MSc in Computer Science 2017-18

Project Dissertation

Project Dissertation title: A Programming Language for Higher-Order Quantum Computation with Indefinite Causal Order

Term and year of submission: Trinity Term 2018

Candidate Name: William Simmons

Title of Degree the dissertation is being submitted under: MSc in Computer Science

Word count: 15,939

Abstract

Programming languages usually impose a definite order over the instructions to be executed, but quantum mechanics permits processes to be combined in a superposition of configurations. This thesis provides the first higher-order quantum programming language featuring indefinite causal orders. This model introduces a new primitive for the quantum switch - a resource that can combine two channels in a superposition of causal orders. The quantum switch itself is generalised from the usual definition to control the connections between higher-order maps. Our type system incorporates a large fragment of linear logic; this enables signalling-aware types, preventing the construction of causal loops where information is sent backwards in time. In developing this model, we prove new results about the behaviour of the direct sum in the category Caus[CP(**FHilb**)] of causal quantum processes.

Acknowledgements

I would like to take this opportunity to give thanks to my supervisor, Prof. Giulio Chiribella, for his patience and enthusiasm with this project, with our meetings proving invaluable for helping to solidify my ideas and giving the project direction.

Further thanks go out to the friends I have made over the year at Worcester College, Balliol College, and beyond, with whom many days of fun turned spare time into a great time.

This work has been partly supported by a grant from the John Templeton Foundation. The opinions expressed in this work are those of the author and do not necessarily reflect the views of the John Templeton Foundation.

Contents

Chapter 1

Introduction

Recent years have seen exciting advances in the production of practical quantum computers. As a result, there has been a surge of interest in providing tools for quantum programming. A number of popular (classical) programming languages have been outfitted with libraries for quantum constructs, e.g. the quantum IO monad in Haskell [4]. We have also seen the emergence of bespoke languages for quantum computing such as Quipper [28] and LIQ$Ui|\rangle$ [50].

Most of the practical quantum programming languages currently available follow the paradigm of "quantum data, classical control" [44]: the computer performs operations on quantum systems, but the instructions are scheduled by a classical device. This gives a definite order over the instructions of a program. However, it is entirely possible in quantum mechanics for two events to be connected in a superposition of causal orders.

Some programming languages such as QML [3] and the language of [52] provide mechanisms for quantum alternation. This is a more general notion, allowing two distinct programs to be executed in superposition coherently with the input state. This allows the expression of programs with indefinite causal order. However, these languages lack the generality of higher-order programming and they can only simulate the effects of indefinite causal order over white-box processes.

This thesis presents a higher-order programming language for quantum computation with indefinite causal order. We achieve this in such a way that permits the modelling and management of black-box processes.

The quantum switch is a computational resource that takes two black-box channels and composes them in a superposition of causal orders. We generalise this here to operating on higher-order processes of type $\hat{A} \multimap \hat{A}$. This generalised switch is provided as a primitive in our programming language. This is a very timely development given the recent interest in theoretical [18, 54] and experimental constructions [38, 41, 41] of the switch, in addition to results on the computational advantages it provides [12, 22, 5].

A notable constraint on the quantum switch is the requirement for the channels to be non-signalling. If this is not satisfied, then some information must be sent backwards in time, violating laws of special relativity ([30] presents a good example from the swap operation). In quantum programming languages, linear type systems have seen great use to enforce compliance with the no-cloning theorem. Similar techniques are used in this thesis to handle the non-signalling constraint by providing signalling-aware types. In particular, we introduce the ability to operate

locally on each side of higher-order bipartite processes, preserving any signalling relation between them.

Our work takes Selinger and Valiron's quantum lambda calculus [45] as a starting point. We extend the computational model from qubit computing to quantum operations in arbitrary finite-dimensional Hilbert spaces. Programs reference a collection of registers, each of which may contain a physical higher-order quantum process. Processes can be built into these registers from their program descriptions, and connected via function application.

The theory behind the signalling-aware types lies in categories of higher-order causal processes [29]. We make progress in this area by adding the direct sum of processes and introduce new results on the signalling induced by quantum alternation.

Chapter 2 presents the necessary foundations on quantum mechanics and computation. In Chapter 3, we cover the required background on signalling between quantum systems, including the construction for categories of causal processes [29] and an introduction to indefinite causal orders. Chapter 4 revues some trends in quantum programming languages and examines the work of Selinger and Valiron [45] as a case study, highlighting its shortcomings as a general model of quantum computation. The main results of the thesis are contained within Chapter 5, bringing together the topics of the previous sections to give the schema for a new programming language and demonstrating the features with some example programs. Chapter 6 goes into some of the design choices in detail, reasoning about them using the category theoretic framework, and presenting useful operational properties of the language. Finally, in Chapter 7 we discuss several avenues down which the language could still be extended to give a more complete picture of quantum computing, followed by some concluding remarks.

Chapter 2

Quantum Structures

In the typical classical setting, every property has a fixed state which will not change unless acted on directly. Furthermore, this state can be freely interrogated without any side-effects. Quantum systems contrast this by being in superpositions of states, possibly entangled with other systems, and observing/measuring is a probabilistic action that irreversibly alters a state. These can be used as computational resources allowing a variant of parallelism via superposition, or giving a new type of shared cryptographic secret in the form of entangled systems.

The simplest non-trivial quantum system is the qubit, with states corresponding to superpositions of two orthonormal basis states. Typical (reversible) transformations are change-of-basis operations, rebalancing the superposition in some way. Observing the system will cause the state to instantaneously snap to one of the basis states - this happens probabilistically depending on the coefficients of superposition prior to the observation. A wide selection of physical systems can exhibit the behaviour of a qubit. Common examples include (superpositions of) particle trajectories, photon polarisation angles, and electronic states of ions.

2.1 Systems and States

Each quantum system is associated with its corresponding Hilbert space. The dimension of the Hilbert space corresponds to the number of degrees of freedom in the system, which may potentially be infinite.

Definition 2.1.1 (Hilbert space)**.** *A Hilbert space $\mathcal{H}$ is an inner product space which is complete in the sense that, given any sequence $v_0, v_1, \ldots \in \mathcal{H}$ where $\sum_{i=0}^{\infty} \|v_i\| < \infty$, there exists some $v \in \mathcal{H}$ such that $\lim_{n\to\infty} \|v - \sum_{i=0}^{n} v_i\| = 0$.*

The internal state of a system can be represented by a normalised density matrix.

Definition 2.1.2 (Density matrix)**.** *A density matrix ρ on a Hilbert space $\mathcal{H}$ is a positive linear map $\mathcal{H} \to \mathcal{H}$ ($\rho = \psi^\dagger \circ \psi$ for some Hilbert-Schmidt operator $\psi : \mathcal{H} \to \mathcal{K}$). ρ is normalised when it satisfies $\mathrm{Tr}[\rho] = 1$ or partial when $\mathrm{Tr}[\rho] \leq 1$.*

The two main ways of combining Hilbert spaces are via tensor product $\otimes$, modelling compound systems, and direct sum $\oplus$, modelling systems in superposition over the component Hilbert spaces.

Trivial systems are those with no inputs or outputs. States have a single degree of freedom, describing the probability of some event occurring - the set of partial

density matrices for the one-dimensional Hilbert space ($\mathbb{C}$) corresponds exactly to the range $[0, 1]$. Taking the direct sum over two trivial state spaces gives a superposition over two of these fixed states, i.e. a qubit. The trivial system allows us to define purity of a state:

Definition 2.1.3 (Purity)**.** *A state ρ is pure when there exists some $\psi : \mathcal{H} \to \mathbb{C}$ such that $\rho = \psi^\dagger \circ \psi$.*

Quantum processes can be desribed by completely positive maps. We will use the notation $\mathcal{L}(\mathcal{H})$ to refer to the set of linear maps from $\mathcal{H}$ to itself.

Definition 2.1.4 (CP-maps)**.** *A completely positive (CP) map from $\mathcal{H}$ to $\mathcal{K}$ is an operator $\Psi : \mathcal{L}(\mathcal{H}) \to \mathcal{L}(\mathcal{K})$ such that, for every $\mathcal{H}'$, if $\rho \in \mathcal{L}(\mathcal{H} \otimes \mathcal{H}')$ is positive, then so is $(\Psi \otimes \mathrm{id}_{\mathcal{H}'})(\rho)$.*

Definition 2.1.5 (Pure Process)**.** *A process $\Psi : \mathcal{L}(H) \to \mathcal{L}(\mathcal{K})$ is pure when, for every $\mathcal{H}'$, if $\rho \in \mathcal{L}(\mathcal{H} \otimes \mathcal{H}')$ is pure, then so is $(\Psi \otimes \mathrm{id}_{\mathcal{H}'})(\rho)$.*

Every linear map $\psi : \mathcal{H} \to \mathcal{K}$ gives rise to a pure process $\hat{\psi} : \mathcal{L}(\mathcal{H}) \to \mathcal{L}(\mathcal{K})$ defined as $\hat{\psi}(\rho) := \psi \circ \rho \circ \psi^\dagger$. Whilst not every process is pure, every CP-map can be represented by a set of pure processes via a (not necessarily unique) Kraus decomposition - $\Psi(\rho) = \sum_i \psi_i^\dagger \circ \rho \circ \psi_i$ for some maps $\psi_i : \mathcal{H} \to \mathcal{K}$ such that $\sum_i \psi_i^\dagger \psi_i = \mathrm{id}_{\mathcal{H}}$.

From these definitions, it is clear that (pure) states are the special case of (pure) processes with a trivial input system.

2.2 Graphical Calculus

Quantum circuits are comprised of collections of processes, connected either in parallel or series. This can be neatly represented by a diagrammatic calculus which makes interchange laws and permutations trivial operations. We will use such diagrams within this thesis to give examples and aid explanations when appropriate. We will assume the convention that time flows from the bottom of a diagram to the top.

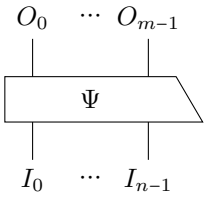
Figure 2.1 demonstrates the graphical representations of quantum processes. A process with inputs $I_0, \ldots, I_{n-1}$ and outputs $O_0, \ldots, O_{m-1}$ is drawn as a box with n input wires and m output wires. Wedges are used to distinguish between a process and its conjugate (flipped horizontally), adjoint (flipped vertically), and transpose (rotated). Each wire carries a density matrix over some Hilbert space. Wires carrying trivial systems are not drawn since there is intuitively nothing to carry.

[21] gives a full introduction to the graphical constructs and these notations.

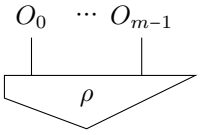
2.3 The Choi-Jamiołkovski Isomorphism

Quantum theory admits a bijection between processes from $\mathcal{L}(\mathcal{H})$ to $\mathcal{L}(\mathcal{K})$ and states in $\mathcal{L}(\mathcal{K} \otimes \mathcal{H})$. This process-state duality is useful to provide a uniform representation of any quantum process as a state in some large Hilbert space.

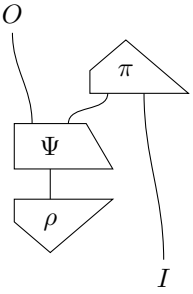
Entanglement describes an interdependence between the individual components of a compound system. For any Hilbert space $\mathcal{H}$ with an orthonormal basis $\{|i\rangle\}_i$,



(a) A general process.



(b) A process with no inputs (i.e. a state).

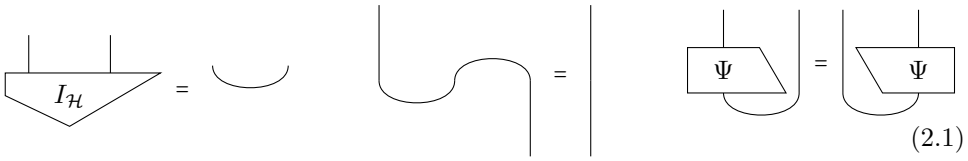


(c) A circuit formed by composing processes.

