb{N}$, $\Phi : \mathcal{M}_k(\mathbb{C}) \otimes A \to \mathcal{M}_k(\mathbb{C}) \otimes B$ is complete positive.*

Remark 2.3.16. *For C* algebras, we could equivalently define positive elements of A as those of the form b^*b for some $b \in A$ [30, 28].*

This notion of positive element does indeed generalise positive semi-definite matrices.

Example 2.3.17. *Positive elements in $\mathcal{M}_n(\mathbb{C})$ are exactly the positive semi-definite matrices (lemma IV.3.2 of [70]).*

Now, to recover the notion of density matrices, we need one more structure:

Definition 2.3.18 (Trace)**.** *For a finite dimensional C* algebra $\bigoplus_i \mathcal{M}_{n_i}(\mathbb{C})$, the trace is defined as $\mathbf{tr}((\rho_i)_i \in \bigoplus_i \mathcal{M}_{n_i}(\mathbb{C})) \triangleq \sum_i \mathbf{tr}(\rho_i)$.*

So given a finite-dimensional C* algebra $\bigoplus_i \mathcal{M}_{n_i}(\mathbb{C})$, a positive element of trace 1 is a vector of positive semi-definite matrices $(\rho_i \in \mathcal{M}_{n_i}(\mathbb{C}))_i$, such that $\sum_i \mathbf{tr}(\rho_i) = 1$. We can then interpret $\mathbf{tr}(\rho_i)$ as the probability of the ith branch, and $\frac{1}{\mathbf{tr}(\rho_i)}\rho_i$ the density operator it corresponds to.

Finally, the tensor product is inherited from the structure of the matrix algebras and distributes over the direct sum:

$$\left(\bigoplus_i \mathcal{M}_{n_i}(\mathbb{C})\right) \otimes \left(\bigoplus_j \mathcal{M}_{m_j}(\mathbb{C})\right) \cong \bigoplus_{i,j} \left[\mathcal{M}_{n_i}(\mathbb{C}) \otimes \mathcal{M}_{m_j}(\mathbb{C})\right] \tag{2.1}$$

Now, we can finally express measurements as CPTP maps, which take a quantum state and output classical information.

Example 2.3.19 (Computational basis measurement)**.** *The computational basis measurement can be written as the map in $\mathcal{M}_2(\mathbb{C}) \to \mathcal{M}_1(\mathbb{C}) \oplus \mathcal{M}_1(\mathbb{C}) = \mathbb{C} \oplus \mathbb{C}$ defined as*

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \mapsto (a, d)$$

or in bra-ket notation:

$$\rho \mapsto (\langle 0|\rho|0\rangle, \langle 1|\rho|1\rangle)$$

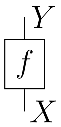
Convention 2.3.20. *From here onwards, we write* **qbit** *to denote* $\mathcal{M}_2(\mathbb{C})$ *and* **bit** *to denote* $\mathbb{C} \oplus \mathbb{C}$*.*

2.3.2 Compact closed categories and String Diagrams: a CQM primer

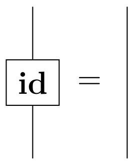
String diagrams are an alternative graphical representation of morphisms in a compact closed category, which is a symmetric monoidal closed category with extra structure. We give a brief presentation below. For more detail, we invite the reader to refer to the textbooks by Coecke and Kissinger [22] and Heunen and Vicary [28].

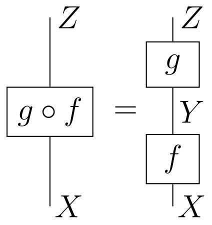
2.3.2.1 Monoidal Categories

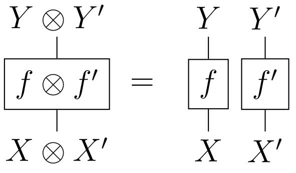
String diagrams are specialised to represent morphisms: wires correspond to objects, and boxes to morphisms.



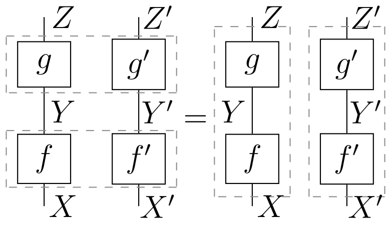
The empty wire is then the identity morphism, composition is done by connecting wires together, and the monoidal product ⊗ is represented by juxtaposing the component string diagrams.





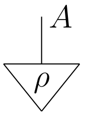


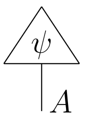
Graphically, the famous interchange law $(g \otimes g') \circ (f \otimes f') = (g \circ f) \otimes (g' \circ f')$ then follows naturally:

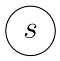


In fact, this is one of the original motivations for using string diagrams for monoidal categories.

The monoidal unit I is represented as the empty diagram. Then, we can represent states ($\rho : I \to A$ morphisms), effects ($\psi : A \to I$) and scalars $s : I \to I$ as follows:





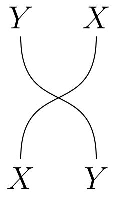


One important fact to know is that the string diagrams are a sound and complete representation of monoidal categories, as per the following theorem:

Theorem 2.3.21 (String diagrams and monoidal categories (theorem 1.8 [28]))**.** *A well-typed equation between morphisms in a monoidal category follows from the axioms iff it holds in the graphical language up to planar isotopy.*

Here, planar isotopy is the fancy way of saying that two diagrams can be continuously deformed into one another within a rectangular plane, with input and output wires touching the lower and upper boundaries of the rectangle.

For symmetric monoidal categories, we can interpret the symmetry morphism $\sigma_{X,Y}$ as



Then, the theorem above holds for three-dimensional isotopy with the additional ability of moving wires past each other.

2.3.2.2 Compact closed categories

Now, we will start defining the relevant categorified notions of linear algebra. We start by defining dual objects, akin to dual vector spaces in linear algebra.

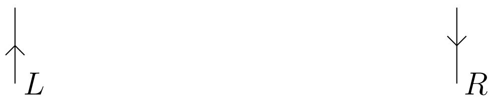
Definition 2.3.22 (Dual object)**.** *In a monoidal category, an object L is the left dual to an object R (or R is a right dual to L), written as $L \dashv R$, when there are morphisms $\eta : I \to R \otimes L$ (the* unit*) and $\epsilon : L \otimes R \to I$ (the* counit*) such that the following diagrams commute:*

$$\begin{tikzcd}
L \arrow[r, "\cong"] \arrow[d, "\mathbf{id}_L"] & L \otimes I \arrow[r, "L\otimes\eta"] & L \otimes (R \otimes L) \arrow[d, "\cong"] \\
L & I \otimes L \arrow[l, "\cong"'] & (L \otimes R) \otimes L \arrow[l, "\epsilon\otimes L"']
\end{tikzcd}$$

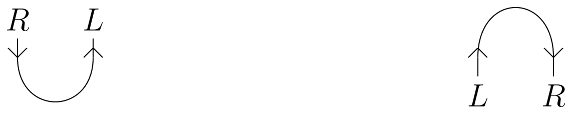
$$\begin{tikzcd}
R \arrow[r, "\cong"] \arrow[d, "\mathbf{id}_R"'] & I \otimes R \arrow[r, "\eta\otimes R"] & (R \otimes L) \otimes R \arrow[d, "\cong"] \\
R & R \otimes I \arrow[l, "\cong"'] & R \otimes (L \otimes R) \arrow[l, "R\otimes\epsilon"']
\end{tikzcd}$$

If R is both right and left dual of L, we say that R is L's dual, written as $R = L^*$.

In the graphical calculus, we express duality $L \dashv R$ using arrows on the wires: left duals have the arrow pointed upwards, whereas right duals have them pointed downwards.

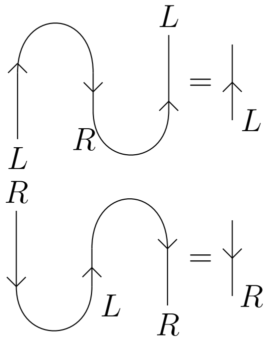


The unit and counit are written as follows:



and are, from here on, referred to as the cup and the cap, respectively.

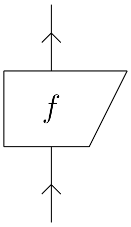
The equations these have to satisfy are then written as written as:



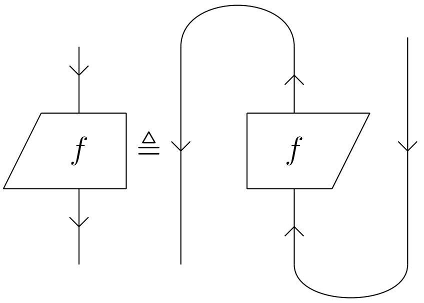
When objects are their own duals, $A \cong A^*$, we can omit the arrows on the cups and caps.

Definition 2.3.23 (Compact closed category)**.** *A (symmetric) compact closed category is a symmetric monoidal category where every object has a dual.*

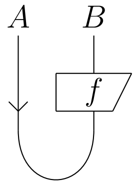
In a compact closed category, the dualising map $(\cdot)^*$ can canonically be extended to a contravariant functor (see [64]), where $f^* : B^* \to A^*$ can intuitively be thought of as the transposition of the map f. Graphically, we now represent morphisms $f : A \to B$ as:



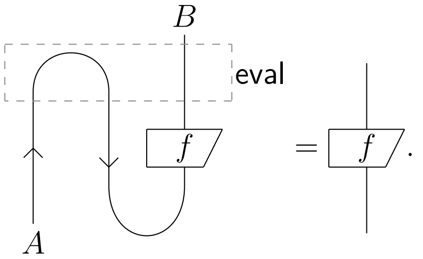
and its dual f^* as:



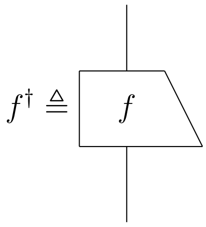
Finally, a symmetric compact closed category is, in particular, symmetric monoidal closed with the internal hom defined by $[A, B] \triangleq A^* \otimes B$. For any morphism $f : A \to B$ one can form the corresponding exponential object in $I \to A^* \otimes B$ using the unit η_A as:



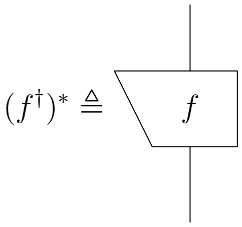
We will refer to the mapping from $\mathcal{C}(A, B)$ to $\mathcal{C}(I, [A, B])$ as the 'abstraction map', because it corresponds to a lambda abstraction. The evaluation map is defined by the corresponding cap $\mathsf{eval} : A \otimes (A^* \otimes B) \to B = (\epsilon_A \otimes B)$, such that we do indeed have:



Remark 2.3.24 (Daggers)**.** *There is one structure that we omitted: adjoints (i.e. the conjugate transpose), which are useful for the analysis of Hilbert spaces. We did so intentionally, for we would not need it in the development. In CQM, they are axiomatised as 'daggers' (§2.3 [28]): a dagger category $\mathcal{C}$ is one equipped with a contravariant functor $\dagger : \mathcal{C} \to \mathcal{C}^{op}$ that is identity on the objects, and a dagger compact closed category is one such that $\dagger$ is compatible with the tensor product and the dual objects. In graphical calculus, one can then write $f^\dagger : B \to A$ as:*



Combining dual objects and daggers gives a notion analogous to taking the complex conjugate of a linear map between Hilbert spaces:



Convention 2.3.25. *In this thesis, we only use the trapezoidal boxes to denote morphisms when we explicitly need compact closure.*

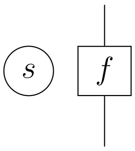
2.3.2.3 Linear structures

In addition, we need a categorified way to describe linear structures. Indeed, if density matrices are states, we want to model the ability of adding them together or multiplying them by a scalar.

Multiplicative structure Recall that scalars in a monoidal category $\mathcal{C}$ are simply morphisms of the form $I \to I$.

Definition 2.3.26 (Scalar multiplication)**.** *Let $s : I \to I$ and $f : A \to B$, we define scalar multiplication $s \bullet f$ as the morphism $A \to I \otimes A \overset{s \otimes f}{\to} I \otimes B \to B$.*

Thus, graphically, scalar multiplication is simply juxtaposing f with s.



Zeroes, Biproducts and Superposition rules We now develop the additive structure.

Definition 2.3.27 (Zero object)**.** *Let $\mathcal{C}$ be a category. A zero object $0 \in \mathcal{C}$ is an object which is both the initial and terminal object. Then, for any A, B, we can note the unique morphism $A \to 0 \to B$ as $0_{A,B}$.*

Definition 2.3.28 (Biproduct)**.** *Let $\mathcal{C}$ be a category. For any $A, B \in \mathcal{C}$, a biproduct $A \oplus B$ is both a product and a coproduct of A and B. We then note the injections $i_1 : A \to A \oplus B$ (resp. $i_2 : B \to A \oplus B$) and $p_1 : A \oplus B \to A$ (resp. $p_2 : A \oplus B \to B$).*

This structure is related to what Selinger calls a commutative monoid enrichment [64], intuitively meaning that each morphism set $\mathcal{C}(A, B)$ is a commutative monoid. This has been explicitly characterised by Heunen and Vicary [28] as a 'superposition rule':

Definition 2.3.29 (Superposition rule)**.** *A superposition rule is an operation $(f, g) \mapsto f + g$ defined for all morphism $A \overset{f,g}{\to} B$ such that $+$ is commutative and associative and has units (named $u_{A,B} : A \to B$). Additionally, $\circ$ distributes over $+$ and units (e.g. $u = f \circ u = u \circ g$ for any suitable f, g).*

Lemma 2.3.30 (Lemma 2.14 of [28])**.** *If $\mathcal{C}$ has a zero object 0, then $u_{A,B} = 0_{A,B}$.*

Biproducts and superposition rules are very much related:

Proposition 2.3.31 (Definition 2.18, Lemma 2.19 of [28])**.** *If a category $\mathcal{C}$ has a zero object 0 and a superposition rule $+$, then a biproduct $A_1 \oplus A_2$ in $\mathcal{C}$ is an object along with injection morphisms $i_k : A_k \to A_1 \oplus A_2$ and projection morphisms $p_k : A_1 \oplus A_2 \to A_k$ ($k = 1, 2$) satisfying*

- $p_k \circ i_k = \mathbf{id}_{A_k}$
- $0_{A_j,A_k} = p_k \circ i_j$ *(for $j, k = 1, 2,\ j \neq k$)*

- $\mathbf{id}_{A_1 \oplus A_2} = i_1 \circ p_1 + i_2 \circ p_2$

Proof. An object as defined satisfies the universal properties of biproducts by using

$$\langle f, g \rangle : X \to A_1 \oplus A_2 = i_1 \circ f + i_2 \circ g$$

for $f : X \to A_1$ and $g : A_2$ and

$$[f, g] : A_1 \oplus A_2 \to X = f \circ p_1 + g \circ p_2$$

for $f : A_1 \to X$ and $g : A_2 \to X$. □

Conversely, a biproduct induces a canonical superposition rule that is compatible with it:
Proposition 2.3.32 (Lemma 2.21 [28] and §2.5 [64])**.** *If a category $\mathcal{C}$ has biproducts, then it has a unique superposition rule defined with:*

$$[\mathbf{id}_B, \mathbf{id}_B] \circ (f \oplus g) \circ \langle \mathbf{id}_A, \mathbf{id}_A \rangle$$

If $\mathcal{C}$ is a monoidal category, and if $\otimes$ distributes over $+$, it also automatically implies that scalar multiplication distributes over with $+$. However, this is not guaranteed. That said, if it were, it would also be compatible with the biproduct:
Proposition 2.3.33. *In a monoidal category with a superposition rule $+$ and biproducts $\oplus$, $\otimes$ distributes over $+$ (i.e. $f \otimes (g + h) = f \otimes g + f \otimes h$ and its symmetric case), iff there is a natural isomorphism $A \otimes (B \oplus C) \cong (A \otimes B) \oplus (A \otimes C)$ (and its symmetric case).*

Proof sketch. For ($\Longrightarrow$), we construct the isomorphism using the universal property of $\oplus$, i.e. $\langle A \otimes p_1, A \otimes p_2 \rangle$ and $[A \otimes i_1, A \otimes i_2]$ and use the assumption to compose to identity.

For ($\Longleftarrow$), we simply decompose $f \otimes (g + h) = f \otimes ([\mathbf{id}, \mathbf{id}] \circ (g \oplus h) \circ \langle \mathbf{id}, \mathbf{id} \rangle)$ and use the property that $\otimes$ distributes over the biproduct $\oplus$ to conclude. □

Matrix notation In a category with biproducts, we can write morphisms in the form of a matrix as follows:
Definition 2.3.34 (Matrix)**.** *For a collection of maps $f_{m,n} : A_m \to B_n$, we define their matrix as a morphism $(f_{m,n}) : \bigoplus_m A_m \to \bigoplus_n B_n$ as follows:*

$$(f_{m,n}) \triangleq \sum_{m,n} (p_m; f_{m,n}; i_n)$$

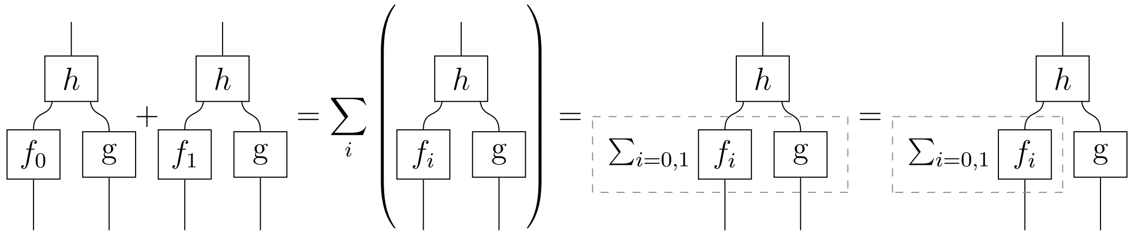
And in fact, all morphisms $f : \bigoplus_m a_m \to \bigoplus_n B_n$ can be uniquely characterised by their components $f_{m,n} : A_m \to B_n = i_m; f; p_n$, by the universal property of biproducts.

Composition can then simply be done as in matrix multiplication:

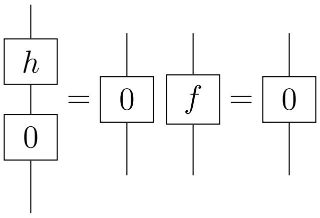
$$\left(\bigoplus_m A_m \xrightarrow{f} \bigoplus_n B_n \xrightarrow{g} \bigoplus_k C_k \right)_{m,k} = \sum_n f_{m,n}; g_{n,k}$$

Interaction with compact closure If a symmetric monoidal category $\mathcal{C}$ with biproducts and a zero object is compact closed, the zero object 0 will be its own dual and $A \otimes 0 \cong 0 \otimes A \cong 0$ (§3.3.1 [28]). Moreover, the monoidal product $\otimes$ distributes over $+$ and $\oplus$. Finally, the duals are preserved by biproducts, i.e. the dual of $A \oplus B$ will be the component-wise dual $A^* \oplus B^*$ (§3.3.2 [28]).

Graphical notation of the additive structure Assume that $\otimes$ distributes over $+$ (and equivalently over the corresponding biproduct $\oplus$). Then, we can write sums directly in diagrams:



Likewise, $f \otimes 0 = 0$ is also not guaranteed, but when it does, we can make 0 boxes absorb everything.



Remark 2.3.35. *We could further axiomatise this sort of category where $\otimes$ distributes over $\oplus$ and where* 0 *absorbs $\otimes$ as a* rig category*, or furthermore as a bipermutative category [30], but we choose not to introduce this formalism because this section focuses on introducing the string diagrams as a tool, not as an object of study.*

Free biproduct completion When we have a category $\mathcal{C}$ with a superposition rule (enriched over commutative monoids), we can construct the biproduct completion $\mathcal{C}^{\oplus}$ as follows (e.g. [64]): objects are finite tuples $[A_1, A_2...A_n]$ for $A_i \in \mathcal{C}$, and morphisms are given by matrices of morphisms in $\mathcal{C}$. Then, $\mathcal{C}^{\oplus}$ has biproducts, and the singleton functor $F : A \mapsto [A]$ is an embedding. When $\mathcal{C}$ is compact closed, and its monoidal product $\otimes$ is linear (distributes over $+$ and is absorbed by 0), then it follows that $\mathcal{C}^{\oplus}$ is compact closed by taking the dual objects component-wise.

2.3.3 The category of finite dimensional C* algebras and CP maps

We are now ready to construct the working category we will use to model quantum computing. We do so using Huot and Staton's construction [30]. An equivalent alternative could be a biproduct completion on the category of matrix algebras, which is the approach taken by [64].

Definition 2.3.36 (Category of finite dimensional C* algebras and CP maps)**.** *The category* **CP** *has, as objects, lists of positive natural numbers, and morphisms from* $[n_i]_i \to [m_j]_j$ *are defined as the complete positive maps* $\bigoplus_i \mathcal{M}_{n_i}(\mathbb{C}) \to \bigoplus_j \mathcal{M}_{m_j}(\mathbb{C})$*. Composition is composition of CP maps.*

This category has the following structure.

Biproduct and zero object $\oplus$ is given by concatenation of lists (identified with the direct sum). If we have a biproduct $M_1 \oplus M_2$, its projections are given by $p_k : (m_1, m_2) \mapsto m_k$, and its injections by $i_1 : m_1 \mapsto (m_1, 0)$ and $i_2 : m_2 \mapsto (m_2, 0)$ where 0 is a list of zero matrices.

The empty list $0 = []$ is the zero object (identified with the trivial C* algebra $\{0\} = M_0(\mathbb{C})$).

We remark that **CP** consists exactly of objects of the form $\mathcal{M}_n(\mathbb{C})$ and their biproducts, and thus we can consider all morphisms $\Phi : [n_i]_i \to [m_j]_j$ as matrices $(\Phi_{i,j})$, and composition as matrix multiplication.

Superposition rule Given $(\Phi_{i,j}), (\Phi_{i,j}) : \bigoplus_i A_i \to \bigoplus_j B_j$, their superposition rule $\Phi + \Psi$ is simply the component-wise addition $(\Phi_{i,j} + \Psi_{i,j})$.

Symmetric Monoidal Structure The monoidal product $\otimes$ is defined on objects as $[n_i]_{i=1...k} \otimes [m_j]_{j=1...p} = [n_1 m_1 ... n_1 m_p ..., n_k m_1 ... n_k m_p]$, and on morphisms as $\Phi : A \to [m]$, $\Psi : B \to [n]$ by $(\Phi \otimes \Psi) : A \otimes B \to [nm] : (M \otimes M') \mapsto (\Phi(M) \otimes \Psi(M'))$. The monoidal unit is then simply $[1]$ (identified with the unit C* algebra $\mathbb{C}$). Notice that this construction strictifies the isomorphism we showed in eq. (2.1). This structure is symmetric because the underlying tensor product is symmetric.

These two structures clearly satisfy that 1- $\otimes$ distributes over $\oplus$, and that 2- $\otimes$ is absorbed by the zero object $[]$. This allows us to use the sum notation and the 0 maps.

Compact closure Every object $X \in \mathbf{CP}$ is self-dual. For singleton objects $[n]$, the cup and cap are defined respectively as $\eta_{[n]} = \sum_{i,j} [(|i\rangle \otimes |i\rangle)(\langle j| \otimes \langle j|)]$ (where the $|i\rangle$'s form a basis of $\mathbb{C}^n$) and $\epsilon_{[n]} : (\rho : [n] \otimes [n]) \mapsto \sum_{i,j} [(\langle i| \otimes \langle i|)\rho(|j\rangle \otimes |j\rangle)]$. For arbitrary biproducts of such singletons, the cup and cap can be defined component-wise.

Remark 2.3.37. *Compact closure only holds for finite dimensional C* algebras; for general infinite dimensional C* algebras, the cups and caps are not well-defined because they would be unbounded linear operators (e.g. [28]).*

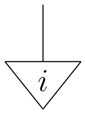
CP as a biproduct completion : We could alternatively construct **CP** as a free finite biproduct completion of **CPM**, the compact closed category of natural numbers and CP maps $\mathcal{M}_n(\mathbb{C}) \to \mathcal{M}_k(\mathbb{C})$ (i.e. the category of unnormalised density operators and CP maps between them). Indeed, **CPM** is symmetric compact closed and has a (finite) superposition rule defined as usual by addition.

States CP maps of the form $\phi : \mathbb{C} \to \mathcal{M}_n(\mathbb{C})$ are exactly the positive elements of $\mathcal{M}_n(\mathbb{C})$ (e.g. example 7.8 in [28]). In fact, this is because ϕ is a linear map, which means that we need only consider the matrix obtained at $\phi(1)$. If ϕ is additionally trace-preserving, then $\mathbf{tr}\phi(1) = 1$, making it a density operator.

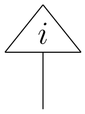
In general, a map $\psi : \mathbb{C} \to \bigoplus_i \mathcal{M}_{n_i}(\mathbb{C})$ corresponds to a list of density matrices $[M_i]_i$; when normalised (when ψ is trace-preserving), we interpret the trace of each component matrix $\mathbf{tr}(M_i)$ as the probability of that i-th branch.

Effects If we reason in terms of the adjoints of the states, we can easily see that the effects $\psi : A \to \mathbb{C}$ correspond to 'choosing a particular measurement outcome', or equivalently the component of a demolishing measurement.

Computational basis states and effects In this thesis, we will write the computational basis states in $\mathbb{C} \to \mathcal{M}_2(\mathbb{C})$ as

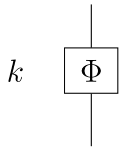


and their corresponding effects $\Phi_i : \mathcal{M}_2(\mathbb{C}) \to \mathbb{C} \triangleq \rho \mapsto \langle i|\rho|i\rangle$ as



for $i = 0, 1$. This can generalise to systems with more qubits in the standard way.

Scalars A scalar is a complete positive map of the form $s : \mathbb{C} \to \mathbb{C}$, which corresponds exactly to a non-negative real number. The number can be taken as $s(1)$ by linearity, and because $1 \geq 0$, by positivity it follows that $s(1) \geq 0$. Thus, we write the scalars directly as the real numbers, without the circle:



CPTP and CPU We give two important subcategories of **CP**:

- The **CPTP** category is the subcategory of **CP** with only the trace preserving maps as morphisms.
- The **CPU** category is the subcategory of **CP** with only the unital maps as morphisms.

Proposition 2.3.38. *We have the isomorphism of categories* $\mathbf{CPU} \cong \mathbf{CPTP}^{op}$.

3
Sums and Tensors of Parameterised Algebraic Theories

In section 2.2 of the background, we informally introduced some basic concepts of universal algebra, as well as how to use algebraic theories to model effects. We have also remarked that simple algebraic theories were insufficient to model more general computational effects. In this chapter, we introduce the generalised notion of algebraic theory we work in, explicitly characterise the semantic object generated by a syntactic theory, and construct the standard ways of combining algebraic theories (sums and tensors, cf. remark 2.2.6). These constructions and, in particular, the tensor, will be useful for giving a precise mathematical characterisation of the algebraic theory we will study in the later chapters, $\mathsf{I/O} \otimes \mathsf{QUANTUM}$.

In more detail, we start in section 3.1 by formally introducing the syntactic notion of *parameterised* algebraic theory presented by Staton in three separate works [67, 69, 68]. This will allow for operations of the form $\mathbf{op}(\vec{p}, (\vec{q_i}.t_i)_i)$, consuming the parameters from $\vec{p}$, choosing one of the i branches, and producing new parameters $\vec{q_i}$, which are then used in the t_i's. In this thesis, we will only concern ourselves with single-sorted theories, i.e. where the parameters are of the same sort (e.g. qubit). Then, in section 3.2, we present the semantic objects (particular kinds of enriched Lawvere theories) corresponding to the syntactic notion of parameterised algebraic theory and demonstrate this correspondence by explicitly constructing the enriched Lawvere theory generated ('presented') by a syntactic parameterised theory (theorem 3.2.21). This will, in section 3.3, allow us to show the existence of sums and tensors for all presented enriched Lawvere theories (theorems 3.3.11 and 3.3.15), which is the main novel result of this chapter.

$$\frac{}{\Gamma, x : p, \Gamma' \mid a_1 ... a_p \vdash x(a_1 ... a_p)} \text{(var)}$$

$$\frac{\Gamma \mid \Delta \vdash t \qquad \iota : \text{structural map}}{\Gamma \mid \Delta_\iota \vdash t_\iota} \text{(struct)}$$

$$\frac{\{\Gamma \mid \Delta, b_1 ... b_{m_i} \vdash t_i\}_{i=1...k} \qquad \mathbf{op} : (p \mid m_1 ... m_k)}{\Gamma \mid \Delta, a_1 ... a_p \vdash \mathbf{op}(a_1 ... a_p; b_1 ... b_{m_1}.t_1, ..., b_1 ... b_{m_k}.t_k)} \text{(op)}$$

Figure 3.1: Typing rules for parameterised algebraic theories

3.1 Syntactic Framework for Single-Sorted Parameterised Algebraic Theories

We start by introducing the framework in which we insert ourselves. This section is a synthesis of the definitions of Staton's framework for PATs [67, 69].

3.1.1 Syntactic framework

Definition 3.1.1 (Arity). *An arity* $(p \mid m_1 ... m_k)$ *is a tuple of a parameter* p *and a list of* k *natural numbers* $m_1, ..., m_k$*.*

We write $\mathbf{op} : (p \mid m_1 ... m_k)$ if $\mathbf{op}$ takes k parameters of arities $m_1, ..., m_k$ and returns a result of arity p. We interpret such an operation as taking p units of resource, choosing a branch $i \in \{1...k\}$, and returning m_i units of the same resource.

Definition 3.1.2 (Signature). *A signature is a set of operations* $\mathcal{O}$ *such that for each* $\mathbf{op} \in \mathcal{O}$ *there is an associated arity, written as* $\mathbf{op} : (p \mid m_1 ... m_k)$*.*

Definition 3.1.3 (Terms). *The terms of the language are defined inductively as follows:*

$$t := \mathbf{op}(a_1 ... a_k; b_1 ... b_{m_1}.t_1, ..., b_1 ... b_{m_k}.t_k) \mid x(a_1 ... a_p) \qquad \textit{(Terms)}$$

where x, y, z *are computation variables and* a, b, c *are parameters. Bound parameters are taken up to* α*-renaming.*

Definition 3.1.4 (Type system). *The typing sequent is* $\Gamma \mid \Delta \vdash t$*, where* Γ *is the 'order 1' context of computation variables associated with their valences (the number of parameters they depend on) and* Δ *is the 'order 0' context of parameters, which we think of as resources. The* Δ *is assumed to contain no duplicates. The typing relation is defined as the least relation satisfying the rules in fig. 3.1 where* ι *is a structural map of resources in a set* I*.*

To make the meaning of structural map precise, we first define substitution as follows:

Definition 3.1.5 (Parameter Substitution)**.** *For any well formed terms* $\Gamma \mid a_1...a_p \vdash t$ *we define* $t[b_1...b_p/a_1...a_p]$ *inductively as:*

$$\begin{aligned}
c[\vec{b}/\vec{a}] &= c && \textit{(if } c \neq a_\ell \textit{ for any } \ell\textit{)}\\
a_\ell[\vec{b}/\vec{a}] &= b_\ell\\
x(c_1...c_q)[\vec{b}/\vec{a}] &= x((c_i[\vec{b}/\vec{a}])_{i=1...q})\\
\mathbf{op}(\vec{c}, (\vec{d}_i.t_i)_i)[\vec{b}/\vec{a}] &= \mathbf{op}((c_j[\vec{b}/\vec{a}])_j, (\vec{d}_i.t_i[\vec{b}/\vec{a}])_i) && \textit{(if for all } i, \ell,\ b_\ell \notin \vec{d}_i \textit{ and } a_\ell \notin \vec{d}_i\textit{)}
\end{aligned}$$

where parameter equality is syntactic.

Definition 3.1.6 (α-equivalence)**.** *We say two well-formed terms* $\Gamma \mid \Delta \vdash t, t'$ *are* α *equivalent written as* $\Gamma \mid \Delta \vdash t \simeq_\alpha t'$ *if they can be* α*-renamed to each other. In other words, the* $\simeq_\alpha$ *relation is the least congruence satisfying*

$$\Gamma \mid \Delta, a_1...a_p \vdash \mathbf{op}(\vec{a}, (\vec{b}_i.t_i)_i) \simeq_\alpha \mathbf{op}(\vec{a}, (\vec{c}_i.t_i[\vec{c}_i/\vec{b}_i])_i).$$

Definition 3.1.7 (Computation variable substitution)**.** *For any well formed terms* $\Gamma \mid \Delta_j \vdash t_j$ *for all* j*, any term* t' *and any computation variable* y_j*, we partially define* $t'\{\Delta_j \vdash t_j/y_j\}_j$ *as follows:*

$$\begin{aligned}
x(a_1...a_p)\{\Delta_j \vdash t_j/y_j\}_j &= x(a_1...a_p) && \textit{(if } \forall j. x \neq y_j\textit{)}\\
x(a_1...a_p)\{\Delta_j \vdash t_j/y_j\}_j &= t_i[a_1...a_p/\Delta_i] && \textit{(if } \exists i.\ y_i = x \wedge p = |\Delta_i|\textit{)}\\
\mathbf{op}(\vec{a}; (b_i.t_i)_i)\{\Delta_j \vdash t_j/y_j\}_j &= \mathbf{op}(\vec{a}; (b_i.t_i\{\Delta_j \vdash t_j/y_j\}_j)_i)
\end{aligned}$$

Then, we enforce that I satisfies the following property:

Definition 3.1.8 (Structural maps)**.** *The set of structural maps* I *must be*

- *isomorphic to the set of all well-formed terms of the form* $x : |\Delta| \mid a_1...a_{|\Delta_\iota|} \vdash x(b_1...b_{|\Delta|})$*, where* $\{b_1...b_{|\Delta|}\} \subseteq \{a_1...a_{|\Delta_\iota|}\}$*;*
- *such that for every instance of the (struct) rule where* ι *is of the form above*

  $$\frac{\Gamma \mid \Delta \vdash t}{\Gamma \mid \Delta_\iota \vdash t_\iota}$$

  we have that

  $$t_\iota = \iota\{\Delta \vdash t/x\} = t[b_1...b_{|\Delta|}/\Delta].$$

We note that the above definition is the unique one satisfying the substitution lemma:

Proposition 3.1.9 (Substitution Lemma (Computation variables))**.** *Assume* $\Gamma \mid \vec{n}_j \vdash t_j$ *for all* j*, and* $(y_j : n_j)_j \mid \vec{p} \vdash t'$*, then* $t'\{\vec{n}_j \vdash t_j/y_j\}_j$ *exists and* $\Gamma \mid \vec{p} \vdash t'\{\vec{n}_j \vdash t_j/y_j\}_j$*.*

Proof. By simple rule induction, and in the (var) case, because the typing relation is compatible with alpha conversion. □

System	Parameters	Rules included	$\mathbf{Ctx_0}$
Bij	Linear	(exch)	**Bij**
Inj	Affine	(exch), (wk)	**Inj**
Cart	Cartesian	(exch), (wk), (contr)	**FinSet**

Table 3.1: Substructural systems

The choice of the structural maps ι depends on the resource we want to model with the parameters. In this text, we only consider three systems of structural maps: Bij for linear parameters, Inj for affine parameters and Cart for cartesian (normal) parameters.

We can alternatively present the (struct) rule as the following more familiar rules:

$$\frac{\Gamma \mid \Delta \vdash t}{\Gamma \mid \sigma(\Delta) \vdash t}\ \text{(exch)} \qquad \frac{\Gamma \mid \Delta \vdash t}{\Gamma \mid \Delta, a \vdash t}\ \text{(wk)} \qquad \frac{\Gamma \mid \Delta, a, b \vdash t}{\Gamma \mid \Delta, a \vdash t[a/b]}\ \text{(contr)}$$

where σ is an arbitrary permutation, and $t[a/b]$ is the syntactic substitution of b by a in t. Note that we use an equivalent, more general exchange rule than the usual one for our convenience. Then, we can equivalently define our three systems by including or excluding the relevant rules – a more familiar way to define 'substructural' systems for type theorists, shown in table 3.1.

The naming of the systems becomes clear given the rules: a 'bijective' structural map simply reorders the input parameters; an 'injective' one allows weakening, hence the discarding of that new parameter introduced by weakening; and a 'cartesian' one further allows contraction, hence the copying of parameters to cover for the contracted ones. We make this idea formal in the next section.

We can now proceed to define an algebraic theory.

Definition 3.1.10 (Axiom)**.** *Given a signature, an axiom is a pair of terms in the same context, written as* $\Gamma \mid \Delta \vdash t = u$*.*

The equivalence relation $\Gamma \mid \Delta \vdash t = u$ is then formed by taking all the substitution instances of the axioms and closing under reflexivity, transitivity, and congruence.

Definition 3.1.11 (Parameterised Algebraic Theory)**.** *A* parameterised algebraic theory *(PAT) in the sense of [67] is a signature coupled with a set of equations associated with it.*

Example 3.1.12 (Qubit quantum computing)**.** *We can model qubit quantum computing using PATs with linear parameters. To do so, we can define operations* new : $(0 \mid 1)$ *which creates a new qubit at* $|0\rangle$*,* measure : $(1 \mid 0, 0)$ *which performs a computational basis measurement on a qubit and chooses the branch according to the result, modelling classical control, and* $\mathsf{apply}_U : (n \mid n)$*, which applies an* $n \times n$ *unitary* U*. Then, we can add equations according to the quantum circuit model. One example could be*

$$\mathsf{new}(q.\mathsf{measure}(q, x, y)) = x$$

which says that creating a new qubit and measuring it directly gives the first branch deterministically. We will present this theory in more detail in section 4.1.1.

3.1.2 Models

We define the model as in [67] for the three systems shown above. We start by defining $(\mathbf{Ctx_0}, +, 0)$, the monoidal category of computations depending on resources, usually taken as a skeletal subcategory of **FinSet**. Its objects are natural numbers $n \in \mathbb{N}$, thought of as computations depending on resources, and its morphisms $n \to m$ are the structural maps of I of the form $x : n \mid a_1...a_m \vdash x(b_1...b_n)$. We can then characterise $\mathbf{Ctx_0}$ explicitly for different types of parameters, as shown in table 3.1.