Figure 2.1: Examples of structures in the graphical calculus.

we can define the maximally entangled vector as $|I_{\mathcal{H}}\rangle = \sum_i |i\rangle \otimes |i\rangle$ and the corresponding state $|I_{\mathcal{H}}\rangle\langle I_{\mathcal{H}}|$. Note that this is not a normalised state in general since $\mathrm{Tr}[|I_{\mathcal{H}}\rangle\langle I_{\mathcal{H}}|] = \dim(\mathcal{H})$.

In the graphical calculus, the maximally entangled state can be represented by a curved piece of wire since combining it with its adjoint yields the identity. It also relates a process to its transpose in a natural manner.



(2.1)

The Choi-Jamiołkovski isomorphism formalises this bijection between maps and states.

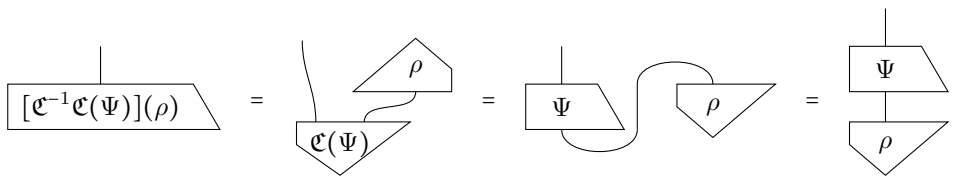
Definition 2.3.1 (Choi-Jamiołkovski Isomorphism)**.** *Given a CP-map* $\Psi : \mathcal{L}(\mathcal{H}) \to \mathcal{L}(\mathcal{K})$*, its Choi-Jamiołkovski operator is defined as*

$$\mathfrak{C}(\Psi) := (\Psi \otimes \mathrm{id}_{\mathcal{H}})(|I_{\mathcal{H}}\rangle\langle I_{\mathcal{H}}|) \tag{2.2}$$

and the inverse operation is

$$[\mathfrak{C}^{-1}(M)](\rho) := \mathrm{Tr}_{\mathcal{H}}[(\mathrm{id}_{\mathcal{K}} \otimes \rho^T)M] \tag{2.3}$$

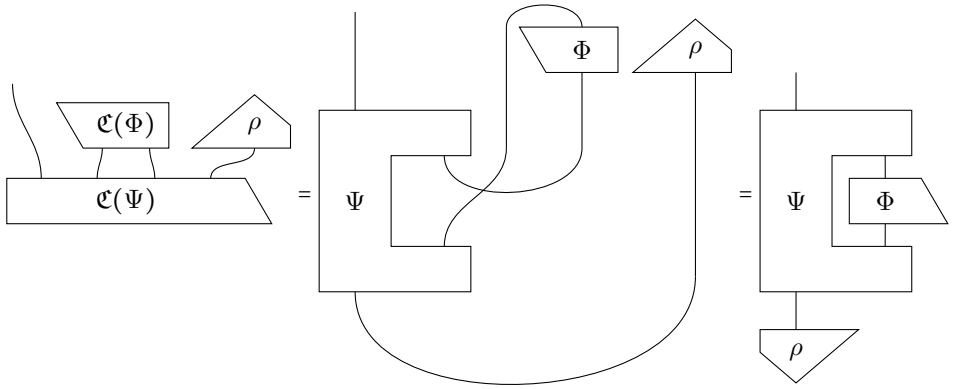
The isomorphism is very clear to see in the graphical calculus (note that the partial trace has been absorbed into the box for the transposed ρ state).



Since this isomorphism can be applied to any quantum process to reduce it to a density matrix, higher-order processes/supermaps [17] (those where the input or

output is a generic process as opposed to a first-order state) can be represented as processes acting on Choi-Jamiołkovski operators. This turns it into a process with first-order inputs and outputs, allowing the isomorphism to be applied to yield a state. Density matrices can hence be used as universal descriptions of any quantum process.

For example, given a second-order process Ψ, the isomorphism works as follows:



2.4 Measurements

The main method of extracting information from a quantum system is to perform a measurement. Any measurement is made with respect to a collection of subspaces spanning the system's Hilbert space - a pure quantum process is applied before projecting into the specified subspaces. This projection causes a probabilistic collapse of the state onto a single subspace. The only information obtained from the measurement is which subspace the state was projected into. An extreme case of this is a measurement on an orthonormal basis, where the collapse yields an exact state.

As an example, suppose we are given a qubit in the normalised pure state $|\phi\rangle\langle\phi|$ where $|\phi\rangle = \alpha|0\rangle + \beta|1\rangle$. Measuring it in the computational basis ($\{|0\rangle, |1\rangle\}$) yields $|0\rangle\langle 0|$ with probability $|\alpha|^2$ and $|1\rangle\langle 1|$ with probability $|\beta|^2$. Note that, in performing this operation, we lose any off-diagonal information (the coefficients of $|0\rangle\langle 1|$ and $|1\rangle\langle 0|$).

Measurements can furthermore be divided into two classes. Demolition measurements completely remove a state, reducing it to the trivial system. On the other hand, non-demolition measurements retain the state after the collapse, allowing it to be used for further processing. Every demolition measurement is equivalent to the corresponding non-demolition measurement followed by discarding the residual quantum system.

The most general definition is a Positive-Operator Valued Measure (POVM):

Definition 2.4.1 (POVM)**.** *A POVM is a collection $\{M_i\}_i$ of measurement operators satisfying*

$$\sum_i M_i^\dagger M_i = \mathrm{id} \tag{2.4}$$

Applying a POVM to a normalised state ρ gives the result i with probability

$$p(\rho, i) \coloneqq \mathrm{Tr}[M_i \rho M_i^\dagger] \tag{2.5}$$

and the state of the system after the measurement is

$$\rho' \coloneqq \frac{M_i \rho M_i^\dagger}{\mathrm{Tr}[M_i \rho M_i^\dagger]} \tag{2.6}$$

If the measurement operators are projectors onto orthogonal subspaces (such as the case for orthonormal basis measurements) then the measurement map from Equation 2.6 will be idempotent - after applying it once, we can continue to apply it without changing the state or outcome.

2.5 No-Go Theorems

Here, we present three no-go theorems that are useful in the remainder of this thesis.

The no-cloning theorem [51] states the impossibility of a physical process which takes a single token of a state ρ and produces two tokens $\rho \otimes \rho$. Data is constantly being copied in classical computers, whether it be through fetching instructions or data from registers/RAM/storage, performing any data analysis whilst retaining the data, or explicitly copying states within some algorithm. However, we find that the map which copies every quantum state is not linear, so no physical process can realise this map. A similar result holds for classical probability distributions - given one source of a distribution, we cannot produce two independent sources. The difference between the probabilistic and quantum cases is that we can broadcast a probability distribution, where each output appears to act like the input distribution locally (ignoring the other output). Broadcasting of states in this sense is not viable with quantum data [10]. In a higher-order context, we find similar no-cloning theorems which prevent us using a single instance of a channel to perfectly simulate the effect of running it on two inputs in parallel [14] or iterating it multiple times on a single input [46].

When we move into higher-order processing, we start to become interested in control structures as our computational primitives. In classical computing, one of the most significant control structures is conditional execution - the controlled execution of some program dependent on some state. There are two ways this can be mapped into a quantum environment: either some test state can be measured and the outcome used to classically control the execution, or the conditional execution is done coherently with the control system retaining any superposition. The latter case gives rise to the idea of a controlled unitary which is the process with Kraus operator $|0\rangle\langle 0| \otimes \mathrm{id} + |1\rangle\langle 1| \otimes U$ for some given unitary U. Whilst each controlled-unitary is a valid quantum process, the no-controlling theorem [6] shows the impossibility of a physical process which transforms an arbitrary $\hat{U}$ into the corresponding controlled unitary. A similar result has been shown for the impossibility of a process producing a fixed superposition of two pure input states [36].

For higher-order processing on classical computers, we can encode instructions as data, so it suffices to build our computers to operate only on simple first-order data units (i.e. bits). Unfortunately, the no-programming theorem [34] means we cannot encode arbitrary channels in finite-dimensional systems - or rather, to reliably encode one of n possible processes, we need to use a Hilbert space with at least n dimensions. Whilst the Choi-Jamiołkovski isomorphism provides such a mechanism

for the purposes of modelling processes, this does not provide a practical encoding scheme for physical processes since we cannot perform the inverse operation with certainty.

Chapter 3

Signalling Structures

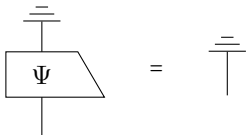
Drawing arbitrary connections between the inputs and outputs of processes could allow us to construct descriptions of circuits which send information into the past with perfect accuracy. However, relativistic physics does not permit information to be sent out of the future light cone of its origin, let alone into the past. At a mathematical level, this constraint is captured by the definitions of causal processes and associated signalling constraints.

3.1 Causality

Every Hilbert space has a unique associated map, called the discarding map, which corresponds to the action of discarding a system living in that space.

Definition 3.1.1 (Discarding Map)**.** *The discarding map* $\bar{\top}_{\mathcal{H}} : \mathcal{L}(\mathcal{H}) \to \mathcal{L}(\mathbb{C})$ *for a given Hilbert space* $\mathcal{H}$ *acts by applying the trace operator to the state -* $\bar{\top}_{\mathcal{H}}(\rho) \coloneqq \mathrm{Tr}[\rho]$*. Applying a discarding map locally on a larger system corresponds to applying a partial trace -* $(\bar{\top}_{\mathcal{H}} \otimes \mathrm{id}_{\mathcal{K}})(\rho) = \mathrm{Tr}_{\mathcal{H}}[\rho]$*.*

Definition 3.1.2 (Causal Process)**.** *A process is causal when applying it and discarding all outputs gives the same effect as discarding all inputs.*



(3.1)

This is exactly the class of processes which can be implemented physically with certainty (as opposed to non-deterministic processes which yield multiple operations, each occurring with some probability). The definition is equivalent to saying that the process is trace-preserving, or (for pure processes) an isometry. As a special case of the causality property, a state is causal iff it is normalised.

A higher-order process is causal when it transforms causal processes into causal processes. For example, we say that a process is second-order if it transforms processes of $A_i \to A_o$ to processes of $B_i \to B_o$. Second-order causality is satisfied by processes $\Phi : (A_i \to A_o) \to (B_i \to B_o)$ where, for all causal $\Psi : A_i \to A_o$, $\Phi(\Psi)$ is causal.

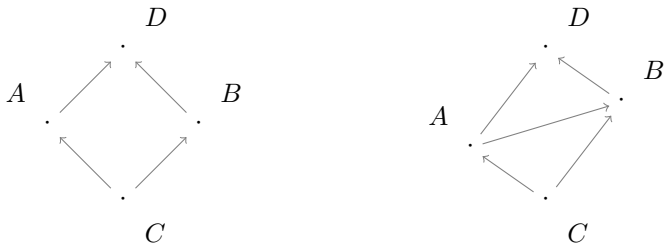


Figure 3.1: Some causal structures in which processes A and B are non-signalling (left) and one-way signalling $A \prec B$ (right). In both cases, it is possible for them to have some shared past C and future D. The graph on the right is an example of a linear causal structure with the unique ordering of events being $C \prec A \prec B \prec D$.

3.2 Causal Structures

We can picture each quantum process as acting in some region of spacetime. Suppose we break all regions down to a collection of specific points called events. Special relativity tells us that each event can only influence those in its future light cone (the region of space-time that can be accessed by travelling no faster than light) and can only be influenced by those in its past light-cone (the region from which it is accessible by travelling no faster than light). We can then construct the relation of possible influences between the events. This gives us a restriction on how we can connect the set of processes together to form a valid circuit.

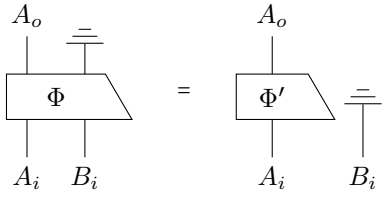
Mathematically, we can represent the relation between spacetime points that can signal to one another by a partial order. Given a discrete model of spacetime, this can be drawn as a directed acyclic graph. Such graphs are referred to as causal structures or quantum causal networks [16]. In particular, linear causal structures are those where there is a unique ordering of the events.

A circuit satisfies a given causal structure if all wires in the circuit lie on edges of the graph. It is possible for each circuit to satisfy multiple causal structures.

3.3 One-Way Signalling

The most common signalling constraints can be examined by considering a bipartite process from a system $A_i \otimes B_i$ to $A_o \otimes B_0$, viewed as a pair of (possibly connected) channels at locations A and B. In the general case, we can imagine that both inputs could influence both outputs. One-way signalling ($A \prec B$) is a property that states the independence of the input of one side (B) of the bipartite process and the output of the other (A) [11, 15].

Definition 3.3.1 (One-Way Signalling)**.** *A causal process $\Phi : A_i \otimes B_i \to A_o \otimes B_o$ is one-way signalling with $A \prec B$ if discarding B_o permits a factorisation of the form:*



(3.2)

Definition 3.3.2 (Non-Signalling)**.** *A causal process $\Phi : A_i \otimes B_i \to A_o \otimes B_o$ is non-signalling when it satisfies the one-way signalling condition with both $A \prec B$ and $B \prec A$.*

It is important to note that the definition of non-signalling is different from that of a product/separable state. It is entirely possible that A and B have some shared cause in the past (as in Figure 3.1) which allows them to be entangled without any signalling occurring between them.

3.4 Causal Categories

Categories are well-known for their generality as mathematical objects and numerous applications across Computer Science; quantum theory is no exception to this. Monoidal categories permit clear diagrammatic reasoning where any combination of the associators and unitors corresponds to a simple graphical identity. Dagger compact closed categories [2] have sufficient structure to represent the process-state duality exhibited by the Choi-Jamiołkovski isomorphism. On top of this, we can build the CPM construction [43] which exactly captures the set of completely positive maps. Finally, Frobenius algebras give rise to classical structures (representing orthonormal bases) which form the basis of the ZX-calculus [20], providing a way of reasoning the equivalence of programs that is complete for the stabilizer fragment [8] and can be extended for completeness of qubit computation [33].

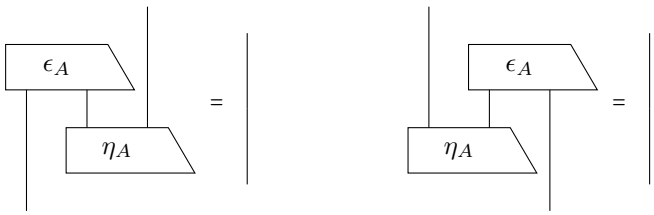
This section covers some of the content of [29] which presents the Caus construction and gives proofs on the representation of signalling constraints in the objects of the category.

3.4.1 The Caus Construction

We will start with compact closed categories and progressively build up more structure. For clarity, we use implicit associators and unitors for the monoidal structure.

Definition 3.4.1 (Compact Closed Category)**.** *A symmetric monoidal category $\mathcal{C}$ with tensor unit I is compact closed if, for every object A, we have a dual object A^* and morphisms $\eta_A : I \to A^* \otimes A$ and $\epsilon_A : A \otimes A^* \to I$ satisfying:*

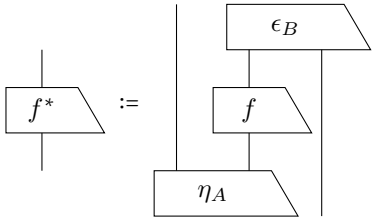
$$(\epsilon_A \otimes \mathrm{id}_A) \circ (\mathrm{id}_A \otimes \eta_A) = \mathrm{id}_A \qquad (\mathrm{id}_{A^*} \otimes \epsilon_A) \circ (\eta_A \otimes \mathrm{id}_{A^*}) = \mathrm{id}_{A^*} \tag{3.3}$$



In every compact closed category, the tensor unit I is isomorphic to its dual ($I \cong I^*$) and we generally identify them. We can use these dualities to introduce the transposition functor.

Definition 3.4.2 (Transposition Functor)**.** *For a compact closed category $\mathcal{C}$, we define the transposition functor $(-)^* : \mathcal{C}^{op} \to \mathcal{C}$ mapping objects A to their duals A^**

and morphisms $f : A \to B$ *to their transpose* $f^* := (\mathrm{id}_{A^*} \otimes \epsilon_B) \circ (\mathrm{id}_{A^*} \otimes f \otimes \mathrm{id}_{B^*}) \circ (\eta_A \otimes \mathrm{id}_{B^*}) : B^* \to A^*$.



(3.4)

The dualities allow us to represent a process $\Phi : A \to B$ as a state $(\Phi \otimes \mathrm{id}_{A^*}) \circ \eta_A : I \to B \otimes A^*$, generalising the Choi-Jamiołkovski isomorphism. We then obtain higher-order processes as morphisms between such types. For example, a second-order process from $A_i \to A_o$ to $B_i \to B_o$ is a morphism of $A_o \otimes A_i^* \to B_o \otimes B_i^*$. We will continue to draw these as combs for clarity.

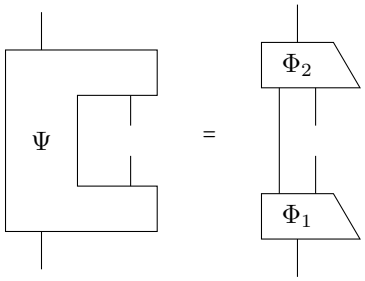
Definition 3.4.3 (Precausal Category)**.** *A compact closed category is precausal if:*

- *It has a discarding process for every system satisfying:*

$$\overline{\top}_{A \otimes B} = \overline{\top}_A \otimes \overline{\top}_B \qquad (3.5) \qquad\qquad \overline{\top}_I = 1 (= \mathrm{id}_I) \qquad (3.6)$$

- *The dimension* $\dim(A) := \overline{\top}_A \circ \overline{\top}_A^*$ *of any non-zero system* A *is an invertible scalar.*
- *It has "enough causal states" (if* $f \circ \rho = g \circ \rho$ *for all causal* ρ*, then* $f = g$*).*
- *Second-order causal processes factorise as:*

$\exists \Phi_1, \Phi_2$ *causal* .



(3.7)

We know that signalling and non-signalling processes occupy the same Hilbert space, so in order to distinguish between them we need to consider more specific sets of processes. We introduce causal sets to solve this problem.

Definition 3.4.4 (Dual Sets)**.** *For any set* $c \subseteq \mathcal{C}(I, A)$*, its dual set* $c^* \subseteq \mathcal{C}(I, A^*)$ *is defined as:*

$$c^* := \{\pi : A^* | \forall \rho \in c . \pi^* \circ \rho = 1\} \qquad (3.8)$$

Furthermore, c *is closed if* $c = c^{**}$ *and it is flat if both* c^* *contains* $\overline{\top}_A$ *and* c *contains* $\overline{\top}_A^*$ *up to invertible scalars.*

From this definition, we can show that $c \subseteq c^{**}$ always holds, and $c^* = c^{***}$ (hence $(-)^{**}$ is an idempotent "closure" operation).

The category Caus[$\mathcal{C}$] of higher-order causal processes for $\mathcal{C}$ can now be defined:

Definition 3.4.5 (Caus Construction)**.** *For a precausal category $\mathcal{C}$, the objects of* Caus$[\mathcal{C}]$ *are pairs $\mathbf{A} \coloneqq (A, c_{\mathbf{A}})$ where the object's causal set $c_{\mathbf{A}} \subseteq \mathcal{C}(I, A)$ is closed and flat. Morphisms $f : \mathbf{A} \to \mathbf{B}$ are morphisms $f \in \mathcal{C}(A, B)$ such that $\forall \rho \in c_{\mathbf{A}}. f \circ \rho \in c_{\mathbf{B}}$.*

Caus$[\mathcal{C}]$ is a symmetric monoidal category with the tensor product defined by:

$$\mathbf{A} \otimes \mathbf{B} \coloneqq (A \otimes B, (c_{\mathbf{A}} \otimes c_{\mathbf{B}})^{**}) \tag{3.9}$$

where $c_{\mathbf{A}} \otimes c_{\mathbf{B}} \coloneqq \{\rho_1 \otimes \rho_2 | \rho_1 \in c_{\mathbf{A}}, \rho_2 \in c_{\mathbf{B}}\}$, and the tensor unit is $\mathbf{I} \coloneqq (I, \{1\})$. We also find dual objects as $\mathbf{A}^* \coloneqq (A^*, c_{\mathbf{A}}^*)$.

It is important to note that CP(**FHilb**) is a precausal category, so it is useful to consider Caus[CP(**FHilb**)] when discussing quantum processes.

3.4.2 $\parr$ and $\multimap$

One important property of this construction is that duals do not distribute over the tensor product; that is, $(c_{\mathbf{A}} \otimes c_{\mathbf{B}})^* \neq c_{\mathbf{A}}^* \otimes c_{\mathbf{B}}^*$. We can bridge this gap by adding a new definition for the "par" of two systems $\mathbf{A} \parr \mathbf{B}$ as the De Morgan dual of the tensor product:

$$c_{\mathbf{A} \parr \mathbf{B}} \coloneqq (c_{\mathbf{A}}^* \otimes c_{\mathbf{B}}^*)^* = \{\rho : A \otimes B | \forall \pi \in c_{\mathbf{A}}^*, \xi \in c_{\mathbf{B}}^*. (\pi^* \otimes \xi^*) \circ \rho = 1\} \tag{3.10}$$

Informally, the definition of $c_{\mathbf{A} \otimes \mathbf{B}}$ lets it be the set of processes which can be embedded in environments matching any causal structure. There should not be any signalling between the $\mathbf{A}$ and $\mathbf{B}$ components as the environment could add signalling in the opposite direction, introducing a causal loop. On the other hand, processes in $c_{\mathbf{A} \parr \mathbf{B}}$ only need to interact nicely with local effects that cannot introduce signalling between $\mathbf{A}$ and $\mathbf{B}$. This means we are free to have any causal relationship between them. Naturally, Caus$[\mathcal{C}]$ has a canonical embedding $\mathbf{A} \otimes \mathbf{B} \to \mathbf{A} \parr \mathbf{B}$.

Definition 3.4.6 (First-Order System)**.** *A system $\mathbf{A} = (A, c_{\mathbf{A}})$ in* Caus$[\mathcal{C}]$ *is first-order if it is of the form $(A, \{\overline{\top}_A\}^*)$.*

Lemma 3.4.7. *For any first-order systems $\mathbf{A}, \mathbf{B}$ in* Caus$[\mathcal{C}]$*, $\mathbf{A} \otimes \mathbf{B} \cong \mathbf{A} \parr \mathbf{B}$.*

This lemma matches our intuition regarding first-order processes as those with trivial inputs - there is only one causal state for the trivial system, so there is no information to signal.

We now have a way to represent functions by the internal hom $\mathbf{A} \multimap \mathbf{B} \coloneqq (\mathbf{A} \otimes \mathbf{B}^*)^* \cong \mathbf{A}^* \parr \mathbf{B}$. This may be read as "processes which eliminate something of type $\mathbf{A}$ and can signal information to an output of type $\mathbf{B}$". Using these operators, we can now construct objects of higher-order types which appropriately describe the signalling between the subsystems.

3.4.3 Signalling Constraints as Types

In the context of causal categories, we say that a process is "of type $\mathbf{A}$" when it is a morphism in Caus$[\mathcal{C}](\mathbf{I}, \mathbf{A})$. The following lemmas are proved in [29] and confirm our informal arguments about the relationships between $\otimes$, $\parr$, and signalling. Let $\mathbf{A}, \mathbf{A}', \mathbf{B}, \mathbf{B}'$ be first-order systems.