- If we want linear parameters, the only structural morphisms allowed will be permutations, i.e. ones of the form $x : n \mid a_{\pi(1)}...a_{\pi(m)} \vdash x(a_1...a_m)$. This corresponds exactly to the category **Bij** of finite sets and bijections.
- If we want affine parameters, we want to not only allow permutations as above but structural morphisms of the form $x : n \mid a_1...a_n...a_m \vdash x(a_1...a_n)$, where we discard the parameters $a_{n+1}...a_m$. In general, these maps correspond exactly to injections from an n element set to an m element set, where we pick which n of the m parameters we wish to preserve. In other words, we get $\mathbf{Ctx_0} = \mathbf{Inj}$.
- Finally, if we want to allow permutations, discarding, but also copying of parameters, we will actually get $\mathbf{Ctx_0} = \mathbf{FinSet}$, whereby when in a morphism $f : n \to m$ two elements in the domain map to the same element from the codomain, we are effectively copying the corresponding resource and assigning it to different parameters.

Intuitively, $\mathbf{Ctx_0}$ can be considered a template category for the computations, specifying how the parameters should interact with each other. Then, when taking models in a category $\mathcal{C}$, we will mandate that it provides an 'implementation' of this structure by specifying an action $\bullet : \mathbf{Ctx_0} \times \mathcal{C} \to \mathcal{C}$. In particular, if X is the carrier object, then $p \bullet X$ will be the concrete representation of computations depending on p parameters.

Definition 3.1.13 (Models)**.** *Let $\mathcal{C}$ be a category with finite products and a $\mathbf{Ctx_0}$-action $\bullet : \mathbf{Ctx_0} \times \mathcal{C} \to \mathcal{C}$ which preserves products on its second argument. Let P be a PAT.*

A model of P in $\mathcal{C}$ is an object $X \in \mathcal{C}$ (the 'carrier') along with an interpretation for each operation $\mathbf{op} : (p \mid m_1...m_k)$, $[\![\mathbf{op}]\!] : \prod_{i=1...k} m_i \bullet X \to p \bullet X$. Furthermore, these morphisms have to satisfy that for every axiom $\Gamma \mid \Delta \vdash t = t'$, $[\![t]\!] = [\![t']\!]$ holds.

It is then a standard exercise to inductively define for each term $(x_i : m_i)_i \mid a_1...a_p \vdash t$ an interpretation $[\![t]\!] : \prod_i m_i \bullet X \to p \bullet X$, which is exactly the same as the one presented later under definition 3.2.7.

Furthermore, by induction one can show that if an equality is derivable in a theory, then it is true in all models (proposition 5 in [67]).

3.2 From presentations to $[\mathbf{Ctx_0}, \mathbf{Set}]$-Enriched Lawvere Theories

As briefly mentioned by Staton in [67], syntactic PATs correspond exactly to a particular kind of enriched Lawevere theory. In this section, we present this semantic object and explicit its relations with its syntactic counterpart.

3.2.1 Enriched Lawvere Theories: enrichment or actegory

One way of presenting this is to directly jump to the notion of enriched lawvere theories in the sense of [68]. We can then give an explicit characterisation of our PATs in terms of $[\mathbf{Ctx_0}, \mathbf{Set}]$-enriched theories.

Definition 3.2.1 (FinProd doctrine, Enriched Lawvere Theory)**.** *Let $\mathcal{V}$ be a category with* FinProd*-basis $\mathbb{F}$ (i.e. the sifted colimit cocompletion of $\mathbb{F}$ yields $\mathcal{V}$). An $\mathcal{V}$-enriched Lawvere theory following the* FinProd *doctrine is a $\mathcal{V}$-enriched category $\mathbb{L}$ with $\mathbb{F}$ powers, equipped with a strictly $\mathbb{F}$ power preserving, identity on object functor $K : \mathbb{F}^{op} \to \mathbb{L}$.*

If we ignore all the big words like 'FinProd doctrine', 'sifted colimits', '$\mathcal{V}$-enriched category' or '$\mathbb{F}$ power', we can see that the structure of this definition is similar to that of ordinary Lawvere theories (cf. definition 2.2.5): we construct a category $\mathbb{F}^{op}$ of computations and structural morphisms between them and use it as a template for the actual theory $\mathbb{L}$ by means of a structure-preserving functor $K : \mathbb{F}^{op} \to \mathbb{L}$.

Thus, instead of trying to dissect this complex definition, we will build the template category $\mathbb{F}^{op}$ from the ground up, adding all the structure that we need in the process. Also, we replace enrichment with the equivalent actegorical construction to circumvent the complicated machinery of enriched category theory.

The first thing to remark is that the category $(\mathbf{Ctx_0}, +, 0)$ defined in the previous section already has a lot of the structures that we need: it specifies exactly how non-branching computations depending on p parameters should relate with each other. However, just like in classical Lawvere theories, we additionally want to model *branching* computations of p parameters. Informally, this would correspond to objects of the form $p \times q \times r$ which is a list of computations depending on p, q and r parameters respectively and do not interact with each other.

The solution is simple. Recall that for classical Lawvere theories, we have effectively taken a free product completion of a single object X to form $\mathbb{F}^{op}$. Likewise, here, we can take the free product completion of $\mathbf{Ctx_0}$, which gives $(\mathbb{F}^{op}, \otimes_{\text{Day}}, \text{よ}(0))$, where $\mathbb{F}$ is the full subcategory of $[\mathbf{Ctx_0}, \mathbf{Set}]$ containing the coproducts of representables. Then, the free product in $\mathbb{F}^{op}$ is used to model branching computations. Note that because we know that the Day convolution preserves colimits in $\mathbb{F}$, it will in particular preserve coproducts. Dually in $\mathbb{F}^{op}$, $\otimes_{\mathrm{Day}}$ will preserve products.

$$(\prod_i \text{よ}(m_i)) \otimes_{\mathrm{Day}} (\prod_j \text{よ}(n_j)) \cong \prod_{i,j}(\text{よ}(m_i) \otimes_{\mathrm{Day}} \text{よ}(n_j)) \cong \prod_{i,j} \text{よ}(m_i + n_j) \tag{3.1}$$

Now $\mathbb{F}^{op}$ has all the structure we wanted. But we have to specify what structure we want to preserve in the actual category, say $\mathbb{L}$. By now, the reader will have guessed our approach: we choose to impose a product-preserving action $\bullet : \mathbb{F}^{op} \times \mathbb{L} \to \mathbb{L}$ as a way of specifying the relationship between computations depending on different resources:

- $\text{よ}(n) \bullet X$ is interpreted as 'making a computation in X dependent on n additional resources' in a way that is compatible with the structure of the resources.
- Product preservation: $(\text{よ}(n) \times \text{よ}(m)) \bullet X = \text{よ}(n) \bullet X \times \text{よ}(m) \bullet X$ is interpreted as an X-computation depending on either n or m additional resoures being the same thing as an X-computation depending on n resources and another one depending on m resources.

Note that eq. (3.1) above precisely allows us to define the additional structure of $\mathbb{F}^{op}$-actions in $\mathbb{F}^{op}$.

Now, as before, we define an Enriched Lawvere theory as follows:

Definition 3.2.2. *(Enriched Lawvere theory as an ordinary category with actions) A* FinProd *doctrine,* $[\mathbf{Ctx_0}, \mathbf{Set}]$*-enriched Lawvere theory* $\mathbb{L}$ *(in the sense of [68]) is*

- *A category* $\mathbb{L}$ *with an* $\mathbb{F}^{op}$*-action* $\bullet : \mathbb{F}^{op} \times \mathbb{L} \to \mathbb{L}$ *which preserves finite products on its left argument;*
- *A functor* $K : \mathbb{F}^{op} \to \mathbb{L}$ *which is identity on objects, and strictly preserves the* $\mathbb{F}^{op}$ *action.*

From the definition, we derive some interesting properties which will be useful later:

- K preserves finite products. This is because $\mathbb{F}^{op}$ has finite products, and every object in $\mathbb{F}^{op}$ is of the form $X \bullet \text{よ}(0)$, and K being identity on object and action preserving, it follows that in $\mathbb{L}$, any two objects X and Y have products $K(X) \times K(Y) = (X \times Y) \bullet K(\text{よ}(0)) = K(X \times Y)$.
- The $\mathbb{F}^{op}$-action $\bullet$ preserves finite products on its right argument. This is because for $X, Y \in \mathbb{L}$ and $A \in \mathbb{F}^{op}$, $A \bullet K(X) \times K(Y)$ is equal to $K(A \otimes_{\mathrm{Day}} (X \times Y))$ because K is product and action preserving. And because $\otimes_{\mathrm{Day}}$ preserves $\times$, we get $K(A \otimes_{\mathrm{Day}} X \times A \otimes_{\mathrm{Day}} Y)$ which is equal to $A \bullet K(X) \times A \bullet K(Y)$ by action preservation.

Remark 3.2.3 (Bifunctoriality of the monoidal action)**.** *In the properties of the monoidal action* $\bullet : \mathbb{F}^{op} \times \mathbb{L} \to \mathbb{L}$*, bifunctoriality corresponds to the following diagram, i.e. making every morphism* $f : X \to Y$ *in* $\mathbb{L}$*, i.e. every term in our theory, commute with morphisms* $s : A \to B$ *in* $\mathbb{F}^{op}$*, i.e. the structural maps.*

$$\begin{tikzcd}
A \bullet X \arrow[r, "A \bullet f"] \arrow[d, "s \bullet X"'] & A \bullet Y \arrow[d, "s \bullet Y"] \\
B \bullet X \arrow[r, "B \bullet f"'] & B \bullet Y
\end{tikzcd}$$

This means that running a term f *and then performing a structural morphism is the same as performing that structural morphism and then running* f*.*

Noting that $\mathbb{F}^{op}$ *has the same objects as* $\mathbb{L}$*, we could ask whether* $\bullet$ *could be extended to a bifunctor* $\odot\ : \mathbb{L} \times \mathbb{L} \to \mathbb{L}$*. In reality, despite being a functor on each of its arguments individually,* $\odot$ *is not a bifunctor. This is intuitively because, in general, we do not wish every term to commute with every other term, i.e. we do not want our theory to be commutative necessarily.*

As before, we can define a model as follows:

Definition 3.2.4 (Model of an enriched Lawvere theory)**.** *A model of an enriched Lawvere theory* $\mathbb{L}$ *is a category* $\mathcal{C}$ *with a finite product preserving* $\mathbb{F}^{op}$*-action, with a functor* $M : \mathbb{L} \to \mathcal{C}$ *which preserves the* $\mathbb{F}^{op}$*-action.*

Finally, for the sake of completeness, we show that this definition is equivalent to the previous one. We omit several technical details, for they are irrelevant to the development of this chapter's content.

Proposition 3.2.5. *The notion of* $[\mathbf{Ctx_0}, \mathbf{Set}]$*-enriched Lawvere theory defined in definition 3.2.2 is equivalent to an* $[\mathbf{Ctx_0}, \mathbf{Set}]$*-enriched Lawvere theory in the sense of definition 3.2.1.*

Proof Sketch following [68]. Follows from proposition 2.7 of [68] which shows that if a category $\mathbb{F}$ sifted-colimit-cocompletes to $\mathcal{V}$, then a category $\mathcal{C}$ with an $\mathbb{F}^{op}$-action $\bullet : \mathbb{F}^{op} \times \mathcal{C} \to \mathcal{C}$ which preserves finite products on its left argument is equivalent to an enrichment of $\mathcal{C}$ in $\mathcal{V}$ with powers in $\mathbb{F}$. In particular, $\mathbb{F}$ defined above does indeed cocomplete to $[\mathbf{Ctx_0}, \mathbf{Set}]$ with sifted colimits, which concludes the proof. □

Remark 3.2.6 (Sound limit doctrines)**.** *In general, for arbitrary an category* $\mathcal{V}$*, we do not obtain a well-defined notion of* $\mathcal{V}$*-enriched Lawvere theory for any choice of* $\mathbb{F}$*. We have to carefully choose* $\mathbb{F}$ *to form a 'basis' of* $\mathcal{V}$ *with respect to a 'sound limit doctrine'* D *in the sense of [68]. This intuitively means that* $\mathbb{F}$ *should be a subcategory of* $\mathcal{V}$ *such that any object in* $\mathcal{V}$ *is a* D*-filtered colimit of objects in* $\mathbb{F}$*, where* D*-filtered colimits are just a special kind of colimit. In our setting, we used the* $\mathsf{D} = \mathsf{FinProd}$ *doctrine, where* $\mathsf{FinProd}$*-filtered colimits correspond to 'sifted' colimits, another special kind of colimit.*

3.2.2 Explicit characterisation of the theory generated from a presentation

Now that we have a semantic notion of algebraic theory, we tie it back to the syntactic notion of PAT. We will show that a PAT can be considered a *presentation* of an underlying Lawvere theory by explicitly constructing the underlying theory generated by a PAT.

Syntactic Category We start by constructing a category directly from the syntax of a PAT, where contexts are objects and terms are morphisms.

Definition 3.2.7 (Syntactic category from a presentation)**.** *Let P be a PAT. We define $\mathbb{L}_P$ as its corresponding syntactic category as follows:*

- *Objects are tuples of natural numbers,*
- *Morphisms $(n_i)_i \to (m_j)_j$ are defined as follows:*

$$\mathbb{L}_P((n_i)_i, (m_j)_j) = \prod_j \mathbb{L}_P((n_i)_i, m_j)$$
$$\mathbb{L}_P((n_i)_i, m) = \{(x_i : n_i)_i \mid \vec{m} \vdash t\}/=$$

  where $=$ is the least congruence relation generated by the axioms.

- *Identity $id : (n_i)_i \to (n_i)_i$ is defined by the term $((x_i : n_i)_i \mid p_1...p_{n_j} \vdash x_j(p_1...p_{n_j}))_j$*
- *Composition $\circ$ is defined by substitution as*

$$((y_j : n_j)_j \mid \vec{p}_k \vdash t'_k)_k \circ (\Gamma \mid \vec{n}_j \vdash t_j)_j \triangleq (\Gamma \mid \vec{p}_k \vdash t'_k\{\vec{n}_j \vdash t_j/y_j\}_j)_k$$

  which is well defined by proposition 3.1.9.

Proposition 3.2.8. *The above definition forms a category.*

Proof. Check that $\circ$ is well-defined modulo the equivalence relation $=$, which follows because it is closed under substitution on both sides and is a congruence.

Also, check associativity and unity of $\circ$. □

Additionally, $\mathbb{L}_P$ has finite products by definition.

Lemma 3.2.9 (Finite Products)**.** *$\mathbb{L}_P$ has finite products.*

Proof. By definition we have that $\mathbb{L}_P(-, (n_i)_i) \cong \prod_i \mathbb{L}_P(-, n_i)$. It then follows (as a consequence of Yoneda's lemma) that $(n_i)_i$ is the finite product of the n_i's. The rest follows naturally by doing tuple concatenations. □

In more detail, we explicitly construct the morphisms corresponding to each term from their typing derivations. As a convention, to distinguish between syntactic objects like contexts Γ, Δ and terms t from semantic objects in $\mathbb{L}_P$, we write the latter objects surrounded by $[\![-]\!]$.

Explicit construction of morphisms from the typing rules. We interpret natural numbers $[\![n]\!] = n$, where the RHS is the object in $\mathbb{L}_P$.

Parameter contexts Δ are interpreted as $[\![\Delta]\!] = [\![|\Delta|]\!]$, where $|\Delta|$ gives the number of parameters of the context Δ.

Computation contexts $\Gamma = (x_i : m_i)_i$ are interpreted as $[\![\Gamma]\!] = \prod_i [\![m_i]\!]$.

An instance of the (var) rule is interpreted simply interpreted as

$$[\![(x_i : m_i)_i \mid a_1 ... a_{m_i} \vdash x_i(a_1 ... a_{m_i})]\!] \triangleq \pi_i : [\![\Gamma]\!] \to [\![m_i]\!]$$

The (struct) rule is interpreted as the following commutative diagram:

$$\begin{tikzcd}
{[\![\Gamma]\!]} \arrow[r, "{[\![t]\!]}"] \arrow[dr, "{[\![t_\iota]\!]}"'] & {[\![\Delta]\!]} \arrow[d, "{[\![\iota]\!]}"] \\
& {[\![\Delta_\iota]\!]}
\end{tikzcd}$$

where ι is taken as a non-branching structural map, or equivalently a morphism in $\mathbf{Ctx_0}$ (which as we shall see later can be embedded into $\mathbb{L}_P$ via $K \circ よ$). The notation follows definition 3.1.4, where Δ_ι is the new context, and t_ι the substituted version of t. The (op) rule is interpreted as the following commutative diagram:

$$\begin{tikzcd}
{[\![\Gamma]\!]} \arrow[r, "{\langle [\![t_i]\!] \rangle_i}"] \arrow[dr, "{[\![\mathbf{op}(\vec{p}, (\vec{m}_i . t_i)_i)]\!]}"'] & {\prod_i [\![\Delta, \vec{m}_i]\!]} \arrow[d, "\Delta \bullet \mathbf{op}"] \\
& {[\![\Delta, \vec{p}]\!]}
\end{tikzcd}$$

where $\Delta \bullet \mathbf{op} = [\![(x_i : m_i) \mid \Delta, \vec{p} \vdash \mathbf{op}(\vec{p}, (\vec{m}_i . x_i(\Delta, \vec{m}_i))_i)]\!]$. We note that this notation comes from the definition of the $\mathbf{Ctx_0}$ action later in this section, where $\mathbf{op}$ would be the morphism $[\![[\![(x_i : m_i) \mid \vec{p} \vdash \mathbf{op}(\vec{p}, (\vec{m}_i . x_i(\vec{m}_i))_i)]\!]]\!]$. □

Structural Morphisms and Canonical Functor $K : \mathbb{F}^{op} \to \mathbb{L}_P$ Recall that morphisms from $\mathbf{Ctx_0}$ are canonically identified with the structural maps from I in the syntactic framework. Its free product completion, $\mathbb{F}^{op}$, contains in addition the morphisms induced from the free product completion, i.e. projections and diagonal morphisms. It turns out that we can also easily identify those additional morphisms with 'branching' structural maps in the syntactic framework as follows:

Proposition 3.2.10 (Empty presentation gives $\mathbb{F}^{op}$)**.** *Let $P_\emptyset$ be the PAT with no operations and no equations. Then, $\mathbb{F}^{op} \cong \mathbb{L}_{P_\emptyset}$.*

Proof. We need only consider the morphisms in both categories, for their objects can be canonically identified.

A morphism in $\mathbb{F}^{op}(\prod_i \text{よ} m_i, \prod_j \text{よ} n_j)$ is a tuple of morphisms in $\mathbb{F}^{op}(\prod_i \text{よ} m_i, \text{よ} n_j)$, and a morphism in $\mathbb{F}^{op}(\prod_i \text{よ} m_i, \text{よ} n)$ is necessarily of the form of a projection, followed by a morphism in $\mathbf{Ctx_0}$ yoneda-embedded in $\mathbb{F}^{op}$, by the properties of the free product completion. Therefore, they can be identified with morphisms in $\mathbb{L}_{P_\emptyset}((m_i)_i, n)$ of the form $(x_i : m_i)_i \mid p_1...p_n \vdash x(a_1...a_{m_j})$ for some j, where $\{a_1...a_{m_j}\} \subseteq \{p_1...p_n\}$. Conversely, by reasoning with the typing rules, it is easy to see that terms of this form are the only morphisms in $\mathbb{L}_{P_\emptyset}((m_i)_i, n)$. □

Corollary 3.2.11 (Explicit characterisation of $\mathbf{Ctx_0}$)**.** $\mathbf{Ctx_0}$ *is isomorphic to the full subcategory of* $\mathbb{L}(P_\emptyset)$ *with only singleton objects.*

Proof. Follows directly from the reasoning in proposition 3.2.10. □

From the above, we can easily deduce that $\mathbb{L}_P$ contains the structure of $\mathbb{F}^{op}$.

Corollary 3.2.12. *There is a canonical identity-on-objects functor* $K : \mathbb{F}^{op} \to \mathbb{L}_P$ *for any presentation* P*.*

Proof. Simply map morphisms as in proposition 3.2.10, and by renaming $\mathbb{L}_P$'s objects as $\prod_i \text{よ}(m_i)$ instead of tuples $(m_i)_i$, we obtain that K is identity on objects. □

As a convention, from here onwards, we will call all morphisms from $\mathbf{Ctx_0}$ and their corresponding image through K as 'non-branching' structural morphisms, and all morphisms from $\mathbb{F}^{op}$ and their corresponding image through K as 'branching' structural morphisms.

$\mathbb{F}^{op}$ action We now proceed to build the $\mathbb{F}^{op}$ action. To simplify the construction, we will first build a $\mathbf{Ctx_0}$ action $\bullet : \mathbf{Ctx_0} \times \mathbb{L}_P \to \mathbb{L}_P$ which preserves products on its second argument, and canonically extend it to an $\mathbb{F}^{op}$ action, by the following lemma:

Lemma 3.2.13. *An* $\mathbb{F}^{op}$*-action restricts to a* $\mathbf{Ctx_0}$ *action preserving products on its second argument, and a* $\mathbf{Ctx_0}$ *action preserving products on its second argument canonically extends to a* $\mathbb{F}^{op}$ *action.*

Proof. Let $\bullet_{\mathbb{F}^{op}} : \mathbb{F}^{op} \times \mathcal{C} \to \mathcal{C}$ and restrict it to the representable objects, we obtain directly an action $\bullet_{\mathbf{Ctx_0}} : \mathbf{Ctx_0} \times \mathcal{C} \to \mathcal{C}$. Then, because $\bullet_{\mathbb{F}^{op}}$ preserves products on its second argument by the reasoning in the previous section, so does $\bullet_{\mathbf{Ctx_0}}$.

Conversely, let $\bullet_{\mathbf{Ctx_0}} : \mathbf{Ctx_0} \times \mathcal{C} \to \mathcal{C}$ be an action that preserves products on its second argument. Then, define:

$$(\prod_i \text{よ}(n_i)) \bullet_{\mathbb{F}^{op}} X \triangleq \prod_i (n_i \bullet_{\mathbf{Ctx_0}} X)$$

Then, we can show it is an action simply because

$$\text{よ}(0) \bullet_{\mathbb{F}^{op}} X = 0 \bullet_{\mathbf{Ctx_0}} X \cong X$$

and

$$\begin{aligned}
&\left[\prod_i (\text{よ}(n_i)) \otimes_{\text{Day}} \prod_j (\text{よ}(m_j))\right] \bullet_{\mathbb{F}^{op}} X \\
&\cong [\prod_{i,j} \text{よ}(n_i + m_j)] \bullet_{\mathbb{F}^{op}} X \\
&= \prod_{i,j} [(n_i + m_j) \bullet_{\mathbf{Ctx_0}} X] \\
&\cong \prod_{i,j} [(n_i + m_j)] \bullet_{\mathbf{Ctx_0}} X \\
&= \prod_i n_i \bullet_{\mathbf{Ctx_0}} (\prod_j m_j \bullet_{\mathbf{Ctx_0}} X) && \text{(product preservation)} \\
&= [\prod_i \text{よ}(n_i)] \bullet_{\mathbb{F}^{op}} ([\prod_j \text{よ}(m_j)] \bullet_{\mathbb{F}^{op}} X)
\end{aligned}$$

□

As noted in remark 3.2.3, the bifunctoriality of $\bullet$ requires every morphism in $\mathbb{L}_P$ to commute with non-branching structural morphisms from $\mathbf{Ctx_0}$. Now, from the previous development, we know that the morphisms in $\mathbf{Ctx_0}$ correspond exactly to the structural maps $x : n \mid \Delta \vdash x(\vec{a})$ where $\vec{a} \subseteq \Delta$. Thus, what we actually need to show is that syntactically, the non-branching structural maps commute with all other terms.

We start by defining what it means for two terms t and u to be sequenced together.

Definition 3.2.14 (Sequencing)**.** *Let P be a presentation and $\mathbb{L}_P$ be its corresponding syntactic category. For any morphism $(\Gamma \mid \Delta \vdash t) : [\![\Gamma]\!] \to [\![\Delta]\!]$ and $(\Gamma' \mid \Delta' \vdash t') : [\![\Gamma]\!] \to [\![\Delta]\!]$ we define the left sequencing $(\Gamma \times \Gamma' \mid \Delta, \Delta' \vdash t \rtimes u) : [\![\Gamma \times \Gamma']\!] \to [\![\Delta, \Delta']\!]$ (where $\Gamma \times \Gamma'$ is such that for every $x : p$ in Γ and $y : q$ in Γ' there is a $xy : p + q$ in $\Gamma \times \Gamma'$) inductively as follows:*

$$\begin{aligned}
\mathbf{op}(\vec{a}, (\vec{b_i}.t_i)_i) \rtimes u &\triangleq \mathbf{op}(\vec{a}, (\vec{b_i}.t_i \rtimes u)_i) \\
x(\vec{p}) \rtimes \mathbf{op}(\vec{c}, (\vec{d_j}.u_j)_j) &\triangleq \mathbf{op}(\vec{c}, ((\vec{d})_j.x(\vec{p}) \rtimes u_j)) \\
x(\vec{p}) \rtimes y(\vec{q}) &\triangleq xy(\vec{p}, \vec{q})
\end{aligned}$$

We can likewise define the right sequencing $t \ltimes u : [\![\Gamma' \times \Gamma]\!] \to [\![\Delta', \Delta]\!]$ which is easily seen to be equivalent. When unambiguous, we uniformly write $t; u$.

Lemma 3.2.15. *Sequencing is well-typed and well-defined.*

Proof. By rule induction. □

Definition 3.2.16 (Commutation of terms in a PAT). *Let* $\Gamma \mid \Delta \vdash t$ *and* $\Gamma' \mid \Delta' \vdash u$ *be two terms in* P. *They are said to commute if*

$$\Gamma \times \Gamma' \mid \Delta, \Delta' \vdash t \rtimes u = u \ltimes t$$

Lemma 3.2.17. *(Every term commutes with structural maps) For any well-typed term of the form* $x : n \mid p_1...p_m \vdash x(q_1...q_n)$ *where* $q_1...q_n$ *comes from* $p_1...p_m$*, and any term* $(y_j : n_j) \mid r_1...r_k \vdash t$ *we have that*

$$(xy_j : n + m_j)_j \mid p_1...p_m, r_1...r_k \vdash x(q_1...q_n) \rtimes t = t \ltimes x(q_1...q_n).$$

Proof. We will prove this for the Bij system, by induction over t. We need only consider the non-branching structural maps of form $x : n \mid p_1...p_m \vdash x(p_{\pi(1)}...p_{\pi(m)})$ where π is a permutation.

Case (var). Then $t = y_j(r_1...r_k)$ where σ is a permutation. So

$$\mathsf{LHS} = x(p_{\pi(1)}...p_{\pi(m)}); y_j(r_1...r_k) = xy_j(\vec{p_\pi}, \vec{r}) = \mathsf{RHS}$$

Case (struct). By IH we have that

$$(xy_j : m + m_j)_j \mid p_1...p_m, r_1...r_k \vdash x(p_1...p_m) \rtimes t = t \ltimes x(p_1...p_m).$$

from which it follows that

$$(xy_j : m + m_j)_j \mid p_{\sigma(1)}...p_{\sigma(m)}, r_1...r_k \vdash x(p_1...p_m) \rtimes t = t \ltimes x(p_1...p_m).$$

where σ is a permutation.

Case (op). So we get $t = \mathbf{op}(a_1...a_n, (b_1...b_{k_\ell}.u_\ell)_\ell)$. By IH we have that for each ℓ,

$$(xy_j : m + m_j)_j \mid p_1...p_m, \Delta, b_1...b_{k_\ell} \vdash x(p_1...p_m) \rtimes u_\ell = u_\ell \ltimes x(p_1...p_m)$$

Moreover, we know that

$$\frac{\forall \ell.\ (xy_j : m + m_j)_j \mid p_1...p_m, \Delta, b_1...b_{k_\ell} \vdash x(p_1...p_m) \rtimes u_\ell}{(xy_j : m + m_j)_j \mid p_1...p_m, \Delta, a_1...a_n \vdash \mathbf{op}(a_1...a_n, (b_1...b_{k_\ell}.x(p_1...p_m) \rtimes u_\ell)_\ell)}$$

and likewise we get $(xy_j : m + m_j)_j \mid p_1...p_m, \Delta, a_1...a_n \vdash \mathbf{op}(a_1...a_n, (b_1...b_{k_\ell}.u_\ell \ltimes x(p_1...p_m))_\ell)$. Hence by congruence we have

$$\mathbf{op}(a_1...a_n, (b_1...b_{k_\ell}.x(p_1...p_m) \rtimes u_\ell)_\ell) = \mathbf{op}(a_1...a_n, (b_1...b_{k_\ell}.u_\ell \ltimes x(p_1...p_m))_\ell)$$

(where we ignore the context for clarity). The rest then follows by definition of left and right sequencing. □

Proposition 3.2.18 (Action)**.** *For any P, $\mathbb{L}_P$ has a $\mathbf{Ctx_0}$ action $\bullet$ such that $m \bullet -$ preserves products.*

Proof. We define $\bullet : \mathbf{Ctx_0} \times \mathbb{L}_P \to \mathbb{L}_P$ as follows:

$$\begin{aligned}
&n \bullet (m_i)_i \triangleq (n + m_i)_i \\
&(x : n \mid a_1...a_p \vdash x(b_1...b_n)) \bullet ((x_i : m_i)_i \mid \vec{p_j} \vdash t_j)_j : n \bullet (m_i)_i \to p \bullet (p_j)_j \\
&\quad \triangleq ((x_i : n + m_i)_i \mid a_1...a_p, \vec{p_j} \vdash x(b_1...b_n) \rtimes t_j)_j
\end{aligned}$$

where we take the $n \to p$ morphism in $\mathbf{Ctx_0}$ as a syntactic term of the form $x : n \mid a_1...a_p \vdash x(b_1...b_n)$ (where $b_1...b_n \in \{a_1...a_p\}$) by corollary 3.2.11. Bifunctoriality is directly shown by lemma 3.2.17.

The laws for the monoidal action can also be easily checked to hold strictly, because $\mathbf{Ctx_0}$ is taken to be strict monoidal.

Finally, let $m \in \mathbf{Ctx_0}$, it can easily be shown that $m \bullet -$ preserves products just from the definition above. □

Corollary 3.2.19. *$\mathbb{L}_P$ has a finite product preserving $\mathbb{F}^{op}$ Action.*

Proof. Follows by building the action $\bullet : \mathbb{F}^{op} \times \mathbb{L}_P \to \mathbb{L}_P$ whereby $\text{よ}(n) \bullet X \triangleq n \bullet X$ and $\prod_i \text{よ}(n_i) \bullet X \triangleq \prod_i (n \bullet X)$. □

Finally, by definition of the $\mathbf{Ctx_0}$ action $\bullet : \mathbf{Ctx_0} \times \mathbb{L}_P \to \mathbb{L}_P$, we already know it is consistent with all the non-branching structural morphisms identified by K. We now show that its canonical extension into the $\mathbb{F}^{op}$ action is additionally consistent with the branching structural morphisms identified by K.

Proposition 3.2.20. *The canonical functor $K : \mathbb{F}^{op} \to \mathbb{L}_P$ strictly preserves $\mathbb{F}^{op}$ actions.*

Proof. In $\mathbb{F}^{op}$ the action $\bullet : \mathbb{F}^{op} \times \mathbb{F}^{op} \to \mathbb{F}^{op}$ is given by Day's convolution $\otimes_{\mathrm{Day}}$. It thus suffices to show that

$$K(\prod_i \text{よ}(m_i) \bullet \prod_j \text{よ}(n_j)) = (\prod_i \text{よ}(m_i)) \bullet K(\prod_j \text{よ}(n_j))$$

which holds simply by definition of K, and because $\bullet : \mathbb{F}^{op} \times \mathbb{L}_P \to \mathbb{L}_P$ preserves products:

$$\begin{aligned}
\mathsf{LHS} &= K(\prod_{i,j} \text{よ}(m_i + n_j)) = (m_i + n_j)_{i,j} \\
&= \prod_i ((m_i + n_j)_j) = \prod_i (\text{よ}(m_i) \bullet K(\prod_j \text{よ}(n_j))) = \mathsf{RHS}
\end{aligned}$$

□

PATs generate Lawvere Theories With all the facts above, we can now conclude with the following theorem:

Theorem 3.2.21. *For any presentation P, $\mathbb{L}_P$ is an $[\mathbf{Ctx_0}, \mathbf{Set}]$-enriched,* FinProd *doctrine Lawvere theory in the sense of definition 3.2.2.*

Proof. Follows from definition 3.2.7, lemma 3.2.9, corollary 3.2.12, corollary 3.2.12 and proposition 3.2.20. □

Additionally, direct from lemma 3.2.13 and theorem 3.2.21 we can also get that the models coincide as follows:

Corollary 3.2.22. *Let P be a presentation, then models of P in $\mathcal{C}$ coincide with models of $\mathbb{L}_P$ in $\mathcal{C}$, for any $\mathcal{C}$ with $\mathbb{F}^{op}$ actions.*

3.3 Sums and Tensors of presented theories

This section concerns the explicit construction of the standard combinations of algebraic theories – the sum and the tensor – in terms of our framework of PATs. We will first generalise the action $\bullet$ to allow for expressing morphisms $f \bullet X$, where f is not necessarily structural. By doing so, we generalise $\bullet$ into a premonoidal product $\odot$. Then, we use this structure to define the sums and tensors of *presented* enriched Lawvere theories.

3.3.1 A premonoidal product from the action and explicit characterisation

Proposition 3.3.1. *Let $\mathbb{L}$ be a $[\mathbf{Ctx_0}, \mathbf{Set}]$-enriched Lawvere theory in the sense of definition 3.2.2. Then we have that*

$$X \bullet K(Y) \cong Y \bullet K(X)$$

for any $X, Y \in \mathbb{F}^{op}$, natural in both X and Y.

Proof.

$$\begin{aligned}
\mathsf{LHS} &= K(X \bullet Y) && (K \text{ strictly preserves actions})\\
&= K(X \otimes_{\mathrm{Day}} Y) && (\text{definition})\\
&\overset{K(\sigma_{X,Y})}{\cong} K(Y \otimes_{\mathrm{Day}} X) && (\text{proposition 2.1.21})\\
&= \mathsf{RHS} && (\text{analogous reasoning}).
\end{aligned}$$

□

Proposition 3.3.2. *Let $\mathbb{L}$ be a $[\mathbf{Ctx_0}, \mathbf{Set}]$-enriched Lawvere theory in the sense of definition 3.2.2. Then the action $\bullet : \mathbb{F}^{op} \times \mathbb{L} \to \mathbb{L}$ extends to an object and morphism map $\odot : \mathbb{L} \times \mathbb{L} \to \mathbb{L}$ such that there are functors $X \rtimes -$ mapping Y to $X \odot Y$ and $- \ltimes X$ mapping Y to $Y \odot X$. We note $\rtimes$ and $\ltimes$ as $\odot$ when unambiguous.*

Proof. Knowing that $K : \mathbb{F}^{op} \to \mathbb{L}$ is identity-on-objects, we can define $K(X) \odot K(Y) \triangleq X \bullet K(Y)$, and get directly from proposition 3.3.1 that $K(X) \odot K(Y) \cong K(Y) \odot K(X)$.

Then, functoriality of $X \odot -$ follows from that of $K(X) \bullet -$. For $- \odot K(X)$, we define $f \odot K(X)$ by its transpose and functoriality follows.

$$\begin{tikzcd}
K(A) \odot K(X) \arrow[r, "f \odot K(X)"] \arrow[d, "\cong"] & K(B) \odot K(X) \\
K(X) \odot K(A) \arrow[r, "K(X) \odot f"'] & K(X) \odot K(B) \arrow[u, "\cong"']
\end{tikzcd}$$

Note that this definition is the unique one making the isomorphism natural. □

This binoidal product does indeed generalise the action $\bullet$. In fact, we can recover the bifunctoriality squares of $\bullet$ by considering the central morphisms of $\mathbb{L}$:

Proposition 3.3.3 (Structural morphisms are central morphisms of $\mathbb{L}$)**.** *In the binoidal category $(\mathbb{L}, \odot)$, for any $f \in \mathbb{F}^{op}$, $K(f)$ is a central morphism.*

Proof. Easy to show one side by bifunctoriality of the action functor $\bullet$, and the other side follows from symmetry (proposition 3.3.1), which is inherited from the symmetry of the Day convolution $\otimes_{\text{Day}}$. □

This binoidal product $\odot$ happens to additionally be premonoidal:

Proposition 3.3.4. *$\odot$ is a symmetric premonoidal product on $\mathbb{L}$ with unit $I = K(\text{よ}(0))$*

Proof. Proposition 3.3.2 shows that $(\mathbb{L}, \odot)$ is binoidal. It now suffices to show that it is premonoidal with respect to I. Then, we can define the symmetry as $\sigma_{KX,KY} \triangleq K(\sigma_{X,Y})$ by proposition 3.3.1, which conveniently is natural by definition in proposition 3.3.2.

We devise the associators and unitors as follows as directly following from those in $\mathbb{F}^{op}$: $\alpha_{KA,KB,KC} \triangleq K(\alpha_{A,B,C})$, $\lambda_{KA} \triangleq K(\lambda_A)$ and $\rho_{KA} \triangleq K(\rho_A)$. They are obviously central isomorphisms (proposition 3.3.3) and satisfy the pentagon and triangle equations by functoriality of K, and because $\mathbb{F}^{op}$ is a monoidal category. Thus, it suffices to convince oneself that each of these morphisms is natural.

To do so, we start by remarking that for any such naturality square, if the morphism f which varies the index object always appears on the right, then everything follows from the definition of the action. One example could be the following square, showing that $\alpha_{KX,KY,KZ}$ is indeed natural in KZ.

$$\begin{tikzcd}
(KX \odot KY) \odot KZ \arrow[r, "\alpha"] \arrow[d, "(KX\odot KY)\odot f"'] & KX \odot (KY \odot KZ) \arrow[d, "KX\odot(KY\odot f)"] \\
(KX \odot KY) \odot KZ' \arrow[r, "\alpha"'] & KX \odot (KY \odot KZ')
\end{tikzcd}$$

Indeed, we know that

$$(KX \odot KY) \odot f = K(X \bullet Y) \odot f = (X \bullet Y) \bullet f = (X \otimes_{\text{Day}} Y) \bullet f \cong X \bullet (Y \bullet f)$$

and that

$$KX \odot (KY \odot f) = X \bullet (Y \bullet f).$$

Moreover, the action $\bullet : \mathbb{F}^{op} \times \mathbb{F}^{op} \to \mathbb{F}^{op}$ of $\mathbb{F}^{op}$ on $\mathbb{F}^{op}$ is defined by $\otimes_{\text{Day}}$, meaning that the isomorphism $\mu_{A,B,X} : A \otimes_{\text{Day}} (B \otimes_{\text{Day}} X) \cong (A \otimes_{\text{Day}} B) \otimes_{\text{Day}} X$ is defined by α^{-1}. Thus, the diagram commutes.

For other naturality squares where f is not on the right most side, we can just use symmetry to send it to the right most side and reason with that. □

Now, we prove some simple and useful properties.