Lemma 3.4.8. *A bipartite process* Φ *is of type* $(\mathbf{A} \multimap \mathbf{A}') \otimes (\mathbf{B} \multimap \mathbf{B}')$ *iff it is causal and non-signalling.*

Lemma 3.4.9. *A bipartite process* Φ *is of type* $\mathbf{A} \multimap (\mathbf{A}' \multimap \mathbf{B}) \multimap \mathbf{B}'$ *iff it is causal and one-way signalling with* $A \prec B$*.*

Lemma 3.4.10. *A bipartite process* Φ *is of type* $(\mathbf{A} \multimap \mathbf{A}') \parr (\mathbf{B} \multimap \mathbf{B}')$ *iff it is causal.*

These results generalise to n-partite processes. In particular, one-way signalling with $A_1 \prec \ldots \prec A_n$ is equivalent to the type $\mathbf{A_1} \multimap (\mathbf{A'_1} \multimap (\ldots) \multimap \mathbf{A_n}) \multimap \mathbf{A'_n}$ which is the general type for an n-comb [13, 16]. This provides a correspondence between the quantum comb formalism and linear causal structures.

3.5 Indefinite Causal Order

If we are given two channels $F, G : \mathcal{L}(\mathcal{H}) \to \mathcal{L}(\mathcal{H})$, there are three ways of combining them classically: $F \circ G$, $G \circ F$, and $F \otimes G$. These correspond to the three possible directed acyclic graphs/causal structures that can be constructed with two points. Choosing one of these combinations dependent on some test produces a process which will not necessarily fit any of these causal structures until the result of the test is determined. If, instead of a classical test, we let this be a coherent quantum controlled-operation, we can obtain a superposition of causal orders of events.

The quantum switch [18] is a quantum process which creates exactly this superposition of orders. It takes two black-box channels and composes them such that the order is controlled coherently by some qubit. The switch is formally defined to be the operation taking channels F, G with some Kraus operators $\{f_i\}_i, \{g_j\}_j$, and produce the channel with Kraus operators:

$$w_{i,j} \coloneqq (g_j \circ f_i) \otimes |0\rangle\langle 0| + (f_i \circ g_j) \otimes |1\rangle\langle 1| \tag{3.11}$$

It should be noted that the switch is only valid when the pair of channels provided are non-signalling. For example, attempting to apply it over the two sides of a swap gate will immediately create a causal loop, even if we fix it to one of the classical orders [30]. Lemma 3.4.8 tells us that the process combining two channels of type $\mathbf{A} \multimap \mathbf{A}$ in a fixed order should be of type $(\mathbf{A} \multimap \mathbf{A}) \otimes (\mathbf{A} \multimap \mathbf{A}) \multimap (\mathbf{A} \multimap \mathbf{A})$. The type of the quantum switch is $\mathbb{C}^2 \multimap \mathbb{C}^2 \parr ((\mathbf{A} \multimap \mathbf{A}) \otimes (\mathbf{A} \multimap \mathbf{A}) \multimap (\mathbf{A} \multimap \mathbf{A}))$. We examine this claim in more detail in Section 6.2.

Several physical embodiments for a switch have been proposed in the literature. Supposing we have wires that can be moved in accordance with the quantum control system, we can arrange them to connect the two input boxes in one order or the other [18]. Strong gravitational forces are capable of tilting the light cones of nearby processes, so by carefully positioning a large mass closer to one process than the other we can control the order in which they occur; entangling the position of this mass with some control system gives the gravitational switch [54]. Laboratory experiments [38, 41, 40] have been able to construct a switch using optical circuits where the control qubit is encoded in the spatial trajectory of a photon and the gates applied affect the photon's polarisation; although, controversially, similar setups [25] have been shown to circumvent the no-controlling theorem. The switch can also be simulated if we have multiple copies of the pure transformations [18].

The quantum switch offers numerous advantages in computation. For example, inserting commuting or anti-commuting channels into a switch induces a phase change on the control qubit which can be measured. This allows for perfect discrimination between the two cases which is not possible when executing them in a fixed order [12]. We also find that, using a switch with a fixed control system, we can construct a channel with non-zero capacity from completely depolarising channels [22], exceeding the bounds predicted by standard information theory.

Chapter 4

Quantum Programming

4.1 Practical Programming

The recent development of quantum computers has tended to fall in line with the QRAM (Quantum Random Access Machine) model [31], whereby each machine instruction corresponds to an elementary operation (state preparation, unitaries, measurement, etc.) on any of the quantum registers. In practice, they tend to be implemented as a coprocessor for a classical device; the host system sends off lists of instructions to the quantum computer in a batch, which then executes them and returns the results of measurements when complete.

Since the turn of the century, a number of quantum programming languages have been proposed. [26] provides a good survey of the early developments in this area. A large number took direct inspiration from or are embedded in popular classical languages, such as QCL [35], Quipper [28], and LIQ$Ui|\rangle$ [50]. These practical languages tend to be designed for completeness with respect to the quantum circuit model, with Quipper extending this to classical-quantum circuits with classically-controlled quantum gates. However, since processes exhibiting indefinite causal structures such as the quantum switch cannot be constructed by any quantum circuit [18], we would need to extend the model in order to capture this.

Several quantum programming languages have given rise to new concepts and type-theoretic constructs for solving problems in quantum computing. The functional language QML [3] introduced the notion of quantum control flow and used a type system inspired by linear logic to prevent the copying of quantum information. Additionally, in the language of [37], we see two connectives for describing pairs of objects, distinguishing between separable and potentially entangled states.

4.2 Modelling Languages

Many of the languages defined for quantum programming were not intended to be convenient for practical use, but instead focussed on other goals such as minimality or the ability to perform model checking.

Lambda calculi, such as those of Van Tonder [49], Selinger and Valiron [45], and Arrighi and Dowek [7], aim to provide a complete model of computation with a minimal syntax and well-defined semantics. They provide a great setting for demonstrating new ideas for type systems or for observing links with logics and

category theory. Such a link exists in the classical case with a correspondence between the simply-typed lambda calculus and cartesian-closed categories [32]. A particular example from [45] is the use of the ! modality to represent copyable objects such as classical data.

Languages such as CQP [27] and the extension of PRISM used by QPMC [24] were designed with the intention of enabling automated verification. In the latter case, the programs are converted to quantum Markov chains and checked against properties described in quantum computational tree logic [23].

Other languages have often been constructed to consider new and interesting concepts in quantum computing. Ying [52] discusses recursively defined programs where the internal control flow is quantum. Under these circumstances, the number of recursive calls the program can make is potentially indefinite. This model uses Fock spaces to handle the undetermined number of potential control systems used by the quantum controlled operations. More recently, [42] provided an alternative view on the construction of programs with quantum recursion using pattern-matching isomorphisms and purely quantum fixpoints.

4.3 Quantum Control Flow

When adopting classical control flow in programs, each run of a program follows a definite computation path. Each branching point in the tree of computation paths is a conditional test - a test state is measured and we take the branch corresponding to the measurement outcome. As mentioned in previous sections, we can (in principle) extend this idea to taking superpositions of the computation paths coherently with the test state, giving a quantum control flow.

This idea was introduced in the QML programming language [3] with the $\mathbf{case}^\circ$ statement. This takes any state of type $A \oplus B$, then runs one branch on the A portion and the other branch on the B portion, preserving any superposition. It requires each of the branches to produce orthogonal first-order states and for each branch to be pure in order for the resulting process to be an isometry. For example, the Hadamard gate can be defined by the program:

$$\begin{aligned} had\ c\ =\ & \mathbf{case}^\circ\ c\ \mathbf{of} \\ & \left\{\mathbf{inl}\ x \Rightarrow \left\{\left(\frac{1}{\sqrt{2}}\right)\ \mathbf{inl}\ x \mid \left(\frac{1}{\sqrt{2}}\right)\ \mathbf{inr}\ x\right\}\right. \\ & \left.\mid \mathbf{inr}\ y \Rightarrow \left\{\left(\frac{1}{\sqrt{2}}\right)\ \mathbf{inl}\ y \mid \left(-\frac{1}{\sqrt{2}}\right)\ \mathbf{inr}\ y\right\}\right\} \end{aligned}$$

A simple way to enforce orthogonality of the branches is to rebuild the decomposed state, giving programs of the form $\mathbf{case}^\circ\ c\ \mathbf{of}\ \{\mathbf{inl}\ x \Rightarrow (\mathbf{inl}\ x, F) \mid \mathbf{inr}\ y \Rightarrow (\mathbf{inr}\ y, G)\}$. This is often referred to as quantum alternation of programs F and G [9, 53]. We can view this as a bipartite process with one side working on the control system (the input c) and the other on an affected system (that which F and G can modify). This is reflected by the Kraus operator of the alternation being $P_0 \otimes F + P_1 \otimes G$ (where P_0 and P_1 are projectors onto orthogonal subspaces spanning the control system). In most cases, we assume that the control system is a qubit, though this need not be the case in general.

A common example is the notion of a controlled-unitary, which is the alternation of a given unitary and the identity map. Despite controlled-unitaries being admissible for any unitary, the no-controlling theorem (as discussed in Section 2.5) states that there is no process taking a single black-box instance of a unitary and producing the corresponding controlled-unitary.

Even with quantum alternation, we require each of the programs being combined to be pure. In theory, we could propose a definition for the alternation of mixed processes based on Kraus operators, taking $\{f_i\}_i$ and $\{g_j\}_j$ and returning $\left\{\frac{1}{\sqrt{|\{g_j\}_j|}}P_0 \otimes f_i + \frac{1}{\sqrt{|\{f_i\}_i|}}P_1 \otimes g_j\right\}_{ij}$. [9] shows why this fails outside of the pure case, where this matches the previous definition.

Quantum alternation is closely related to the matrix direct sum: given pure processes $\hat{\psi_0} : \mathcal{L}(\mathcal{H}_0) \to \mathcal{L}(\mathcal{K}_0)$ and $\hat{\psi_1} : \mathcal{L}(\mathcal{H}_1) \to \mathcal{L}(\mathcal{K}_1)$, their direct sum is the process with block matrix form of Equation 4.1.

$$[\hat{\psi_0} \oplus \hat{\psi_1}](\rho) \coloneqq \begin{pmatrix} \psi_0 & 0 \\ 0 & \psi_1 \end{pmatrix} \rho \begin{pmatrix} \psi_0 & 0 \\ 0 & \psi_1 \end{pmatrix}^\dagger : \mathcal{L}(\mathcal{H}_0 \oplus \mathcal{H}_1) \to \mathcal{L}(\mathcal{K}_0 \oplus \mathcal{K}_1) \tag{4.1}$$

In fact, when $\mathcal{H}_0 = \mathcal{H}_1$ and $\mathcal{K}_0 = \mathcal{K}_1$, this is equivalent to the qubit-controlled alternation of $\hat{\psi_0}$ and $\hat{\psi_1}$. To see this, we note that $\mathcal{H} \oplus \mathcal{H} \cong (I \otimes \mathcal{H}) \oplus (I \otimes \mathcal{H}) \cong (I \oplus I) \otimes \mathcal{H} \cong \mathbb{C}^2 \otimes \mathcal{H}$ via unitality and distributivity. We can then observe that the Kraus operator of the direct sum process is exactly $|0\rangle\langle 0| \otimes \psi_0 + |1\rangle\langle 1| \otimes \psi_1$. Quantum alternation with more general control systems can arise from the matrix direct sum where $\mathcal{H}_0$ and $\mathcal{H}_1$ are not identical, but possibly share some subspace.

We find that the quantum switch is given by quantum alternation of the higher-order maps denoted by $\lambda x.\lambda y.\lambda z.x(yz)$ and $\lambda x.\lambda y.\lambda z.y(xz)$. The switch and its direct generalisation to n channels only give superpositions over linear causal structures, though one can imagine a higher-order version of the **case**$^\circ$ statement allowing more complex scenarios with indefinite causal ordering.