Lemma 3.3.5. *$\odot$ preserves products on both arguments*

Proof. Follows from the definition of $\bullet$ and symmetry. □

Lemma 3.3.6. *$K : \mathbb{F}^{op} \to \mathbb{L}$ is strong symmetric premonoidal.*

Proof.

$$K(X \otimes_{\text{Day}} Y) = K(X \bullet Y) = X \bullet K(Y) = K(X) \odot K(Y)$$

Moreover it is symmetric because symmetry is defined directly as $\sigma_{KX,KY} = K(\sigma_{X,Y})$ □

Lemma 3.3.7. *If a functor $F : \mathbb{L} \to \mathbb{L}'$ preserves the action, then F preserves the premonoidal product.*

Proof.

$$F(K(X) \odot K(Y)) = F(X \bullet K(Y)) = X \bullet F(K(Y)) = X \bullet K'(Y) = K'(X) \odot K'(Y)$$

□

3.3.2 Translation of Sequencing and Premonoidality

The reason why $\odot$ has to be premonoidal (and not monoidal) becomes clear when considering remark 3.2.3: $\odot$ is premonoidal because the bifunctoriality condition corresponds to commutativity of terms, which does not always hold. To make this explicit, we formulate the correspondences between syntactic sequencing defined in definition 3.2.14 and the premonoidal product.

Theorem 3.3.8 (Translation of sequencing)**.** *Let P be a presentation and $\mathbb{L}_P$ be its corresponding Lawvere theory. Let $\Gamma \mid \Delta \vdash t$ and $\Gamma' \mid \Delta' \vdash u$ be terms in P. Then the morphism $[\![\Gamma \times \Gamma' \mid \Delta, \Delta' \vdash t \rtimes u]\!]$ is exactly:*

$$[\![\Gamma \times \Gamma']\!] = [\![\Gamma]\!] \odot [\![\Gamma']\!] \xrightarrow{[\![\Gamma]\!]\odot u} [\![\Gamma]\!] \odot [\![\Delta']\!] \xrightarrow{t\odot[\![\Delta']\!]} [\![\Delta]\!] \odot [\![\Delta']\!] = \text{よ}(\Delta + \Delta')$$

Proof. First, it is easy to see that $[\![\Gamma \times \Gamma']\!] = [\![\Gamma]\!] \odot [\![\Gamma']\!]$ and $[\![\Delta]\!] \odot [\![\Delta']\!] = \text{よ}(\Delta + \Delta')$ hold by definition. For the rest, we proceed by double rule induction over t and u. Note that to avoid cluttering, we drop the $[\![\cdot]\!]$ notation when there is no ambiguity.

Case (op) on t:

We know that $\Gamma' \mid \Delta' \vdash u$ and

$$\frac{\Gamma \mid \Delta, \vec{m}_i \vdash t_i}{\Gamma \mid \Delta, \vec{p} \vdash \mathbf{op}(\vec{p}, (\vec{m}_i.\ t_i))}$$

We are also assuming with (IH) that $\Gamma \times \Gamma' \mid \Delta, \vec{m}_i, \Delta' \vdash t_i; u$ is such that the following commutes:

$$\begin{tikzcd}
\Gamma \odot \Gamma' \arrow[r, "t_i;u"] \arrow[d, "\Gamma\odot u"'] & \text{よ}(\Delta + m_i) \odot \text{よ}(\Delta') \\
\Gamma \odot \text{よ}(\Delta') \arrow[ur, "t_i\odot\text{よ}(\Delta')"'] &
\end{tikzcd}$$

where the $[\![\cdot]\!]$ has been omitted to avoid cluttered notations.

We wish to show that the following commutes:

$$\begin{tikzcd}
\Gamma \odot \Gamma' \arrow[r, "\Gamma\odot u"] \arrow[d, "\Gamma\odot u"'] & \Gamma \odot \text{よ}(\Delta') \arrow[r, "\langle t_i\rangle_i\odot\text{よ}(\Delta')"] \arrow[dl, dash, "="'] \arrow[dr, phantom, "(\mathrm{I})"] & \prod_i \text{よ}(\Delta + m_i) \odot \text{よ}(\Delta') \arrow[d, "\text{よ}(\Delta)\odot\mathbf{op}\odot\text{よ}(\Delta')"] \\
\Gamma \odot \text{よ}(\Delta') \arrow[rr, "t\odot\text{よ}(\Delta')"'] & & \text{よ}(\Delta + p + \Delta')
\end{tikzcd}$$

where the upper path corresponds to the typing of $t; u = \mathbf{op}(\vec{p}, (\vec{m}_i.t_i; u)_i)$, and the lower path is the target. This does indeed commute because (I) commutes by left functoriality of $\odot$ and by definition of the morphisms in the syntactic category.

Case (var) on t, (op) on u:

We know that

$$\frac{}{(x_i : m_i)_i \mid a_1...a_{m_i} \vdash x(a_1...a_{m_i})} \qquad \frac{\Gamma' \mid \Delta', \vec{n}_j \vdash u_j}{\Gamma' \mid \Delta', \vec{q} \vdash \mathbf{op}(\vec{q}, (\vec{n}_j.\ u_j))}$$

And by (IH) we know that $(x_i : m_i)_i \times \Gamma' \mid \vec{a}, \Delta', \vec{n}_j \vdash x_i(\vec{a}); u_j$ the following commutes for all j:

$$\begin{tikzcd}
\prod_i \text{よ}(m_i) \odot \Gamma' \arrow[r, "x(\vec{a});u_j"] \arrow[d, "\prod_i \text{よ}(m_i)\odot u_j"'] & \text{よ}(m_i) \odot \text{よ}(\Delta') \odot \text{よ}(n_j) \\
\prod_i \text{よ}(m_i) \odot \text{よ}(\Delta') \odot \text{よ}(n_j) \arrow[ur, "\pi_i \odot \text{よ}(\Delta') \odot \text{よ}(n_j)"'] &
\end{tikzcd}$$

We wish to prove that

$$\begin{tikzcd}
\prod_i \text{よ}(m_i) \odot \Gamma' \arrow[r, "\prod_i \text{よ}(m_i)\odot\langle u_j\rangle_j"] \arrow[dd, "\prod_i \text{よ}(m_i)\odot u"'] & \prod_i \text{よ}(m_i) \odot \prod_j[\text{よ}(\Delta') \odot \text{よ}(n_j)] \arrow[d, "\pi_i\odot\prod_j[\text{よ}(\Delta')\odot\text{よ}(n_j)]"] \arrow[ddl, "\prod_i \text{よ}(m_i)\odot\mathbf{op}"'] \\
\arrow[ur, phantom, "\text{(I)}"] & \text{よ}(m_i) \odot \prod_j[\text{よ}(\Delta') \odot \text{よ}(n_j)] \arrow[d, "\text{よ}(m_i)\odot\mathbf{op}"] \arrow[dl, phantom, "\text{(II)}"] \\
\prod_i \text{よ}(m_i) \odot \text{よ}(\Delta') \odot \text{よ}(q) \arrow[r, "\pi_i\odot\text{よ}(\Delta')\odot q"'] & \text{よ}(m_i) \odot \text{よ}(\Delta') \odot \text{よ}(q)
\end{tikzcd}$$

where the upper path corresponds to the interpretation of the typing of $t; u = \mathbf{op}(\vec{q}, (\vec{n}_j.x(a_1...a_{m_i}); u_j)_j)$ and the bottom path is the target. Here, (I) simply follows from the definition of the morphisms in the syntactic category and the right functoriality of $\odot$, and (II) follows from the naturality of $\pi_i \bullet -$ (because $\pi_i \in \mathbb{F}^{op}(\prod_i \text{よ}(m_i), m_i)$), or alternatively from lemma 3.2.17.

Case (var) on t, (var) on u:

We need only show:

$$\begin{tikzcd}
\prod_i \text{よ}(m_i) \odot \prod_j \text{よ}(n_j) \arrow[d, "\prod_i \text{よ}(m_i)\odot\pi_{j^*}"'] \arrow[dr, "xy(\vec{m}_{i^*}, \vec{n}_{j^*})"] & \\
\prod_i \text{よ}(m_i) \odot \text{よ}(n_{j^*}) \arrow[r, "\pi_{i^*}\odot\text{よ}(n_{j^*})"'] & \text{よ}(m_{i^*}) \odot \text{よ}(n_{j^*})
\end{tikzcd}$$

which is follows from the definition.

Case (struct) on t: We assume that $\Gamma' \mid \Delta' \vdash u$ and

$$\frac{\Gamma \mid \Delta \vdash t}{\Gamma \mid \Delta_\iota \vdash t_\iota}$$

By (IH) we assume:

$$\begin{tikzcd}
\Gamma \odot \Gamma' \arrow[r, "\Gamma\odot u"] \arrow[dr, "t;u"'] & \Gamma \odot \text{よ}(\Delta') \arrow[d, "t\odot \text{よ}(\Delta')"] \\
 & \text{よ}(\Delta) \odot \text{よ}(\Delta')
\end{tikzcd}$$

We wish to prove that

$$\begin{tikzcd}
\Gamma \odot \Gamma' \arrow[r, "\Gamma\odot u"] \arrow[d, "\Gamma\odot u"'] & \Gamma \odot \text{よ}(\Delta') \arrow[r, "t\odot \text{よ}(\Delta')"] \arrow[dl, dash, "="] \arrow[dr, phantom, "\text{(I)}"] & \text{よ}(\Delta) \odot \text{よ}(\Delta') \arrow[d, "\iota\odot \text{よ}(\Delta')"] \\
\Gamma \odot \text{よ}(\Delta') \arrow[rr, "t_\iota\odot \text{よ}(\Delta')"'] & & \text{よ}(\Delta_\iota) \odot \text{よ}(\Delta')
\end{tikzcd}$$

where again, the upper path is the interpretation of $t_\iota; u$ and the lower path is the target. Here, (I) commutes because of the definition of the morphisms in the syntactic category and of the left functoriality of $\odot$.

Case (struct) on u: We assume that $\Gamma \mid \Delta \vdash t$ and

$$\frac{\Gamma' \mid \Delta' \vdash u}{\Gamma' \mid \Delta'_\iota \vdash u_\iota}$$

By (IH) we assume:

$$\begin{tikzcd}
\Gamma \odot \Gamma' \arrow[r, "\Gamma\odot u"] \arrow[dr, "t;u"'] & \Gamma \odot \text{よ}(\Delta') \arrow[d, "t\odot \text{よ}(\Delta')"] \\
 & \text{よ}(\Delta) \odot \text{よ}(\Delta')
\end{tikzcd}$$

We wish to show that

$$\begin{tikzcd}
\Gamma \odot \Gamma' \arrow[r, "\Gamma\odot u"] \arrow[d, "\Gamma\odot u_\iota"'] \arrow[dr, phantom, "\text{(I)}"] & \Gamma \odot \text{よ}(\Delta') \arrow[r, "t\odot \text{よ}(\Delta')"] \arrow[dl, "\Gamma\odot\iota"] \arrow[dr, phantom, "\text{(II)}"] & \text{よ}(\Delta) \odot \text{よ}(\Delta') \arrow[d, "\text{よ}(\Delta)\odot\iota"] \\
\Gamma \odot \text{よ}(\Delta'_\iota) \arrow[rr, "t\odot \text{よ}(\Delta'_\iota)"'] & & \text{よ}(\Delta) \odot \text{よ}(\Delta'_\iota)
\end{tikzcd}$$

This does indeed commute because (I) commutes by definition of the morphisms in the syntactic category and the right functoriality of $\odot$; (II) commutes because of lemma 3.2.17 and by taking the square through the symmetry of the premonoidal product (proposition 3.3.1). □

This explicit translation of sequencing directly shows that syntactic commutation is equivalent to a bifunctoriality square of $\odot$.

Corollary 3.3.9 (Commutation). *$\Gamma \times \Gamma' \mid \Delta, \Delta' \vdash t \rtimes u = u \ltimes t$ in P if and only if the following commutes in $\mathbb{L}_P$.*

$$\begin{tikzcd}
\Gamma \odot \Gamma' \arrow[r, "t\odot\Gamma'"] \arrow[d, "\Gamma\odot u"'] & p \odot \Gamma' \arrow[d, "p\odot u"] \\
\Gamma \odot q \arrow[r, "t\odot\Gamma'"'] & p \odot q
\end{tikzcd}$$

Proof. Follows directly from theorem 3.3.8. □

3.3.3 Sums

Using the results from the previous sections, we are now ready to define the sum and tensor of presentations.

Definition 3.3.10 (Sum of presentations). *Let P and P' be two PATs. The sum $P + P'$ is the PAT with operations $\mathcal{O} \sqcup \mathcal{O}'$, and containing all axioms from P and P', with no additional axiom.*

Theorem 3.3.11 (Sum of presented theories is sum of presentations). *$\mathbb{L}_{P+P'}$ is the coproduct of $\mathbb{L}_P$ and $\mathbb{L}_{P'}$.*

Proof. First, there are canonical functors **inl** : $\mathbb{L}_P \to \mathbb{L}_{P+P'}$ and **inr** : $\mathbb{L}_{P'} \to \mathbb{L}_{P+P'}$ directly embedding terms in P and P' to terms in $P + P'$ respectively.

We need only show the universal property:

$$\begin{tikzcd}
\mathbb{L}(P) \arrow[r, "\mathbf{inl}"] \arrow[dr, "F"'] & \mathbb{L}(P+P') \arrow[d, dashed, "H"] & \mathbb{L}(P') \arrow[l, "\mathbf{inr}"'] \arrow[dl, "G"] \\
& \mathbb{L}' &
\end{tikzcd}$$

Assuming we have functors F and G into $\mathbb{L}'$ as shown above, we can simply construct H as identity on objects, product preserving and action preserving, and the following map on morphisms:

$$H(\langle t_j \rangle_j : \prod_i \text{よ}(m_i) \to \prod_j \text{よ}(n_j)) = \langle H(t_j) \rangle_j$$

and

$$H(\Gamma \mid \Delta \vdash t) = \begin{cases} (\text{よ}(\Delta) \odot F(\mathbf{op})) \circ \langle H(t_k) \rangle_k & \text{if } t = \mathbf{op}(\vec{p}, (\vec{m}_k.t_k)_k) \\ (\text{よ}(\Delta) \odot G(\mathbf{op}')) \circ \langle H(t_\ell) \rangle_\ell & \text{if } t = \mathbf{op}'(\vec{q}, (\vec{n}_\ell.t_\ell)_\ell) \\ K'(\iota) \circ H(t') & \text{if } t = t'_\iota \text{ under (struct)} \\ \pi_k & \text{if } t = (x_k : m_k)_k \mid \vec{a} \vdash x_k(\vec{a}) \end{cases}$$

where $K' : \mathbb{F}^{op} \to \mathbb{L}'$ is the functor associated with $\mathbb{L}'$, and **op** and **op**$'$ stand for operations from P and P', respectively.

Uniqueness is easy to check: Let $H' : \mathbb{L}(P + P') \to \mathbb{L}'$, then H' is identity-on-objects and preserves products (follows from the definition). Thus it suffices to show that H and H' coincide on individual terms $\Gamma \mid \Delta \vdash t$, which we can easily do by rule induction:

Case (op) with **op** from P:

$$\begin{aligned}
& H'(\Gamma \mid \Delta, \vec{p} \vdash \mathbf{op}(\vec{p}, (\vec{m}_k.\ t_k)_k)) \\
&= H'((\text{よ}(\Delta) \odot \mathbf{op}) \circ \langle \Gamma \mid \Delta, \vec{m}_k \vdash t_k \rangle_k) && \text{(def of morphisms in definition 3.2.7)} \\
&= H'((\text{よ}(\Delta) \odot \mathbf{op})) \circ H'(\langle \Gamma \mid \Delta, \vec{m}_k \vdash t_k \rangle_k) && \text{(Functoriality of } H'\text{)} \\
&= H'((\text{よ}(\Delta) \odot \mathbf{op})) \circ \langle H'(\Gamma \mid \Delta, \vec{m}_k \vdash t_k) \rangle_k && \text{(Product preservation)} \\
&= \ \text{よ}(\Delta) \odot H'(\mathbf{op}) \circ \langle H'(\Gamma \mid \Delta, \vec{m}_k \vdash t_k) \rangle_k && \text{(lemma 3.3.7)} \\
&= \ \text{よ}(\Delta) \odot F(\mathbf{op}) \circ \langle H'(\Gamma \mid \Delta, \vec{m}_k \vdash t_k) \rangle_k && \text{(Coproduct property)} \\
&= H(\Gamma \mid \Delta, \vec{p} \vdash \mathbf{op}(\vec{p}, (\vec{m}_k.\ t_k)_k)) && \text{(By (IH) and def of } H\text{)}
\end{aligned}$$

Case (op) with $\mathbf{op}'$ from P': analogous.

Case (var): structural morphisms are obviously preserved because the functors H and H' both have to commute with $K_{\mathbb{L}_{P+P'}}$ and K'.

Case (struct):

$$\begin{aligned}
H'(\Gamma \mid \Delta_\iota \vdash t_\iota) &= H'(K_{\mathbb{L}_{P+P'}}(\iota) \circ (\Gamma \mid \Delta \vdash t)) && \text{(def of morphisms in definition 3.2.7)} \\
&= H'(K_{\mathbb{L}_{P+P'}}(\iota)) \circ H'(t) && \text{(Functoriality)} \\
&= K'(\iota) \circ H'(t) && \text{(Because } H' \text{ commutes with the } K\text{'s)} \\
&= H(t_\iota) && \text{(IH)}
\end{aligned}$$

□

3.3.4 Tensors

Definition 3.3.12 (Tensor of presentations)**.** *Let P and P' be two PATs. The sum $P \otimes P'$ is the PAT with operations $\mathcal{O} \sqcup \mathcal{O}'$, and containing all axioms from P and P', and with additional axioms making all operation in $\mathcal{O}$ commute with those in $\mathcal{O}'$, i.e., for any* $\mathbf{op} : (p \mid m_1 ... m_k)$ *and* $\mathbf{op} : (q \mid n_1 ... n_\ell)$ *we have the following equation:*

$$\begin{aligned}
& (x_{i,j} : m_i + n_j)_{i,j}\ \vec{a}, \vec{b} \vdash \\
& \qquad \mathbf{op}(\vec{a}, (\vec{m}_i.\ \mathbf{op}'(\vec{b}, (\vec{n}_j.\ x_{i,j}(\vec{m}_i, \vec{n}_j))_j))_i) = \mathbf{op}'(\vec{b}, (\vec{n}_j.\ \mathbf{op}(\vec{a}, (\vec{m}_i.\ x_{i,j}(\vec{m}_i, \vec{n}_j))_i))_j)
\end{aligned}$$

Lemma 3.3.13. *Let t be a term in P, u be in P', then in $P \otimes P'$ we have that $t; u = u; t$.*

Proof. Firstly, by rule induction, we can establish that for any term t of P and any $\mathbf{op}$ of P' we have that $t; \mathbf{op} = \mathbf{op}; t$, and likewise that for any u of P' and any $\mathbf{op}$ of P we have that $u; \mathbf{op} = \mathbf{op}; u$.

Then, the lemma can be shown by rule induction on both t and u, using these facts as well as lemma 3.2.17. □

Definition 3.3.14. *Let $\mathbb{L}$ and $\mathbb{L}'$ be enriched Lawvere theories in the sense of definition 3.2.2. Their tensor, $\mathbb{L} \otimes \mathbb{L}'$, if it exists, is the Lawvere theory satisfying the universal property that for any $f : X \to Y$ in $\mathbb{L}$ and $g : A \to B$ in $\mathbb{L}'$, the following diagram commutes in $\mathbb{L}'$:*

$$\begin{tikzcd}
X \odot A \arrow[r, "X \odot g"] \arrow[d, "f \odot A"'] & X \odot B \arrow[d, "f \odot B"] \\
Y \odot A \arrow[r, "Y \odot g"'] & Y \odot B
\end{tikzcd}$$

Theorem 3.3.15 (Tensor of presented theories is the tensor of presentations)**.** $\mathbb{L}_{P \otimes P'}$ *is* $\mathbb{L}_P \otimes \mathbb{L}'_P$.

Proof. The proof of universal property is then summarised by the following diagram, which we shall argue commutes.

$$\begin{tikzcd}
 & \mathbb{L}_{P+P'} \arrow[d, dashed, tail, "I"] \arrow[dd, dashed, bend right, "{[F,G]}"'] & \\
\mathbb{L}_P \arrow[ur, "\mathbf{inl}"] \arrow[r, "\ell" description] \arrow[dr, "F"'] & \mathbb{L}_{P \otimes P'} \arrow[d, dashed, red, "H"] & \mathbb{L}_{P'} \arrow[ul, "\mathbf{inr}"'] \arrow[l, "r" description] \arrow[dl, "G"] \\
 & M &
\end{tikzcd}$$

Firstly, it is easy to see by lemma 3.3.13 and corollary 3.3.9 that $\mathbb{L}_{P \otimes P}$ satisfies the property given in definition 3.3.14.

Then, we can define ℓ and r as the canonical embedding functors from $\mathbb{L}_P$ and $\mathbb{L}_{P'}$ to $\mathbb{L}_{P \otimes P'}$.

Now assume we have a diagram $F : \mathbb{L}_P \to M$ and $G : \mathbb{L}_{P'} \to M$ such that the diagram commutes. Then, there is a functor $H : \mathbb{L}_{P \otimes P'} \to M$ such that $H \circ \ell = F$ and $H \circ r = G$, defined exactly analogously to that of $[F, G]$. Therefore, by definition, H satisfies that $H \circ I = [F, G]$, where I is the canonical identity on object and full functor in $\mathbb{L}_{P+P'} \to \mathbb{L}_{P \otimes P'}$.

Let $H' : \mathbb{L}_{P \otimes P'} \to M$ such that $H' \circ \ell = F$ and $H' \circ r = G$, it follows that $H \circ I = H' \circ I$ by universal property of $\mathbb{L}_{P+P'}$, and thus $H = H'$ because I is epic. □

4

An Algebraic Theory for Quantum Communication

This chapter concerns the main object of study of the dissertation: an algebraic theory for quantum communication. By quantum communication, we really mean quantum I/O, i.e. outputting and inputting qubits to and from an unknown environment. In this chapter, we study such programs 'freely', in the sense that we start with a complete axiomatisation of qubit quantum computing (the $\mathsf{QUANTUM}$ theory from [67]), and simply adjoin two operations for sending and receiving qubits respectively (the $\mathsf{I/O}$ theory). Concretely, we choose to work with $\mathsf{I/O} \otimes \mathsf{QUANTUM}$, a tensor of the two theories in the sense of chapter 3, and argue for its merits (section 4.1). Then, in section 4.2, we give a non-trivial operational semantics for quantum communication. We then use it in section 4.3 to define a suitable notion of program equivalence and show that it is a model of $\mathsf{I/O} \otimes \mathsf{QUANTUM}$ (theorem 4.3.25). This allows us to demonstrate that not only does our theory not collapse into triviality (unlike the Eckmann-Hilton result cf. proposition 4.2.1), it actually soundly axiomatises a reasonable model of quantum communication.

4.1 The Theory

4.1.1 Starting point: algebraic theory for qubit quantum computing

Staton [67] presented in his work an algebraic theory for qubit quantum computing with programming constructs stemming from the quantum circuit model. Note that the parameters here are treated *linearly*, conformably to the no-cloning theorem of quantum theory, which means that we are in the Bij system as per the notation introduced in chapter 3.

$$\mathsf{apply}_X(a, a.\mathsf{measure}(a,x,y)) = \mathsf{measure}(a,y,x) \tag{A}$$

$$\begin{aligned}&\mathsf{measure}(a, \mathsf{apply}_U(\vec{b}, \vec{b}.x(\vec{b}), \mathsf{apply}_V(\vec{b}, \vec{b}.y(\vec{b})))) \\ &= \mathsf{apply}_{D(U,V)}((a,\vec{b}), (a,\vec{b}).\mathsf{measure}(a, x(\vec{b}), y(\vec{b})))\end{aligned} \tag{B}$$

$$\mathsf{apply}_U(\vec{a}, \vec{a}.\mathsf{discard}_n(\vec{a}, x)) = \mathsf{discard}_n(\vec{a}, x) \tag{C}$$

$$\mathsf{new}(a.\mathsf{measure}(a,x,y)) = x \tag{D}$$

$$\mathsf{new}(a.\mathsf{apply}_{D(U,V)}((a,\vec{b}), (a,\vec{b}).x(a,\vec{b}))) = \mathsf{apply}_U(\vec{b}, \vec{b}.\mathsf{new}(a.x(a,\vec{b}))) \tag{E}$$

$$\mathsf{apply}_{\mathbf{swap}}((a,b), (a,b).x(a,b)) = x(b,a) \tag{F}$$

$$\mathsf{apply}_I(\vec{a}, \vec{a}.x(\vec{a})) = x(\vec{a}) \tag{G}$$

$$\mathsf{apply}_{VU}(\vec{a}, \vec{a}.x(\vec{a})) = \mathsf{apply}_U(\vec{a}, \vec{a}.\mathsf{apply}_V(\vec{a}, \vec{a}.x(\vec{a}))) \tag{H}$$

$$\mathsf{apply}_{U\otimes V}((\vec{a},\vec{b}), (\vec{a},\vec{b}).x(\vec{a},\vec{b})) = \mathsf{apply}_U(\vec{a}, \vec{a}.\mathsf{apply}_V(\vec{b}, \vec{b}.x(\vec{a},\vec{b}))) \tag{I}$$

$$\begin{aligned}&\mathsf{measure}(a, \mathsf{measure}(b,u,v), \mathsf{measure}(b,x,y)) \\ &= \mathsf{measure}(b, \mathsf{measure}(a,u,x), \mathsf{measure}(a,v,y))\end{aligned} \tag{J}$$

$$\mathsf{new}(a.\mathsf{new}(b.x(a,b))) = \mathsf{new}(b.\mathsf{new}(a.x(a,b))) \tag{K}$$

$$\mathsf{new}(a.\mathsf{measure}(b, x(a), y(a))) = \mathsf{measure}(b, \mathsf{new}(a.x(a)), \mathsf{new}(a, y(a))) \tag{L}$$

Figure 4.1: Algebraic Theory for Qubit Quantum Computing in [67]

Definition 4.1.1 (Algebraic Theory for Quantum Computing, Section 3 [67]). QUANTUM *is a* Bij *PAT. The signature is given by operations* new : $(0 \mid 1)$*,* measure : $(1 \mid 0, 0)$ *and for every* $2^n \times 2^n$ *unitary* U*, an operation* $\mathsf{apply}_U : (n \mid n)$*. Equations are given in fig. 4.1, where* X *is the* X *gate,* $D(U,V)$ *is the matrix* $\left(\begin{smallmatrix} U & 0 \\ 0 & V \end{smallmatrix}\right)$*, and* $\mathsf{discard}_n$ *is defined as follows for* $n \in \mathbb{N}$*:*

$$\begin{aligned}\mathsf{discard}_0(-, t) &= t \\ \mathsf{discard}_{n+1}((a,\vec{b}), t) &= \mathsf{measure}(a, \mathsf{discard}_n(\vec{b}, t), \mathsf{discard}_n(\vec{b}, t))\end{aligned}$$

This theory admits the complex numbers $\mathbb{C}$ as a fully complete model in **CPU** in the following sense:

Definition 4.1.2 (Def 7, [67]). *A model* X *in a category is* fully complete *if for all contexts* $((x_i : m_i)_i \mid a_1 ... a_p)$*,*

- *for every morphism* $f : \prod_i (m_i \bullet X) \to p \bullet X$ *there is a term* $(x_i : m_i)_i \mid a_1 ... a_p \vdash t$ *such that* $[\![t]\!] = f$*;*
- *for all terms* $(x_i : m_i)_i \mid a_1 ... a_p \vdash t, u$ *if* $[\![t]\!] = [\![u]\!]$ *then* $t = u$ *is derivable.*

Theorem 4.1.3 (Thm 11, [67]). *The complex numbers* $\mathbb{C} \in$ **CPU** *form a fully complete model, where the* **Bij** *action is given by* $m \bullet X \triangleq M_{2^m}(X)$*.*

4.1.2 The theory to be studied: $\mathsf{I/O} \otimes \mathsf{QUANTUM}$

We now proceed to define the actual theory to be studied.

We model communication using operations $\mathsf{in} : (0 \mid 1)$ and $\mathsf{out} : (1 \mid 0)$, where one is able to send and receive qubits. This can be modelled as the theory with these two operations and no equations, as per the following definition.

Definition 4.1.4 (Algebraic theory for communications (e.g. [55] example 2.2))**.** $\mathsf{I/O}$ *is the theory with operations* in *and* out *and no equations.*

The question we wish to ask then becomes: how should we combine $\mathsf{I/O}$ and $\mathsf{QUANTUM}$? The section title spoils the answer: we used the tensor (definition 3.3.12, definition 3.3.14). Let us try to understand this decision in more detail.

The most flexible construction is the sum (definition 3.3.10): combining the two theories together without adding any equations. In fact, this is the standard combination suggested by Hyland, Plotkin and Power [31]. However, they do so in the context of classical, interactive I/O – like user inputs in a classical program. Unfortunately, the sum is insufficient for modelling quantum phenomena[1].

Consider the following equation:

$$\begin{aligned}
&\mathsf{new}(q.\mathsf{apply}_H(q, q.\mathsf{new}(p.\mathsf{apply}_{CX}((q,p),(q,p).\mathsf{measure}(q,\mathsf{out}(p,x),\mathsf{out}(p,x))))))\\
&\quad = \mathsf{new}(q.\mathsf{apply}_H(q, q.\mathsf{measure}(q, \mathsf{new}(p.\mathsf{out}(p,x)), \mathsf{new}(p.\mathsf{apply}_X(p, p.\mathsf{out}(p,x)))))))
\end{aligned} \tag{4.1}$$

where H is the Hadamard gate, X is the X gate, and CX is the controlled X gate with $CX = D(\mathbf{id}, X)$, more widely known as the CNOT gate.

This equation is a slight twist to a well-known fact in quantum computing: preparing an entangled pair of qubits, sending one out and then discarding one of them should be equivalent to performing a fair coin toss and then sending the result out as a qubit. This is counterintuitive – the measurement from the discarding operation has been 'pulled back in time'– and makes the axiomatisation of any notion of communication subtle.

For this equality to hold, we need the following axiom, which only holds in $\mathsf{I/O} \otimes \mathsf{QUANTUM}$ and not $\mathsf{I/O} + \mathsf{QUANTUM}$.

$$\mathsf{measure}(q, \mathsf{out}(p,x), \mathsf{out}(p,y)) = \mathsf{out}(p, \mathsf{measure}(q,x,y)) \tag{4.2}$$

[1] or for the category theorist, it is insufficient to model the structure of a symmetric compact closed category

$$\mathsf{new}(a.\mathsf{in}(b.x(a,b))) = \mathsf{in}(b.\mathsf{new}(a.x(a,b))) \tag{U}$$

$$\mathsf{apply}_U(\vec{a}, \mathsf{in}(b.x(\vec{a},b))) = \mathsf{in}(b.\mathsf{apply}_U(\vec{a}), x(\vec{a},b)) \tag{V}$$

$$\mathsf{measure}(a, \mathsf{in}(b.x(b)), \mathsf{in}(b.y(b))) = \mathsf{in}(b.\mathsf{measure}(a, x(b), y(b))) \tag{W}$$

$$\mathsf{new}(a.\mathsf{out}(b, x(a))) = \mathsf{out}(b, \mathsf{new}(a.x(a))) \tag{X}$$

$$\mathsf{apply}_U(\vec{a}, \mathsf{out}(b, x(\vec{a}))) = \mathsf{out}(b, \mathsf{apply}_U(\vec{a}), x(\vec{a})) \tag{Y}$$

$$\mathsf{measure}(a, \mathsf{out}(b,x), \mathsf{out}(b,y)) = \mathsf{out}(b, \mathsf{measure}(a,x,y)) \tag{Z}$$

Figure 4.2: Commutativity equations in $\mathsf{I/O} \otimes \mathsf{QUANTUM}$

The proof for eq. (4.1) would then go as follows:

$$\begin{aligned}
&\mathsf{measure}(q, \mathsf{new}(p.\mathsf{out}(p,x)), \mathsf{new}(p.\mathsf{apply}_X(p, p.\mathsf{out}(p,x))))\\
&= \mathsf{new}(p.\mathsf{measure}(q, \mathsf{apply}_{\mathbf{id}}(p, p.\mathsf{out}(p,x)), \mathsf{new}(p.\mathsf{apply}_X(p, p.\mathsf{out}(p,x)))))\\
&= \mathsf{new}(p.\mathsf{apply}_{CX}(p, p.\mathsf{measure}(q, \mathsf{out}(p,x), \mathsf{out}(p,x))))\\
&= \mathsf{new}(p.\mathsf{apply}_{CX}(p, p.\mathsf{measure}(q, \mathsf{out}(p,x), \mathsf{out}(p,x))))\\
&\overset{\text{eq. (4.2)}}{=} \mathsf{new}(p.\mathsf{apply}_{CX}(p, p.\mathsf{out}(p, \mathsf{discard}(q,x))))
\end{aligned}$$

which implies that $\mathsf{LHS} = \mathsf{RHS}$.

In fact, in general, we would want all of $\mathsf{QUANTUM}$'s operations to commute with inputs and outputs. Not only does this model quantum phenomena better, but it also makes intuitive sense: local operations should commute with unrelated I/O. Therefore, we add the equations shown in fig. 4.2, which, by definition 3.3.12, gives us exactly the theory $\mathsf{I/O} \otimes \mathsf{QUANTUM}$.

Are there any more equations to add?

Both $\mathsf{I/O}$ and $\mathsf{QUANTUM}$ are standard theories modelling communication and quantum computing respectively. However, between $\mathsf{I/O}$ and $\mathsf{QUANTUM}$ operations, we do need to ask ourselves whether we want to allow some notion of *deferred measurement* (e.g. §4.4 of [42] or [46]). Deferred measurement refers to the fact that a measurement can always be deferred in a quantum process. In particular, classically controlled processes depending on the result of a measurement can be replaced with a quantumly controlled process followed by a measurement, as embodied in axiom (B):

$$\begin{aligned}
&\mathsf{measure}(a, \mathsf{apply}_U(\vec{b}, \vec{b}.x(\vec{b}), \mathsf{apply}_V(\vec{b}, \vec{b}.y(\vec{b}))))\\
&= \mathsf{apply}_{D(U,V)}((a,\vec{b}), (a,\vec{b}).\mathsf{measure}(a, x(\vec{b}), y(\vec{b})))
\end{aligned}$$

The question we have to ask ourselves is whether there is a coherent way of deferring measurements through (possibly classically controlled) I/O operations.

Consider the term

$$\mathsf{measure}(q, \mathsf{in}(p.x(p)), y)$$

which measures a qubit q, and inputs a new qubit p only when the measurement outcome is $|0\rangle$.

Should we be able to defer this measurement?

The naive way of deferring would be allowing a rule like

$$\mathsf{measure}(q, \mathsf{in}(p.x(p)), y) = \mathsf{in}(\mathsf{measure}(q, x(p), \mathsf{discard}(p, y)))$$

which is problematic firstly because on the RHS, discarding introduces decoherence, but more importantly because it is problematic for an input to be done regardless of the measurement outcome. It then follows that

$$\begin{aligned}\mathsf{in}(p.\mathsf{measure}(q, x(p), y(p))) &= \mathsf{measure}(q, \mathsf{in}(p, x(p)), \mathsf{in}(p, y(p))) \\ &= \mathsf{in}(p, \mathsf{measure}(q, x(p), \mathsf{discard}(p, \mathsf{in}(p, y(p)))))\end{aligned}$$

which is clearly nonsense.

A better way would be to have a notion of *quantum-controlled* input $\mathsf{cin}(q, (q, p).t)$ (which, let's say, does the input if q is 0), and equation

$$\mathsf{measure}(q, \mathsf{in}(p.x(p)), y) = \mathsf{cin}(q, (q, p).\mathsf{measure}(q, x(p), \mathsf{discard}(p, y)))$$

but we make the design decision **not** to allow this. As mentioned in the related works section, quantum controlled quantum communication [35] is an active area of research, and we choose to leave its development as a programming language primitive to future work.

4.2 An Operational Semantics for $\mathsf{I/O} \otimes \mathsf{QUANTUM}$

Defining an algebraic theory is not sufficient to claim that it is well-formed. Indeed, the following famous result shows that if a theory has two unital binary operations which commute with each other, then those two operations coincide and are additionally associative and commutative. In other words, the two binary operations 'collapse' into one.

Proposition 4.2.1 (Eckmann-Hilton argument (e.g.[47]))**.** *Let $\mathbb{L}$ be the (unparameterised) algebraic theory of a unital magma, i.e. a theory with operations $\mu : 2$ and $\eta : 0$ such that the following unital law holds:*

$$\mu(\eta, x) = \mu(x, \eta) = x.$$

Then, in $\mathbb{L} \otimes \mathbb{L}$, the two magma structures collapse into a single structure satisfying the laws of a commutative monoid.

Proof sketch. Let us call the multiplication and unit of the first theory $\odot$ and $1_\odot$, and those of the second theory $\otimes$ and $1_\otimes$. We will demonstrate that the two structures $(\otimes, 1_\otimes)$ and $(\odot, 1_\odot)$ coincide and will, in fact, be commutative and associative. To do so, recall that the tensor product implies that the following interchange law holds:

$$(x \otimes y) \odot (z \otimes w) = (x \odot z) \otimes (y \odot w)$$

We first observe that the units coincide:

$$1_\odot = 1_\odot \odot 1_\odot = (1_\otimes \otimes 1_\odot) \odot (1_\odot \otimes 1_\otimes) \overset{\text{interchange}}{=} (1_\otimes \odot 1_\odot) \otimes (1_\odot \odot 1_\otimes) = 1_\otimes \otimes 1_\otimes = 1_\otimes$$

Then, we can show that the two operators coincide too:

$$x \odot y = (x \otimes 1) \odot (1 \otimes y) \overset{\text{interchange}}{=} (x \odot 1) \otimes (1 \odot y) = x \otimes y$$

and that it is commutative:

$$x \odot y = (1 \otimes x) \odot (y \otimes 1) \overset{\text{interchange}}{=}= (1 \odot y) \otimes (x \odot 1) = y \otimes x$$

Finally, associativity also holds:

$$(x \otimes y) \otimes z = (x \otimes y) \otimes (1 \otimes z) \overset{\text{interchange}}{=} (x \otimes 1) \otimes (y \otimes z) = x \otimes (y \otimes z)$$

□

To show that operations in I/O ⊗ QUANTUM do not similarly collapse, we need to exhibit a non-trivial model. In this section, we make the first steps by giving an ad hoc operational semantics to the signature of I/O ⊗ QUANTUM, which corresponds to the 'semantics we had in mind' when defining our algebraic theory of study.

4.2.1 Syntax and Type System

The syntax of our language is exactly that of parameterised algebraic theories in section 3.1.1, with operations as in definition 4.1.1. For our convenience, however, we redefine the type system.

Definition 4.2.2 (Type System). *The typing judgement $\Gamma \mid \Delta \vdash t$ is defined just as definition 3.1.4 and the typing relation is the smallest satisfying the rules in fig. 4.3.*

Lemma 4.2.3 (Admissibility of (struct) rule). *The (struct) rule of* Bij *shown below holds for the typing relation of definition 4.2.2.*

$$\frac{\Gamma \mid \Delta \vdash t}{\Gamma \mid \sigma(\Delta) \vdash t}\ \textit{(exch)}$$

$$\frac{}{\Gamma, x : p, \Gamma' \mid a_{\sigma(1)} ... a_{\sigma(p)} \vdash x(a_1 ... a_p)}\ (\text{var})$$

$$\frac{\Gamma \mid \Delta, q \vdash t}{\Gamma \mid \Delta \vdash \mathsf{in}(q.t)}\ (\text{in}) \qquad \frac{\Gamma \mid \Delta, \Delta' \vdash t}{\Gamma \mid \Delta, q, \Delta' \vdash \mathsf{out}(q, t)}\ (\text{out}) \qquad \frac{\Gamma \mid \Delta, q \vdash t}{\Gamma \mid \Delta \vdash \mathsf{new}(q.t)}\ (\text{new})$$

$$\frac{\Gamma \mid \sigma(\Delta, \vec{b}) \vdash t}{\Gamma \mid \sigma(\Delta, \vec{a}) \vdash \mathsf{apply}_U(\vec{a}, \vec{b}.t)}\ (\text{apply}) \qquad \frac{(\Gamma \mid \Delta, \Delta' \vdash t_i)_{i=0,1}}{\Gamma \mid \Delta, q, \Delta' \vdash \mathsf{measure}(q, t_0, t_1)}\ (\text{measure})$$

Figure 4.3: Alternative typing rules for $\mathsf{I/O}\otimes\mathsf{QUANTUM}$, where σ is taken as an arbitrary permutation.

Proof. By rule induction over $\Gamma \mid \Delta \vdash t$.

Case (var): trivial. Both $\Gamma \mid \Delta \vdash x(\Delta)$ and $\Gamma \mid \sigma(\Delta) \vdash x(\Delta)$ hold directly by (var).

Case (out): assuming that

$$\Gamma \mid \Delta, \Delta' \vdash t \tag{4.3}$$

$$\Gamma \mid \Delta, q, \Delta' \vdash \mathsf{out}(q, t) \tag{4.4}$$

we want to show that $\Gamma \mid \sigma(\Delta, q, \Delta') \vdash \mathsf{out}(q, t)$.

Let $\sigma(\Delta, q, \Delta') = \Omega, q, \Omega'$. Then, from eq. (4.3) and (IH) we get

$$\Gamma \mid \Omega, \Omega' \vdash t \tag{4.5}$$

which by (out) proves

$$\Gamma \mid \Omega, q, \Omega' \vdash \mathsf{out}(q, t) \tag{4.6}$$

as required.

Case (measure): analogous.

Case (in): assume

$$\Gamma \mid \Delta, q \vdash t \tag{4.7}$$

$$\Gamma \mid \Delta \vdash \mathsf{in}(q.t) \tag{4.8}$$

Then by eq. (4.7) and (IH) it follows that

$$\Gamma \mid \sigma(\Delta), q \vdash t \tag{4.9}$$

from which

$$\Gamma \mid \sigma(\Delta) \vdash \mathsf{in}(q.t) \tag{4.10}$$

follows by (in).

Case (new): analogous.

Case (apply): assume that

$$\Gamma \mid \sigma'(\Delta, b_1 ... b_n) \vdash t \tag{4.11}$$

$$\Gamma \mid \sigma'(\Delta, a_1 ... a_n) \vdash \mathsf{apply}_U(\vec{a}, \vec{b}.t) \tag{4.12}$$

where σ' is an arbitrary permutation.

By eq. (4.11) and (IH) it follows that

$$\Gamma \mid \sigma''(\Delta, b_1 ... b_n) \vdash t \tag{4.13}$$

where $\sigma'' = \sigma \circ \sigma'$. Thus it follows that

$$\Gamma \mid \sigma''(\Delta, a_1 ... a_n) \vdash \mathsf{apply}_U(\vec{a}, \vec{b}.t) \tag{4.14}$$

□

Theorem 4.2.4 (Equivalence with definition 3.1.4)**.** *The typing relation defined in definition 4.2.2 is equivalent to that of definition 3.1.4 instantiated with the signature of* I/O ⊗ QUANTUM*.*

Proof. On the one hand, every rule in definition 4.2.2 can be derived by definition 3.1.4. On the other hand, every rule in definition 3.1.4, including (struct) (cf. lemma 4.2.3), is derivable from definition 4.2.2. Thus, these two relations are the same. □

4.2.2 Operational Semantics

4.2.2.1 A quantum stream

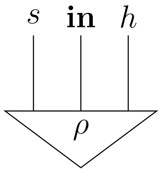
Classically, communication channels are operationally modelled by input/output streams. Then, one can imagine configurations of the machine to look like

$$\langle \Sigma, \texttt{stream\_state}, c \rangle$$

where c is the program and Σ is the store.

In this section, we wish to construct a quantum version of such a stream. The challenge is threefold:

1. The existence of quantum effects means that we can't, in general, separate the state of the store and that of the steam; we need a global state. We can then imagine a state of the form:



where s is the state accessible to the program (which we shall call 'program state'), h is the hidden stream state (the 'hidden state') and **in** : **qbit** is the next input qubit. (Note that here the string diagram would be taken in **CP**.)

2. The stream must be able to evolve with a general CPTP map every time an input or an output is performed.

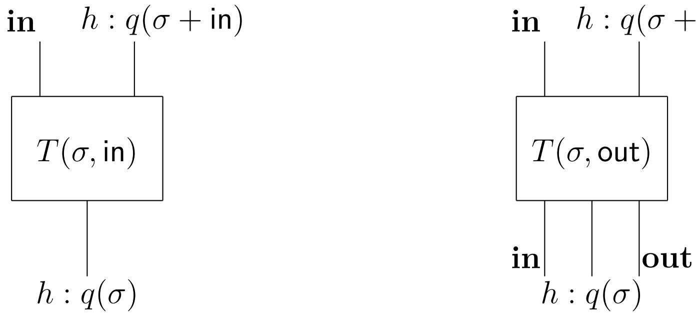
3. Apart from the possible information sharing with the program state due to entanglement, the evolution of the hidden should only depend on whether an input or an output was performed by the program. The program's branching information should not affect the stream. In other words, if, after a measurement, the program only performs an input in one branch, the state of the stream should not be affected in the other branch.

To fulfil these requirements, we define the stream as follows.

Definition 4.2.5 (Quantum stream)**.** *Let* $X_0 \in \mathbf{Ob\ CP}$. *A quantum stream in* $\mathsf{Stream}(X_0)$ *consists of*

- *A map* $q : \{\mathsf{in}, \mathsf{out}\}^* \to \mathbf{Ob\ CP}$ *such that the initial state* ϵ *satisfies* $q(\epsilon) = X_0$*;*
- *For every* $\sigma \in \{\mathsf{in}, \mathsf{out}\}^*$*, a morphism* $T(\sigma, \mathsf{in}) \in \mathbf{CPTP}(q(\sigma), \mathbf{qbit} \otimes q(\sigma + \mathsf{in}))$*, and a morphism* $T(\sigma, \mathsf{out}) \in \mathbf{CPTP}(\mathbf{qbit} \otimes q(\sigma) \otimes \mathbf{qbit}, \mathbf{qbit} \otimes q(\sigma + \mathsf{out}))$

Intuitively, the stream is a state machine, where each state is uniquely determined by its trace $\sigma \in \{\mathsf{in}, \mathsf{out}\}^*$. Every state σ has a set hidden state shape $q(\sigma) \in \mathbf{Ob}\ (\mathbf{CP})$ and two quantum maps, $T(\sigma, \mathsf{in})$ and $T(\sigma, \mathsf{out})$ shown below, performed when the program ask for an input or gives an output respectively.



in
$h : q(\sigma + \mathsf{in})$
$T(\sigma, \mathsf{in})$
$h : q(\sigma)$
in
$h : q(\sigma + \mathsf{out})$
$T(\sigma, \mathsf{out})$
in
out
$h : q(\sigma)$

Indeed, $T(\sigma, \mathsf{in})$ takes the previous hidden state and gives the new hidden state and provides a new input qubit; $T(\sigma, \mathsf{out})$ takes the previous hidden state, the previous unused input, and the new qubit output by the program to give a new hidden state and input qubit.

4.2.2.2 Transition system

We now have all the tools to define the operational semantics.

Definition 4.2.6 (Configuration). *A configuration is a tuple $\langle \rho, \sigma, c \rangle$ where*

- *ρ is a* normalised *state in* **CP** *of the form $I \to S \otimes \mathbf{qbit} \otimes H$ where $S = \mathbf{qbit}^{\otimes n} = \mathcal{M}_{2^n}(\mathbb{C})$ is the shape of the current state (which we restrict to a tensor of qubits) and $H \in \mathbf{Ob\ CP}$ is the shape of the hidden state, with no restrictions.*
- *An (implicit) list Δ_ρ of n labels (qubit names) for each qubit in the program state of ρ.*
- *$\sigma \in \{\mathsf{in}, \mathsf{out}\}^*$ is the current trace*
- *and c is a program.*

We denote the set of all configurations by **Config**.

Convention 4.2.7 (Names on string diagrams). *When drawing string diagrams, we label each wire by the corresponding qubit name. We also use the letter s for a general program state wire,* **in** *for the input wire and h for the hidden state wire.*

Convention 4.2.8. *For any state ρ we write $\tilde{\rho}$ to denote the underlying morphism in the* **CP** *category, forgetting the names.*

Definition 4.2.9 (Operational Semantics). *Let $t \in \mathsf{Stream}(X_0)$ be an arbitrary stream. The small-step operational semantics is given by a transition relation $\rightarrow^t \subseteq \mathbf{Config} \times (0, 1] \times \mathbf{Config}$ such that either ϕ does not reduce, or $\sum_{\phi \rightarrow^t_p \phi'} p = 1$.*

We define the relation as the least one satisfying the rules in $fig.$ 4.4.

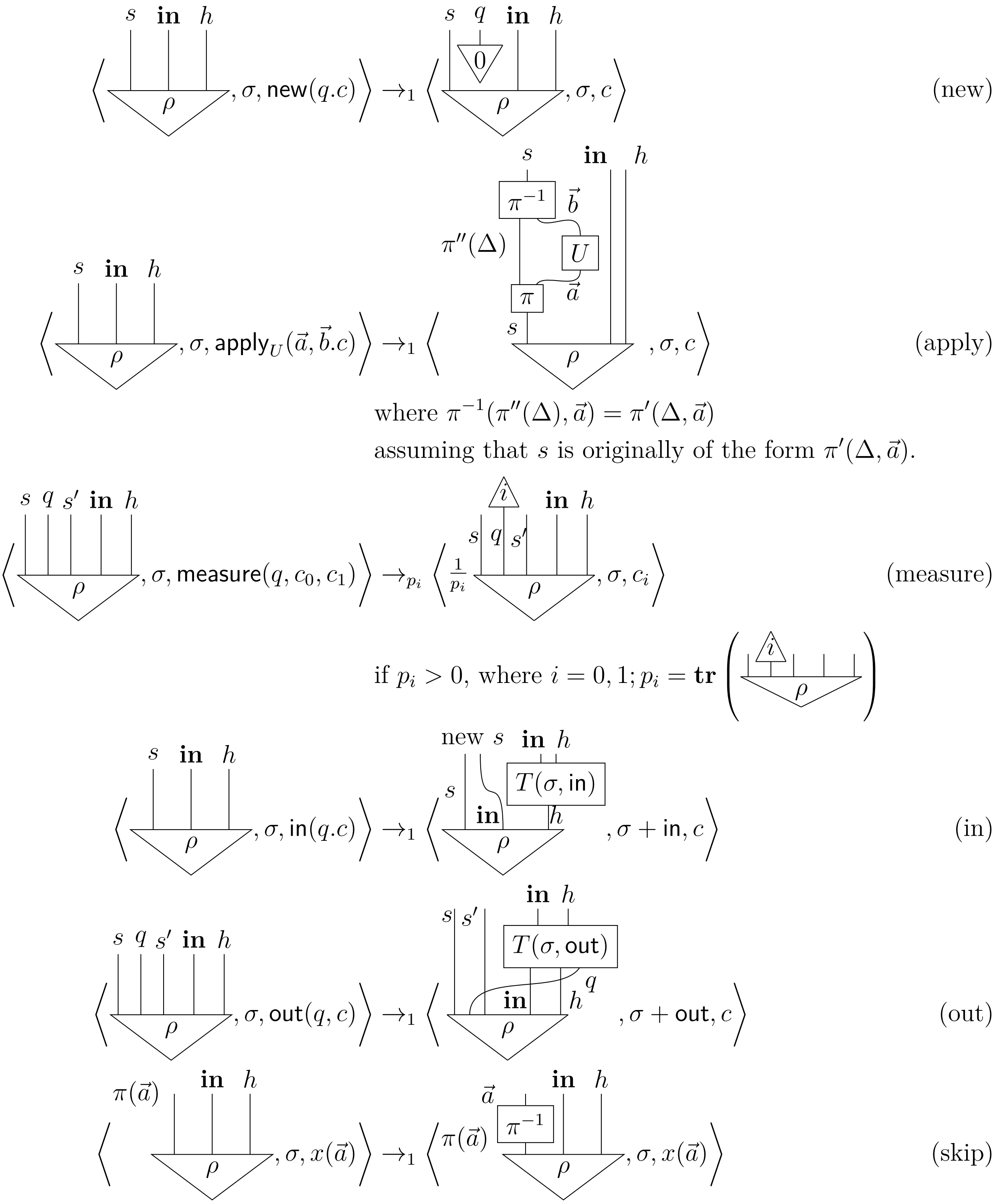
Perhaps it is worth noting that in the (apply) case of the operational semantics, we choose an arbitrary permutation π that permutes all the required qubits to the right in the right order and leaves the rest of the state in another permutation π''. It might seem like there are multiple possible choices of π, but in reality, regardless of the choice we make, the resulting state will mathematically be the same. Therefore, that transition is indeed deterministic and well-formed.

Remark 4.2.10 (Probabilistic interpretation). *Equivalently we can define transition as a probability distribution $D_{\rightarrow^t}$: **Config** $\rightharpoonup \mathcal{D}(\mathbf{Config})$ (where $\mathcal{D}(\mathbf{Config})$ is the set of discrete probability distributions over* **Config***). As a notational convenience, we write*

- *$\phi \not\rightarrow$ meaning that ϕ does not reduce, i.e. $D_{\rightarrow^t}(\phi) \uparrow$*
- *$\Pr(\phi \rightarrow^t \phi')$ denotes $\Pr_{\Phi' \sim D_{\rightarrow^t}(\phi)}(\Phi' = \phi')$, defined when $D_{\rightarrow^t}(\phi) \downarrow$.*

Convention 4.2.11 (skip). *We write $\langle \rho, \sigma, \mathsf{skip}_x \rangle$ to mean $\langle \rho, \sigma, x(\Delta_\rho) \rangle$.*

Proposition 4.2.12 (Well-definedness of the operational semantics). *Let $\phi \in$ **Config** and $t \in \mathsf{Stream}(X_0)$ for some $X_0 \in \mathbf{Ob\ CP}$. Then, either $\phi \not\rightarrow^t$ or $\sum_{\phi \rightarrow^t_p \phi'} p = 1$. Furthermore, if $\phi \rightarrow^t_p \phi' = \langle \rho', \sigma', c' \rangle$, then $\mathbf{tr}(\rho') = 1$.*

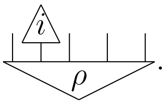


where $\pi^{-1}(\pi''(\Delta), \vec{a}) = \pi'(\Delta, \vec{a})$
assuming that s is originally of the form $\pi'(\Delta, \vec{a})$.

if $p_i > 0$, where $i = 0, 1; p_i = \mathbf{tr}$

Figure 4.4: Operational semantics for quantum communication

Proof. For convenience, we write

$$\phi_i =$$ 

For the first statement, we need only check the (measure) and (apply) cases. The (measure) case follows because

$$p_0 + p_1 = \sum_{i=0,1} \mathbf{tr}\,(\phi_i) = 1$$

by definition of traces. The (apply) case follows because the permutation π is taken as the unique one moving the qubits in $\vec{a}$ to the right and reorders them, leaving the qubits in Δ in the same order as in s.

For the second statement, at every rule but (measure), the map applied to the state is a morphism in **CPTP**. in the (measure) case, assuming the trace on the left was 1, the trace of ϕ_i would indeed be $p_i = \mathbf{tr}(\phi_i)$, and thus $\frac{1}{p_i}\phi_i$ has trace 1. □

4.2.3 Properties

Definition 4.2.13 (State shape). *Let $\rho : I \to S \otimes \mathbf{qbit} \otimes H$ be a normalised state. We write $\rho :_t \Delta, \sigma$ where $t = (q, T) \in \mathsf{Stream}(X_0)$ is a stream, Δ is a parameter context and $\sigma \in \{\mathsf{in}, \mathsf{out}\}^*$ is a trace, if*

- $H = q(\sigma)$
- $|\Delta| = S$ *and* $\Delta = \Delta_\rho$.

Let $\Gamma \mid \Delta \vdash c$, $X_0 \in \mathbf{Ob\ CP}$, $t = (q, T) \in \mathsf{Stream}(X_0)$, $\sigma \in \{\mathsf{in}, \mathsf{out}\}^*$, $\rho :_t (\Delta, \sigma)$. Let $\phi = \langle \rho, \sigma, c \rangle$.

Theorem 4.2.14 (Progress). *Either $\phi \nrightarrow$ or $\phi \to \phi'$ for some $\phi' \in$ **Config**.*

Proof. By straightforward rule induction on the typing derivation of c, simply because in every case of the operational semantics, the state ρ on the left-hand side can be set to satisfy $\rho :_t (\Delta, \sigma)$. □

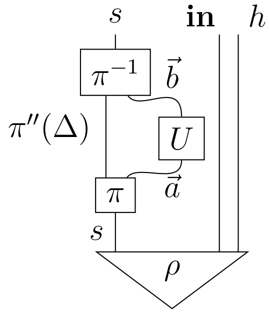
Theorem 4.2.15 (Type Preservation). *If $\phi \to^t_p \phi' = \langle \rho', \sigma', c' \rangle$ then $\Gamma \mid \Delta' \vdash c'$ and $\rho' :_t (\Delta', \sigma')$.*

Proof. By induction over $\to^t$.

Case (skip): Firstly, by inversion, we get that $\Gamma, x : n, \Gamma' \mid \pi(\vec{a}) \vdash x(\vec{a})$ which implies (by e.g. lemma 4.2.3) that $\Gamma, x : n, \Gamma' \mid \vec{a} \vdash x(\vec{a})$. Moreover, $\rho' :_t (\vec{a}, \sigma)$ holds very clearly.

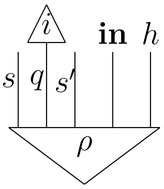
Case (new): We know that $\Gamma \mid \Delta \vdash \mathsf{new}(q.c)$. By inversion we get $\Gamma \mid \Delta, q \vdash c$. And indeed, we have that $\rho' :_t ((\Delta, q), \sigma)$ by inspection of its shape.

Case (apply): We know that $\Gamma \mid \pi'(\Delta, \vec{a}) \vdash \mathsf{apply}_U(\vec{a}, \vec{b}.c)$. By inversion we get $\Gamma \mid \pi'(\Delta, \vec{b}) \vdash c$. Moreover, by inspection of ρ' we get $\rho' :_t \pi'(\Delta, \vec{b}), \sigma$. Indeed, if in the state

$\rho' =$ 

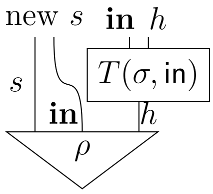
we have that s is initially of the shape $\pi'(\Delta, \vec{a})$, then the whole operation just replaces all the a's with the corresponding b's, which means that the final s is of the shape $\pi'(\Delta, \vec{b})$.

Case (measure): We know that $\Gamma \mid \Delta, q, \Delta' \vdash \mathsf{measure}(q, c_0, c_1)$. By inversion we get $\Gamma \mid \Delta, \Delta' \vdash c_i$ for $i = 0, 1$. Moreover, the final state will look like

$\rho' = \frac{1}{p_i}$ 

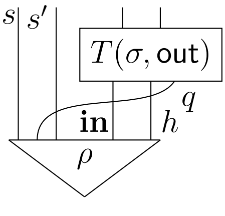
for any $i = 0, 1$. Since we know that $\rho :_t (\Delta, q, \Delta'), \sigma$, the shape of the final state will be $\rho' :_t (\Delta, \Delta'), \sigma$ necessarily.

Case (in): We know that $\Gamma \mid \Delta \vdash \mathsf{in}(q.c)$, so by inversion we get $\Gamma \mid \Delta, q \vdash c$. Moreover, given that the initial state satisfies $\rho :_t \Delta, \sigma$, and by inspection on the shape of the final state

$\rho' =$ 

we know that new s will be of the shape Δ, q. Moreover, the new hidden state along with the input will be of the shape $\mathbf{qbit} \otimes q(\sigma + \mathsf{in})$, and thus we do indeed have that $\rho' :_t (\Delta, q), (\sigma + \mathsf{in})$.

Case (out): We know that $\Gamma \mid \Delta, q, \Delta' \vdash \mathsf{out}(q, c)$, so by inversion we get that $\Gamma \mid \Delta, \Delta' \vdash c$. Moreover given that $\rho :_t \Delta, \sigma$, and that the final state looks like

$\rho' =$  ,

we know that the new program state is of the shape Δ, Δ', and moreover, by definition, we know that the new input and hiddel state will be of the shape $\mathbf{qbit} \otimes q(\sigma + \mathsf{out})$ by definition of the quantum stream. □

Now that we get type safety, let's prove termination.

Theorem 4.2.16 (Termination)**.** *For $\Gamma \mid \Delta \vdash c$, $\phi = \langle \rho, \sigma, c \rangle$ terminates (reaches some state $\phi_f \not\rightarrow$) with probability 1.*

Proof. By simple rule induction over the typing derivation. For the base case (the (var) rule), the (skip) reduction rule applies, after which the configuration does not reduce anymore. For all the other rules, assuming that the continuations terminate w.p. 1, the probability that the original term c terminates is also 1 because by progress (theorem 4.2.14), it necessarily steps to one of the continuations, along with a new state which by type preservation (theorem 4.2.15) satisfies the premise for the IH. Moreover, because the one-step transition gives a probability distribution, the resulting probability of terminating is preserved. □

With this property, for any well typed term and well formed configuration compatible with it, we can treat its end states as a probability distribution.

4.3 Program Equivalence

To exhibit a model of our $\mathsf{I/O} \otimes \mathsf{QUANTUM}$ theory, we now need to define a notion of program equivalence which is well-behaved (is a congruence with respect to typing and is compatible with substitution and α-equivalence) and satisfies our axioms.

4.3.1 Notions of chained reduction

Definition 4.3.1 (Paths)**.** *Let $\phi, \phi' \in$ **Config**. We write $\pi : \phi \rightarrow^* \phi'$ to denote a* particular *reduction path from ϕ to ϕ', and we write*

$$\Pr(\pi) \triangleq \prod_{i=1\ldots n} \Pr(\phi_{i-1} \rightarrow \phi_i)$$

where $\phi_0 = \phi$ and $\phi_n = \phi'$.

Definition 4.3.2 (Probabilistic reduction)**.** *Let $\phi, \phi' \in$ **Config**. We write $\phi \downarrow^t_p \phi'$ iff $\phi' \not\rightarrow$ and*

$$p = \sum_{\pi : \phi \rightarrow^* \phi'} \Pr(\pi)$$

In other words, p is the probability that a reduction from ϕ lands on ϕ', which we can denote as $\Pr(\phi \rightarrow^* \phi') \triangleq p$ (as per the law of total probabilities).

Lemma 4.3.3 (Induction principle for $\downarrow^t_p$)**.** *Let $\phi \in$ **Config**. If $(\phi \rightarrow_{q_i} \phi'_i)_{i \in J}$ then*

$$\phi \downarrow^t_p \phi_f \iff \forall i \in J.\ \phi \rightarrow_{q_i} \phi' \downarrow^t_{p_i} \phi_f$$

where $p = \sum_{i \in J} q_i p_i$.

Proof. Follows directly from the probabilistic interpretation and the law of total probabilities. □

4.3.2 A first notion of program equivalence

A first naive definition of program equivalence is as follows:

Definition 4.3.4 (Probabilistic equivalence)**.** *We write*

$$\Gamma \mid \Delta \vdash c_1 \simeq_{pr} c_2$$

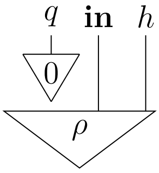
to mean that $\Gamma \mid \Delta \vdash c_1, c_2$*, and for all* $X_0 \in \mathbf{Ob\ CP}$*,* $t = (q, T) \in \mathsf{Stream}(X_0)$*,* $\sigma \in \{\mathsf{in}, \mathsf{out}\}^*$ *and* $\rho :_t \Delta, \sigma$*, for all* ρ_f*,* σ_f *and* $(x : n) \in \Gamma$*,*

$$\langle \rho, \sigma, c_1 \rangle \downarrow^t_p \langle \rho_f, \sigma_f, \mathsf{skip}_x \rangle$$

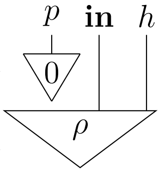
iff

$$\langle \rho, \sigma, c_2 \rangle \downarrow^t_p \langle \rho_f, \sigma_f, \mathsf{skip}_x \rangle$$

The issue with this definition is that it unfortunately does not respect α-equivalence. Consider the α-equivalent terms $x : 1 \mid \cdot \vdash \mathsf{new}(q.x(q)) \simeq_\alpha \mathsf{new}(p.x(p))$. It is clearly not possible to show that these two terms are equivalent under $\simeq_{\text{pr}}$ because for any suitable ρ and σ,

$\langle \rho, \sigma, \mathsf{new}(q.x(q)) \rangle \downarrow^t_1 \Big\langle$  $, \sigma, x(q) \Big\rangle$

and

$\langle \rho, \sigma, \mathsf{new}(p.x(p)) \rangle \downarrow^t_1 \Big\langle$  $, \sigma, x(p) \Big\rangle$

which are different.

One solution could be to define a notion of equivalence which allows different name lists on the final state ρ_f, as long as $\tilde{\rho}_f = \tilde{\rho}'_f$. We get the following notion of equivalence:

Definition 4.3.5 (Probabilistic skip equivalence)**.** *We write*

$$\Gamma \mid \Delta \vdash c_1 \simeq^{\mathsf{skip}}_{pr} c_2$$

to mean that $\Gamma \mid \Delta \vdash c_1, c_2$*, and for all* $X_0 \in \mathbf{Ob\ CP}$*,* $t = (q, T) \in \mathsf{Stream}(X_0)$*,* $\sigma \in \{\mathsf{in}, \mathsf{out}\}^*$*,* $\rho :_t \Delta, \sigma$*,* σ_f *and* $(x : n) \in \Gamma$*,*

$$\Pr\left(\langle \rho, \sigma, c_1 \rangle \to^* \langle [\tilde{\rho}_f], \sigma_f, \mathsf{skip}_x \rangle\right) = \Pr\left(\langle \rho, \sigma, c_2 \rangle \to^* \langle [\tilde{\rho}_f], \sigma_f, \mathsf{skip}_x \rangle\right)$$

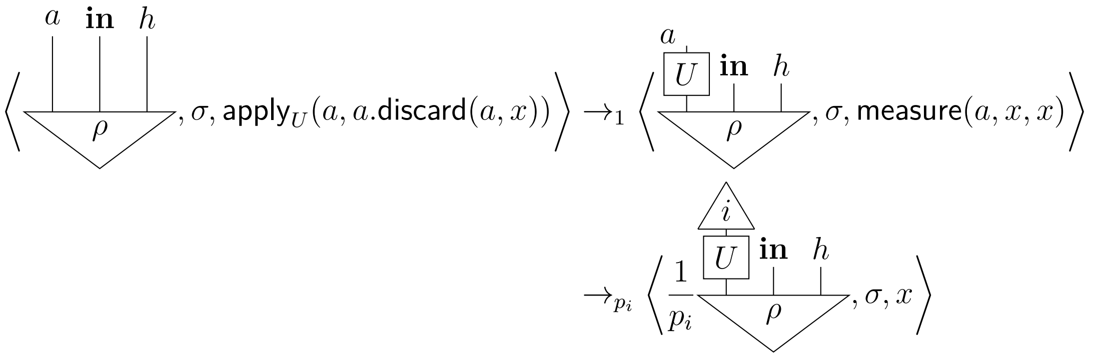
where the event $\phi \to^* \langle [\tilde{\rho}_f], \sigma_f, \mathsf{skip}_x \rangle$ *is that of* ϕ *terminating on some configuration* $\langle \rho', \sigma', x(\Delta_{\rho'}) \rangle$ *such that* $\tilde{\rho}' = \tilde{\rho}_f$ *and* $\sigma' = \sigma_f$*.*

This does solve the α-equivalence problem. Indeed, by ignoring the qubit names at the final state, we are allowing final states resulting from α-equivalent programs to be equated.

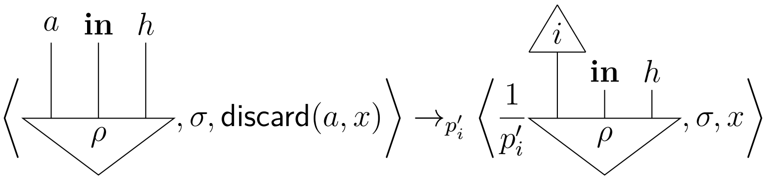
But this probabilistic skip equivalence fails to verify a key equation exhibiting quantum behaviour: eq. (C). We demonstrate this by considering the following equation:

$$x : 1 \mid a \vdash \mathsf{apply}_U(a, a.\mathsf{discard}(a, x)) = \mathsf{discard}(a, x)$$

where U is a single qubit unitary. Reasoning about its reduction paths gives



whereas



for $i = 0, 1$. Unfortunately, because we cannot reason about the sum of density matrices, it is not possible to show equivalence in the sense of definition 4.3.5.

4.3.3 A 'quantum' equivalence

This leads us naturally to the following definition:

Definition 4.3.6 (Quantum reduction)**.** *Let* $\Gamma \mid \Delta \vdash c$*,* $X_0 \in \mathbf{Ob\ CP}$*,* $t = (q, T) \in \mathsf{Stream}(X_0)$*,* $\sigma \in \{\mathsf{in}, \mathsf{out}\}^*$*,* $\rho :_t (\Delta, \sigma)$*. Let* $\phi = \langle \rho, \sigma, c \rangle$*.*

For $(x : n) \in \Gamma$*, we write* $\phi \Downarrow_p^{\Gamma, t} [\tilde{\rho}_f, \sigma_f, \mathsf{skip}_x]$ *to mean*

$$p = \Pr(\phi \to^* \langle -, \sigma_f, \mathsf{skip}_x \rangle) = \sum_{\psi \in \mathbf{Config}} \Pr(\phi \to^* \langle \psi, \sigma_f, x(\Delta_\psi) \rangle)$$

by the law of total probability, and

$$\tilde{\rho}_f = \sum_{\psi} \Pr(\phi \to^* \langle \psi, \sigma_f, x(\Delta_\psi) \rangle) \cdot \tilde{\psi}.$$

Note that this sum is well defined because by type preservation, any such ψ *satisfies* $\psi :_t (\Delta_\psi, \sigma_f)$ *where* $|\Delta_\psi| = n$*. Thus for any such* ψ *we have* $\tilde{\psi} : I \to \mathbf{qbit}^{\otimes n} \otimes \mathbf{qbit} \otimes q(\sigma_f)$*.*

Lemma 4.3.7 (Induction principle for $\Downarrow_p^{\Gamma,t}$)**.** *Let ϕ and ϕ_0, ϕ_1 be well-typed configurations and assume that $\phi \to_{q_i} \phi_i$ ($i \in J$ and J is finite) where $\sum_i q_i = 1$. Then*

$$\phi \Downarrow_p^{\Gamma,t} [\tilde{\rho}_f, \sigma_f, \mathsf{skip}_x] \iff \forall i \in J. \left(\phi \to_{q_i} \phi_i \Downarrow_{p_i}^{\Gamma,t} [\tilde{\rho}_i, \sigma_f, \mathsf{skip}_x]\right)$$

with $p = \sum_{i \in J} q_i p_i$ and $\tilde{\rho}_f = \frac{1}{p}(\sum_{i \in J} q_i p_i \tilde{\rho}_i)$

Proof. Follows directly from the probabilistic interpretation and the law of total probabilities. □

Definition 4.3.8 (Quantum Equivalence $\simeq_{qu}$)**.** *We write*

$$\Gamma \mid \Delta \vdash c_1 \simeq_{qu} c_2$$

to mean that $\Gamma \mid \Delta \vdash c_1, c_2$, and for all $X_0 \in \mathbf{Ob\ CP}$, $t = (q, T) \in \mathsf{Stream}(X_0)$, $\sigma \in \{\mathsf{in}, \mathsf{out}\}^*$, $\rho :_t \Delta, \sigma$, σ_f and $(x : n) \in \Gamma$,

$$\langle \rho, \sigma, c_1 \rangle \Downarrow_p^{\Gamma,t} [\tilde{\rho}_f, \sigma_f, \mathsf{skip}_x] \iff \langle \rho, \sigma, c_2 \rangle \Downarrow_p^{\Gamma,t} [\tilde{\rho}_f, \sigma_f, \mathsf{skip}_x].$$

Equivalently it means that

$$\sum_\psi \Pr(\langle \rho, \sigma, c_1 \rangle \to^* \langle \psi, \sigma_f, x(\Delta_\psi) \rangle) \cdot \tilde{\psi} = \sum_\psi \Pr(\langle \rho, \sigma, c_2 \rangle \to^* \langle \psi, \sigma_f, x(\Delta_\psi) \rangle) \cdot \tilde{\psi}$$

Remark: if we take the trace on both sides, we get the following equality

$$\Pr(\langle \rho, \sigma, c_1 \rangle \to^* \langle [-], \sigma_f, \mathsf{skip}_x \rangle) = \Pr(\langle \rho, \sigma, c_2 \rangle \to^* \langle [-], \sigma_f, \mathsf{skip}_x \rangle)$$

where the event $\phi \to^* \langle [-], \sigma_f, \mathsf{skip}_x \rangle$ means that ϕ terminates on some state of the form $\langle \rho', \sigma_f, x(\Delta_{\rho'}) \rangle$ with no constraints.

4.3.4 Basic Properties

This definition does indeed satisfy basic desired properties.

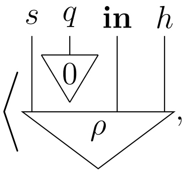
Proposition 4.3.9 (Congruence)**.** *The relation $\Gamma \mid \Delta \vdash c_1 \simeq_{qu} c_2$ is a congruence.*

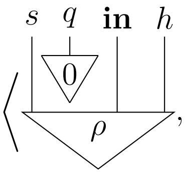
Proof. By induction. We only show the (var), (new) and (measure) cases; all the other cases are analogous to (new).

Case (var): Indeed it is easy to see that $\Gamma \mid \pi(\vec{a}) \vdash x(\vec{a}) \simeq_{qu} x(\vec{a})$: indeed both sides type check and by definition reduce to the same thing with the same probability.

Case (new): Assume that $\Gamma \mid \Delta, q \vdash c_1 \simeq_{qu} c_2$ we wish to show that $\Gamma \mid \Delta \vdash \mathsf{new}(q.c_1) \simeq_{qu} \mathsf{new}(q.c_2)$. Indeed, both sides type check. Moreover, let $\phi_1 = \langle \rho, \sigma, \mathsf{new}(q.c_1) \rangle$ and $\phi_2 =$

$\langle \rho, \sigma, \mathsf{new}(q.c_2)\rangle$ for $\rho :_t \Delta, \sigma$. Then it follows that

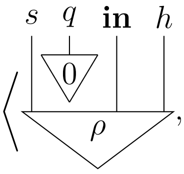
$\phi_1 \Downarrow_p^{\Gamma,t} [\tilde{\rho}_f, \sigma_f, \mathsf{skip}_x] \iff \Big\langle$  $, \sigma, c_1 \Big\rangle \Downarrow_p^{\Gamma,t} [\tilde{\rho}_f, \sigma_f, \mathsf{skip}_x]$ (by lemma 4.3.7)

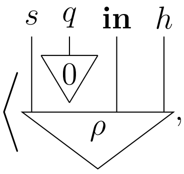
$\iff \Big\langle$  $, \sigma, c_2 \Big\rangle \Downarrow_p^{\Gamma,t} [\tilde{\rho}_f, \sigma_f, \mathsf{skip}_x]$ (IH)

$\iff \phi_2 \Downarrow_p^{\Gamma,t} [\tilde{\rho}_f, \sigma_f, \mathsf{skip}_x]$ (lemma 4.3.7)

Case (measure): Assume that $\Gamma \mid \Delta, \Delta' \vdash c_0 \simeq_{qu} c_1$ and $\Gamma \mid \Delta, \Delta' \vdash d_0 \simeq_{qu} d_1$ we wish to show that $\Gamma \mid \Delta, q, \Delta' \vdash \mathsf{measure}(q, c_0, c_1) = \mathsf{measure}(q, d_0, d_1)$. Indeed, both sides type check. Moreover, let $\phi = \langle \rho, \sigma, \mathsf{measure}(q, c_0, c_1)\rangle$ and $\phi' = \langle \rho, \sigma, \mathsf{measure}(q, d_0, d_1)\rangle$. It follows that

$\phi \Downarrow_p^{\Gamma,t} [\tilde{\rho}_f, \sigma_f, \mathsf{skip}_x]$

$\iff \forall i = 0, 1. \Big\langle$  $, \sigma, c_i \Big\rangle \Downarrow_p^{\Gamma,t} [\tilde{\rho}_f, \sigma_f, \mathsf{skip}_x]$ (by lemma 4.3.7)

$\iff \forall i = 0, 1. \Big\langle$  $, \sigma, d_i \Big\rangle \Downarrow_p^{\Gamma,t} [\tilde{\rho}_f, \sigma_f, \mathsf{skip}_x]$ (IH)

$\iff \phi' \Downarrow_p^{\Gamma,t} [\tilde{\rho}_f, \sigma_f, \mathsf{skip}_x]$ (lemma 4.3.7)

□

Proposition 4.3.10 (Compatibility with $\simeq_\alpha$). *If* $\Gamma \mid \Delta \vdash c_1 \simeq_\alpha c_2$ *then* $\Gamma \mid \Delta \vdash c_1 \simeq_{qu} c_2$.