4.4 Case Study: Quantum Lambda Calculus

Motivation for the work presented in this thesis came from evaluating the quantum lambda calculus by Selinger and Valiron [45]. Rather than capturing the entire program and state in a single term like other lambda calculi, this uses a larger structure called the quantum closure. Each closure is a tuple $[Q, L, M]$ consisting of a vector $Q \in \otimes_{i=1}^{n}\mathbb{C}^2$, a list L of n variables, and a term M with free variables in L. This resembles the QRAM model: Q describes the state of a collection of qubit registers (called the quantum array) which are referenced by the program described by M.

Whilst the quantum lambda calculus introduced a number of innovative features, this section presents some shortcomings of the language to highlight the importance of the specific developments made in this thesis.

Selinger and Valiron's aim was to create a complete model of qubit quantum computing with classical control structures. In the advent of theoretical benefits [12, 22, 5] and experimental results [38, 41, 40] relating to indefinite causal orders of events, there is interest in adding these into our computational models. Whilst the choice of classical control flow makes the calculus easy to understand and provide physical realisations of programs, this restriction to the quantum circuit model

prevents the construction of the quantum switch and indefinite orders [18]. More generally, in the interest of introducing quantum control flow, constructing a controlled unitary from a classical representation of the corresponding unitary would require substantial knowledge of the specific set of elementary gates used in the language.

A prerequisite to introducing the quantum switch or a mechanism for alternation would be the reinterpretation of the $\oplus$ operator from classical sum types to the direct sum of the subsystem Hilbert spaces, or, more specifically, allowing arbitrary superpositions of states of the same type. This presents us with a choice for how to decompose values of type $A \oplus B$: coherent decomposition would preserve any superposition (as in QML's **case**$^\circ$ statement), whilst decoherent decomposition would add a measurement which projects the value to either an object of type A or one of B.

The main reason for focussing on qubit computation is in the simplicity, scalability, and convenience of qubit computers (especially given the number of physical systems which can be used to encode them), in much the same way that classical computers operating on bits are often easier to design and more reliable than using larger units. From the perspective of quantum mechanics, however, systems with two degrees of freedom are not particularly special. We also know from the no-programming theorem that first-order systems alone are not sufficient for representing higher-order state in a reliable and compact manner. One could imagine a more general language would permit higher-order mixed processes over any finite-dimensional Hilbert space to be embedded into programs, as opposed to the current model with only pure states from $\mathbb{C}^2$.

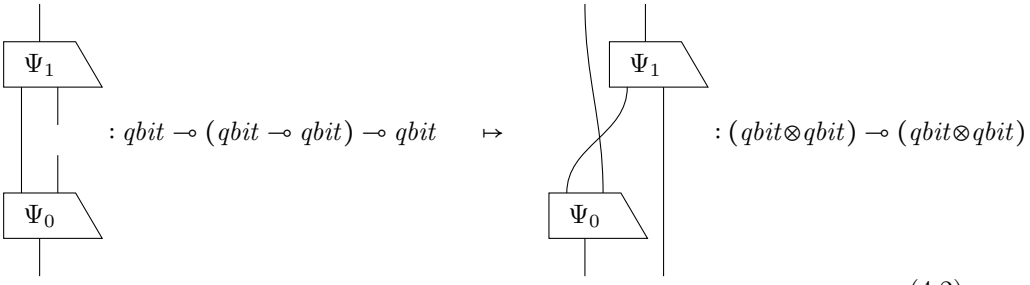
The semantics of qubit measurement in the quantum lambda calculus appears as a demolition measurement from the perspective of the lambda term, but the measured qubit persists in the environment. On the one hand, this allows us to keep track of the measurement results, giving each computation path a unique signature recording all instances of non-deterministic actions taken. However, this also means the use of ancillas has a lasting effect on the state of the program which is generally not desirable from a programmer's perspective.

Furthermore, this calculus is missing some canonical morphisms between types which occur in the standard categorical interpretations. In particular, we note that the relation between isomorphic types is not a congruence. For example, we can exhibit an isomorphism between $qbit$ and $\top \multimap qbit$ where $\top$ denotes the type of a trivial system. There are no terms of type $!qbit$ as this would indicate the presence of copyable quantum data, going against the no-cloning theorem. However, there do exist terms of type $!(\top \multimap qbit)$, such as the program generating a qubit in the standard state $|0\rangle\langle 0|$. Clearly, no isomorphism can exist here between $!qbit$ and $!(\top \multimap qbit)$.

The type system of the quantum lambda calculus is not rich enough to describe signalling to the extent discussed in section 3.4.3. We cannot identify A^* with the type $A \multimap \top$ as we could not construct the canonical map from $(A^*)^*$ to A. As a result, we are not able to encode $A \parr B$ as the De Morgan dual of $\otimes$ via $((A \multimap \top) \otimes (B \multimap \top)) \multimap \top$.

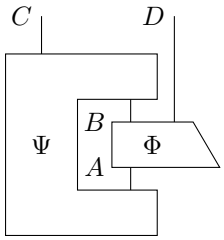
Even when types exist which represent some signalling constraints, we may not have all canonical morphisms between them. For instance, there is no morphism which takes the 2-comb of qubits $(qbit \multimap (qbit \multimap qbit) \multimap qbit)$ to the generic bipartite channel type $(qbit \otimes qbit \multimap qbit \otimes qbit)$ which does not involve discard-

ing systems and replacing them by fixed states, despite the canonical morphism appearing trivial in graphical notation:



(4.2)

This can be solved by allowing the local use of a term $\Phi : A \multimap B \otimes D$ as the input to $\Psi : (A \multimap B) \multimap C$ as shown below.



(4.3)

Suppose Ψ is the comb built from Ψ_0 and Ψ_1 in the example (4.2) and Ψ is a swap gate (both with an input state provided). This diagram then allows us to build the solution on the right hand side of 4.2. Moreover, local application in this manner can be used to solve the problem of identifying A^* with $A \multimap \top$.

Chapter 5

A Programming Language with Indefinite Causal Order

This chapter covers the primary novel contribution of this thesis: a programming language capable of describing processes with indefinite causal structure. We adopt a collection of elementary higher-order circuits and provide methods of composing them, following the operations available in a laboratory setup. Given the physical admissibility of indefinite causal orders, we add a higher-order quantum switch as a primitive construction.

This language addresses the issues covered in Section 4.4:

- We enable arbitrary higher-order quantum processes to be embedded into programs, using the Choi-Jamiołkovski isomorphism to describe the state of a higher-order register in a succinct manner.
- We provide a restricted model of quantum direct sums - restricted in the sense that we can only achieve superposition of processes with linear types (not containing !).
- We introduce the $\parr$ operator to the language of types to express compound systems with potential for signalling. Objects of this form can then be handled by local application, or converted to a pair type if both subsystems are first-order.

5.1 Computational Model

At any point in a laboratory experiment, there will be a collection of physical quantum systems and a sequence of instructions still to be completed. In this programming language, we say that each of these physical systems is contained within a quantum register. We refer to the collection of registers as the quantum array. We assume that each quantum register can store any higher-order process as a black-box, including first-order states and channels as special cases. Each quantum register is assumed to be completely disconnected from the others in the sense that there is no physical connection between them that permits signalling, although entanglement between the contents of registers is allowed. When we introduce the semantics of the language, we will represent the state of the quantum array by the combined Choi-Jamiołkovski operator over the registers.

We supplement this state by a term of the language representing the program to be executed. The quantum registers are labelled by variables which may be referenced within the program.

In Section 5.5, we provide operational semantics for the language in the form of a reduction relation. This breaks computation down into individual elementary steps, such as applying a process to an input state or measuring a system. Computation stops once we reach a value.

A key feature is the ability to take a program which describes some quantum state or process and build it into a quantum register. In particular, we build processes by starting with an identity channel and applying the process to the output. This breaks down the procedure of building a circuit into multiple, simpler computation steps. In any implementation of the programming language, we are given a collection of elementary circuits which we can build into a quantum register in a single step.

The only point in a program at which we may proceed non-deterministically is when performing a measurement. We only provide measurements with two outcomes corresponding to projecting a system living in a Hilbert space $\mathcal{H}_A \oplus \mathcal{H}_B$ into either $\mathcal{H}_A$ or $\mathcal{H}_B$. If more than two measurement outcomes are required, multiple measurements can be used in succession.

With regards to terminology, in the following we will refer to processes and circuits synonymously. We do not assume they necessarily fit the traditional quantum circuit model as they may include processes exhibiting indefinite causal order such as the quantum switch.

5.2 Type Hierarchy

This language utilises a hierarchy of type classes, with certain operations only valid on the more specific classes. General types $(A, B, \ldots)$ encompass all types representable within the language. Quantum types $(\hat{A}, \hat{B}, \ldots)$ correspond to the physical processes which we can hold in a quantum register. Such processes can be described by a Choi-Jamiołkovski operator. Finally, the first-order types $(\tilde{A}, \tilde{B}, \ldots)$ are those quantum types which contain no inputs, and so we allow the construction of the isomorphism $\tilde{A} \otimes \tilde{B} \cong \tilde{A} \parr \tilde{B}$ as in Lemma 3.4.7.

Definition 5.2.1. *The sets of general, quantum, and first-order types are respectively defined by the following abstract syntaxes:*

$$
\begin{aligned}
A, B &::= \top \mid A \otimes B \mid A \oplus B \mid A \parr B \mid A \multimap B \mid !A \\
\hat{A}, \hat{B} &::= \top \mid \hat{A} \otimes \hat{B} \mid \hat{A} \oplus \hat{B} \mid \hat{A} \parr \hat{B} \mid \hat{A} \multimap \hat{B} \\
\tilde{A}, \tilde{B} &::= \top \mid \tilde{A} \otimes \tilde{B} \mid \tilde{A} \oplus \tilde{B}
\end{aligned}
$$

Qubits are represented in this type system as the first-order type $\top \oplus \top$. These type operators are not strictly symmetric/associative/distributive - for instance, qutrits (systems in a superposition over three orthogonal states) can be either $(\top \oplus \top) \oplus \top$ or $\top \oplus (\top \oplus \top)$ and the distinction is crucial for specifying a unique measurement operation. If we have the qutrit $p\,|0\rangle\langle 0| + q\,|1\rangle\langle 1| + r\,|2\rangle\langle 2|$ as type $(\top \oplus \top) \oplus \top$ and measure it, we will either get $p\,|0\rangle\langle 0| + q\,|1\rangle\langle 1|$ or $r\,|2\rangle\langle 2|$, whereas measuring it as $\top \oplus (\top \oplus \top)$ will give either $p\,|0\rangle\langle 0|$ or $q\,|1\rangle\langle 1| + r\,|2\rangle\langle 2|$.

5.3 Higher-Order Quantum Switch

The quantum switch is an exotic computational resource which permits two channels to be composed in a superposition of classical orders, mediated by an external control system. The traditional definition requires the channels to be over first-order systems, each matching some type $\tilde{A} \multimap \tilde{A}$. However, superpositions are possible in instances of any quantum type $\hat{A}$ as they are associated to finite-dimensional Hilbert spaces, and furthermore there are two canonical ways to classically compose any two processes of the form $\hat{A} \multimap \hat{A}$. This sets the stage for the higher-order switch that is presented in this programming language.