Proof outline. Assume that $\Gamma \mid \Delta \vdash c_1 \simeq_\alpha c_2$. Because both $\simeq_\alpha$ and $\simeq_{qu}$ are congruences, we need only show that if $c_1 = \mathbf{op}(\vec{a}, (\vec{b}_i.c_i)_i)$ and $c_2 = \mathbf{op}(\vec{a}, (\vec{d}_i.c_i[\vec{d}_i/\vec{b}_i]))$ then $c_1 \simeq_{qu} c_2$.

And indeed, this is the case simply because they are the same program, so they will reduce to the same configurations with the same probability; and because the definition of $\simeq_{qu}$ ignores the names at the end, it follows easily that $c_1 \simeq_{qu} c_2$. □

Proposition 4.3.11 (Relating probabilistic and quantum equivalences). *If* $c_1 \simeq_{pr}^{\mathsf{skip}} c_2$ *then* $c_1 \simeq_{qu} c_2$.

Proof. Follows directly from definitions. Indeed if $\Gamma \mid \Delta \vdash c_1 \simeq_{pr}^{\mathsf{skip}} c_2$, we have that

$$\Pr\left(\langle \rho, \sigma, c_1\rangle \to^* \langle[\tilde{\rho}_f], \sigma_f, \mathsf{skip}_x\rangle\right) = \Pr\left(\langle \rho, \sigma, c_2\rangle \to^* \langle[\tilde{\rho}_f], \sigma_f, \mathsf{skip}_x\rangle\right)$$

And so, by the law of total probability, it follows that

$$\Pr\left(\langle \rho, \sigma, c_1\rangle \rightarrow^* \langle [-], \sigma_f, \mathsf{skip}_x\rangle\right) = \Pr\left(\langle \rho, \sigma, c_2\rangle \rightarrow^* \langle [-], \sigma_f, \mathsf{skip}_x\rangle\right)$$

and

$$\begin{aligned}\sum_{\psi} \Pr(\langle \rho, \sigma, c_1\rangle \rightarrow^* \langle \psi, \sigma_f, x(\Delta_\psi)\rangle) \cdot \tilde{\psi} &= \sum_{\tilde{\psi}} \Pr(\langle \rho, \sigma, c_1\rangle \rightarrow^* \left\langle [\tilde{\psi}], \sigma_f, \mathsf{skip}_x\right\rangle) \cdot \tilde{\psi} \\ &= \sum_{\tilde{\psi}} \Pr(\langle \rho, \sigma, c_2\rangle \rightarrow^* \left\langle [\tilde{\psi}], \sigma_f, \mathsf{skip}_x\right\rangle) \cdot \tilde{\psi} \\ &= \sum_{\psi} \Pr(\langle \rho, \sigma, c_2\rangle \rightarrow^* \langle \psi, \sigma_f, x(\Delta_\psi)\rangle) \cdot \tilde{\psi}\end{aligned}$$

as required. □

4.3.5 Reduction Paths and Compatibility with Substitution

Finally, we have to prove the compatibility of $\simeq_{qu}$ with substitution, i.e. if two programs are equivalent, then instantiating a computation variable with an arbitrary continuation preserves that equivalence. This is actually not a trivial fact to prove. In fact, suppose two programs c and c' are equivalent under $\simeq_{qu}$, and further suppose that under state ρ_I, stream t and σ_I we have two distinct paths in c: $\pi_{i=1,2} : \langle \rho_I, \sigma_I, c\rangle \rightarrow^*_{p_i} \langle \rho_i, \sigma_f, \mathsf{skip}_x\rangle$, as well as two distinct paths from c': $\pi'_{i=1,2} : \langle \rho_I, \sigma_I, c'\rangle \rightarrow^*_{p'_i} \langle \rho'_i, \sigma_f, \mathsf{skip}_x\rangle$ such that $\sum_i p_i\rho_i = \sum_i p'_i\rho'_i$, but the ρ_i's and the ρ'_i's are distinct states. Intuitively, it is unclear whether there exists a continuation that we can insert in place of x that can distinguish the ρ_i's from the ρ'_i's, but if this were the case, we would have broken the equivalence between the substitutions of c and c'.

It turns out that this can never happen, and we will show this fact mathematically. For this purpose, we first need to define an abstract notion of reduction paths that does not depend on an initial state.

4.3.5.1 Abstract reduction paths overapproximate concrete reduction sequences

We begin by giving a symbolic notion of reduction paths generated from a program c.

Definition 4.3.12 (Abstract Reduction paths)**.** *An abstract reduction path P from a term $\Gamma \mid \Delta \vdash c$ is informally a sequence of terms $c_0 \rightarrow c_1 \rightarrow \ldots \rightarrow c_n$ such that $c_0 = c$, $c_n = \mathsf{skip}_x$ for some x and each measurement is paired with its measurement outcome. We formally define the set of reduction paths from $\Gamma \mid \Delta \vdash c$ as* $\mathsf{Path}\,(c)$ *as in fig. 4.5.*

Conventionally, we call an abstract reduction path from a term c a 'path', whereas concrete reduction sequences from a given configuration $\langle \rho, \sigma, c\rangle$ will be referred to as 'sequences'.

Remark 4.3.13. *Paths $P \in \mathsf{Path}\,(c)$ are exactly the linear subtrees of c's typing derivation, i.e. subtrees which can be written as a list from c to its root.*

$$\begin{aligned}\mathsf{Path}\,(T \equiv \Gamma \mid \Delta, q, \Delta' \vdash \mathsf{measure}(q, c_0, c_1)) = \{T \to^{(0)} L_0 \mid L_0 \in \mathsf{Path}\,(c_0)\} \\ \cup\, \{T \to^{(1)} L_1 \mid L_1 \in \mathsf{Path}\,(c_1)\}\end{aligned}$$

$$\begin{aligned}\mathsf{Path}\,(T \equiv \Gamma, x : n, \Gamma' \mid \vec{a} \vdash x(\vec{a})) &= \{T\} \\ \mathsf{Path}\,(T \equiv \Gamma, x : n, \Gamma' \mid \pi(\vec{a}) \vdash x(\vec{a})) &= \{T \to \Gamma, x : n, \Gamma' \mid \vec{a} \vdash x(\vec{a})\} \\ &\quad \text{(where } \pi \neq \mathbf{id} \text{ is a permutation)} \\ \mathsf{Path}\,(T) &= \{T \to L \mid L \in \mathsf{Path}\,(c)\} \\ &\quad \text{(for any other } T \text{ of the form } \Gamma \mid \Delta \vdash \mathbf{op}(\vec{a}, \vec{b}.c)).\end{aligned}$$

Figure 4.5: Definition for the Reduction Path $\mathsf{Path}\,(T)$ where T is an instance of the typing relation.

For any path $P \in \mathsf{Path}\,(c)$, we write $\sigma(P) \in \{\mathsf{in}, \mathsf{out}\}^*$ as the sequence of inputs and outputs which happen in P, and define it as follows:

$$\begin{aligned}\sigma(\Gamma \mid \Delta \vdash \mathsf{in}(q.c) \to P) &= \mathsf{in} + \sigma(P) \\ \sigma(\Gamma \mid \Delta, q, \Delta' \vdash \mathsf{out}(q, c) \to P) &= \mathsf{out} + \sigma(P) \\ \sigma(\Gamma, x : n, \Gamma' \mid \pi(\vec{a}) \vdash x(\vec{a})) &= \epsilon \\ \sigma((\Gamma \mid \Delta \vdash c) \to P) &= \sigma(P) \qquad\qquad (c \neq \mathsf{in}, c \neq \mathsf{out})\end{aligned}$$

For any path P, we also write $x_P : n_P$ to denote the computation variable called by P at the end.

Proposition 4.3.14. *Given a term $\Gamma \mid \Delta \vdash c$, a path $P \in \mathsf{Path}\,(c)$ is fully determined by a list of measurement outcomes, one for each measurement along the path.*

Proof. Evident, because measurement is the only case where there is a choice of the branch to take when constructing the path. □

Definition 4.3.15 (Reduction sequence induced by a path)**.** *Let $\Gamma \mid \Delta \vdash c$ and $P \in \mathsf{Path}\,(c)$, a reduction sequence $\pi = \langle \rho_0, \sigma_0, c_0 \rangle \to \ldots \to \langle \rho_n, \sigma_n, c_n \rangle$ where $c_0 = c$ and $c_n = \mathsf{skip}_x$ where $x : n \in \Gamma$ is said to be induced by $P = c = d_0 \to d_1 \ldots \to d_n = \mathsf{skip}_{x_P}$ if*

- *$x_P = x$,*
- *$c_i = d_i$ for all $i = 0 \ldots n$,*
- *$\sigma_n = \sigma_0 + \sigma(P)$*
- *For each measurement along π, say $\phi \to \phi'$ such that the measurement outcome is i, the corresponding $c \to_j c'$ in P satisfies $j = i$.*

We denote this as $\pi : P$.

The reason why we create this formalism is demonstrated in the following proposition: we wish to say that the possible concrete reductions from a given configuration can be safely overapproximated by the abstract, symbolic reduction paths of the program itself. Note that we distinguish between concrete reduction sequences with the same sequence of configurations but different measurement outcomes at each measurement, for it wil be vital in the subsequent proof.

Proposition 4.3.16 (Concrete reduction sequences are induced by abstract reduction paths)**.** *Let $\Gamma \mid \Delta \vdash c$. Let $t \in \mathsf{Stream}X_0$, $\sigma_0 \in \{\mathsf{in}, \mathsf{out}\}^*$ and $\rho_0 :_t \Delta, \sigma_0$. Then there is an injection from every possible concrete reduction sequence π from $\langle \rho_0, \sigma_0, c\rangle$, to the abstract reduction paths $\mathsf{Path}\,(c)$. In other words, for every reduction sequence π there is a distinct unique path $P^{\langle \rho_0,\sigma_0,c\rangle}(\pi) \in \mathsf{Path}\,(c)$ such that $\pi : P^{\langle \rho_0,\sigma_0,c\rangle}(\pi)$.*

Proof. By induction over the structure of c.

Case (var): If $\Gamma \mid \Delta \vdash x(\Delta)$ for some $x \in \Delta$, any configuration $\langle \rho_0, \sigma_0, x(\Delta)\rangle$ will not reduce. The corresponding singleton reduction path will thus be induced by the unique element in $\mathsf{Path}\,(\Gamma \mid \Delta \vdash x(\Delta)) = \{\Gamma \mid \Delta \vdash x(\Delta)\}$.

If we have $\Gamma \mid \pi(\Delta) \vdash x(\Delta)$, then for any ρ_0, σ_0, the reduction path will look like $\pi = \langle \rho_0, \sigma_0, x(\Delta)\rangle \to \langle \pi^{-1} \circ \rho_0, \sigma_0, x(\Delta)\rangle$. On the other hand, $\mathsf{Path}\,(c) = \{\Gamma \mid \pi(\Delta) \vdash x(\Delta) \to \Gamma \mid \Delta \vdash x(\Delta)\}$. Call P its unique element, then we clearly have that $\pi : P$.

Case (measure): We assume that $\Gamma \mid \Delta, \Delta' \vdash c_i$ for $i = 0, 1$ as well as $\Gamma \mid \Delta \vdash \mathsf{measure}(q, c_0, c_1)$. By (IH) we have that for every $i = 0, 1$ and every valid configuration $\langle \rho, \sigma, c_i\rangle$, there is an injection $\pi \mapsto P^{\langle \rho,\sigma,c_i\rangle}(\pi)$ from its possible reduction sequences to $\mathsf{Path}\,(c_i)$.

Now let ρ, σ be such that $\phi = \langle \rho, \sigma, \mathsf{measure}(q, c_0, c_1)\rangle$ is well-formed. Then for reduction path π starting at ϕ we know that either $\pi = \phi \to \langle \rho_0, \sigma, c_0\rangle$ or $\pi = \phi \to \langle \rho_1, \sigma, c_1\rangle$ (where ρ_i is the state given measurement outcome i), if their probabilities are not zero. On the other hand, we know that $\mathsf{Path}\,(\mathsf{measure}(q, c_0, c_1)) = \{\mathsf{measure}(q, c_0, c_1) \to^{(0)} \mathsf{Path}\,(c_0)\} \cup \{\mathsf{measure}(q, c_0, c_1) \to^{(1)} \mathsf{Path}\,(c_1)\}$. Therefore, we can simply construct P^ϕ as the map

$$P^\phi : \pi \mapsto \begin{cases} \mathsf{measure}(q, c_0, c_1) \to^{(0)} P^{\phi_0}(\pi') & (\text{if } \pi = \phi \to \pi' \text{ with meas. outcome } 0) \\ \mathsf{measure}(q, c_0, c_1) \to^{(1)} P^{\phi_1}(\pi') & (\text{if } \pi = \phi \to \pi' \text{ with meas. outcome } 1) \end{cases}$$

where $\phi_0 = \langle \rho_0, \sigma, c_0\rangle$ and $\phi_1 = \langle \rho_1, \sigma, c_1\rangle$. This map is injective by definition, because we are distinguishing between concrete reduction sequences with different measurement outcomes. Note that it is here that we are approximating: when doing a measurement, one of the branches might have probability 0, in which case all paths corresponding to the branch will not have a corresponding concrete reduction sequence.

Case (in): We assume that $\Gamma \mid \Delta, q \vdash c$ and $\Gamma \mid \Delta \vdash \mathsf{in}(q.c)$. Furthermore, (IH) says that for every valid configuration $\langle \rho, \sigma, c\rangle$, there is an injection $\pi \mapsto P^{\langle \rho,\sigma,c\rangle}(\pi)$ from its possible reduction sequences to $\mathsf{Path}\,(c)$.

Now let ρ_0, σ_0 such that $\phi_0 = \langle \rho_0, \sigma_0, \mathsf{in}(q.c)\rangle$ is well-formed. Then, we know from inversion that every path π starting at ϕ_0 is of the form $\phi_0 \to_1 \langle \rho_1, \sigma_0 + \mathsf{in}, c\rangle$, where ρ_1 is the evolved state, which is well-formed by type preservation. On the other hand, we know that $\mathsf{Path}\,(\mathsf{in}(q.c)) = \{\mathsf{in}(q.c) \to P \in \mathsf{Path}\,(c)\}$. Thus, we can inject all reduction sequences from $\langle \rho_0, \sigma_0, \mathsf{in}(q.c)\rangle$ to $\mathsf{Path}\,(\mathsf{in}(q.c))$ with the map

$$P^{\phi_0} : \pi \mapsto \mathsf{in}(q.c) \to P^{\langle \rho_1, \sigma_0 + \mathsf{in}, c\rangle}(\pi')$$

where $\pi = \phi_0 \to \pi'$ and π' is a sequence starting at $\langle \rho_1, \sigma_0 + \mathsf{in}, c\rangle$.

Case (out), (new), (apply): analogous, because they are all deterministic. □

Remark 4.3.17. *We can also reason about the inverse of the map defined in the above proposition. In fact, every path $P \in \mathsf{Path}\,(c)$ along with a starting state $\phi = \langle \rho_I, \sigma_I, c\rangle$ canonically induces a reduction sequence $\pi^{\phi}_P : P$, which may not be well-defined if $\Pr(\pi^{\phi}_P) = 0$.*

4.3.5.2 State evolution only depends on the abstract path and the stream state

Perhaps surprisingly at first, not only do reduction paths completely determine concrete reduction sequences, they also fully determine the evolution of the global quantum state given a concrete stream $t \in \mathsf{Stream}X_0$ and a stream state $\sigma \in \{\mathsf{in}, \mathsf{out}\}^*$. Intuitively, this is because choosing a path P chooses the measurement outcomes, hence grounds all the possible non-determinism in a concrete reduction sequence (proposition 4.3.14), and the map that such a path P would give can be taken directly from accumulating the CP map that the operational semantics applies to the state at every step. Of course, P might have probability 0, but in this case, the resulting accumulated map will be a 0 map.

More formally, let us define $[\![P]\!]^{(q,T),\sigma}$, the interpretation of P under stream (q, T) and trace σ. By convention, we interpret contexts as follows:

- For any $n \in \mathbb{N}$, $[\![n]\!] \triangleq \mathbf{qbit}^{\otimes n}$.
- For any parameter context Δ, $[\![\Delta]\!] \triangleq [\![|\Delta|]\!]$, where $|\Delta|$ is the size of Δ.

Definition 4.3.18 (Interpretation of paths with concrete streams)**.** *Let $\Gamma \mid \Delta \vdash c$, $P \in \mathsf{Path}\,(c)$, and concrete stream $t = (q, T) \in \mathsf{Stream}X_0$. The interpretation of P in t at state σ is given by a morphism $[\![P]\!]^{(q,T),\sigma} : [\![\Delta]\!] \otimes \mathbf{qbit} \otimes q(\sigma) \to [\![n_P]\!] \otimes \mathbf{qbit} \otimes q(\sigma + \sigma(P))$ in $\mathbf{CP}$ defined as in fig. 4.6.*

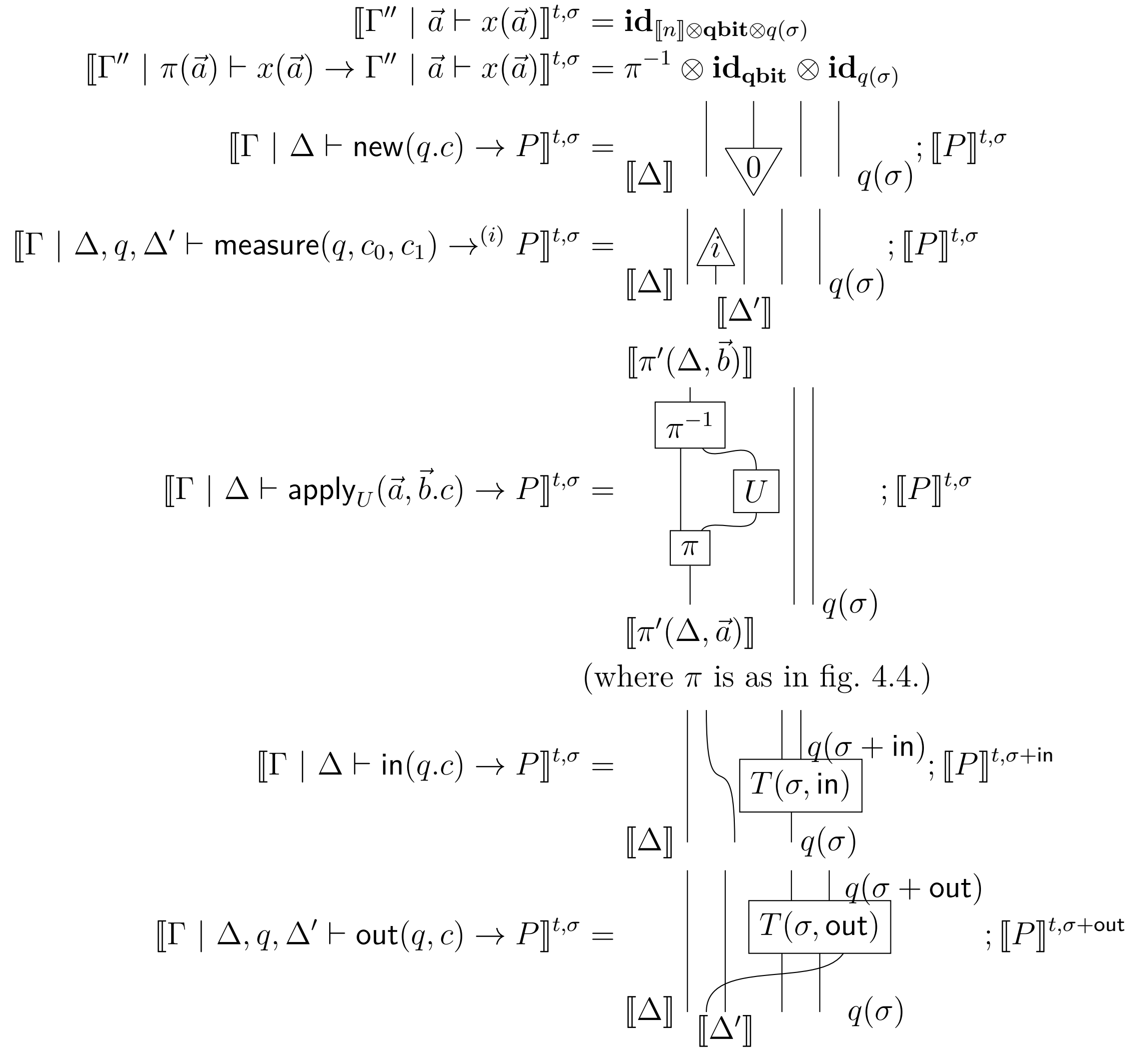
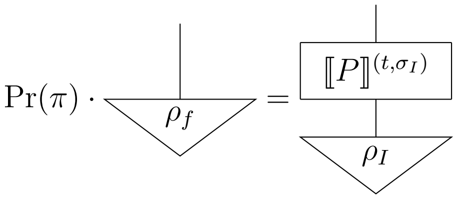

$[\![\Gamma'' \mid \vec{a} \vdash x(\vec{a})]\!]^{t,\sigma} = \mathbf{id}_{[\![n]\!]\otimes\mathbf{qbit}\otimes q(\sigma)}$
$[\![\Gamma'' \mid \pi(\vec{a}) \vdash x(\vec{a}) \to \Gamma'' \mid \vec{a} \vdash x(\vec{a})]\!]^{t,\sigma} = \pi^{-1} \otimes \mathbf{id}_{\mathbf{qbit}} \otimes \mathbf{id}_{q(\sigma)}$
$[\![\Gamma \mid \Delta \vdash \mathsf{new}(q.c) \to P]\!]^{t,\sigma} =$
$[\![\Delta]\!]$
0
$q(\sigma)$
$; [\![P]\!]^{t,\sigma}$
$[\![\Gamma \mid \Delta, q, \Delta' \vdash \mathsf{measure}(q, c_0, c_1) \to^{(i)} P]\!]^{t,\sigma} =$
$[\![\Delta]\!]$
i
$[\![\Delta']\!]$
$q(\sigma)$
$; [\![P]\!]^{t,\sigma}$
$[\![\pi'(\Delta, \vec{b})]\!]$
π^{-1}
U
π
$[\![\Gamma \mid \Delta \vdash \mathsf{apply}_U(\vec{a}, \vec{b}.c) \to P]\!]^{t,\sigma} =$
$; [\![P]\!]^{t,\sigma}$
$q(\sigma)$
$[\![\pi'(\Delta, \vec{a})]\!]$
(where π is as in fig. 4.4.)
$[\![\Gamma \mid \Delta \vdash \mathsf{in}(q.c) \to P]\!]^{t,\sigma} =$
$q(\sigma + \mathsf{in})$
$; [\![P]\!]^{t,\sigma+\mathsf{in}}$
$T(\sigma, \mathsf{in})$
$[\![\Delta]\!]$
$q(\sigma)$
$q(\sigma + \mathsf{out})$
$[\![\Gamma \mid \Delta, q, \Delta' \vdash \mathsf{out}(q, c) \to P]\!]^{t,\sigma} =$
$T(\sigma, \mathsf{out})$
$; [\![P]\!]^{t,\sigma+\mathsf{out}}$
$[\![\Delta]\!]$
$[\![\Delta']\!]$
$q(\sigma)$

Figure 4.6: Interpretation of path P with stream $t = (q, T)$ at state σ. Note that unlabelled wires are of type **qbit**, and that $\Gamma'' \triangleq \Gamma, x : n, \Gamma'$

Proposition 4.3.19 (State evolution given the path and the stream)**.** *Let* $\Gamma \mid \Delta \vdash c$ *and* $P \in \mathsf{Path}\,(c)$*. Let* $t = (q, T) \in \mathsf{Stream}X_0$ *be a stream. Also let* π *be* $\langle \rho_I, \sigma_I, c \rangle \to \ldots \to \left\langle \rho_f, \sigma_f, \mathsf{skip}_{x_P} \right\rangle$*. Let* $\pi : P$*, then*

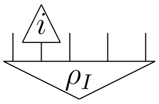


Proof. The proof is by induction on the structure of P, and follows directly from the definition of the operational semantics.

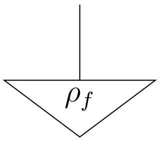
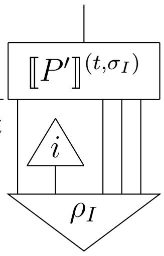
For $P = \Gamma, x : n, \Gamma' \mid \vec{a} \vdash x(\vec{a})$, the path has no transition, thus π will also be a singleton sequence, meaning that $\rho_f = \mathbf{id} \circ \rho_I$ as required.

For $P = \Gamma, x : n, \Gamma' \mid \pi(\vec{a}) \vdash x(\vec{a}) \to \Gamma, x : n, \Gamma' \mid \vec{a} \vdash x(\vec{a})$, we get one deterministic transition which indeed applies π^{-1} on ρ_I to obtain ρ_f, as required.

For all the deterministic cases (every case apart from measurement), the result follows from the definition of the operational semantics.

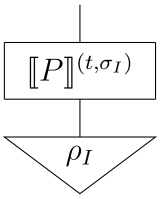
Finally, for $P = \Gamma \mid \Delta, q, \Delta' \vdash \mathsf{measure}(q, c_0, c_1) \to^{(i)} P'$ where $P' \in \mathsf{Path}\,(c_i)$, the proof goes as follows. By reasoning on (measure) rule of the operational semantics, we know that π is of the form $\langle \rho_I, \sigma_I, \mathsf{measure}(q, c_0, c_1)\rangle \to_{p_i} \pi'$ where $p_i > 0$, $\pi' : P'$ is some reduction sequence starting from $\left\langle \frac{1}{p_i}\rho, \sigma_I, c_i \right\rangle$, $p_i = \mathbf{tr}\rho$, and $\rho = \frac{1}{p_i}$ .

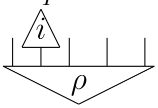
By (IH), we get that

$\mathrm{Pr}(\pi')$  $= \dfrac{1}{p_i}$ 

And multiplying p_i on both sides and using the definition of $\mathrm{Pr}(\pi)$ and $[\![P]\!]^{t,\sigma_I}$ the equality follows. □

Remark 4.3.20. *It is also worth noting that if* $\mathrm{Pr}(\pi_P^{\langle \rho_I, \sigma_I, c\rangle}) = 0$ *with* $P \in \mathsf{Path}\,(c)$*, then*

 $= 0$

holds. Informally, this is because the reduction sequence $\pi_P^{\langle \rho_I, \sigma_I, c\rangle}$ *will be well-defined up to a measurement where the branch taken in* $\pi_P^{\langle \rho_I, \sigma_I, c\rangle}$ *has* 0 *probability, meaning that the trace of the corresponding post-measurement state*  *is* 0*. By lemma 2.3.6, we thus know that the resulting state is the* 0 *matrix, which implies that the final state is the* 0 *matrix.*

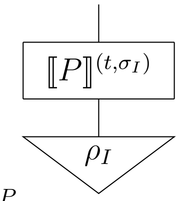
Corollary 4.3.21. *We have the equality*

$$\sum_{\pi : \langle \rho_I, \sigma_I, c\rangle \to^* \left\langle \rho_f, \sigma_I + \sigma, \mathsf{skip}_x \right\rangle} \mathrm{Pr}(\pi) \cdot \tilde{\rho}_f = \sum_{P \in \mathsf{Path}(c), x_P = x, \sigma_P = \sigma} [\![P]\!]^{t,\sigma_I} \circ \rho_I$$

Proof.

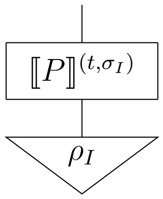
$$\mathsf{LHS} = \sum_{\substack{P \in \mathsf{Path}(c) \\ \text{s.t. } x_P = x, \sigma_P = \sigma \\ \exists \pi : \langle \rho_I, \sigma_I, c\rangle \to^* \left\langle \rho_f, \sigma_I + \sigma, \mathsf{skip}_x \right\rangle . \\ P^{\left\langle \rho_I, \sigma_I, c\right\rangle}(\pi) = P}} \mathrm{Pr}(\pi) \cdot \tilde{\rho}_f \qquad \text{(proposition 4.3.16)}$$

$$= \sum_{\substack{P \in \mathsf{Path}(c) \\ \text{s.t. } x_P = x, \sigma_P = \sigma \\ \exists \pi . P^{\left\langle \rho_I, \sigma_I, c \right\rangle}(\pi) = P}}$$



(proposition 4.3.19)

$$= \sum_{\substack{P \in \mathsf{Path}(c) \\ \text{s.t. } x_P = x \ \wedge \ \sigma_P = \sigma}}$$



(cf. remark 4.3.20)

$$= \mathsf{RHS}$$

□

4.3.5.3 Compatibility with substitution

With the above results, compatibility with substitution falls out naturally as a corollary.

Proposition 4.3.22 (Compatibility with substitution)**.** *If* $\Gamma, x : n, \Gamma' \mid \Delta \vdash c_1 \simeq_{qu} c_2$ *and* $\Gamma, \Gamma' \mid \Delta' \vdash c$ *then* $\Gamma, \Gamma' \mid \Delta \vdash c_1\{\Delta' \vdash c/x\} \simeq_{qu} c_2\{\Delta' \vdash c/x\}$.

Proof. For our convenience, we use the following notation:

- We note $c_1\{\Delta' \vdash c/x\}$ as c_1' and resp for c_2'.
- The initial states are noted as $\phi_1 = \langle \rho_I, \sigma_I, c_1' \rangle$ and $\phi_2 = \langle \rho_I, \sigma_I, c_2' \rangle$.
- We write $\bar{\phi}_1$ (resp $\bar{\phi}_2$) for $\langle \rho_I, \sigma_I, c_1 \rangle$ (resp $\langle \rho_I, \sigma_I, c_2 \rangle$) as the initial state if we were to run the programs before substitution.

Firstly, the assumption means that for all σ and for all $z \in (\Gamma, x : n, \Gamma')$ and $\sigma \in \{\mathsf{in}, \mathsf{out}\}^*$,

$$\sum_{\psi} \Pr(\bar{\phi}_1 \to^* \langle \psi, \sigma, \mathsf{skip}_x \rangle) \cdot \tilde{\psi} = \sum_{\psi'} \Pr(\bar{\phi}_2 \to^* \langle \psi', \sigma, \mathsf{skip}_x \rangle) \cdot \tilde{\psi}'$$

We wish to show that for any $y \in (\Gamma, \Gamma')$ and any $\sigma_f \in \{\mathsf{in}, \mathsf{out}\}^*$,

$$\sum_{\rho} \Pr(\phi_1 \to^* \left\langle \rho, \sigma_f, \mathsf{skip}_y \right\rangle) \cdot \tilde{\rho} = \sum_{\rho'} \Pr(\phi_2 \to^* \left\langle \rho', \sigma_f, \mathsf{skip}_y \right\rangle) \cdot \tilde{\rho}$$

Let's focus on studying the LHS. Firstly, given an execution sequence of the form $\phi_1 \to^* \langle \rho, \sigma_f, y \rangle$, we know that either it followed a path completely captured in the reduction tree for $\bar{\phi}$, or it followed a path in the reduction tree of $\bar{\phi}$ up to a configuration of the form $\langle \psi, \sigma, \mathsf{skip}_x \rangle$ for some σ and ψ, then continues according to ϕ's reduction tree from $\langle \psi, \sigma, c[\Delta_\psi/\Delta'] \rangle$ to a configuration of the form $\left\langle \rho, \sigma_f, \mathsf{skip}_y \right\rangle$.

In other words, we get that

$$\begin{aligned}\mathsf{LHS} &= \sum_{\rho} \Pr(\bar{\phi}_1 \to^* \left\langle \rho, \sigma_f, \mathsf{skip}_y \right\rangle) \cdot \tilde{\rho} \\ &\quad + \sum_{\rho,\psi,\sigma} \Pr(\bar{\phi}_1 \to^* \langle \psi, \sigma, \mathsf{skip}_x \rangle \wedge \langle \psi, \sigma, c[\Delta_\psi/\Delta'] \rangle \to^* \left\langle \rho, \sigma_f, \mathsf{skip}_y \right\rangle) \cdot \tilde{\rho}.\end{aligned}$$

Now, by assumption, we know that

$$\sum_{\rho} \Pr(\bar{\phi}_1 \to^* \left\langle \rho, \sigma, \mathsf{skip}_y \right\rangle) \cdot \tilde{\rho} = \sum_{\rho} \Pr(\bar{\phi}_2 \to^* \left\langle \rho, \sigma, \mathsf{skip}_y \right\rangle) \cdot \tilde{\rho}$$

Thus it suffices to show

$$\begin{aligned}&\sum_{\rho,\psi,\sigma} \Pr(\bar{\phi}_1 \to^* \langle \psi, \sigma, \mathsf{skip}_x \rangle \wedge \langle \psi, \sigma, c[\Delta_\psi/\Delta'] \rangle \to^* \left\langle \rho, \sigma_f, \mathsf{skip}_y \right\rangle) \cdot \tilde{\rho} \\ &= \sum_{\rho,\psi,\sigma} \Pr(\bar{\phi}_2 \to^* \langle \psi, \sigma, \mathsf{skip}_x \rangle \wedge \langle \psi, \sigma, c[\Delta_\psi/\Delta'] \rangle \to^* \left\langle \rho, \sigma_f, \mathsf{skip}_y \right\rangle) \cdot \tilde{\rho}.\end{aligned}$$

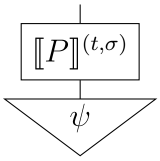
Let's consider the LHS.

$$\mathsf{LHS} = \sum_{\sigma} \sum_{\psi} \Pr(\bar{\phi}_1 \to^* \langle \psi, \sigma, \mathsf{skip}_x \rangle) \sum_{\rho} \left[\Pr(\langle \psi, \sigma, c[\Delta_\psi/\Delta'] \rangle \to^* \left\langle \rho, \sigma_f, \mathsf{skip}_y \right\rangle) \cdot \tilde{\rho} \right]$$

Let σ and ψ be fixed and arbitrary. Then

$$\begin{aligned}&\sum_{\rho} \Pr\left(\langle \psi, \sigma, c[\Delta_\psi/\Delta'] \rangle \to^* \left\langle \rho, \sigma_f, \mathsf{skip}_y \right\rangle \right) \cdot \tilde{\rho} \\ &= \sum_{\substack{\pi : \left\langle \psi, \sigma, c[\Delta_\psi/\Delta'] \right\rangle \to^* \left\langle \rho, \sigma_f, \mathsf{skip}_y \right\rangle \\ \text{for some end state } \rho}} \Pr(\pi) \cdot \tilde{\rho}\end{aligned}$$

$$= \sum_{\substack{P \in \mathsf{Path}\left(c[\Delta_\psi/\Delta']\right) \\ \text{s.t. } x_P = y \wedge \sigma + \sigma_P = \sigma_f}}$$

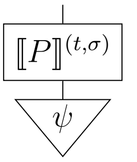


(corollary 4.3.21)

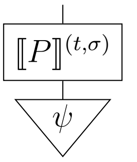
Note that in this sum, the side condition on P does not actually depend on ψ. Indeed, $[\![P]\!]^{t,\sigma}$ is invariant under α conversion, which means that the names in Δ_ψ are irrelevant to the proof.

It now follows that

$$\mathsf{LHS} = \sum_{\sigma} \sum_{\psi} \Pr(\bar{\phi}_1 \to^* \langle \psi, \sigma, \mathsf{skip}_x \rangle) \quad \textstyle\sum_P$$



$$= \sum_{\sigma} \sum_{\psi} \Pr(\bar{\phi}_2 \to^* \langle \psi, \sigma, \mathsf{skip}_x \rangle) \quad \textstyle\sum_P$$



(assumption)

$$= \mathsf{RHS}$$

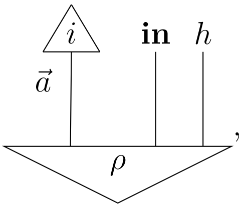
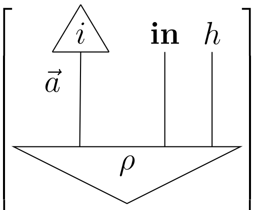
(by same reasoning)

□

4.3.6 Satisfying equations of $\mathsf{I/O} \otimes \mathsf{QUANTUM}$

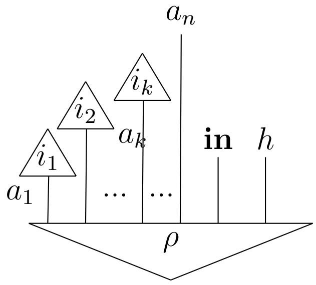
The final step to show that the system we have is a model is to show that the quantum equivalence satisfies the equations of $\mathsf{I/O} \otimes \mathsf{QUANTUM}$.

Lemma 4.3.23. *Consider the term $x : n \mid \vec{a} \vdash \mathsf{discard}_n(\vec{a}.x)$. Let $\rho :_t (\vec{a}), \sigma$, then*

$\langle \rho, \sigma, \mathsf{discard}_n(\vec{a}.x)\rangle \downarrow^t_{p_i} \left\langle \frac{1}{p_i}\right.$  $\left., \sigma, x \right\rangle$ *where* $p_i = \mathbf{tr}\Big[$  $\Big]$

for $i = 0...n-1$.