Given any process of a quantum type $\hat{A}$, we can use the Choi-Jamiołkovski isomorphism to describe it by a state $\rho \in \mathcal{L}(\mathcal{H}_{\hat{A}})$. Suppose we are given two transformations of type $\hat{A} \multimap \hat{A}$, each of which can be described as CP-maps in $\mathcal{L}(\mathcal{H}_{\hat{A}}) \to \mathcal{L}(\mathcal{H}_{\hat{A}})$ and consequently permit some Kraus decompositions $\{f_i\}_i, \{g_j\}_j$. Inserting these into a quantum switch gives the process with Kraus operators $\{w_{ij}\}_{ij}$ given by:

$$w_{ij} = (g_j \circ f_i) \otimes P_0 + (f_i \circ g_j) \otimes P_1 \tag{5.1}$$

where P_0 and P_1 are projectors onto orthogonal subspaces of the control system. This control system need not be a qubit, but instead can be any first-order system of the form $\tilde{A} \oplus \tilde{B}$, where P_0 and P_1 are the projectors onto the spaces of $\tilde{A}$ and $\tilde{B}$ respectively. When $\tilde{A} = \tilde{B} = \top$, these correspond exactly to the projectors onto the computational basis of a qubit ($P_0 = |0\rangle\langle 0|$ and $P_1 = |1\rangle\langle 1|$), as in the usual definition of the switch.

For a more concrete example, we look at the case where Φ_0 and Φ_1 are second-order processes. Figure 5.1 shows the two ways to compose such processes classically.

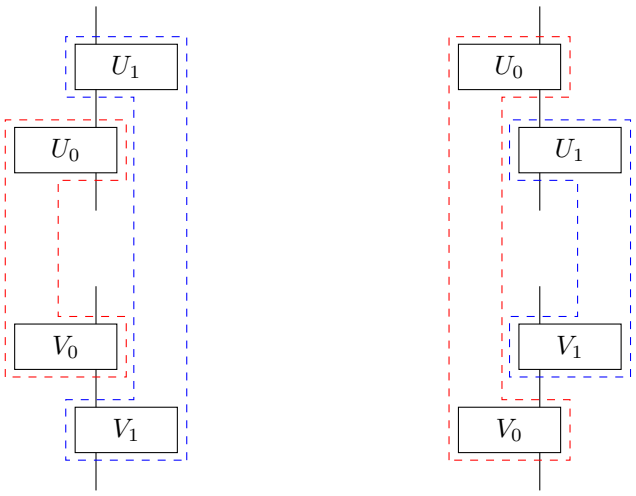


Figure 5.1: The two ways of composing second-order processes classically.

Suppose each Φ_i decomposes into a pair of unitaries U_i and V_i with no memory channel. From the diagrammatic interpretation, we can see that this could equivalently be achieved by using a pair of first-order switches, with one switching U_0 and U_1 and the other switching V_0 and V_1. These are synchronised by using the same control system for both switches (i.e. feeding the control output of one into the control input of the other), but they need to switch U_0/V_0 and U_1/V_1 in opposite

orders. This conforms with categorical interpretations of quantum mechanics where composing processes acts contravariantly on the corresponding processes between the dual objects (e.g. the Hilbert spaces for inputs). It is unknown whether every instance of a higher-order switch can be broken down into multiple first-order switches or simulated in such a manner.

We can also simulate the switch over pure second-order processes using a second copy of the processes and controlled-swap gates as shown in Figure 5.2. This construction exactly matches the simulation of the two-first order switches using the technique from [18].

5.4 Terms and Types

The terms of the language mostly follow those of other lambda calculi with some new additional terms: syntax for promotion and dereliction, the quantum switch, building circuits from their instructions, and handling $^2\mathcal{Y}$-systems.

Definition 5.4.1. *The set of terms for the language are defined by the following abstract syntax:*

$$\begin{aligned} M, N, P ::= {} & U \mid x \mid \hat{x} \mid * \mid \mathit{Choi}\ \hat{x}\ \mathit{into}\ M \mid \lambda x : A.M \mid MN \\ & \mid \langle M, N \rangle \mid \mathit{let}\ \langle x, y \rangle = M\ \mathit{in}\ N \\ & \mid \mathit{inj}_{-\oplus B}(M) \mid \mathit{inj}_{A\oplus -}(M) \mid \mathit{match}\ P\ \mathit{with}\ (x \mapsto M \mid y \mapsto N) \\ & \mid \mathit{let\ rec}\ fx : A = M\ \mathit{in}\ N \mid \mathit{switch}\ M\ \mathit{and}\ N\ \mathit{by}\ P \\ & \mid \mathit{promote}(M) \mid \mathit{derelict}(M) \mid \mathit{build}(M) \\ & \mid \mathit{separate}(M) \mid \mathit{mix}(M) \mid \mathit{local}\ P\ \mathit{in}\ (x \mapsto M \mid y \mapsto N) \end{aligned}$$

We shall provide some intuitive interpretations of these constructions:

- The following constructs retain their standard interpretations from other lambda calculi:

  Lambda abstractions $\lambda x : A.M$ and application MN;

  Pair constructors $\langle M, N \rangle$;

  Pair destructors $\mathit{let}\ \langle x, y \rangle = M\ \mathit{in}\ N$;

  Direct sum constructors $\mathit{inj}_{-\oplus B}(M)$, $\mathit{inj}_{A\oplus -}(M)$;

  Trivial systems/units $*$;

  Recursive function definitions $\mathit{let\ rec}\ fx : A = M\ \mathit{in}\ N$.

- Variables come in two flavours: term variables x and register variables $\hat{x}$. The term variables are bound in the surrounding term, providing a placeholder for another term to be substituted in at a later point. On the other hand, each register variable refers to the quantum register with the same name.

- U represents the name of an elementary operation. These can be built into a quantum register in a single computation step. The set of elementary operations should be specified for any given implementation of this programming language. This will typically include standard unitaries, such as the Pauli

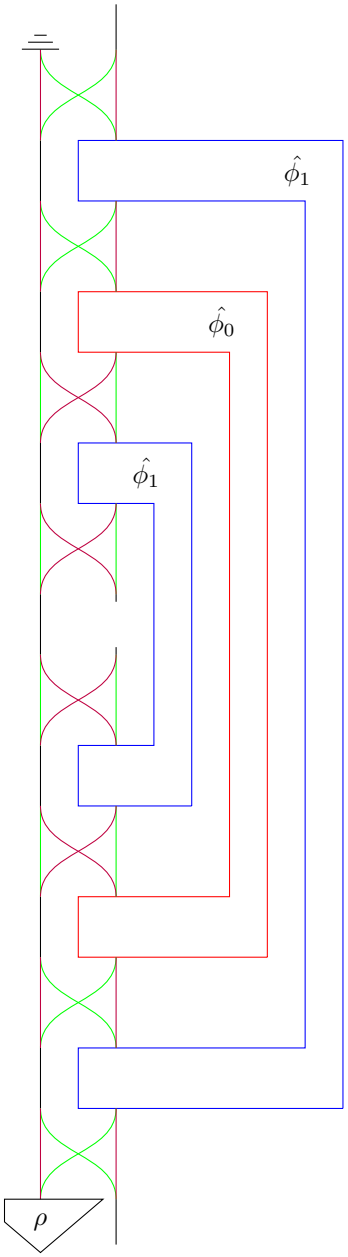


Figure 5.2: Simulation of a second-order switch using a second copy and controlled-swaps. $\hat{\phi}_0$ and $\hat{\phi}_1$ are the two pure transformations being switched. For the controlled-swaps, the purple paths are taken if the qubit control system is in $|0\rangle\langle 0|$ and the green for $|1\rangle\langle 1|$. ρ is some initial state for the ancilla.

gates for qubits, as well as some key isomorphisms between types which only perform a permutation of the basis states, including:

Associativity $(\hat{A} \oplus \hat{B}) \oplus \hat{C} \cong \hat{A} \oplus (\hat{B} \oplus \hat{C})$

Distributivity $(\hat{A} \oplus \hat{B}) \otimes \hat{C} \cong (\hat{A} \otimes \hat{C}) \oplus (\hat{B} \otimes \hat{C})$

Unitality $\hat{A} \otimes \top \cong \hat{A}$

Side-channel equivalence $\hat{A} \multimap \hat{B} \parr \hat{C} \cong (\hat{A} \multimap \hat{B}) \parr \hat{C}$

- Given a term P of type $\hat{A} \oplus \hat{B}$, we can decompose this using terms of the form *match* P *with* $(x \mapsto M \mid y \mapsto N)$. This performs a projective measurement onto the $\hat{A}$ and $\hat{B}$ subspaces. Depending on the measurement outcome, this would apply $\lambda x.M$ to the output if it is of type $\hat{A}$ or $\lambda y.N$ for $\hat{B}$. We extend this to when P is of a general type $A \oplus B$: if the type does not permit superposition then P either returns an A object or a B object, so we choose M or N appropriately.

- *build* takes a description of a circuit and builds it in a register in the quantum array. In order to build a lambda abstraction $\lambda x.M$ of type $\hat{A} \multimap \hat{B}$, we start with the identity process on $\hat{A}$ in the quantum register and then apply M to the output of this process. Since we model processes by their Choi-Jamiołkovski operator, at any point we will have an operator on the Hilbert space $\mathcal{H}_{\hat{C}} \otimes \mathcal{H}_{\hat{A}}$. *Choi* $\hat{x}$ *into* M represents such an intermediate point, where $\hat{x}$ is the free input of type $\hat{A}$ (treated as a separate register) and M is the remainder of the function to be applied. *build* and *Choi* $\hat{x}$ *into* M are necessary for the reduction mechanism but do not need to directly be used by the programmer.

- *promote* and *derelict* are the constructor and destructor for the ! modality. Promotion takes a self-contained program and makes it a copyable value, and dereliction recovers the program from one of these values. A key feature of promotion is that it will prevent any internal computation from occurring until dereliction is applied - it immediately generates a value (a !-suspension) much like a lambda abstraction. We resume the suspended computation upon dereliction.

- *switch* M *and* N *by* P takes a higher-order quantum switch (of the form discussed in Section 5.3) and applies it to the processes M and N with control system P. We could have included this as one of the elementary circuits U, but we keep it separate to emphasise that the quantum switch is intended to be a new way of composing processes.

- With regards to handling $\parr$-systems, *mix* allows us to replace an instance of $\otimes$ with one of $\parr$, in accordance with the mix rule of linear logic or the canonical map in Caus$[\mathcal{C}]$ from $\mathbf{A} \otimes \mathbf{B}$ to $\mathbf{A} \parr \mathbf{B}$. Recall from Lemma 3.4.7 that this becomes an isomorphism $\mathbf{A} \otimes \mathbf{B} \cong \mathbf{A} \parr \mathbf{B}$ when $\mathbf{A}$ and $\mathbf{B}$ are first-order systems. *separate* realises the other direction of this isomorphism, allowing us to convert any value of the first-order type $\tilde{A} \parr \tilde{B}$ to $\tilde{A} \otimes \tilde{B}$. When P gives us a value of type $\hat{A} \parr \hat{B}$, we use *local* P *in* $(x \mapsto M \mid y \mapsto N)$ to perform local transformations; $\lambda x.M$ is applied on the $\hat{A}$ subsystem and $\lambda y.N$ on $\hat{B}$.

We forbid substitution in place of register variables, allowing it only in place of term variables. Beyond this caveat, substitution and α-equivalence follow their standard definitions.

The type system for this language also falls in line with those for other lambda calculi.

Definition 5.4.2 (Typing Context)**.** *A typing context Δ is a finite set $\{x_1 : A_1, \ldots, x_n : A_n\}$ of pairs mapping variables to types ($\Delta(x_i) = A_i$), such that no variable appears in more than one pair. This can include both term variables and register variables.*

Remark. *We use $!\Delta$ for contexts of the form $\{x_i :!A_i\}_i$. If Δ and Γ range over disjoint sets of variables, then we write Δ, Γ for the union of the contexts. $|\Delta|$ is the set of variable names occurring in Δ.*

Definition 5.4.3 (Typing Judgement)**.** *A typing judgement $\Delta \rhd M : A$ is a sequent consisting of a typing context Δ, a term M, and a type A. We interpret this judgement to say that M returns a value of type A, assuming the free variables of M are typed according to Δ.*

Figure 5.3 details the typing rules for this programming language. In the case of the elementary unitaries, each operation U is provided with a predefined type A_U.

As an example, we can consider the algorithm from [12] for discrimination between commuting and anti-commuting channels. Let's define the shorthand types $qbit := \top \oplus \top$ and $chan := qbit \multimap qbit$. Suppose we have some program $\rhd \pi : chan \multimap \top$ which eliminates qubit channels by providing some input state and discarding the output, and elementary unitaries $\rhd \mathbf{LUnit} : (\top \parr qbit) \multimap qbit$ (absorbing a trivial system) and $\rhd \mathbf{H} : chan$ (a Hadamard gate).

$$\lambda a.\lambda b.\mathbf{LUnit}[local\ (switch\ a\ and\ b\ by\ \mathbf{H} inj_{-\oplus\top}(*)) \quad (5.2)$$
$$in\ (x \mapsto \pi x \mid y \mapsto match\ \mathbf{H} y\ with\ (x' \mapsto inj_{-\oplus\top}(x') \mid y' \mapsto inj_{\top\oplus-}(y')))]$$

This program can be typed as $chan \multimap chan \multimap qbit$, as shown by the derivation in Figure 5.4. The intention behind this algorithm is that either a and b or commuting or anti-commuting, with each case inducing a different measurable phase change on the control qubit.

5.5 Reduction

We adapt the notion of a quantum closure from [45] to the following:

Definition 5.5.1 (Quantum Closure)**.** *A quantum closure is a tuple $[\rho, L, M]$ consisting of:*

- *A Choi-Jamiołkovski operator ρ representing the state of the quantum array;*
- *An ordered list L of maps from distinct register variables to (quantum) types;*
- *A term M with all its free variables in L.*

  A quantum closure is well-formed $[\rho L, M] : A$ when $\Delta_L \rhd M : A$, where the type context Δ_L is obtained by forgetting the order of L.

$$\frac{}{!\Delta \rhd U : A_U}\ (U) \qquad \frac{}{!\Delta \rhd * : \top}\ (\top) \qquad \frac{}{!\Delta, x : A \rhd x : A}\ (\mathrm{var}_1) \qquad \frac{}{!\Delta, \hat{x} : A \rhd \hat{x} : A}\ (\mathrm{var}_2)$$

$$\frac{\Delta, x : A \rhd M : B}{\Delta \rhd \lambda x : A.M : A \multimap B}\ (\lambda) \qquad \frac{!\Delta, \Gamma_1 \rhd M : A \multimap B \qquad !\Delta, \Gamma_2 \rhd N : A}{!\Delta, \Gamma_1, \Gamma_2 \rhd MN : B}\ (\mathrm{app})$$

$$\frac{\Delta \rhd M : \hat{B}}{\Delta, \hat{x} : \hat{A} \rhd \mathit{Choi}\ \hat{x}\ \mathit{into}\ M : \hat{A} \multimap \hat{B}}\ (\mathit{Choi}) \qquad \frac{!\Delta, \Gamma_1 \rhd M : A \qquad !\Delta, \Gamma_2 \rhd N : B}{!\Delta, \Gamma_1, \Gamma_2 \rhd \langle M, N \rangle : A \otimes B}\ (\otimes.I)$$

$$\frac{!\Delta, \Gamma_1 \rhd M : A \otimes B \qquad !\Delta, \Gamma_2, x : A, y : B \rhd N : C}{!\Delta, \Gamma_1, \Gamma_2 \rhd \mathit{let}\ \langle x, y \rangle = M\ \mathit{in}\ N : C}\ (\otimes.E)$$

$$\frac{\Delta \rhd M : A}{\Delta \rhd \mathit{inj}_{-\oplus B}(M) : A \oplus B}\ (\oplus.I_1) \qquad \frac{\Delta \rhd M : B}{\Delta \rhd \mathit{inj}_{A\oplus -}(M) : A \oplus B}\ (\oplus.I_2)$$

$$\frac{!\Delta, \Gamma_1 \rhd P : A \oplus B \qquad !\Delta, \Gamma_2, x : A \rhd M : C \qquad !\Delta, \Gamma_2, y : B \rhd N : C}{!\Delta, \Gamma_1, \Gamma_2 \rhd \mathit{match}\ P\ \mathit{with}\ (x \mapsto M \mid y \mapsto N) : C}\ (\oplus.E)$$

$$\frac{!\Delta, f :!(A \multimap B), x : A \rhd M : B \qquad !\Delta, \Gamma, f :!(A \multimap B) \rhd N : C}{!\Delta, \Gamma \rhd \mathit{let\ rec}\ fx : A = M\ \mathit{in}\ N : C}\ (\mathit{rec})$$

$$\frac{!\Delta \rhd M : A}{!\Delta \rhd \mathit{promote}(M) :!A}\ (!.I) \qquad \frac{\Delta \rhd M :!A}{\Delta \rhd \mathit{derelict}(M) : A}\ (!.E) \qquad \frac{\Delta \rhd M : \hat{A}}{\Delta \rhd \mathit{build}(M) : \hat{A}}\ (\mathit{build})$$

$$\frac{!\Delta, \Gamma_0 \rhd M : \hat{A} \multimap \hat{A} \qquad !\Delta, \Gamma_1 \rhd N : \hat{A} \multimap \hat{A} \qquad !\Delta, \Gamma_2 \rhd P : \tilde{B} \oplus \tilde{C}}{!\Delta, \Gamma_0, \Gamma_1, \Gamma_2 \rhd \mathit{switch}\ M\ \mathit{and}\ N\ \mathit{by}\ P : (\hat{A} \multimap \hat{A}) \parr (\tilde{B} \oplus \tilde{C})}\ (\mathit{switch})$$

$$\frac{\Delta \rhd M : \tilde{A} \parr \tilde{B}}{\Delta \rhd \mathit{separate}(M) : \tilde{A} \otimes \tilde{B}}\ (\mathit{separate}) \qquad \frac{\Delta \rhd M : \hat{A} \otimes \hat{B}}{\Delta \rhd \mathit{mix}(M) : \hat{A} \parr \hat{B}}\ (\mathit{mix})$$

$$\frac{!\Delta, \Gamma_0 \rhd P : \hat{A} \parr \hat{B} \qquad !\Delta, \Gamma_1, x : \hat{A} \rhd M : \hat{C} \qquad !\Delta, \Gamma_2, y : \hat{B} \rhd N : \hat{D}}{!\Delta, \Gamma_0, \Gamma_1, \Gamma_2 \rhd \mathit{local}\ P\ \mathit{in}\ (x \mapsto M \mid y \mapsto N) : \hat{C} \parr \hat{D}}\ (\mathit{local})$$

Figure 5.3: The rules of the type system.

Given a closure $[\rho, L, M]$, we assume that any processes from distinct registers of the quantum array are completely disconnected, in the sense that we can build an "overall type" for L by taking the tensor product over all of its elements.

This Choi-Jamiołkovski operator represents processes in the expected manner:

- Trivial systems ($\top$) are associated to the Hilbert space $\mathcal{H}_\top = \mathbb{C}$.
- Direct sums $\hat{A} \oplus \hat{B}$ are associated to $\mathcal{H}_{\hat{A}\oplus\hat{B}} = \mathcal{H}_{\hat{A}} \oplus \mathcal{H}_{\hat{B}}$.
- Pairs $\hat{A} \otimes \hat{B}$ are associated to $\mathcal{H}_{\hat{A}\otimes\hat{B}} = \mathcal{H}_{\hat{A}} \otimes \mathcal{H}_{\hat{B}}$. Connected pairs $\hat{A} \parr \hat{B}$ also live in space, but contain a larger family of possible operators.
- Functions $\hat{A} \multimap \hat{B}$ are associated to $\mathcal{H}_{\hat{A}\multimap\hat{B}} = \mathcal{H}_{\hat{A}} \otimes \mathcal{H}_{\hat{B}}$ in accordance with the Choi-Jamiołkovski isomorphism.

We will define an operational semantics for this programming language in the form of a reduction relation between quantum closures. Allowing measurement of quantum states introduces non-determinism into our reduction.

$$\dfrac{\dfrac{}{a : chan \rhd a : chan} \quad \dfrac{}{b : chan \rhd b : chan} \quad \dfrac{\dfrac{}{\rhd \mathbf{H} : chan} \quad \dfrac{\dfrac{}{\rhd * : \top}}{\rhd inj_{-\oplus\top}(*) : qbit}}{\rhd \mathbf{H} inj_{-\oplus\top}(*) : qbit}}{a : chan, b : chan \rhd switch\ a\ and\ b\ by\ \mathbf{H} inj_{-\oplus\top}(*) : chan \parr qbit}$$

$$\dfrac{\dfrac{\dfrac{}{\rhd \mathbf{H} : chan} \quad \dfrac{}{y : qbit \rhd y : qbit}}{y : qbit \rhd \mathbf{H} y : qbit} \quad \dfrac{\dfrac{}{x' : \top \rhd x' : \top}}{x' : \top \rhd inj_{-\oplus\top}(x') : qbit} \quad \dfrac{\dfrac{}{y' : \top \rhd y' : \top}}{y' : \top \rhd inj_{\top\oplus-}(y') : qbit}}{y : qbit \rhd match\ \mathbf{H} y\ with\ (x' \mapsto inj_{-\oplus\top}(x') \mid y' \mapsto inj_{\top\oplus-}(y')) : qbit}$$

$$\dfrac{a : chan, b : chan \rhd \cdots : chan \parr qbit \quad \dfrac{\dfrac{}{\rhd \pi : chan \multimap \top} \quad \dfrac{}{x : chan \rhd x : chan}}{x : chan \rhd \pi x : \top} \quad y : qbit \rhd \cdots : qbit}{a : chan, b : chan \rhd local\ (\cdots)\ in\ (x \mapsto \pi x \mid y \mapsto \cdots) : \top \parr qbit}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\rhd \mathbf{LUnit} : (\top \parr qbit) \multimap qbit} \quad a : chan, b : chan \rhd \cdots : \top \parr qbit}{a : chan, b : chan \rhd \mathbf{LUnit}[\cdots] : qbit}}{a : chan \rhd \lambda b.\mathbf{LUnit}[\cdots] : chan \multimap qbit}}{\rhd \lambda a.\lambda b.\mathbf{LUnit}[\cdots] : chan \multimap chan \multimap qbit}$$

Figure 5.4: A complete derivation of the type of Program 5.2.

It is important to note that, even if the end value of a term is copyable, the term itself may still reference existing quantum processes before reaching this value (e.g. $z : \top \oplus \top, a : \top \multimap !A, b : \top \multimap !A \rhd match\ z\ with\ (x \mapsto ax \mid y \mapsto by) : !A$). Non-linear substitution of such a term would require the referenced quantum process (z) to be copied to each location, violating the no-cloning theorem. Building a call-by-value scheme into the reduction relation prevents this case from occurring.

Definition 5.5.2. *The set of values of the language is defined by the following abstract syntax:*

$$V, W ::= U \mid x \mid \hat{x} \mid * \mid \lambda x : A.M \mid promote(M) \mid \langle V, W \rangle \mid inj_{-\oplus B}(V) \mid inj_{A\oplus-}(V)$$

Descriptive values are defined by the following fragment of the values:

$$V^*, W^* ::= U \mid x \mid * \mid \lambda x : A.M \mid promote(M) \mid \langle V, W \rangle \mid inj_{-\oplus B}(V) \mid inj_{A\oplus-}(V)$$

From these definitions, we see that a value is either a descriptive value or a register variable. From the perspective of the programmer, the contents of the registers can be treated as black-boxes. Descriptive values are not black-boxes since they describe some structure within the value. For example, if we reach the value $inj_{-\oplus B}(V) : A \oplus B$, we know that the state lives entirely in the A subspace. We use the distinction between descriptive values and register variables in the reduction rules to indicate how we handle registers differently from a circuit description.

We define the reduction relation $\rightarrow$ between quantum closures inductively over the syntax of the program term. At any point, we may apply α-equivalence or a permutation of L and ρ, so the following rules are specified without reference

to specific variable names and orders. We also note that any variable introduced during reduction is assumed to be fresh (it does not already appear in the rest of the quantum closure).