Proof. We write each $i = 0...n-1$ as an n bit number $i_1 i_2 ... i_n$ where $i_k = 0, 1$. For convenience, let's write

$\rho_{i_1...i_k} =$ 

and $q_{i_1...i_k} = \mathbf{tr}(\rho_{i_1...i_k})$.

$$\begin{aligned}
\langle \rho, \sigma, \mathsf{discard}_n(\vec{a}.x)\rangle &\rightarrow_{q_{i_1}} \left\langle \frac{1}{q_{i_1}} \rho_{i_1}, \sigma, \mathsf{discard}_{n-1}(a_2...a_n.x)\right\rangle \\
&\rightarrow_{q_{i_1 i_2}/q_{i_1}} \left\langle \frac{1}{q_{i_1 i_2}} \rho_{i_1 i_2}, \sigma, \mathsf{discard}_{n-2}(a_3...a_n.x)\right\rangle \\
... &\rightarrow_{q_{i_1...i_k}/q_{i_1...i_{k-1}}} \left\langle \frac{1}{q_{i_1...i_k}} \rho_{i_1 i_k}, \sigma, \mathsf{discard}_{n-k}(a_{k+1}...a_n.x)\right\rangle \\
... &\rightarrow_{q_{i_1...i_n}/q_{i_1...i_{n-1}}} \left\langle \frac{1}{q_{i_1...i_n}} \rho_{i_1 i_n}, \sigma, x\right\rangle
\end{aligned}$$

Thus with probability

$$p_i = q_{i_1} \times q_{i_1 i_2}/q_{i_1} ... \times q_{i_1...i_n}/q_{i_1...i_{n-1}} = q_{i_1...i_n}$$

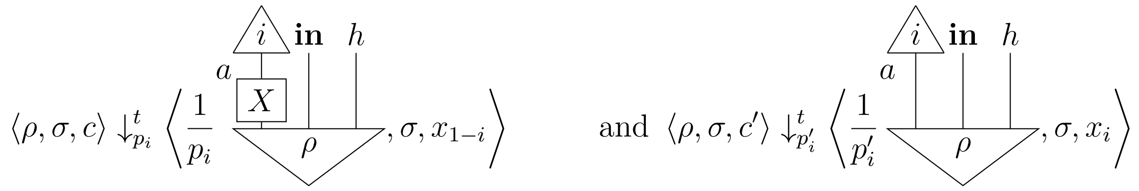
(unless one of the probabilities is 0, in which case we don't need to consider it) the configuration $\langle \rho, \sigma, \mathsf{discard}_n(\vec{a}.x)\rangle$ terminates on state $\left\langle \frac{1}{p_i} \rho_{i_1...i_n}, \sigma, x \right\rangle$ as required. □

Proposition 4.3.24 (Satisfying equations of $\mathsf{I/O} \otimes \mathsf{QUANTUM}$). *$\simeq_{qu}$ satisfies all equations of $\mathsf{I/O} \otimes \mathsf{QUANTUM}$.*

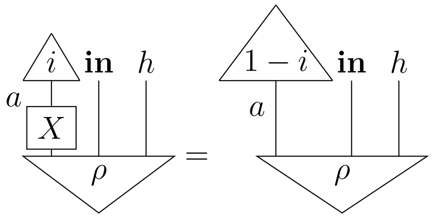
Proof. We simply show that $\simeq_{qu}$ satisfies all equations by showing the reduction sequences. For convenience, we write c for the LHS and c' for the RHS of the equations.

Equation (A): $c = \mathsf{apply}_X(a, a.\mathsf{measure}(a, x_1, x_0))$ and $c' = \mathsf{measure}(a, x_0, x_1)$.

Then we have for arbitrary $\rho :_t (a), \sigma$ we get


$\langle \rho, \sigma, c\rangle \downarrow^t_{p_i}$
$\frac{1}{p_i}$
i
in
h
a
X
ρ
$, \sigma, x_{1-i}$
and $\langle \rho, \sigma, c'\rangle \downarrow^t_{p'_i}$
$\frac{1}{p'_i}$
i
in
h
a
ρ
$, \sigma, x_i$

for $i = 0, 1$, whereby because



this actually gives the same reduction sequences with the same probability ($p_{1-i} = p'_i$). Thus, $c \simeq^{\mathsf{skip}}_{pr} c'$ holds, which implies that $c \simeq_{qu} c'$.

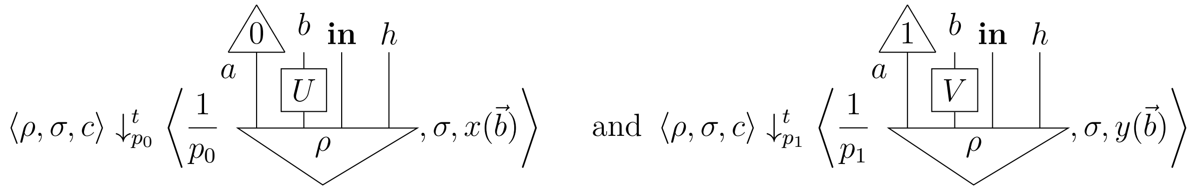
Equation (B): here we have

$$c = \mathsf{measure}(a, \mathsf{apply}_U(\vec{b}, \vec{b}.x(\vec{b}), \mathsf{apply}_V(\vec{b}, \vec{b}.y(\vec{b}))))$$

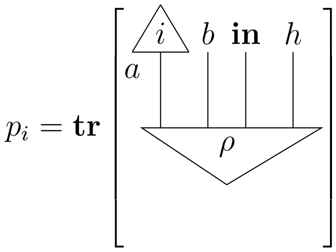
and

$$c' = \mathsf{apply}_{D(U,V)}((a, \vec{b}), (a, \vec{b}).\mathsf{measure}(a, x(\vec{b}), y(\vec{b}))).$$

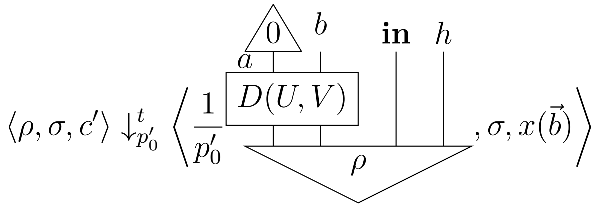
First, let's look at c.



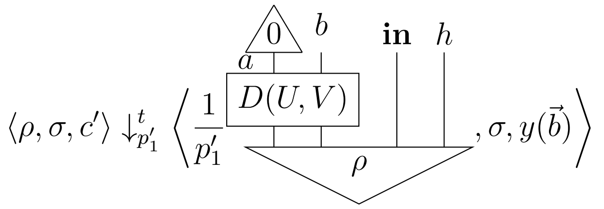
where



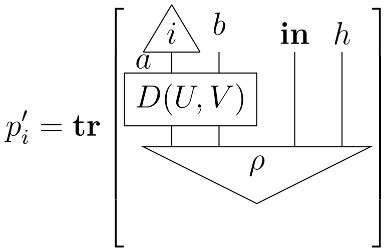
On the other hand, we have that


$\langle \rho, \sigma, c' \rangle \downarrow^t_{p'_0} \left\langle \frac{1}{p'_0} \right.$
0
b
a
$D(U,V)$
in h
ρ
$\left. , \sigma, x(\vec{b}) \right\rangle$

and

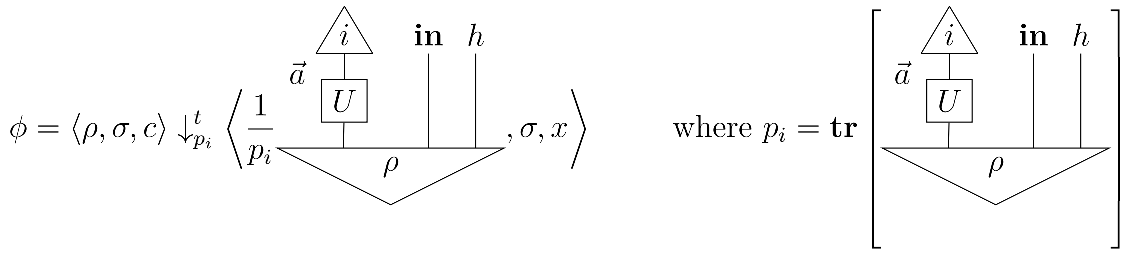

$\langle \rho, \sigma, c' \rangle \downarrow^t_{p'_1} \left\langle \frac{1}{p'_1} \right.$
0
b
a
$D(U,V)$
in h
ρ
$\left. , \sigma, y(\vec{b}) \right\rangle$

where

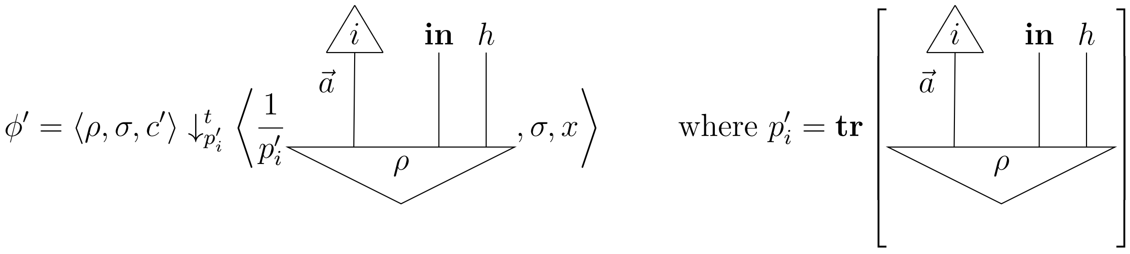


It is easy to see that along both branches, we get the same state, by definition of controlled gates $D(U,V)$. Moreover, $p_i = p'_i$ because U and V are unitaries, thus trace-preserving.

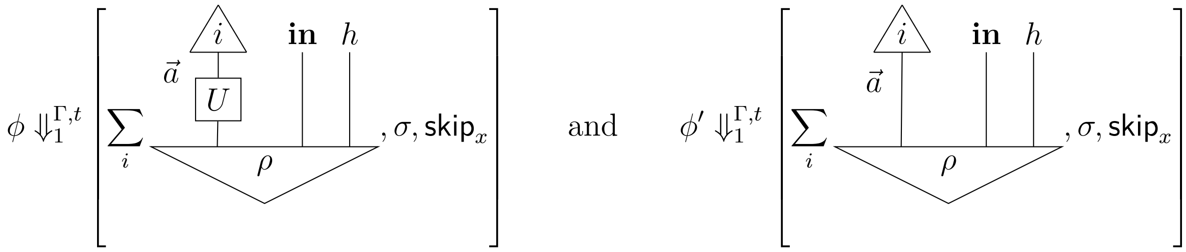
Equation (C): $c = \mathsf{apply}_U(\vec{a}, \vec{a}.\mathsf{discard}_n(\vec{a}, x))$ and $c' = \mathsf{discard}_n(\vec{a}, x)$. First, let $\rho :_t (\vec{a}), \sigma$, we know by lemma 4.3.23 that


$\phi = \langle \rho, \sigma, c \rangle \downarrow^t_{p_i} \left\langle \frac{1}{p_i} \right.$
i
$\vec{a}$
U
in h
ρ
$\left. , \sigma, x \right\rangle$
where $p_i = \mathbf{tr}$
i
$\vec{a}$
U
in h
ρ

and

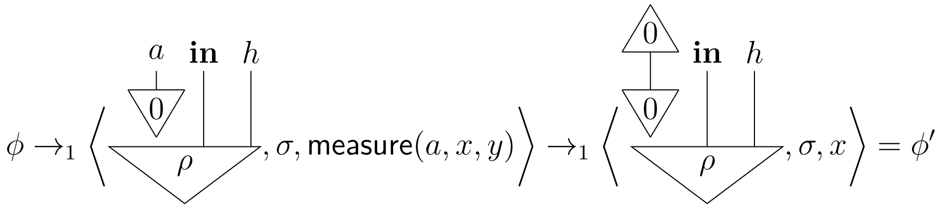

$\phi' = \langle \rho, \sigma, c' \rangle \downarrow^t_{p'_i} \left\langle \frac{1}{p'_i} \right.$
i
$\vec{a}$
in h
ρ
$\left. , \sigma, x \right\rangle$
where $p'_i = \mathbf{tr}$
i
$\vec{a}$
in h
ρ

for $i = 0 \ldots n-1$. Now because $\sum_i p_i = \sum_i p'_i = 1$, it follows that



and by recognising the sum over the projection operators $\langle i| - |i\rangle$ as a trace operator, we can see that the two states are equal because unitaries are trace-preserving. Hence $c \simeq_{qu} c'$.

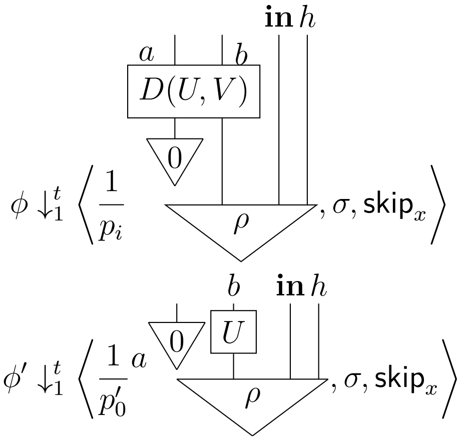
Equation (D): $c = \mathsf{new}(a.\mathsf{measure}(a, x, y))$, $c' = x$.



Equation (E): we have

$$c = \mathsf{new}(a.\mathsf{apply}_{D(U,V)}((a, \vec{b}), (a, \vec{b}).x(a, \vec{b})))$$

$$c' = \mathsf{apply}_U(\vec{b}, \vec{b}.\mathsf{new}(a.x(a, \vec{b})))$$



which are equal states by definition of $D(U, V)$.

Equations (F)-(I): These are properties of unitaries which are easily checked because of the definition of our operational semantics.

Equations (J)-(L): These are commutative equations which follow from the fact that the category **CP** in which we're taking the state is monoidal.

Equations (U)-(Z): Likewise, these can be shown to follow from the fact that **CP** is monoidal. Here, we show the proof for (W) and (X).

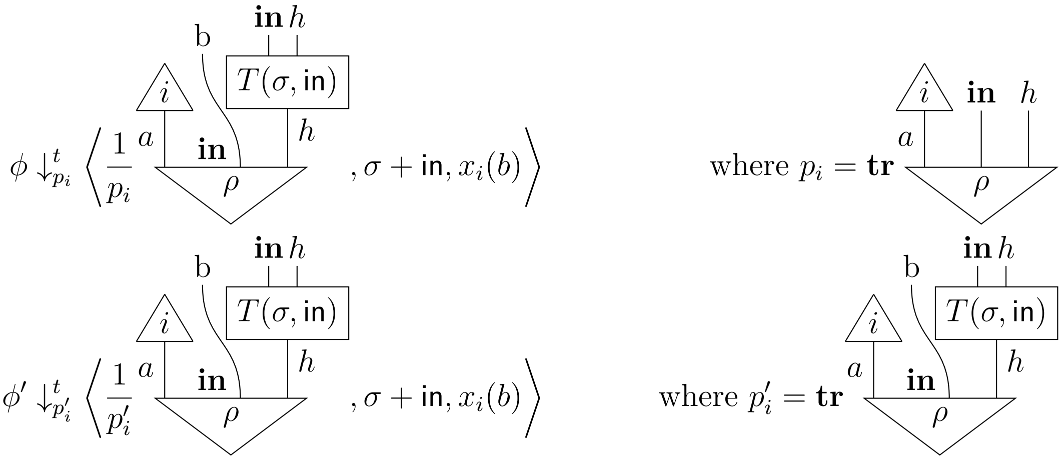
Equation (W): We have

$$c = \mathsf{measure}(a, \mathsf{in}(b.x_0(b)), \mathsf{in}(b.x_1(b)))$$

and

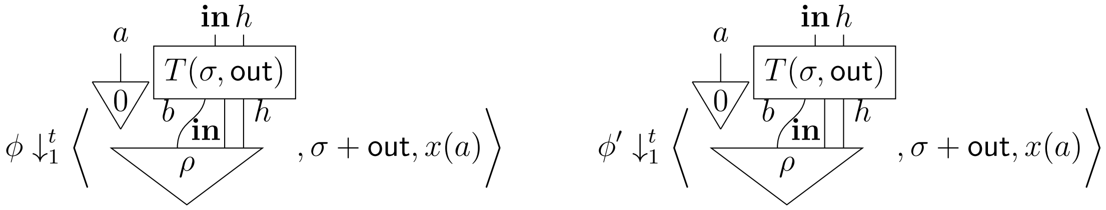
$$c' = \mathsf{in}(b.\mathsf{measure}(a, x_0(b), x_1(b)))$$

.



So it suffices to show that $p_i = p'_i$, which follows from the fact that $T(\sigma, \mathsf{in})$ is in **CPTP**, hence trace-preserving.

Equation (X): $c = \mathsf{new}(a.\mathsf{out}(b, x(a)))$, $c' = \mathsf{out}(b, \mathsf{new}(a.x(a)))$.


$\phi \downarrow^t_1 \Big\langle$
a
0
in h
$T(\sigma, \mathsf{out})$
b
in
h
ρ
$, \sigma + \mathsf{out}, x(a) \Big\rangle$
$\phi' \downarrow^t_1 \Big\langle$
a
0
in h
$T(\sigma, \mathsf{out})$
b
in
h
ρ
$, \sigma + \mathsf{out}, x(a) \Big\rangle$

so we are done. □

Theorem 4.3.25 (Model)**.** *Terms modulo quantum equivalence is a model of* $\mathsf{I/O} \otimes \mathsf{QUANTUM}$.

Proof. We take the syntactic category exactly as in definition 3.2.7. As our terms are exactly those of a Bij PAT, we need only show our notion of equivalence $\simeq_{qu}$ satisfies all the axioms (proposition 4.3.24), be compatible with substitution (proposition 4.3.22) and α-equivalence (proposition 4.3.10), be a congruence (proposition 4.3.9), and be reflexive (follows from congruence) and transitive (easy to verify). Thus, we can define the canonical syntactic **Bij** action on those terms, making our terms modulo quantum equivalence a non-trivial model of $\mathsf{I/O} \otimes \mathsf{QUANTUM}$. □

5
A Monadic Denotational Semantics

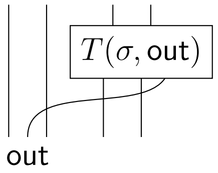
The previous chapter has seen an ad hoc operational semantics as a model of our algebraic theory, i.e. an implementation of our notion of computation. It nicely demonstrates the model of computation and the program equivalence that we wished to axiomatise. In this chapter, we present a complementary model of $\mathsf{I/O} \otimes \mathsf{QUANTUM}$ based on an elegant monadic denotational semantics. We do so by building on the insights from section 4.3.5 (section 5.1). Then, in section 5.2, we draw knowledge from existing literature on modelling the $\mathsf{I/O}$ theory as a monad and explicitly construct one in $\mathbf{CP}^{\infty}$, which is $\mathbf{CP}$ freely extended with infinite biproducts [52]. In section 5.3, we show that our denotational semantics is adequate (theorem 5.3.4) and fully abstract (theorem 5.3.6) for the operational semantics. Then, in section 5.4, we use the previous results to deduce that our monadic semantics give a model of $\mathsf{I/O} \otimes \mathsf{QUANTUM}$ (theorem 5.4.1, corollary 5.4.2). Finally, still in section 5.4, we briefly discuss the possibility of our monad being additionally fully complete for $\mathsf{I/O} \otimes \mathsf{QUANTUM}$ (conjecture 5.4.3).

Convention 5.0.1. *In this chapter, to avoid confusion due to overloaded Greek letters, we uniformly write* proj *and* inj *to denote projection and injection morphisms for biproducts.*

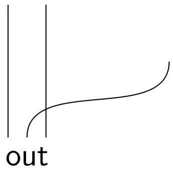
5.1 Motivation

In section 4.3.5, we have defined a notion of abstract reduction paths $P \in \mathsf{Path}\,(c)$, and a way of interpreting these paths concretely given a concrete stream $[\![P]\!]^{t,\sigma} : [\![\Delta]\!] \otimes \mathbf{qbit} \otimes q(\sigma) \to [\![n_P]\!] \otimes \mathbf{qbit} \otimes q(\sigma + \sigma(P))$. One thing we could have noted is that given a path P, the map $[\![P]\!]^{t,\sigma}$ is constructed in exactly the same way for every concrete stream. Is it possible to potentially abstract away from concrete streams?

Perhaps there is. In constructing P, it is easy to see that the stream is only really needed when we are performing an input or an output (fig. 4.6). Consider $[\![\mathsf{out}(q,c) \to P]\!]^{t,\sigma}$, which simply performs the following map of type $[\![\Delta]\!]\otimes\mathbf{qbit}\otimes[\![\Delta']\!]\otimes\mathbf{qbit}\otimes q(\sigma) \to [\![\Delta]\!]\otimes[\![\Delta']\!]\otimes\mathbf{qbit}\otimes q(\sigma{+}\mathsf{out})$ before applying the rest of the path:

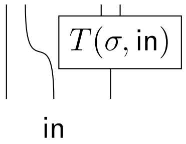


If we ignore the stream, we get the following map of type $[\![\Delta]\!]\otimes\mathbf{qbit}\otimes[\![\Delta']\!] \to [\![\Delta]\!]\otimes[\![\Delta']\!]\otimes\mathbf{qbit}^{\mathsf{out}}$.

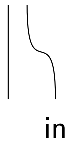


Perhaps this map does not do much, but if we write out the entire inductive definition, we would be applying the continuation on $[\![\Delta]\!] \otimes [\![\Delta']\!]$ only. In other words, the dangling qubit corresponds to an *open output*.

Likewise, consider $[\![\mathsf{in}(q.c) \to P]\!]^{t,\sigma}$, which performs the following map of type $[\![\Delta]\!]\otimes\mathbf{qbit}\otimes q(\sigma) \to [\![\Delta]\!] \otimes \mathbf{qbit} \otimes \mathbf{qbit} \otimes q(\sigma + \mathsf{in})$ before applying the continuation.



If we also ignore the stream here, we get the following map of type $[\![\Delta]\!] \otimes \mathbf{qbit} \to [\![\Delta]\!] \otimes \mathbf{qbit}$.



Knowing that the previous map would have had as codomain $[\![\Delta]\!]$, we have simply added a qubit wire as *open input.*

Here comes the first insight and the first difficulty: we can ignore the stream's evolution and accumulate open input and output wires, but at first glance, it is unclear how we can make sure that we also accumulate a causal ordering of the inputs and outputs based on the trace of that path $\sigma(P)$.

The second difficulty comes from the proof of proposition 4.3.22. Recall that the notion of program equivalence we want to model does not compare the results of individual reduction paths P. Instead, it classifies them by the trace they generate $\sigma(P)$ as well as the continuation x_P that they end up calling. Therefore, in a denotational semantics, not only do we need to record the result for all the paths, we would also want a classification by trace σ and continuation variable x to happen.

5.2 Constructing the Quantum I/O Monad

In search of a solution, we turn to existing literature on the $\mathsf{I/O}$ theory, its corresponding monad, and the relevant constructions.

5.2.1 The Free I/O Monad and the Resumptions Monad

Hyland, Plotkin and Power [31] studied the general $\mathsf{I/O}$ theory (with operations $\mathsf{in} : (\mathsf{unit} \mid I)$ and $\mathsf{out} : (O \mid \mathsf{unit})$, I and O are sets, and no equations). They gave the corresponding monad $T_{\mathsf{I/O}} : \mathbf{Set} \to \mathbf{Set}$ as a 'free' monad, defined as follows. Let $F : \mathbf{Set} \to \mathbf{Set}$ be such that $F(Y) = (I \to Y) + (O \times Y)$, and let $F_X : \mathbf{Set} \to \mathbf{Set}$ be the functor $F_X(Y) = F(Y) + X$. Then, the free I/O monad, if it exists, is defined as $T_{\mathsf{I/O}}(X) \triangleq \mu F_X$, where μF_X is the (countable directed) colimit of the following diagram:

$$0 \xrightarrow{[]_{F0}} F0 \xrightarrow{F([]_{F0})} FF0 \xrightarrow{FF([]_{F0})} FFF0 \qquad \ldots$$

where $[]_A : 0 \to A$ is the unique function from the empty set to A. In $\mathbf{Set}$, this monad does exist and is exactly the monad generated by the general $\mathsf{I/O}$ theory briefly presented above. We often write it using a least fixed point notation as $T_{\mathsf{I/O}}X = \mu Y.I \to Y + O \times Y + X$.

In the same work, the authors also explore combining the general $\mathsf{I/O}$ theory with other theories. However, when combining $\mathsf{I/O}$ with another theory $\mathbb{L}$, they always opt for the *sum* $\mathsf{I/O}+\mathbb{L}$ rather than the tensor. The monad they then obtain, $T_{\mathsf{I/O}+\mathbb{L}} : \mathbf{Set} \to \mathbf{Set} = \mu Y.T_{\mathbb{L}}(I \to Y + O \times Y + X)$, is a special case of the widely studied 'resumptions' monad transformer [15, 31], building the sum of a monad T and the free monad of an endofunctor F as $T_{\text{res}}(X) = \mu Y.T(F(Y) + X)$. Intuitively, this monad allows one to interleave effects from $\mathbb{L}$ with effects from $\mathsf{I/O}$, i.e. doing something from $\mathbb{L}$, then performing an I/O operation, and then doing something else from $\mathbb{L}$, and so on.

In this work, however, we chose to study the tensor $\mathsf{I/O} \otimes \mathsf{QUANTUM}$, which needs to satisfy additional commutativity equations. While quotienting the sum monad could potentially give what we wanted, the result will not be an explicit characterisation of our notion of computation, which makes it unsatisfactory. Instead, our strategy is informally as follows. We recall the commonly cited identity on tensors of Lawvere theories (e.g. [32]):

$$\mathsf{Mod}(\mathbb{L} \otimes \mathbb{L}', \mathcal{C}) \simeq \mathsf{Mod}(\mathbb{L}, \mathsf{Mod}(\mathbb{L}', \mathcal{C}))$$

where $\mathsf{Mod}(\mathbb{L}, \mathcal{C})$ is the category of models of the Lawevere theory $\mathbb{L}$ in the category $\mathcal{C}$. While we have not proven this equivalence for our notion of parameterised algebraic theory, we conjecture it to be true. Furthermore, recall that Staton has given a fully complete model of $\mathsf{QUANTUM}$ in $\mathbf{CPU} \cong \mathbf{CPTP}^{op} \subset \mathbf{CP}$ [67]. Therefore, we will attempt to construct the free $\mathsf{I/O}$ monad directly in $\mathbf{CP}$.

Recalling **CP**'s compact closed structure, let $F_{\mathsf{I/O}} : \mathbf{CP} \to \mathbf{CP}$ be defined as

$$Y \mapsto [I, Y] \oplus (O \otimes Y) = (I \otimes Y) \oplus (O \otimes Y) = (\mathbf{qbit}^{\mathsf{in}} \otimes Y) \oplus (\mathbf{qbit}^{\mathsf{out}} \otimes Y).$$

Note that the distinction between $\mathbf{qbit}^{\mathsf{in}}$ and $\mathbf{qbit}^{\mathsf{out}}$ is purely for our convenience and has no mathematical meaning. Then, an I/O monad would be defined as

$$T(X) = \mu Y.(\mathbf{qbit}^{\mathsf{in}} \otimes Y) \oplus (\mathbf{qbit}^{\mathsf{out}} \otimes Y) \oplus X$$

which, if we unroll the fixed point, gives

$$\begin{aligned} T(X) &= X \oplus (\mathbf{qbit}^{\mathsf{in}} \otimes X) \oplus (\mathbf{qbit}^{\mathsf{out}} \otimes X) \\ &\quad \oplus (\mathbf{qbit}^{\mathsf{in}} \otimes \mathbf{qbit}^{\mathsf{in}} \otimes X) \oplus (\mathbf{qbit}^{\mathsf{in}} \otimes \mathbf{qbit}^{\mathsf{out}} \otimes X) \\ &\quad \oplus (\mathbf{qbit}^{\mathsf{out}} \otimes \mathbf{qbit}^{\mathsf{in}} \otimes X) \oplus (\mathbf{qbit}^{\mathsf{out}} \otimes \mathbf{qbit}^{\mathsf{in}} \otimes X) \oplus ... \\ &= \bigoplus_{\sigma \in \{\mathsf{in},\mathsf{out}\}^*} [\mathsf{Aux}(\sigma) \otimes X] \end{aligned}$$

where Aux is inductively defined by

$$\begin{aligned} \mathsf{Aux}(\epsilon) &= \mathbb{C} \\ \mathsf{Aux}(\mathsf{in} + \sigma) &= \mathbf{qbit}^{\mathsf{in}} \otimes \mathsf{Aux}(\sigma) \\ \mathsf{Aux}(\mathsf{out} + \sigma) &= \mathbf{qbit}^{\mathsf{out}} \otimes \mathsf{Aux}(\sigma) \end{aligned}$$

This definition is quite ideal. The first difficulty is solved because we will know exactly the causal ordering of the inputs and outputs given the index of the corresponding object in the biproduct. The second requirement is also met. If we are able to interpret terms $\Gamma \mid \Delta \vdash c$ as morphisms in the opposite of the Kleisli category Kl_T, then we would have $[\![c]\!] : [\![\Delta]\!] \to T([\![\Gamma]\!])$, where $T([\![\Gamma]\!])$ can be explicitly written as

$$T([\![\Gamma]\!]) = \bigoplus_{\sigma} \mathsf{Aux}(\sigma) \otimes [\![\Gamma]\!] = \bigoplus_{\sigma} \mathsf{Aux}(\sigma) \otimes (\bigoplus_{x:n \in \Gamma} [\![n]\!]) \cong \bigoplus_{\sigma, x \in \Gamma} \mathsf{Aux}(\sigma) \otimes [\![n_x]\!]$$

(where we denote by n_x the natural number such that $(x : n_x) \in \Gamma$). Intuitively, this means that c is expected to be translated to a map which takes a state of $|\Delta|$ qubits as input and gives a list of possible resulting states (with $\mathsf{Aux}(\sigma)$ open wires and n_x qubit outputs), indexed by the I/O trace σ and the continuation that was called x. Recalling the discussion from the previous section, this happens to be exactly what we need to resolve the second difficulty.

However, unfortunately, this monad is *not* well-defined because **CP** does not have infinite biproducts. Luckily, as we shall sketch in the next section, there is a way of constructing a version of **CP** with infinite biproducts.

5.2.2 Infinite Biproducts for CP

In this section, we will sketch the construction of $\mathbf{CP}^\infty$, a category of infinite biproducts of matrix algebras $\mathcal{M}_n(\mathbb{C})$ and CP maps between them, exactly following Pagani, Selinger and Valiron [52]. The reader may safely skip this section and assume the existence of such a category, for its construction is irrelevant to the subsequent proofs.

As in section 2.3.3 where we construct **CP** as a finite biproduct completion, our construction starts with a category **CPM**, where objects are natural numbers n and morphisms $n \to k$ are CP maps $\mathcal{M}_n(\mathbb{C}) \to \mathcal{M}_k(\mathbb{C})$. Recall that we can freely complete **CPM** with finite biproducts only because finite 'superpositions' (sums) of morphisms $\sum_{i \in I} f_i$ (where I is finite, and $f_i : n \to m$ for all i) exist in **CPM**, canonically defined by addition. To obtain infinite biproducts, however, we require the existence of infinite sums, i.e. sums of the form $\sum_{i \in C} f_i$ where C is infinite. Because these are not well defined in **CPM**, our first step will be to fix this.

Completing CPM with dcpo enrichment under the Löwner order We briefly describe the construction of $\overline{\mathbf{CPM}}$ exactly following [52]. First, every homset $\mathbf{CPM}(n,m)$ is partially ordered by the Löwner order $\sqsubseteq$, defined as follows: $A \sqsupseteq B$ iff $A - B$ is positive semi-definite (or equivalently a positive element of the C* algebra) [63]. In fact, $(\mathbf{CPM}(n,m), \sqsubseteq)$ is, in addition, a bounded directed complete partial order (bdcpo). This means that it has a minimum element (the 0 map) and any subset D that is bounded (there is an $f \in \mathbf{CPM}(n,m)$ such that for all $g \in D$, $g \sqsubseteq f$) and directed (for any $f, g \in D$, there exists an $h \in D$ such that $f \sqsubseteq h$ and $g \sqsubseteq h$) has a least upper bound $\bigsqcup D$.

As mentioned by [52], $\mathbf{CPM}(n,m)$ is unfortunately not quite a *dcpo* (directed complete partial order), because there does exist unbounded directed subsets, and they do not have least upper bounds. However, if it were a dcpo, we would be able to define infinite sums. Let C be an infinite set and $f_i \in \mathbf{CPM}(n,m)$ for all $i \in C$. Their infinite sum can be defined as

$$\sum_{i \in C} f_i = \bigsqcup_{S \subseteq_{\text{fin}} C} \sum_{i \in S} f_i = \bigsqcup D$$

where $D = \{\sum_{i \in S} f_i \mid S \subseteq C \wedge S \text{ finite}\}$ is a indeed directed subset of $\mathbf{CPM}(n,m)$ under the Löwner order.

To complete $\mathbf{CPM}(n,m)$ to a dcpo, Pagani, Selinger and Valiron propose to use the D-completion of Zhao and Fan [74]. The construction itself is sketched in [52]. Instead, we content ourselves with some of its properties. Given a partially ordered set (poset) P, its D-completion $c(P)$ has the following universal property: there is a canonical injection map $\iota : P \to c(P)$ which is Scott-continuous (i.e. preserving existing least upper bounds), and for every other dcpo Q and

Scott continuous map $f : P \to Q$, there is a unique Scott-continuous $g : c(P) \to Q$ such that the following commutes.

$$\begin{tikzcd}
P \arrow[r, "\iota"] \arrow[dr, "f"'] & c(P) \arrow[d, dashed, "\exists! g"] \\
 & Q
\end{tikzcd}$$

In particular, if P is a bdcpo, then all the least upper bounds of the bounded directed subsets will be preserved, and the only additional elements contained by $c(P)$ are the elements ‘at infinity’.

Finally, we can define the D-completed dcpo-enriched category $\overline{\mathbf{CPM}}$ as the category with the same objects as $\mathbf{CPM}$, morphisms $\overline{\mathbf{CPM}}(n, m) = c(\mathbf{CPM}(n, m))$, and all categorical constructions are extended using the universal property of the D-completion. The definition of the infinite indexed sum above is one example.

Completing $\overline{\mathbf{CPM}}$ with infinite biproducts Now that we have infinite sums, we can construct the infinite biproduct completion, $\mathbf{CP}^\infty$. Its objects are indexed families of natural numbers N, i.e. an index set I_N along with a natural number d^N_i for each $i \in I_N$. Morphisms $\phi : N \to M$ are exactly the matrices $(\phi_{a,b})_{a \in I_N, b \in I_M}$, where $\phi_{a,b} \in \overline{\mathbf{CPM}}(d^N_a, d^M_b)$. Composition is then simply matrix composition, and identity is the diagonal matrix $(\mathbf{id}_N)_{a,a'} = \delta_{a,a'} \cdot \mathbf{id}_{d^N_a}$.
Remark 5.2.1. *It has been pointed out that a dcpo completion following Pagani et al. [52] is ‘too powerful’ in a sense, for it allows for all infinite sums, and thus all infinite biproducts. However, in our use case, we only need countable biproducts to express a biproduct indexed by* $\{\mathsf{in}, \mathsf{out}\}^*$. *Thus, it is possible that an ω-cpo completion of the homsets along with a countable biproduct completion – if well-behaved – would be more suitable. Noteworthy is that the rest of the proof does not depend on the specific construction, which means that the reader can safely ignore this subtlety.*

$\mathbf{CP}^\infty$ is additionally a biproduct compact closed category.

- Biproducts: Let S be a possibly infinite set and $\{N^{(i)}\}_{i \in S}$ be a family of objects in $\mathbf{CP}^\infty$. Then, the biproduct $P = \bigoplus_{i \in S} N^{(i)}$ is defined by an index set $I_P = \bigcup_{i \in S} \{i\} \times I_{N^{(i)}}$, and $d^P_{(i,j)} = d^{N^{(i)}}_j$. The projections proj^j and injections inj^j ($j \in I$) are defined by $\mathsf{proj}^j_{(i,a),a'} = \mathsf{inj}^j_{a',(i,a)} = \delta_{j,i}\delta_{a,a'}\mathbf{id}$. The zero object can be defined by the empty index set (the empty biproduct).
- Symmetric monoidal structure: The bifunctor $\otimes : \mathbf{CP}^\infty \times \mathbf{CP}^\infty \to \mathbf{CP}^\infty$ can be defined simply by inheriting the tensor product from matrix algebras and making it distribute over the free biproduct. Concretely, $N \otimes M$ is defined by $I_{N \otimes M} = I_N \times I_M$ and $d^{N \otimes M}_{(a,b)} = d^N_a \cdot d^M_b$. On morphisms, we define tensors component-wise, using the tensor product from $\overline{\mathbf{CPM}}$, which is the tensor in $\mathbf{CPM}$ made compatible with the infinite elements from the D-completion. The tensor unit is the object $\mathbb{C}$ (with $I_{\mathbb{C}} = \{*\}$ and $d^{\mathbb{C}}_* = 1$). Finally, the

associativity, unit and symmetry natural isomorphisms are inherited from **CPM** and defined component-wise.

- Compact closure: as we have a skeleton category, every object coincides with its dual: $N^* = N$. The unit and counit are defined component-wise.

Finally, we define a few useful objects.

- The qubit type **qbit** will correspond to the object **qbit** such that $I_{\mathbf{qbit}} = \{*\}$ and $d_*^{\mathbf{qbit}} = 2$. We write $\mathbf{qbit}^{\otimes n}$ to express the n-way tensor product of **qbit**, which corresponds to the object such that $I_{\mathbf{qbit}^{\otimes n}} = \{*\}$ and $d_*^{\mathbf{qbit}^{\otimes n}} = 2^n$.
- The bit type **bit** will correspond to the object **bit** such that $I_{\mathbf{bit}} = \{0, 1\}$ and $d_0^{\mathbf{bit}} = d_1^{\mathbf{qbit}} = 1$. It is isomorphic to $\mathbb{C} \oplus \mathbb{C}$. We write $\mathbf{bit}^{\otimes n}$ to express the n-way tensor product of **bit**, which corresponds to the object such that $I_{\mathbf{bit}^{\otimes n}} = [2^n]$ and $d_i^{\mathbf{bit}^{\otimes n}} = 1$.

5.2.3 Explicit Definition in $\mathbf{CP}^\infty$ and Properties

Now, we can proceed to explicitly define our monad $T : \mathbf{CP}^\infty \to \mathbf{CP}^\infty$.

Definition 5.2.2 (Quantum I/O monad)**.** *Let $T : \mathbf{CP}^\infty \to \mathbf{CP}^\infty$ be the functor defined by*

$$T(X) = \bigoplus_{\sigma \in \{\mathsf{in}, \mathsf{out}\}^*} \mathsf{Aux}(\sigma) \otimes X.$$

It has the structure of a monad with $\eta_X : X \to T(X) \triangleq \mathsf{inj}^\epsilon$ and $\mu_X : TTX \to TX$ is defined by

$$(\mu_X)_{(\sigma,\sigma'),\sigma''} \triangleq \delta_{(\sigma+\sigma'),\sigma''}\mathbf{id}.$$

We can easily check that this forms a monad.

Remark 5.2.3. *Rather deceptive is the fact that when working with this monad, both the input and the output qubits appear as output wires. This is very useful because, this way, we can determine the order in which inputs and outputs happen canonically by their index in the biproduct. This does also mean that if we want to use the accumulated map, we would need to bend all the input wires down using the cup (i.e. the evaluation map in the symmetric monoidal closed structure of $\mathbf{CP}^\infty$).*

We can further prove the following interesting property:

Lemma 5.2.4. *The following commutes for any $\sigma \in \{\mathsf{in}, \mathsf{out}\}^*$:*

$$\begin{tikzcd}
\mathsf{Aux}(\sigma) \otimes X \arrow[r, "\mathsf{inj}^\sigma"] \arrow[d, "\mathsf{Aux}(\sigma)\otimes f"'] & T(X) \arrow[d, "f^\dagger"] \\
\mathsf{Aux}(\sigma) \otimes T(Y) \arrow[r, "\mu_{(\sigma,-),-}"'] & T(Y)
\end{tikzcd}$$

where $(\mu_X)_{(\sigma,-),-} : \mathsf{Aux}(\sigma) \otimes T(X) \to T(X)$ is the map defined by $((\mu_X)_{(\sigma,-),-})_{\sigma',\sigma''} \triangleq (\mu_X)_{(\sigma,\sigma'),\sigma''}$.

Proof. We already know that this commutes when $\sigma = \epsilon$, because we recover the definition of the Kleisli extension $f^\dagger = T(A) \overset{T(f)}{\to} T(T(B)) \overset{\mu}{\to} T(B)$. In general, that this diagram commutes can be derived explicitly. Let $g = \mathbf{id} \otimes f; \mu_{(\sigma,-),-}$, $h = \mathsf{inj}^\sigma; f^\dagger$, then:

$$g_{*,\sigma''} = \sum_{\sigma'} (\mathbf{id} \otimes f)_{*,\sigma'} \cdot \mu_{(\sigma,\sigma'),\sigma''} = \sum_{\sigma'} \delta_{\sigma+\sigma',\sigma''} (\mathbf{id} \otimes f)_{*,\sigma'}$$

and

$$\begin{aligned} h_{*,\sigma''} &= \sum_{\sigma'} \mathsf{inj}^\sigma_{*,\sigma'}; f^\dagger_{\sigma',\sigma''} \\ &= f^\dagger_{\sigma,\sigma''} \\ &= \sum_{\sigma_1,\sigma_2} (T(f))_{(\sigma,*),(\sigma_1,\sigma_2)}; \mu_{(\sigma_1,\sigma_2),\sigma''} \\ &= \sum_{\sigma_1,\sigma_2} \delta_{\sigma,\sigma_1} \cdot (\mathbf{id} \otimes f)_{*,\sigma_2}; \delta_{\sigma_1+\sigma_2,\sigma''} \\ &= \sum_{\sigma_2} \delta_{\sigma+\sigma_2,\sigma''} (\mathbf{id} \otimes f)_{*,\sigma_2}. \end{aligned}$$

Thus $h = g$. Note that we have treated X as a single object instead of an indexed family, which is without loss of generality and simplifies the notation. □

5.2.4 Denotational Semantics

We can now give the terms of $\mathsf{I/O} \otimes \mathsf{QUANTUM}$ a monadic denotational semantics by interpreting it in Kl_T^{op}.

Definition 5.2.5 (Monadic Semantics)**.** *The monadic semantics of terms from* $\mathsf{I/O} \otimes \mathsf{QUANTUM}$ *is given as follows:*

- *A parameter context Δ is interpreted as $[\![\Delta]\!] = \mathbf{qbit}^{\otimes|\Delta|}$,*
- *A computation context $\Gamma = (x_1 : n_1 ... x_k : n_k)$ is interpreted as $[\![\Gamma]\!] = \bigoplus_{i=1}^k [\![n_i]\!]$, where $[\![n]\!] = \mathbf{qbit}^{\otimes n}$.*
- *Every operation $\mathbf{op} : (p \mid m_1 ... m_k)$ is interpreted as $[\![\mathbf{op}]\!] : [\![p]\!] \to T(\bigoplus_{i=1}^k [\![m_i]\!])$, as in fig. 5.1.*
- *Every term $\Gamma \mid \Delta \vdash c$ is interpreted as $[\![c]\!] : [\![\Delta]\!] \to T([\![\Gamma]\!])$ in a standard way.*

Perhaps the first thing to note is how deceptively simple and elegant the semantics appear. We draw the reader's attention to the definition of output and input. For the output, the map is simply identity, but importantly, it no longer appears in the type of the resulting object $T(\mathbb{C})$. In other words, it becomes an open wire hidden under the monad T, and is no longer accessible to the user. For the input, we use the cup to connect the input qubit on the left to the qubit that is accessible to the program on the right. This corresponds exactly to the abstraction map in the compact closed structure (see section 2.3.2.2). As we shall see later, when evaluating

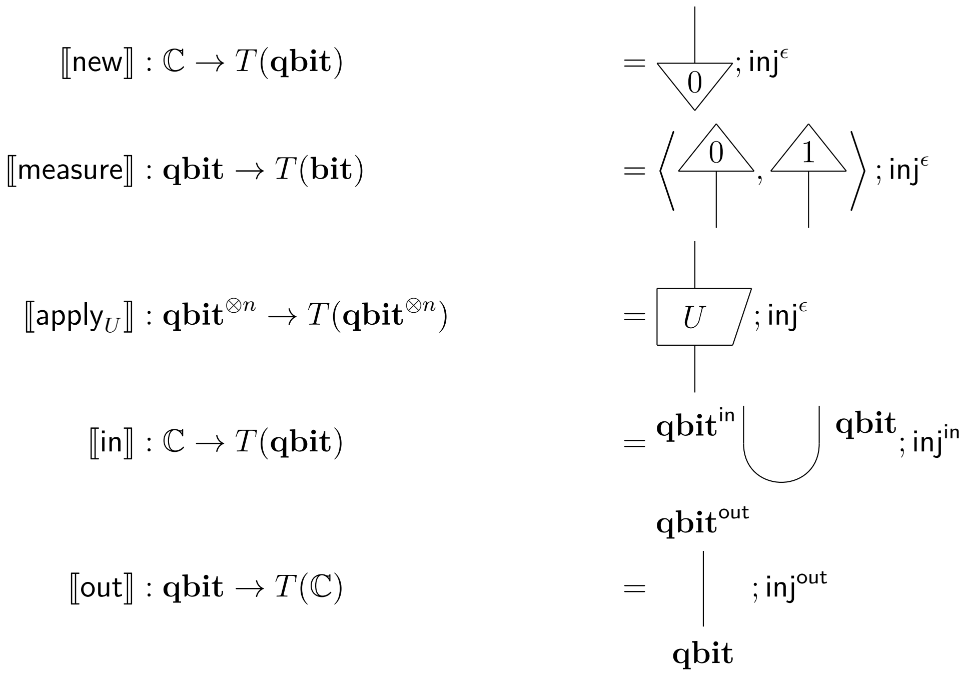

$[\![\mathsf{new}]\!] : \mathbb{C} \to T(\mathbf{qbit})$
$=$
0
; inj^{ϵ}
$[\![\mathsf{measure}]\!] : \mathbf{qbit} \to T(\mathbf{bit})$
$=$
0
1
; inj^{ϵ}
$[\![\mathsf{apply}_U]\!] : \mathbf{qbit}^{\otimes n} \to T(\mathbf{qbit}^{\otimes n})$
$=$
U
; inj^{ϵ}
$[\![\mathsf{in}]\!] : \mathbb{C} \to T(\mathbf{qbit})$
$=$
$\mathbf{qbit}^{\mathsf{in}}$
$\mathbf{qbit}$
; $\mathsf{inj}^{\mathsf{in}}$
$\mathbf{qbit}^{\mathsf{out}}$
$[\![\mathsf{out}]\!] : \mathbf{qbit} \to T(\mathbb{C})$
$=$
; $\mathsf{inj}^{\mathsf{out}}$
$\mathbf{qbit}$

Figure 5.1: Monadic semantics for operations

the denotation by coupling it with a concrete qubit stream, we will use the evaluation map to connect the input wires to the actual inputs.

To understand the full power of these semantics, we explicitly characterise the terms inductively defined by the typing rules. This will result in an induction principle for well-typed terms. Note that we do so based on the new type system in definition 4.2.2 instead of the original one, for it will be more convenient for later proofs.

Induction principle for $[\![c]\!]$ in the new type system. For each rule, we give a commutative diagram, where the blue path gives the standard definition, and the red path gives the induction principle.

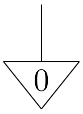
Rule (new):

$$\frac{\Gamma \mid \Delta, q \vdash c}{\Gamma \mid \Delta \vdash \mathsf{new}(q.c)}$$

The morphism $[\![\mathsf{new}(q.c)]\!] : [\![\Delta]\!] \to T([\![\Gamma]\!])$ can be formed according to the following commutative diagram:

$$\begin{tikzcd}
 & & \llbracket \Delta \otimes \mathbf{qbit} \rrbracket \arrow[d, "\eta = \mathsf{inj}^{\epsilon}"'] \arrow[dr, "\llbracket c \rrbracket"] & \\
\llbracket \Delta \rrbracket \arrow[urr, "\llbracket \Delta \rrbracket \otimes \underline{\mathsf{new}}"] \arrow[r, "\llbracket \Delta \rrbracket \otimes \llbracket \mathsf{new} \rrbracket"'] \arrow[rrr, bend right, "\llbracket \mathsf{new}(q.c) \rrbracket"'] & \llbracket \Delta \rrbracket \otimes T(\mathbf{qbit}) \arrow[r, "\cong"'] & T(\llbracket \Delta \rrbracket \otimes \mathbf{qbit}) \arrow[r, dashed, "\llbracket c \rrbracket^{\dagger}"] & T\llbracket \Gamma \rrbracket
\end{tikzcd}$$

where

$\underline{\mathsf{new}} =$ .

The top left triangle commutes by definition, and the right triangle commutes by property of the Kleisli triple.

Rule (apply): Let U be the CP map corresponding to an n-dimensional unitary matrix.

$$\frac{\Gamma \mid \pi(\Delta, \vec{b}) \vdash c}{\Gamma \mid \pi(\Delta, \vec{a}) \vdash \mathsf{apply}_U(\vec{a}, \vec{b}.c)}$$

The morphism $\llbracket \mathsf{apply}_U \rrbracket : \llbracket \pi(\Delta, \vec{a}) \rrbracket \to T(\llbracket \Gamma \rrbracket)$ is defined as follows:

$$\begin{tikzcd}
\llbracket \pi(\Delta, \vec{a}) \rrbracket \arrow[d, "\pi'"] \arrow[rrrd, bend left, "\llbracket \mathsf{apply}_U(\vec{a}, \vec{b}.c) \rrbracket"] & & & \\
\llbracket \pi''(\Delta) \rrbracket \otimes \llbracket \vec{a} \rrbracket \arrow[r, "\mathbf{id}_{\llbracket \pi''(\Delta) \rrbracket} \otimes \llbracket \mathsf{apply}_U \rrbracket"'] \arrow[dd, "\llbracket \pi''(\Delta) \rrbracket \otimes U"'] & \llbracket \pi''(\Delta) \rrbracket \otimes T(\llbracket \vec{b} \rrbracket) \arrow[d, "\cong"'] & T\llbracket \pi(\Delta, \vec{b}) \rrbracket \arrow[r, "\llbracket c \rrbracket^{\dagger}"] & T\llbracket \Gamma \rrbracket \\
 & T(\llbracket \pi''(\Delta) \rrbracket \otimes \llbracket \vec{b} \rrbracket) \arrow[ur, "T((\pi')^{-1})"'] & & \\
\llbracket \pi''(\Delta) \rrbracket \otimes \llbracket \vec{b} \rrbracket \arrow[ur, "\eta = \mathsf{inj}^{\epsilon}"'] \arrow[rrr, "(\pi')^{-1}"'] & & & \llbracket \pi(\Delta, \vec{b}) \rrbracket \arrow[uul, "\eta = \mathsf{inj}^{\epsilon}"] \arrow[uu, "\llbracket c \rrbracket"']
\end{tikzcd}$$

where π' is the permutation on $\pi(\Delta, \vec{a})$ qubits that moves all the qubits from $\vec{a}$ to the right and leaves the rest intact, and π'' is the permutation such that $\pi''(\Delta)$ is exactly $\pi(\Delta, \vec{a})$ with all the qubits from $\vec{a}$ filtered out. Note that any other choice of π' and π'' such that $\pi'(\pi(\Delta, \vec{a})) = (\pi''(\Delta), \vec{a})$ would be equivalent, as remarked under definition 4.2.9.

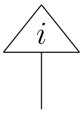
Rule (measure):

$$\frac{(\Gamma \mid \Delta, \Delta' \vdash c_i)_{i=0,1}}{\Gamma \mid \Delta, q, \Delta' \vdash \mathsf{measure}(q, c_0, c_1)}$$

The morphism $\llbracket \mathsf{measure}(q, c_0, c_1) \rrbracket : \llbracket \Delta, q, \Delta' \rrbracket \to T(\Gamma)$ is defined as

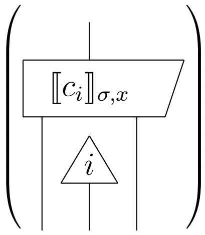
$$\begin{tikzcd}
\llbracket\Delta\rrbracket \otimes \mathbf{bit} \otimes \llbracket\Delta'\rrbracket \arrow[r, "\cong"] & \llbracket\Delta,\Delta'\rrbracket \oplus \llbracket\Delta,\Delta'\rrbracket \arrow[r, "{[\llbracket c_0\rrbracket,\llbracket c_1\rrbracket]}"] \arrow[dd, "\eta=\mathsf{inj}^{\epsilon}"] & T\llbracket\Gamma\rrbracket \\
\llbracket\Delta, q, \Delta'\rrbracket \arrow[u, "\llbracket\Delta\rrbracket\otimes\langle\underline{\mathsf{measure}_0},\underline{\mathsf{measure}_1}\rangle\otimes\llbracket\Delta'\rrbracket"] \arrow[d, "\llbracket\Delta\rrbracket\otimes\llbracket\mathsf{measure}\rrbracket\otimes\llbracket\Delta'\rrbracket"'] & & \\
\llbracket\Delta\rrbracket \otimes T(\mathbf{bit}) \otimes \llbracket\Delta'\rrbracket \arrow[r, "\cong"'] & T(\llbracket\Delta,\Delta'\rrbracket \oplus \llbracket\Delta,\Delta'\rrbracket) \arrow[uur, "{[\llbracket c_0\rrbracket,\llbracket c_1\rrbracket]^{\dagger}}"'] &
\end{tikzcd}$$

where

$\underline{\mathsf{measure}_i} =$ 

for $i = 0, 1$.

If we represent it explicitly, we get exactly the notion of classical control that we would expect.

$\llbracket \mathsf{measure}(q, c_0, c_1) \rrbracket_{\sigma,x} = \sum_{i=0,1}$ 

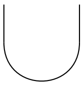
Rule (in):

$$\frac{\Gamma \mid \Delta, q \vdash c}{\Gamma \mid \Delta \vdash \mathsf{in}(q.c)}$$

The morphism $\llbracket \mathsf{in}(q.c) \rrbracket : \llbracket \Delta \rrbracket \to T\llbracket \Gamma \rrbracket$ can be defined as follows:

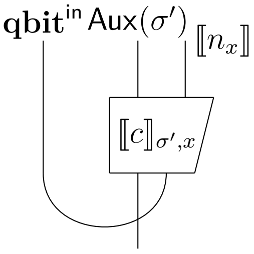
$$\begin{tikzcd}
\llbracket\Delta\rrbracket \arrow[rr, "\llbracket\Delta\rrbracket\otimes\underline{\mathsf{in}}"] \arrow[dr, "\llbracket\Delta\rrbracket\otimes\llbracket\mathsf{in}\rrbracket"] \arrow[dddr, bend right, "\llbracket\mathsf{in}(q.c)\rrbracket"'] & & \llbracket\Delta\rrbracket \otimes \mathbf{qbit}^{\mathsf{in}} \otimes \mathbf{qbit} \arrow[d, "\cong"] \\
& \llbracket\Delta\rrbracket \otimes T(\mathbf{qbit}) \arrow[d, "\cong"] & \mathbf{qbit}^{\mathsf{in}} \otimes \llbracket\Delta\rrbracket \otimes \mathbf{qbit} \arrow[dl, "\mathsf{inj}^{\mathsf{in}}"] \arrow[dd, "\mathbf{qbit}^{\mathsf{in}}\otimes\llbracket c\rrbracket"] \\
& T\llbracket\Delta, q\rrbracket \arrow[d, dashed, "\llbracket c\rrbracket^{\dagger}"] & \\
& T\llbracket\Gamma\rrbracket & \mathbf{qbit}^{\mathsf{in}} \otimes T\llbracket\Gamma\rrbracket \arrow[l, "(\mu_{\llbracket\Gamma\rrbracket})_{(\mathsf{in},-),-}"]
\end{tikzcd}$$

where

$\underline{\mathsf{in}} = \mathbf{qbit}^{\mathsf{in}}$  $\mathbf{qbit}$.

In other words, if we use the matrix notation, we have that

$[\![\mathsf{in}(q.c)]\!]_{\sigma,x} = \Bigg\{$



if $\exists\sigma'.\ \sigma = \mathsf{in} + \sigma'$,

0 otherwise.

Rule (out):

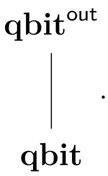
$$\frac{\Gamma \mid \Delta, \Delta' \vdash c}{\Gamma \mid \Delta, q, \Delta' \vdash \mathsf{out}(q, c)}$$

The morphism $[\![\mathsf{out}(q,c)]\!] : [\![\Delta, q, \Delta']\!] \to T[\![\Gamma]\!]$ can then be given as:

$$\begin{tikzcd}
{[\![\Delta]\!] \otimes \mathbf{qbit} \otimes [\![\Delta']\!]} \arrow[rr, "{[\![\Delta]\!]\otimes\underline{\mathsf{out}}\otimes[\![\Delta']\!]}"] \arrow[dr, "{[\![\Delta]\!]\otimes[\![\mathsf{out}]\!]\otimes[\![\Delta']\!]}"] \arrow[ddrd, bend right, "{[\![\mathsf{out}(q,c)]\!]}"'] & & {[\![\Delta]\!] \otimes \mathbf{qbit}^{\mathsf{out}} \otimes [\![\Delta']\!]} \arrow[d, "\cong"] \\
& {[\![\Delta]\!] \otimes T(\mathbb{C}) \otimes [\![\Delta']\!]} \arrow[d, "\cong"] & {\mathbf{qbit}^{\mathsf{out}} \otimes [\![\Delta, \Delta']\!]} \arrow[dl, "\mathsf{inj}^{\mathsf{out}}"'] \arrow[dd, "{\mathbf{qbit}^{\mathsf{out}}\otimes[\![c]\!]}"] \\
& {T[\![\Delta, \Delta']\!]} \arrow[d, "{[\![c]\!]^\dagger}"] & \\
& {T[\![\Gamma]\!]} & {\mathbf{qbit}^{\mathsf{out}} \otimes T[\![\Gamma]\!]} \arrow[l, "{(\mu_{[\![\Gamma]\!]})_{(\mathsf{out},-),-}}"]
\end{tikzcd}$$

where

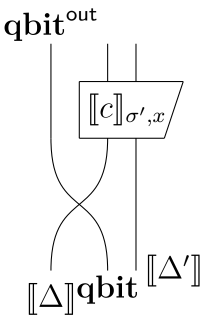
$\underline{\mathsf{out}} =$



.

Explicitly, this can be written as

$[\![\mathsf{out}(q,c)]\!]_{\sigma,x} = \Bigg\{$



if $\exists\sigma'.\ \sigma = \mathsf{out} + \sigma'$,

0 otherwise.

Rule (var): Finally we look at the base case. Assuming that

$$\frac{}{\Gamma, x : n, \Gamma' \mid \pi(\vec{a}) \vdash x(\vec{a})}$$

we can get its semantics $[\![\Gamma, x : n, \Gamma' \mid \pi(\vec{a}) \vdash x(\vec{a})]\!] : [\![n]\!] \to T[\![\Gamma, x : n, \Gamma']\!]$ as

$$\begin{tikzcd}
{[\![n]\!]} \arrow[r, "\pi^{-1}"] \arrow[rrd, "{[\![\Gamma'' \mid \pi(\vec{a}) \vdash x(\vec{a})]\!]}"'] & {[\![n]\!]} \arrow[r, "\mathsf{inj}^x"] & {[\![\Gamma]\!]} \arrow[d, "\eta = \mathsf{inj}^\epsilon"] \\
 & & T[\![\Gamma]\!]
\end{tikzcd}$$

where $\Gamma'' = \Gamma, x : n, \Gamma'$. □

5.3 Results with respect to the Operational Semantics

We are now ready to relate our operational and denotational semantics. To do so, we will first define how to run the program with a concrete stream, i.e. a notion of 'parallel composition' with a concrete stream. Concretely, let $\Gamma \mid \Delta \vdash c$ be an arbitrary program. For every stream $t \in \mathsf{Stream}(X_0)$ where $X_0 \in \mathbf{CP}$ and every initial stream state $\sigma_I \in \{\mathsf{in}, \mathsf{out}\}^*$, we will define a morphism $\mathsf{Run}(c, t, \sigma_I)_{\sigma,x} : [\![\Delta]\!] \otimes \mathbf{qbit} \otimes q(\sigma_I) \to [\![n_x]\!] \otimes \mathbf{qbit} \otimes q(\sigma_I + \sigma)$ for every $\sigma \in \{\mathsf{in}, \mathsf{out}\}^*$, $(x : n_x) \in \Gamma$. Note that if a path P is such that $\sigma(P) = \sigma$ and $x_P = x$, then $[\![P]\!]^{t,\sigma_I}$ will be exactly of that same type!

Further, recall that our monad seems to classify the different reduction paths of a program by trace σ and continuation variable x, and in the final object $T[\![\Gamma]\!]$, all corresponding reduction sequences seem to have their end states added together, weighted by their respective probabilities. It turns out that this is exactly the case, and it is our main result in this section:

Theorem 5.3.1 (Sum of Paths and Runs Coincide)**.** *Let $\Gamma \mid \Delta \vdash c$ be a program. Then, for all $X_0 \in \mathbf{CP}$, stream $t = (q, T) \in \mathsf{Stream}(X_0)$, initial state $\sigma_I \in \{\mathsf{in}, \mathsf{out}\}^*$, path trace $\sigma \in \{\mathsf{in}, \mathsf{out}\}^*$ and computation variable $(x : n_x \in \Gamma)$,*

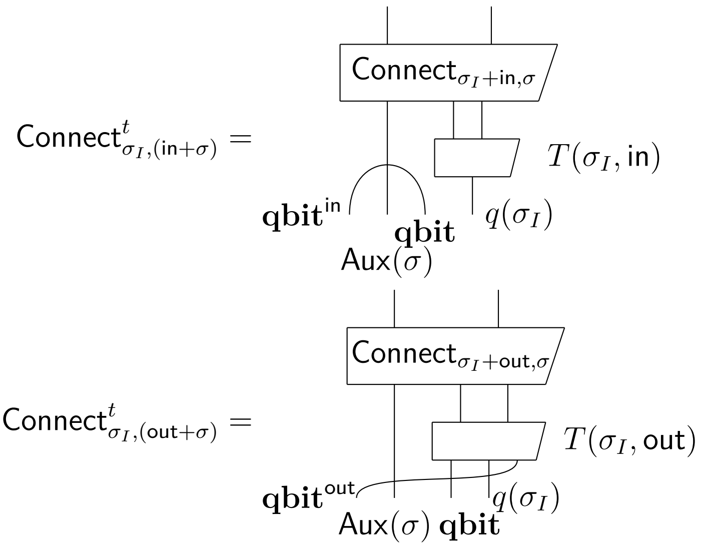
$$\mathsf{Run}(c, t, \sigma_I)_{\sigma,x} = \sum_{\substack{P \in \mathsf{Path}(c) \\ x_P = x \\ \sigma(P) = \sigma}} [\![P]\!]^{t,\sigma_I}.$$

Proof Sketch. Induction over the structure of c, using all the induction principles previously developed. □

And from this result, we can derive adequacy and full abstraction using previous results in section 4.3.5.

5.3.1 Preliminaries

To define Run, we start by defining a gadget $\mathsf{Connect}$, which takes the I/O output $\mathsf{Aux}(\sigma)$ of the component of $T([\![\Gamma]\!])$ at σ, x, and connects them to the concrete stream according to the trace σ.


Connect$^t_{\sigma_I,(\mathsf{in}+\sigma)}$ =
Connect$_{\sigma_I+\mathsf{in},\sigma}$
$T(\sigma_I,\mathsf{in})$
qbit$^{\mathsf{in}}$
qbit
$q(\sigma_I)$
Aux(σ)
Connect$^t_{\sigma_I,(\mathsf{out}+\sigma)}$ =
Connect$_{\sigma_I+\mathsf{out},\sigma}$
$T(\sigma_I,\mathsf{out})$
qbit$^{\mathsf{out}}$
$q(\sigma_I)$
Aux(σ) qbit

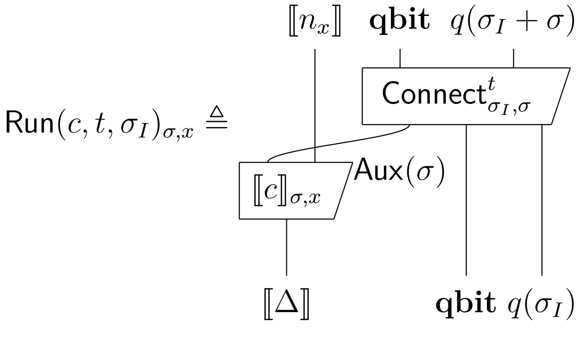
$$\mathsf{Connect}^t_{\sigma_I,\epsilon} = \mathsf{Aux}(\epsilon) \otimes \mathbf{qbit} \otimes q(\sigma_I) \xrightarrow{\cong} \mathbf{qbit} \otimes q(\sigma_I) \qquad \text{(because } \mathsf{Aux}(\epsilon) = \mathbb{C}\text{)}$$

Figure 5.2: Definition of the Connect gadget

Definition 5.3.2 (The Connect gadget). *Let $t = (q, T) \in \mathsf{Stream}(X_0)$ be a stream, $\sigma_I \in$ traceset be an initial stream state, and σ be the trace of the path in question, we define $\mathsf{Connect}^t_{\sigma_I,\sigma}$ as a morphism in $\mathsf{Aux}(\sigma) \otimes \mathbf{qbit} \otimes q(\sigma_I) \to \mathbf{qbit} \otimes q(\sigma_I + \sigma)$ inductively over the structure of σ in fig. 5.2.*

Note that, as promised, when dealing with an input, we are applying the cup to bend it back and feed it with the actual qubit input from the stream.

Definition 5.3.3 (Parallel Composition with Streams using Run). *Let $\Gamma \mid \Delta \vdash c$. Let $t \in \mathsf{Stream}(X_0)$ be a stream, $\sigma_I \in \{\mathsf{in}, \mathsf{out}\}^*$ be an initial state, and further let $\sigma \in \{\mathsf{in}, \mathsf{out}\}^*$ and $(x : n_x) \in \Gamma$. Then, $\mathsf{Run}(c, t, \sigma_I)_{\sigma,x} : [\![\Delta]\!] \otimes \mathbf{qbit} \otimes q(\sigma_I) \to [\![n_x]\!] \otimes \mathbf{qbit} \otimes q(\sigma_I + \sigma)$ is defined as follows:*


$[\![n_x]\!]$ qbit $q(\sigma_I+\sigma)$
Run$(c,t,\sigma_I)_{\sigma,x} \triangleq$
Connect$^t_{\sigma_I,\sigma}$
$[\![c]\!]_{\sigma,x}$
Aux(σ)
$[\![\Delta]\!]$
qbit $q(\sigma_I)$

5.3.2 Proof of Main Result

Proof of theorem 5.3.1. By induction over the structure of c.

Case (var):

$$\frac{}{\Gamma, x : n, \Gamma' \mid \pi(\vec{a}) \vdash x(\vec{a})}$$

We know from the definition of Run and the induction principle of $[\![c]\!]$ that

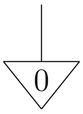
$$\mathsf{Run}(c,t,\sigma_I)_{\sigma,y} = \delta_{\sigma,\epsilon}\delta_{x,y}(\pi^{-1}\otimes \mathbf{id}_{\mathbf{qbit}\otimes q(\sigma_I)}).$$

If we let $P_0 \in \mathsf{Path}\,(c)$ be the unique element of $\mathsf{Path}\,(c)$, we know that $x_{P_0} = x$ and $\sigma(P_0) = \epsilon$, and $[\![P]\!]^{t,\sigma_I} = (\pi^{-1}\otimes \mathbf{id}_{\mathbf{qbit}\otimes q(\sigma_I)})$. Thus $\mathsf{LHS} = \mathsf{RHS}$ follows.

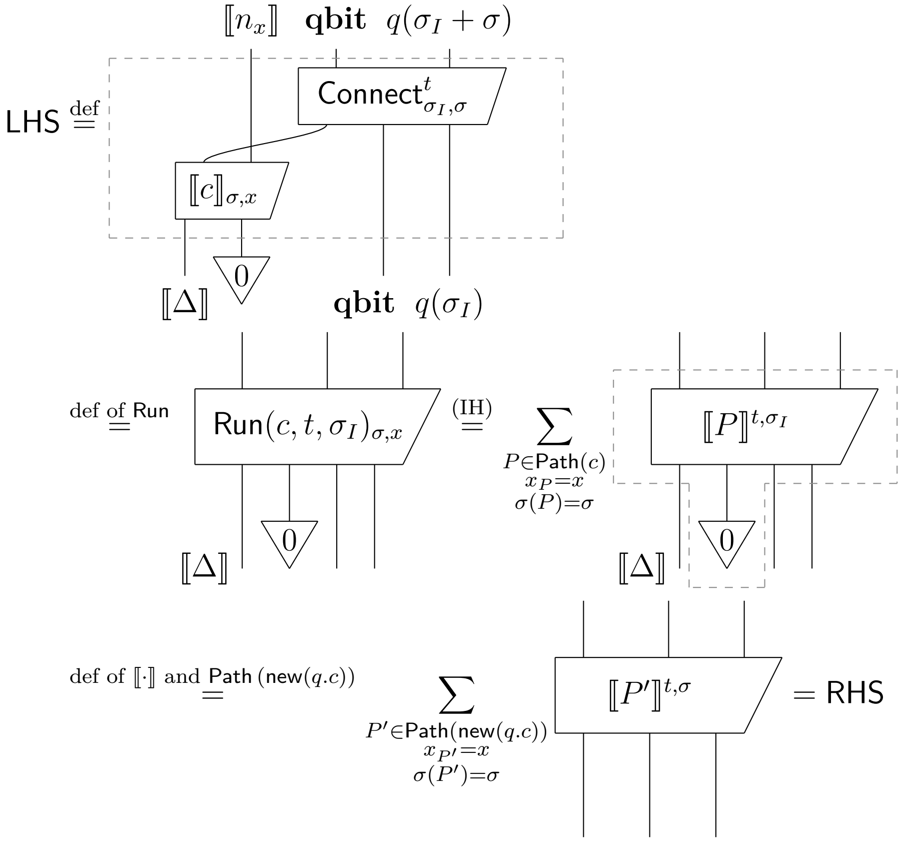
Case (new):

$$\frac{\Gamma \mid \Delta, q \vdash c}{\Gamma \mid \Delta \vdash \mathsf{new}(q.c)}$$

We know that $[\![\mathsf{new}(q.c)]\!]_{\sigma,x}$ is defined by

$$[\![\mathsf{new}(q.c)]\!]_{\sigma,x} = [\![\sigma]\!] \otimes$$  $$; [\![c]\!]_{\sigma,x}$$

and thus, we get



$[\![n_x]\!]$
qbit
$q(\sigma_I+\sigma)$
LHS $\overset{\text{def}}{=}$
$\mathsf{Connect}^t_{\sigma_I,\sigma}$
$[\![c]\!]_{\sigma,x}$
0
$[\![\Delta]\!]$
qbit
$q(\sigma_I)$
$\overset{\text{def of Run}}{=}$
$\mathsf{Run}(c,t,\sigma_I)_{\sigma,x}$
0
$[\![\Delta]\!]$
$\overset{\text{(IH)}}{=}$
$\sum_{\substack{P\in\mathsf{Path}(c)\\ x_P=x\\ \sigma(P)=\sigma}}$
$[\![P]\!]^{t,\sigma_I}$
0
$[\![\Delta]\!]$
$\overset{\text{def of } [\![\cdot]\!] \text{ and Path (new}(q.c))}{=}$
$\sum_{\substack{P'\in\mathsf{Path}(\mathsf{new}(q.c))\\ x_{P'}=x\\ \sigma(P')=\sigma}}$
$[\![P']\!]^{t,\sigma}$
= RHS

Case (apply):

$$\frac{\Gamma \mid \pi(\Delta, \vec{b}) \vdash c}{\Gamma \mid \pi(\Delta, \vec{a}) \vdash \mathsf{apply}_U(\vec{a}, \vec{b}.c)}$$

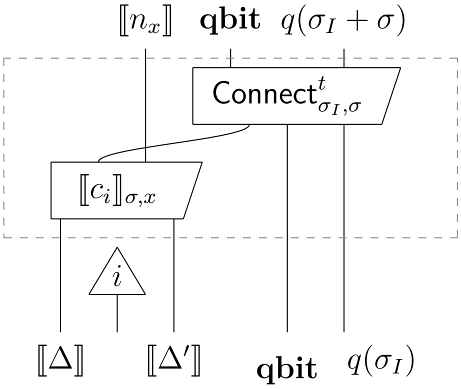
This case is analogous to the previous case, with a deterministic transition and without any addition to the trace.

Case (measure):

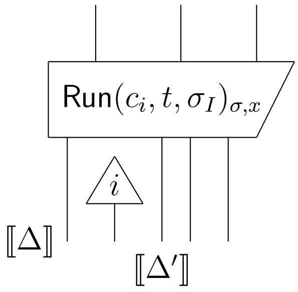
$$\frac{(\Gamma \mid \Delta, \Delta' \vdash c_i)_{i=0,1}}{\Gamma \mid \Delta, q, \Delta' \vdash \mathsf{measure}(q, c_0, c_1)}$$

We have

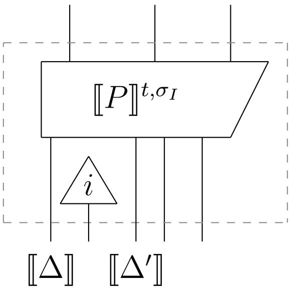
$\mathsf{LHS} \overset{\text{ind. principle}}{=} \sum_{i=0,1}$



$\overset{\text{def of Run}}{=} \sum_{i=0,1}$



$\overset{(\text{IH})}{=} \sum_{i=0,1} \sum_{\substack{P \in \mathsf{Path}(c_i) \\ x_P = x \\ \sigma(P) = \sigma}}$



$$\overset{\text{def of } [\![\cdot]\!] \text{ for paths}}{=} \sum_{i=0,1} \sum_{\substack{P \in \mathsf{Path}(c_i) \\ x_P = x \\ \sigma(P) = \sigma}} [\![\mathsf{measure}(q, c_0, c_1) \rightarrow^{(i)} P]\!]^{t,\sigma_I}$$

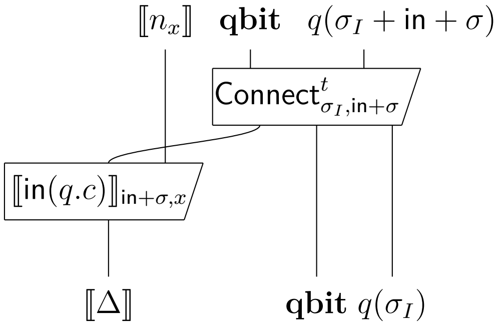
$$\overset{\text{def of Path}\,(\mathsf{measure}(q,c_0,c_1))}{=} \sum_{\substack{P' \in \mathsf{Path}(\mathsf{measure}(q,c_0,c_1)) \\ x_{P'} = x \\ \sigma(P') = \sigma}} [\![P]\!]^{t,\sigma_I} = \mathsf{RHS}$$

Case (in):

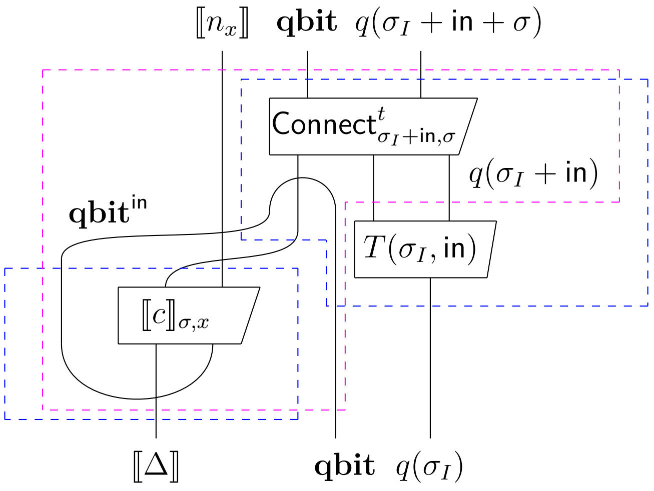
$$\frac{\Gamma \mid \Delta, q \vdash c}{\Gamma \mid \Delta \vdash \mathsf{in}(q.c)}$$

For a trace of the form $\mathsf{in} + \sigma$ we first expand the definitions:

$\mathsf{LHS} \overset{\text{def}}{=}$

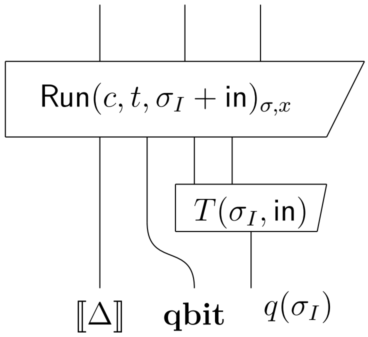

⟦n_x⟧
qbit
$q(\sigma_I + \mathsf{in} + \sigma)$
$\mathsf{Connect}^t_{\sigma_I,\mathsf{in}+\sigma}$
⟦$\mathsf{in}(q.c)$⟧$_{\mathsf{in}+\sigma,x}$
⟦Δ⟧
qbit
$q(\sigma_I)$

$\overset{\text{def } (\llbracket\cdot\rrbracket \text{ and } \mathsf{Connect})}{=}$

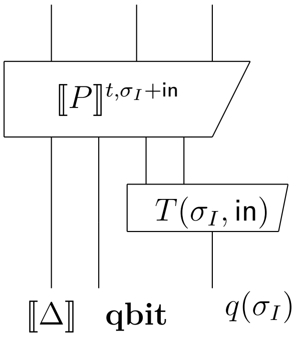

$\llbracket n_x \rrbracket$
qbit
$q(\sigma_I + \mathsf{in} + \sigma)$
$\mathsf{Connect}^t_{\sigma_I+\mathsf{in},\sigma}$
$q(\sigma_I + \mathsf{in})$
$\mathbf{qbit}^{\mathsf{in}}$
$T(\sigma_I, \mathsf{in})$
$\llbracket c \rrbracket_{\sigma,x}$
$\llbracket \Delta \rrbracket$
qbit
$q(\sigma_I)$

where the blue dashed boxes correspond to those in the previous step. We remark that the pink box corresponds exactly to $\mathsf{Run}(c, t, \sigma_I + \mathsf{in})_{\sigma,x}$, we get

$\mathsf{LHS} \overset{\text{def of Run}}{=}$


$\mathsf{Run}(c, t, \sigma_I + \mathsf{in})_{\sigma,x}$
$T(\sigma_I, \mathsf{in})$
$\llbracket \Delta \rrbracket$
qbit
$q(\sigma_I)$

$\overset{\text{(IH)}}{=} \sum_{\substack{P\in\mathsf{Path}(c)\\ x_P=x\\ \sigma(P)=\sigma}}$


$\llbracket P \rrbracket^{t,\sigma_I+\mathsf{in}}$
$T(\sigma_I, \mathsf{in})$
$\llbracket \Delta \rrbracket$
qbit
$q(\sigma_I)$

$$\overset{\text{def of } \llbracket \mathsf{in}(q.c) \to P \rrbracket}{=} \sum_{\substack{P\in\mathsf{Path}(c)\\ x_P=x\\ \sigma(P)=\sigma}} \llbracket \mathsf{in}(q.c) \to P \rrbracket^{t,\sigma_I} \overset{\text{def of } \mathsf{Path}\,(\mathsf{in}(q.c))}{=} \sum_{\substack{P'\in\mathsf{Path}(\mathsf{in}(q.c))\\ x_{P'}=x\\ \sigma(P')=\sigma}} \llbracket P' \rrbracket^{t,\sigma_I}$$

$$= \mathsf{RHS}$$

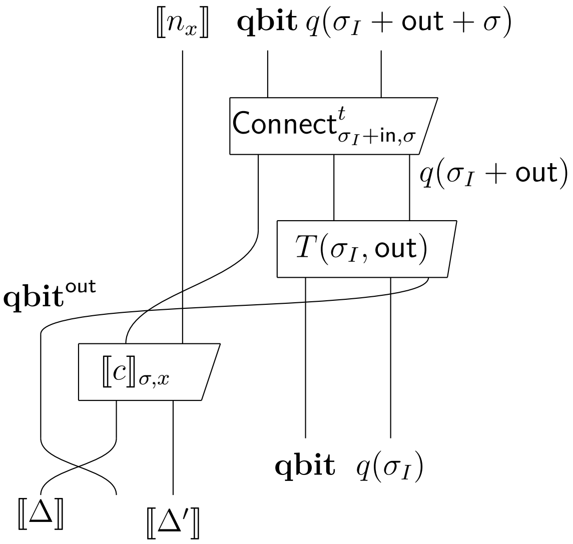
Otherwise if the trace σ does not start with in, then both the left and right hand side will be 0: the LHS gives 0 because $\llbracket \in (q.c) \rrbracket_{\sigma,x}$ will be 0, and the RHS gives 0 there is no path in $\mathsf{Path}\,(\mathsf{in}(q.c))$ such that its trace $\sigma(P)$ does not start with in.

Case (out):

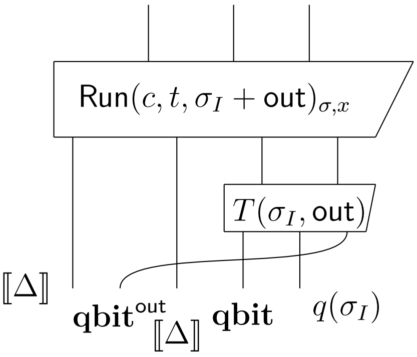
$$\frac{\Gamma \mid \Delta, \Delta' \vdash c}{\Gamma \mid \Delta, q, \Delta' \vdash \mathsf{out}(q, c)}$$

This case is analogous to the (in) case. We include it for the sake of completeness. When the trace is not of the form $\mathsf{out} + \sigma$, then both sides give 0 with the same reasoning as above. When it is, we get

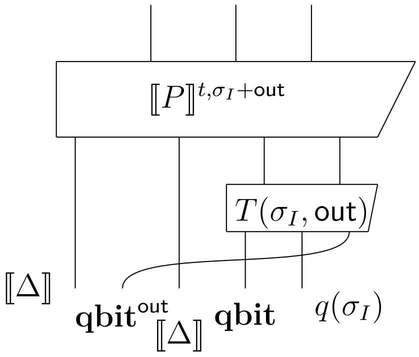
LHS $\overset{\text{def}}{=}$


$[\![n_x]\!]$
qbit $q(\sigma_I + \mathsf{out} + \sigma)$
$\mathsf{Connect}^t_{\sigma_I+\mathsf{in},\sigma}$
$q(\sigma_I + \mathsf{out})$
$T(\sigma_I, \mathsf{out})$
$\mathbf{qbit}^{\mathsf{out}}$
$[\![c]\!]_{\sigma,x}$
qbit $q(\sigma_I)$
$[\![\Delta]\!]$
$[\![\Delta']\!]$

$\overset{\text{def of }\mathsf{Run}}{=}$


$\mathsf{Run}(c, t, \sigma_I + \mathsf{out})_{\sigma,x}$
$T(\sigma_I, \mathsf{out})$
$[\![\Delta]\!]$
$\mathbf{qbit}^{\mathsf{out}}$
$[\![\Delta]\!]$
qbit
$q(\sigma_I)$

$\overset{\text{(IH)}}{=} \sum_{\substack{P \in \mathsf{Path}(c) \\ x_P = x \\ \sigma(P) = \sigma}}$


$[\![P]\!]^{t,\sigma_I+\mathsf{out}}$
$T(\sigma_I, \mathsf{out})$
$[\![\Delta]\!]$
$\mathbf{qbit}^{\mathsf{out}}$
$[\![\Delta]\!]$
qbit
$q(\sigma_I)$

$$\overset{\text{def of }[\![\mathsf{out}(q,c) \to P]\!]^{t,\sigma_I}\text{ and }\mathsf{Path}\,(\mathsf{out}(q,c))}{=} \sum_{\substack{P \in \mathsf{Path}(\mathsf{out}(q,c)) \\ x_P = x \\ \sigma(P) = \sigma}} [\![P]\!]^{t,\sigma_I}$$

$= \mathsf{RHS}$

□

5.3.3 Corollaries: Adequacy and Full Abstraction

This result is significant, because from it we can easily derive adequacy and full abstraction using the results from the previous sections.

Theorem 5.3.4 (Adequacy)**.** *Let $\Gamma \mid \Delta \vdash c, c'$. Then, $[\![c]\!] = [\![c']\!]$ implies $c \simeq_{qu} c'$.*

Proof. Assume $[\![c]\!] = [\![c']\!]$. Let $t \in \mathsf{Stream}(X_0)$ be an arbitrary stream and σ_I be the initial stream state, and let $\sigma \in \{\mathsf{in}, \mathsf{out}\}^*$, $x : n_x \in \Gamma$. Then, by definition we know that $\mathsf{Run}(c, t, \sigma_I)_{\sigma,x} = \mathsf{Run}(c', t, \sigma_I)_{\sigma,x}$. By theorem 5.3.1, it follows that

$$\sum_{\substack{P \in \mathsf{Path}(c) \\ x_P = x \\ \sigma(P) = \sigma}} [\![P]\!]^{t,\sigma_I} = \sum_{\substack{P \in \mathsf{Path}(c') \\ x_P = x \\ \sigma(P) = \sigma}} [\![P]\!]^{t,\sigma_I}$$

from which it follows by corollary 4.3.21 that

$$\sum_{\pi:\langle\rho_I,\sigma_I,c\rangle\to^*\left\langle\rho_f,\sigma_I+\sigma,\mathsf{skip}_x\right\rangle} \Pr(\pi)\cdot\tilde{\rho}_f = \sum_{\pi:\langle\rho_I,\sigma_I,c'\rangle\to^*\left\langle\rho_f,\sigma_I+\sigma,\mathsf{skip}_x\right\rangle} \Pr(\pi)\cdot\tilde{\rho}_f$$

for any $\rho_I :_t \Delta, \sigma_I$. We can now derive that

$$\sum_{\rho_f}\Pr(\langle\rho_I,\sigma_I,c\rangle \to^* \langle\rho_f,\sigma_I+\sigma,\mathsf{skip}_x\rangle)\cdot\tilde{\rho}_f$$

$$= \sum_{\rho_f}\Pr(\langle\rho_I,\sigma_I,c'\rangle \to^* \langle\rho_f,\sigma_I+\sigma,\mathsf{skip}_x\rangle)\cdot\tilde{\rho}_f$$

which is exactly $c \simeq_{qu} c'$. □

For full abstraction, we need the following additional fact:

Lemma 5.3.5. *For any finite dimensional*[1] *system of $A, B \in \mathbf{CP}^\infty$ (i.e. for any system in* **CP***), let $f, g : A \to B$. Then $f = g \iff \forall\psi : \mathbb{C} \to A. f\circ\psi = g\circ\psi$.*

Proof Sketch. One direction is trivial. For the converse, we have to use the fact that the positive elements given by morphisms $\psi : \mathbb{C} \to A$ span the entire space of A. Because we are working with finite dimensions, we can start with matrix algebras $A = \mathcal{M}_n(\mathbb{C})$, which is an object of the form $I_A = \{*\}$ and $d^A_* = n$.

And indeed, the CP maps in $\mathbb{C} \to \mathcal{M}_n(\mathbb{C})$ do form a basis: the explicit construction is given in Theorem 6.24 of [22]. This easily extends to finite biproducts of matrix algebras $\bigoplus_i \mathcal{M}_{k_i}(\mathbb{C})$ (objects with $I_A = [n]$ and $d^A_i = k_i$), because its basis can be easily constructed from its components. □

Theorem 5.3.6 (Full Abstraction)**.** *Let $\Gamma \mid \Delta \vdash c, c'$. Then, $c \simeq_{qu} c'$ implies $[\![c]\!] = [\![c']\!]$.*

Proof. By doing the reverse of the reasoning above (possible because we had only done a chain of equalities), we obtain from $c \simeq_{qu} c'$ that

$$\mathsf{Run}(c,t,\sigma_I)_{\sigma,x}\circ\rho_I = \mathsf{Run}(c',t,\sigma_I)_{\sigma,x}\circ\rho_I$$

for all $t \in \mathsf{Stream}(X_0), \sigma_I, \sigma \in \{\mathsf{in},\mathsf{out}\}^*, x : n_x \in \Gamma, \rho_I : I \to [\![\Delta]\!] \otimes \mathbf{qbit} \otimes q(\sigma_I)$ where ρ_I is normalised. From the above lemma, we then obtain that

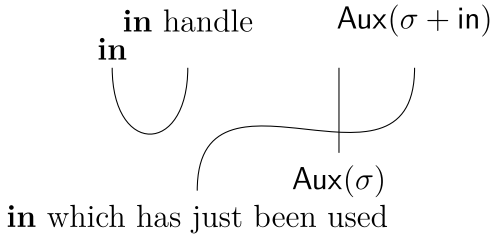
$$\mathsf{Run}(c,t,\sigma_I)_{\sigma,x} = \mathsf{Run}(c',t,\sigma_I)_{\sigma,x}.$$

where, of course, the normalisation of the ρ_I's is irrelevant in forming a basis. Now, we wish to show that this implies that $[\![c]\!] = [\![c']\!]$. This is informally the case because we can instantiate the stream as a trivial stream that stores all the outputs when receiving them, as well as all inputs,

[1] Only studying the finite-dimensional case is a simplification that suffices for our purposes.

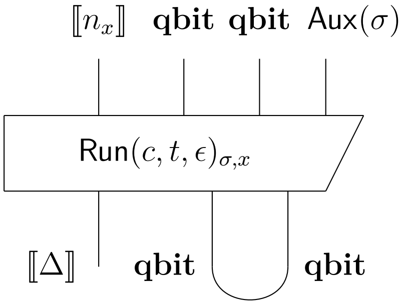
by using a Bell state. Formally, we define $q(\sigma) = \mathbf{qbit} \otimes \mathsf{Aux}(\sigma)$. Intuitively, this additional qubit on the right will be at all times entangled to the qubit, which will be the next input to the program. We call it the 'handle' of the next input qubit. Then, when such an input happens, we store this handle as the rightmost qubit of our stream state and create another entangled pair of qubits, using one as the next input and one as the handle. More formally, we have that for all σ,

$T(\sigma, \mathsf{in}) =$

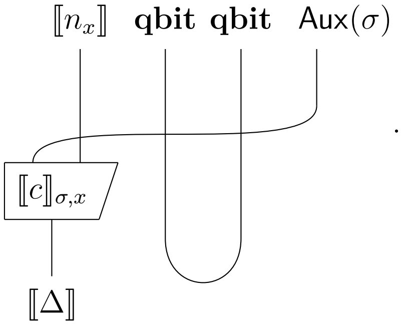


where we have ignored the normalising factor for the Bell state, necessary to make this map trace preserving. The output case, on the other hand, is trivial. We can just define $T(\sigma, \mathsf{out}) = \mathbf{id}_{\mathbf{qbit} \otimes q(\sigma) \otimes \mathbf{qbit}}$, which automatically stores the output as the rightmost qubit of the state.

Now, consider the following morphism, where $t = (q, T)$ is the stream defined above.



By a simple induction, one can easily show that it will be equal to exactly



Finally, our premises imply that the instantiation of this diagram with c and c' are equal. Thus, it follows trivially that $[\![c]\!] = [\![c']\!]$, just as required. □

5.4 Results with respect to the PAT

From full abstraction, one can then easily show that $\mathbb{C} \in \mathsf{Kl}_T^{op}$ forms a model of $\mathsf{I/O} \otimes \mathsf{QUANTUM}$.

Theorem 5.4.1 (Soundness)**.** *The following implication holds:*

$$\Gamma \mid \Delta \vdash c = c' \implies [\![c]\!] = [\![c']\!].$$

Proof. This simply follows from the soundness result of the quantum equivalence $\simeq_{qu}$, given in theorem 4.3.25, and full abstraction given in theorem 5.3.6. □

Corollary 5.4.2 (Model)**.** *The object $\mathbb{C}$ forms a model of PAT in the category Kl_T^{op}.*

Proof. The category $\mathcal{C} = \mathsf{Kl}_T^{op}$ has products inherited from the biproduct in $\mathbf{CP}^\infty$. Moreover, it does indeed have the structure of a **Bij** action $\bullet : \mathbf{Bij} \times \mathcal{C} \to \mathcal{C}$ whereby $p \bullet X = \mathbf{qbit}^{\otimes p} \otimes X$. Finally, we can indeed interpret each term $\Gamma \mid \Delta \vdash c$ in $\mathcal{C}$ as $[\![c]\!]$, and the soundness theorem above shows that this interpretation respects the equational theory. □

When it comes to completeness, we conjecture it to hold.

Conjecture 5.4.3. *The converse of theorem 5.4.1 holds.*

Indeed, this statement being true will be unsurprising, for we constructed our monad $T : \mathbf{CP}^\infty \to \mathbf{CP}^\infty$ as the free monad corresponding to the $\mathsf{I/O}$ theory on a category that fully captures the $\mathsf{QUANTUM}$ theory. This is exactly how one would construct a monad for a tensor of algebraic theories because, in general, $\mathsf{Mod}(\mathbb{L} \otimes \mathbb{L}', \mathcal{C}) = \mathsf{Mod}(\mathbb{L}, \mathsf{Mod}(\mathbb{L}', \mathcal{C}))$. That this conjecture does not hold could conceivably be due to two potential reasons: either the above equality does not hold, which means that the PAT framework presented in [67] is somehow not well formed, which is unlikely; or $\mathbf{CP}^\infty$ has additional structure which is not axiomatised by the equational theory. The interesting question, in the latter case, would then be whether we could extend our theory to capture these additional equalities. We shall leave this question to future work.

6 Conclusion

6.1 Summary

This dissertation achieved two things.

First, we proved the existence of sums and tensors for the FinProd doctrine $[\mathbf{Ctx_0}, \mathbf{Set}]$ enriched Lawvere theories presented by parameterised algebraic theories specified in Staton's framework [67]. The well-formedness of such standard constructions provides evidence that the framework in question is well-behaved.

Second, we proposed the first algebraic theory for qubit quantum computing and classically controlled qubit quantum communication as a tensor in the above sense: $\mathsf{I/O} \otimes \mathsf{QUANTUM}$. We studied the theory by giving two complementary models: a quantum stream-based operational semantics along with a program equivalence compatible with quantum theory and a monadic denotational semantics based on a standard I/O monad taken on $\mathbf{CP}^\infty$, the category of matrix algebras and CP maps freely completed with infinite biproducts [52]. We further showed that the denotational semantics is adequate and fully complete with respect to the operational semantics. The results are summarised in table 6.1.

$\Longrightarrow$?	$c = c'$	$c \simeq_{qu} c'$	$[\![c]\!] = [\![c']\!]$
$c = c'$	✓	✓(Theorem 4.3.25)	✓(Corollary 5.4.2)
$c \simeq_{qu} c'$	? (Conjecture 5.4.3)	✓	✓(Theorem 5.3.6)
$[\![c]\!] = [\![c']\!]$	? (Conjecture 5.4.3)	✓(Theorem 5.3.4)	✓

Table 6.1: Summary of Results in Chapters 4 and 5. Guide to read this table: whether we know that A implies B is indicated in the cell on row A, column B.

6.2 Future Work

The immediate next step is to prove full completeness of the quantum I/O monad we developed with respect to $\mathsf{I/O} \otimes \mathsf{QUANTUM}$ (conjecture 5.4.3). This is crucial, for it is the only missing piece to the puzzle.

It would also be interesting to understand how this work relates mathematically to various other related lines of work. Questions we would like to ask include:

- How does the quantum I/O monad we constructed relate to higher order causal quantum processes as defined by Kissinger and Uijlen [34]? Can I model the causal order specified by the I/O trace $\sigma \in \{\mathsf{in}, \mathsf{out}\}^*$ within that framework?
- It was mentioned previously that our object of study can be seen as the local sublanguage of quantum process calculi. Another question to ask would thus be: how does the $\mathsf{I/O} \otimes \mathsf{QUANTUM}$ theory relate to the notions of bisimilarity proposed for various quantum process calculi?

Additionally, there are potentially interesting extensions of variations of the theory we should consider.

- There have been studies in quantum information theory about quantum controlled I/O [35, 2]. How would such a mechanism look like in the context of a programming language? What would happen if we allowed it in our theory?
- Another direction of extensions would be to try to move closer to giving an algebraic theory for quantum concurrency instead of just communication. This will involve extending the algebraic theory to allow for multiple channels, non-determinism, and parallelism, the latter two of which are known to have subtle interactions with quantum effects. Modelling parallelism is an especially difficult question because the $||$ operator is not algebraic and, therefore, cannot be straightforwardly incorporated in an algebraic theory as an operation [66, 25]. While there do exist workarounds, such as in Abadi et al.'s work on cooperative threading [1], they remain an unsatisfactory solution, especially for modelling process-calculus-style parallel composition.

Finally, in the literature of algebraic effects, while models characterise the effectful computation, comodels [57] and their generalisations ('runners' [4]) specify the environment in which those computations are run. Thus, it is potentially interesting to look into the structure of the quantum stream we defined to see if it corresponds to a notion of a comodel. Then, we could potentially achieve a notion of parallelism if we could instantiate comodels with programs satisfying certain conditions (e.g. its I/O should be complementary to the I/O calls from the other program).

References

[1] Martín Abadi and Gordon D. Plotkin. “A Model of Cooperative Threads”. In: *Log. Methods Comput. Sci.* 6.4 (2010). DOI: 10.2168/LMCS-6(4:2)2010. URL: https://doi.org/10.2168/LMCS-6(4:2)2010.

[2] Alastair A. Abbott et al. “Communication through coherent control of quantum channels”. In: *Quantum* 4 (Sept. 2020), p. 333. ISSN: 2521-327X. DOI: 10.22331/q-2020-09-24-333. URL: http://dx.doi.org/10.22331/q-2020-09-24-333.

[3] Samson Abramsky and Bob Coecke. *A categorical semantics of quantum protocols.* 2007. arXiv: quant-ph/0402130 [quant-ph]. URL: https://arxiv.org/abs/quant-ph/0402130.

[4] Danel Ahman and Andrej Bauer. “Runners in Action”. In: *Programming Languages and Systems.* Ed. by Peter Müller. Cham: Springer International Publishing, 2020, pp. 29–55. ISBN: 978-3-030-44914-8.

[5] Thorsten Altenkirch and Alexander S. Green. “The Quantum IO Monad”. In: *Semantic Techniques in Quantum Computation.* Ed. by Simon Gay and IanEditors Mackie. Cambridge University Press, 2009, pp. 173–205.

[6] S.O. Anderson and A.J. Power. “A representable approach to finite nondeterminism”. In: *Theoretical Computer Science* 177.1 (1997), pp. 3–25. ISSN: 0304-3975. DOI: https://doi.org/10.1016/S0304-3975(96)00232-0. URL: https://www.sciencedirect.com/science/article/pii/S0304397596002320.

[7] Miriam Backens and Aleks Kissinger. “ZH: A Complete Graphical Calculus for Quantum Computations Involving Classical Non-linearity”. In: *Electronic Proceedings in Theoretical Computer Science* 287 (Jan. 2019), pp. 23–42. ISSN: 2075-2180. DOI: 10.4204/eptcs.287.2. URL: http://dx.doi.org/10.4204/EPTCS.287.2.

[8] Andrej Bauer. *What is algebraic about algebraic effects and handlers?* 2019. arXiv: 1807.05923 [cs.LO]. URL: https://arxiv.org/abs/1807.05923.

[9] Charles H. Bennett and Gilles Brassard. “Quantum cryptography: Public key distribution and coin tossing”. In: *Theoretical Computer Science* 560 (2014). Theoretical Aspects of Quantum Cryptography – celebrating 30 years of BB84, pp. 7–11. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2014.05.025. URL: https://www.sciencedirect.com/science/article/pii/S0304397514004241.

[10] Charles H. Bennett et al. “Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels”. In: *Phys. Rev. Lett.* 70 (13 Mar. 1993), pp. 1895–1899. DOI: 10.1103/PhysRevLett.70.1895. URL: https://link.aps.org/doi/10.1103/PhysRevLett.70.1895.

[11] Benjamin Bichsel et al. “Silq: a high-level quantum language with safe uncomputation and intuitive semantics”. In: *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation.* PLDI 2020. London, UK: Association for Computing Machinery, 2020, pp. 286–300. ISBN: 9781450376136. DOI: 10.1145/3385412.3386007. URL: https://doi.org/10.1145/3385412.3386007.

[12] Marcello Caleffi et al. “Distributed quantum computing: A survey”. In: *Computer Networks* 254 (Dec. 2024), p. 110672. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2024.110672. URL: http://dx.doi.org/10.1016/j.comnet.2024.110672.

[13] Matteo Capucci and Bruno Gavranovi. *Actegories for the Working Amthematician.* 2023. arXiv: 2203.16351 [math.CT]. URL: https://arxiv.org/abs/2203.16351.

[14] Jacques Carette et al. “With a Few Square Roots, Quantum Computing Is as Easy as Pi”. In: *Proceedings of the ACM on Programming Languages* 8.POPL (Jan. 2024), pp. 546–574. ISSN: 2475-1421. DOI: 10.1145/3632861. URL: http://dx.doi.org/10.1145/3632861.

[15] Pietro Cenciarelli and Eugenio Moggi. “A syntactic approach to modularity in denotational semantics”. In: *CTCS 1993* (1993).

[16] Lorenzo Ceragioli et al. “A Coalgebraic Model of Quantum Bisimulation”. In: ().

[17] Lorenzo Ceragioli et al. “Quantum Bisimilarity via Barbs and Contexts: Curbing the Power of Non-deterministic Observers”. In: *Proc. ACM Program. Lang.* 8.POPL (Jan. 2024). DOI: 10.1145/3632885. URL: https://doi.org/10.1145/3632885.

[18] G. Chiribella, G. M. D’ Ariano, and P. Perinotti. “Quantum Circuit Architecture”. In: *Physical Review Letters* 101.6 (Aug. 2008). ISSN: 1079-7114. DOI: 10.1103/physrevlett.101.060401. URL: http://dx.doi.org/10.1103/PhysRevLett.101.060401.

[19] Kenta Cho. “Semantics for a Quantum Programming Language by Operator Algebras”. In: *New Generation Computing* 34.1 (2016), pp. 25–68. DOI: 10.1007/s00354-016-0204-3. URL: https://doi.org/10.1007/s00354-016-0204-3.

[20] Bob Coecke and Ross Duncan. “Interacting Quantum Observables”. In: *Automata, Languages and Programming.* Ed. by Luca Aceto et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 298–310. ISBN: 978-3-540-70583-3.

[21] Bob Coecke and Ross Duncan. “Interacting quantum observables: categorical algebra and diagrammatics”. In: *New Journal of Physics* 13.4 (Apr. 2011), p. 043016. DOI: 10.1088/1367-2630/13/4/043016. URL: https://dx.doi.org/10.1088/1367-2630/13/4/043016.

[22] Bob Coecke and Aleks Kissinger. *Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning.* Cambridge University Press, 2017.

[23] Bob Coecke, Eric Oliver Paquette, and Dusko Pavlovic. *Classical and quantum structuralism.* 2009. arXiv: 0904.1997 [quant-ph]. URL: https://arxiv.org/abs/0904.1997.

[24] Simon Gay and Rajagopal Nagarajan. *Communicating Quantum Processes.* 2004. arXiv: quant-ph/0409052 [quant-ph]. URL: https://arxiv.org/abs/quant-ph/0409052.

[25] Rob van Glabbeek and Gordon Plotkin. “On CSP and the Algebraic Theory of Effects”. In: *Reflections on the Work of C.A.R. Hoare.* Springer London, 2010, pp. 333–369. ISBN: 9781848829121. DOI: 10.1007/978-1-84882-912-1_15. URL: http://dx.doi.org/10.1007/978-1-84882-912-1_15.

[26] Alexander S. Green et al. “Quipper: A Scalable Quantum Programming Language”. In: *CoRR* abs/1304.3390 (2013). arXiv: 1304.3390. URL: http://arxiv.org/abs/1304.3390.

[27] Amar Hadzihasanovic. *A Diagrammatic Axiomatisation for Qubit Entanglement.* 2015. arXiv: 1501.07082 [cs.LO]. URL: https://arxiv.org/abs/1501.07082.

[28] Chris Heunen and Jamie Vicary. *Categories for Quantum Theory: An Introduction.* Oxford University Press, Nov. 2019. ISBN: 9780198739623. DOI: 10.1093/oso/9780198739623.001.0001. eprint: https://academic.oup.com/book/43710/book-pdf/50991591/9780191060069_web.pdf. URL: https://doi.org/10.1093/oso/9780198739623.001.0001.

[29] C. A. R. Hoare. “Communicating sequential processes”. In: *Commun. ACM* 21.8 (Aug. 1978), pp. 666–677. ISSN: 0001-0782. DOI: 10.1145/359576.359585. URL: https://doi.org/10.1145/359576.359585.

[30] Mathieu Huot and Sam Staton. “Quantum channels as a categorical completion”. In: *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science.* LICS ’19. Vancouver, Canada: IEEE Press, 2021.

[31] Martin Hyland, Gordon Plotkin, and John Power. “Combining effects: Sum and tensor”. In: *Theoretical Computer Science* 357.1 (2006). Clifford Lectures and the Mathematical Foundations of Programming Semantics, pp. 70–99. ISSN: 0304-3975. DOI: `https://doi.org/10.1016/j.tcs.2006.03.013`. URL: `https://www.sciencedirect.com/science/article/pii/S0304397506002659`.

[32] Martin Hyland and John Power. “The Category Theoretic Understanding of Universal Algebra: Lawvere Theories and Monads”. In: *Electronic Notes in Theoretical Computer Science* 172 (2007). Computation, Meaning, and Logic: Articles dedicated to Gordon Plotkin, pp. 437–458. ISSN: 1571-0661. DOI: `https://doi.org/10.1016/j.entcs.2007.02.019`. URL: `https://www.sciencedirect.com/science/article/pii/S1571066107000874`.

[33] Geun Bin IM and G.M. Kelly. “A universal property of the convolution monoidal structure”. In: *Journal of Pure and Applied Algebra* 43.1 (1986), pp. 75–88. ISSN: 0022-4049. DOI: `https://doi.org/10.1016/0022-4049(86)90005-8`. URL: `https://www.sciencedirect.com/science/article/pii/0022404986900058`.

[34] Aleks Kissinger and Sander Uijlen. “A categorical semantics for causal structure”. In: *Logical Methods in Computer Science* 15 (2019).

[35] Hlér Kristjánsson et al. “Resource theories of communication”. In: *New Journal of Physics* 22.7 (July 2020), p. 073014. DOI: `10.1088/1367-2630/ab8ef7`. URL: `https://dx.doi.org/10.1088/1367-2630/ab8ef7`.

[36] Marie Lalire and Philippe Jorrand. *A Process Algebraic Approach to Concurrent and Distributed Quantum Computation: Operational Semantics.* 2004. arXiv: `quant-ph/0407005 [quant-ph]`. URL: `https://arxiv.org/abs/quant-ph/0407005`.

[37] F. William Lawvere. “Functorial Semantics of Algebraic Theories”. In: *Proceedings of the National Academy of Sciences of the United States of America* 50.5 (1963), pp. 869–872. ISSN: 00278424, 10916490. URL: `http://www.jstor.org/stable/71935` (visited on 08/15/2024).

[38] Saunders MacLane. *Categories for the Working Mathematician.* Graduate Texts in Mathematics, Vol. 5. New York: Springer-Verlag, 1971, pp. ix+262.

[39] R. Milner. *A Calculus of Communicating Systems.* Berlin, Heidelberg: Springer-Verlag, 1982. ISBN: 0387102353.

[40] Robin Milner, Joachim Parrow, and David Walker. “A calculus of mobile processes, I”. In: *Information and Computation* 100.1 (1992), pp. 1–40. ISSN: 0890-5401. DOI: `https://doi.org/10.1016/0890-5401(92)90008-4`. URL: `https://www.sciencedirect.com/science/article/pii/0890540192900084`.

[41] Eugenio Moggi. “Notions of computation and monads”. In: *Information and Computation* 93.1 (1991). Selections from 1989 IEEE Symposium on Logic in Computer Science, pp. 55–92. ISSN: 0890-5401. DOI: `https://doi.org/10.1016/0890-5401(91)90052-4`. URL: `https://www.sciencedirect.com/science/article/pii/0890540191900524`.

[42] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition.* Cambridge University Press, 2010.

[43] nLab authors. *closed monoidal category.* `https://ncatlab.org/nlab/show/closed+monoidal+category`. Revision 56. Aug. 2024.

[44] nLab authors. *closed monoidal structure on presheaves.* `https://ncatlab.org/nlab/show/closed+monoidal+structure+on+presheaves`. Revision 21. Aug. 2024.

[45] nLab authors. *Day convolution.* `https://ncatlab.org/nlab/show/Day+convolution`. Revision 80. Aug. 2024.

[46] nLab authors. *deferred measurement principle.* `https://ncatlab.org/nlab/show/deferred+measurement+principle`. Revision 13. Aug. 2024.

[47] nLab authors. *Eckmann-Hilton argument.* https://ncatlab.org/nlab/show/Eckmann-Hilton+argument. Revision 37. Aug. 2024.

[48] nLab authors. *free cocompletion.* https://ncatlab.org/nlab/show/free+cocompletion. Revision 76. Aug. 2024.

[49] nLab authors. *monoidal category.* https://ncatlab.org/nlab/show/monoidal+category. Revision 169. Aug. 2024.

[50] nLab authors. *star-algebra.* https://ncatlab.org/nlab/show/star-algebra. Revision 24. Aug. 2024.

[51] nLab authors. *Yoneda lemma.* https://ncatlab.org/nlab/show/Yoneda+lemma. Revision 83. Aug. 2024.

[52] Michele Pagani, Peter Selinger, and Benot Valiron. "Applying quantitative semantics to higher-order quantum computing". In: *CoRR* abs/1311.2290 (2013). arXiv: 1311.2290. URL: http://arxiv.org/abs/1311.2290.

[53] Jennifer Paykin, Robert Rand, and Steve Zdancewic. "QWIRE: a core language for quantum circuits". In: *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages.* POPL '17. Paris, France: Association for Computing Machinery, 2017, pp. 846–858. ISBN: 9781450346603. DOI: 10.1145/3009837.3009894. URL: https://doi.org/10.1145/3009837.3009894.

[54] Paolo Perrone. *Starting Category Theory.* WORLD SCIENTIFIC, 2024. DOI: 10.1142/13670. eprint: https://www.worldscientific.com/doi/pdf/10.1142/13670. URL: https://www.worldscientific.com/doi/abs/10.1142/13670.

[55] Gordon Plotkin and John Power. "Computational Effects and Operations: An Overview". In: *Electron. Notes Theor. Comput. Sci.* 73 (Oct. 2004), pp. 149–163. ISSN: 1571-0661. DOI: 10.1016/j.entcs.2004.08.008. URL: https://doi.org/10.1016/j.entcs.2004.08.008.

[56] Gordon Plotkin and John Power. "Notions of Computation Determine Monads". In: *Foundations of Software Science and Computation Structures.* Ed. by Mogens Nielsen and Uffe Engberg. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 342–356. ISBN: 978-3-540-45931-6.

[57] Gordon Plotkin and John Power. "Tensors of Comodels and Models for Operational Semantics". In: *Electronic Notes in Theoretical Computer Science* 218 (2008). Proceedings of the 24th Conference on the Mathematical Foundations of Programming Semantics (MFPS XXIV), pp. 295–311. ISSN: 1571-0661. DOI: https://doi.org/10.1016/j.entcs.2008.10.018. URL: https://www.sciencedirect.com/science/article/pii/S157106610800412X.

[58] John Power. "Premonoidal categories as categories with algebraic structure". In: *Theoretical Computer Science* 278.1 (2002). Mathematical Foundations of Programming Semantics 1996, pp. 303–321. ISSN: 0304-3975. DOI: https://doi.org/10.1016/S0304-3975(00)00340-6. URL: https://www.sciencedirect.com/science/article/pii/S0304397500003406.

[59] JOHN POWER and EDMUND ROBINSON. "Premonoidal categories and notions of computation". In: *Mathematical Structures in Computer Science* 7.5 (1997), pp. 453–468. DOI: 10.1017/S0960129597002375.

[60] Matija Pretnar. "An Introduction to Algebraic Effects and Handlers. Invited tutorial paper". In: *Electronic Notes in Theoretical Computer Science* 319 (2015). The 31st Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXI)., pp. 19–35. ISSN: 1571-0661. DOI: https://doi.org/10.1016/j.entcs.2015.12.003. URL: https://www.sciencedirect.com/science/article/pii/S1571066115000705.

[61] E. Riehl. *Category theory in context.* Aurora: Dover modern math originals. Dover Publications, 2017. ISBN: 978-0-486-82080-4.

[62] Hisham Sati and Urs Schreiber. *The Quantum Monadology.* 2023. arXiv: 2310.15735 [quant-ph]. URL: https://arxiv.org/abs/2310.15735.

[63] PETER SELINGER. "Towards a quantum programming language". In: *Mathematical Structures in Computer Science* 14.4 (2004), pp. 527–586. DOI: 10.1017/S0960129504004256.

[64] Peter Selinger. "Dagger Compact Closed Categories and Completely Positive Maps: (Extended Abstract)". In: *Electronic Notes in Theoretical Computer Science* 170 (2007). Proceedings of the 3rd International Workshop on Quantum Programming Languages (QPL 2005), pp. 139–163. ISSN: 1571-0661. DOI: `https://doi.org/10.1016/j.entcs.2006.12.018`. URL: `https://www.sciencedirect.com/science/article/pii/S1571066107000606`.

[65] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM Journal on Computing* 26.5 (Oct. 1997), pp. 1484–1509. ISSN: 1095-7111. DOI: `10.1137/s0097539795293172`. URL: `http://dx.doi.org/10.1137/S0097539795293172`.

[66] Ian Stark. "Free-Algebra Models for the π-Calculus". In: *Foundations of Software Science and Computation Structures: Proceedings of FOSSACS 2005*. Lecture Notes in Computer Science 3441. Springer-Verlag, 2005, pp. 155–169. URL: `http://www.inf.ed.ac.uk/~stark/freamp.html`.

[67] Sam Staton. "Algebraic Effects, Linearity, and Quantum Programming Languages". In: *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '15. Mumbai, India: Association for Computing Machinery, 2015, pp. 395–406. ISBN: 9781450333009. DOI: `10.1145/2676726.2676999`. URL: `https://doi.org/10.1145/2676726.2676999`.

[68] Sam Staton. "Freyd categories are Enriched Lawvere Theories". In: *Electronic Notes in Theoretical Computer Science* 303 (2014). Proceedings of the Workshop on Algebra, Coalgebra and Topology (WACT 2013), pp. 197–206. ISSN: 1571-0661. DOI: `https://doi.org/10.1016/j.entcs.2014.02.010`. URL: `https://www.sciencedirect.com/science/article/pii/S157106611400036X`.

[69] Sam Staton. "Instances of Computational Effects: An Algebraic Perspective". In: *2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*. 2013, pp. 519–519. DOI: `10.1109/LICS.2013.58`.

[70] Masamichi Takesaki. *Theory of Operator Algebras I*. 1st ed. Springer Science and Business Media, Dec. 2012. ISBN: 9781461261889.

[71] Philip Wadler. "The essence of functional programming". In: *Proceedings of the 19th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '92. Albuquerque, New Mexico, USA: Association for Computing Machinery, 1992, pp. 1–14. ISBN: 0897914538. DOI: `10.1145/143165.143169`. URL: `https://doi.org/10.1145/143165.143169`.

[72] Mark M. Wilde. *Quantum Information Theory*. 2nd ed. Cambridge University Press, 2017.

[73] Mingsheng Ying et al. *An Algebra of Quantum Processes*. 2010. arXiv: `0707.0330 [quant-ph]`. URL: `https://arxiv.org/abs/0707.0330`.

[74] Dongsheng Zhao and Taihe Fan. "Dcpo-completion of posets". In: *Theoretical Computer Science* 411.22 (2010), pp. 2167–2173. ISSN: 0304-3975. DOI: `https://doi.org/10.1016/j.tcs.2010.02.020`. URL: `https://www.sciencedirect.com/science/article/pii/S0304397510001155`.