We use the notation $M[V/x]$ to indicate the substitution of V in place of every occurrence of x in M.

Application

Here we present the semantics regarding application of processes. These rules cover the cases of classically applying some function and composing two processes from quantum registers.

If $\rho = \sum_i E_i \otimes F_i \otimes X_i$ (where the E_i represent environment states, F_i the function, and X_i the argument), then we define $\rho_{app} := \sum_i E_i \otimes \mathrm{Tr}_{\mathcal{H}_{\hat{A}}}[(\mathrm{id}_{\mathcal{H}_{\hat{B}}} \otimes X_i^T)F_i]$ in accordance with the inverse Choi-Jamiołkovski transformation.

$$[\rho, L, (\lambda x : A.M)V] \to [\rho, L, M[V/x]]$$
$$[\rho, [\ldots, \hat{x} : \hat{A} \multimap \hat{B}, \hat{y} : \hat{A}], \hat{x}\hat{y}] \to [\rho_{app}, [\ldots, \hat{z} : \hat{B}], \hat{z}]$$

If we want to apply one of the elementary circuits, it must first be built into a register. Moreover, when we have a physical process in a register, we can only run it on physical inputs, so we must build the argument.

$$[\rho, L, UV] \to [\rho, L, \mathit{build}(U)V]$$
$$[\rho, L, \hat{x}V^*] \to [\rho, L, \hat{x}\ \mathit{build}(V^*)]$$

We enforce the call-by-value reduction scheme by first reducing the argument to a value. We then reduce the function to a point where we can perform the application.

$$\frac{[\rho, L, N] \to [\rho', L', N']}{[\rho, L, MN] \to [\rho', L', MN']}$$

$$\frac{[\rho, L, M] \to [\rho', L', M']}{[\rho, L, MV] \to [\rho', L', M'V]}$$

Tensor Products

The two forms a tensor product value can take are descriptive pairs $\langle V, W\rangle$ and register variables. Given a descriptive pair $\langle V, W\rangle$, we immediately have a decomposition into V and W. A tensor product in a register corresponds to a pair of distinct systems, so we can decompose products by splitting them into two registers.

$$[\rho, L, \mathit{let}\ \langle x, y\rangle = \langle V, W\rangle\ \mathit{in}\ N] \to [\rho, L, N[V/x, W/y]]$$
$$[\rho, [\ldots, \hat{z} : \hat{A} \otimes \hat{B}], \mathit{let}\ \langle x, y\rangle = \hat{z}\ \mathit{in}\ N] \to [\rho, [\ldots, \hat{x} : \hat{A}, \hat{y} : \hat{B}], N[\hat{x}/x, \hat{y}/y]]$$

A tensor product system must be reduced to a value before it can be decomposed and used. In order to give a unique order of reduction, we require that the second term in a pair is reduced as much as possible before we start operating on the first term.

$$\frac{[\rho, L, M] \to [\rho', L', M']}{[\rho, L, let\ \langle x, y\rangle = M\ in\ N] \to [\rho', L', let\ \langle x, y\rangle = M'\ in\ N]}$$

$$\frac{[\rho, L, N] \to [\rho', L', N']}{[\rho, L, \langle M, N\rangle] \to [\rho', L', \langle M, N'\rangle]}$$

$$\frac{[\rho, L, M] \to [\rho', L', M']}{[\rho, L, \langle M, V\rangle] \to [\rho', L', \langle M', V\rangle]}$$

Direct Sums

We always decompose a direct sum in the form of a test followed by a choice of programs. When we have a descriptive value $inj_{-\oplus B}(V)$ or $inj_{A\oplus -}(V)$, only one outcome of the test is possible.

When we measure a system in a register, either outcome could be possible. This introduces non-determinism into the reduction. Since the Choi-Jamiołkovski operator in the closure is for the tensor product of all objects in the quantum array, we can use distributivity $(\hat{A} \oplus \hat{B}) \otimes \hat{C} \cong (\hat{A} \otimes \hat{C}) \oplus (\hat{B} \otimes \hat{C})$ to show that it is always some block matrix of the form:

$$\begin{pmatrix} \rho_l & \rho_{off_1} \\ \rho_{off_2} & \rho_r \end{pmatrix}$$

Upon measurement, we project onto either the left or the right side. This will always lose any information in the off-diagonal space.

$$\begin{aligned}
&[\rho, L, match\ inj_{-\oplus B}(V)\ with\ (x \mapsto M \mid y \mapsto N)] \to [\rho, L, M[V/x]] \\
&[\rho, L, match\ inj_{A\oplus -}(V)\ with\ (x \mapsto M \mid y \mapsto N)] \to [\rho, L, N[V/y]] \\
&\qquad [\rho_l \oplus 0 + 0 \oplus \rho_r + \rho_{off}, [\ldots, \hat{z} : \hat{A} \oplus \hat{B}], \\
&\qquad\qquad match\ \hat{z}\ with\ (x \mapsto M \mid y \mapsto N)] \to [\rho_l, [\ldots, \hat{x} : \hat{A}], M[\hat{x}/x]] \\
&\qquad [\rho_l \oplus 0 + 0 \oplus \rho_r + \rho_{off}, [\ldots, \hat{z} : \hat{A} \oplus \hat{B}], \\
&\qquad\qquad match\ \hat{z}\ with\ (x \mapsto M \mid y \mapsto N)] \to [\rho_r, [\ldots, \hat{y} : \hat{B}], N[\hat{y}/y]]
\end{aligned}$$

We cannot know which branch of a $match\ P\ with\ (x \mapsto M \mid y \mapsto N)$ statement we will take until we have determined the test P, so we prioritise reducing the test over either of the branches.

$$\frac{[\rho, L, P] \to [\rho', L', P']}{[\rho, L, match\ P\ with\ (x \mapsto M \mid y \mapsto N)] \to [\rho', L', match\ P'\ with\ (x \mapsto M \mid y \mapsto N)]}$$

$$\frac{[\rho, L, M] \to [\rho', L', M']}{[\rho, L, inj_{-\oplus B}(M)] \to [\rho', L', inj_{-\oplus B}(M')]}$$

$$\frac{[\rho, L, M] \to [\rho', L', M']}{[\rho, L, inj_{A\oplus -}(M)] \to [\rho', L', inj_{A\oplus -}(M')]}$$

Recursion

Recursion follows a classical procedure of copying the definition to every call location, as opposed to schemes where the fixpoint is determined in a single step of computation.

$$[\rho, L, \mathit{let\ rec}\ fx : A = M\ \mathit{in}\ N] \to [\rho, L, N[\mathit{promote}(\lambda x : A.\mathit{let\ rec}\ fx : A = M\ \mathit{in}\ M)/f]]$$

Choi-Jamiołkovski Operators

When building a circuit with a non-trivial input, we always maintain the handle to the input of the circuit and apply the function on the output handle. We want to be able to consider the final result as a black-box that is completely contained in a single register, so we must build any descriptive value we reach. Once the circuit is built (the output is reduced to a register variable), it is simply relabelled as a single unit.

$$[\rho, [\ldots, \hat{x} : \hat{B}, \hat{y} : \hat{A}], \mathit{Choi}\ \hat{y}\ \mathit{into}\ \hat{x}] \to [\rho, [\ldots, \hat{x} : \hat{A} \multimap \hat{B}], \hat{x}]$$
$$[\rho, L, \mathit{Choi}\ \hat{x}\ \mathit{into}\ V^*] \to [\rho, L, \mathit{Choi}\ \hat{x}\ \mathit{into}\ \mathit{build}(V^*)]$$

$$\frac{[\rho, L, M] \to [\rho', L', M']}{[\rho, L, \mathit{Choi}\ \hat{x}\ \mathit{into}\ M] \to [\rho', L', \mathit{Choi}\ \hat{x}\ \mathit{into}\ M']}$$

!-Suspensions

We use *derelict* to resume the suspended computation M captured in $\mathit{promote}(M)$.

$$[\rho, L, \mathit{derelict}(\mathit{promote}(M))] \to [\rho, L, M]$$

$$\frac{[\rho, L, M] \to [\rho', L', M']}{[\rho, L, \mathit{derelict}(M)] \to [\rho', L', \mathit{derelict}(M')]}$$

Building Circuits

Let $\mathfrak{C}(\mathcal{U})$ be the Choi-Jamiołkovski operator for the unitary process referred to by U. As with previous cases where we were reducing a program to a circuit, if we ever reach a descriptive value, we must recursively build it. In particular, we build a trivial system by placing a fixed process into a quantum register. If we reach a register variable, this is already built so there is nothing more to do. We use the $\mathit{Choi}\ \hat{x}\ \mathit{into}\ M$ construct to help us break down the procedure of building $\lambda x.M$ into simpler steps, starting with the Choi-Jamiołkovski operator of the identity process $(|I_{\mathcal{H}_{\hat{A}}}\rangle\langle I_{\mathcal{H}_{\hat{A}}}|)$.

$$
\begin{aligned}
[\rho, [\ldots], \mathit{build}(U)] &\to [\rho \otimes \mathfrak{C}[\mathcal{U}], [\ldots, \hat{x} : A_U], \hat{x}] \\
[\rho, [\ldots], \mathit{build}(*)] &\to [\rho, [\ldots, \hat{x} : \top], \hat{x}] \\
[\rho, L, \mathit{build}(\hat{x})] &\to [\rho, L, \hat{x}] \\
[\rho, [\ldots], \mathit{build}(\lambda x : \hat{A}.M)] &\to [\rho \otimes |I_{\mathcal{H}_{\hat{A}}}\rangle\langle I_{\mathcal{H}_{\hat{A}}}|, [\ldots, \hat{x} : \hat{A}, \hat{y} : \hat{A}], \\
&\qquad \mathit{Choi}\ \hat{y}\ \mathit{into}\ M[\hat{x}/x]] \\
[\rho, L, \mathit{build}(\langle V, W^* \rangle)] &\to [\rho, L, \mathit{build}(\langle V, \mathit{build}(W^*) \rangle)] \\
[\rho, L, \mathit{build}(\langle V^*, \hat{x} \rangle)] &\to [\rho, L, \mathit{build}(\langle \mathit{build}(V^*), \hat{x} \rangle)] \\
[\rho, [\ldots, \hat{x} : \hat{A}, \hat{y} : \hat{B}], \mathit{build}(\langle \hat{x}, \hat{y} \rangle)] &\to [\rho, [\ldots, \hat{z} : \hat{A} \otimes \hat{B}], \hat{z}] \\
[\rho, L, \mathit{build}(\mathit{inj}_{-\oplus B}(V^*))] &\to [\rho, L, \mathit{build}(\mathit{inj}_{-\oplus B}(\mathit{build}(V^*)))] \\
[\rho, L, \mathit{build}(\mathit{inj}_{A\oplus -}(V^*))] &\to [\rho, L, \mathit{build}(\mathit{inj}_{A\oplus -}(\mathit{build}(V^*)))] \\
[\rho, [\ldots, \hat{x} : \hat{A}], \mathit{build}(\mathit{inj}_{-\oplus \hat{B}}(\hat{x}))] &\to [\rho \oplus 0, [\ldots, \hat{x} : \hat{A} \oplus \hat{B}], \hat{x}] \\
[\rho, [\ldots, \hat{x} : \hat{B}], \mathit{build}(\mathit{inj}_{\hat{A}\oplus -}(\hat{x}))] &\to [0 \oplus \rho, [\ldots, \hat{x} : \hat{A} \oplus \hat{B}], \hat{x}]
\end{aligned}
$$

$$
\frac{[\rho, L, M] \to [\rho', L', M']}{[\rho, L, \mathit{build}(M)] \to [\rho', L', \mathit{build}(M')]}
$$

Quantum Switch

switch $\hat{x}$ *and* $\hat{y}$ *by* $\hat{z}$ uses a quantum switch to connect the processes in registers $\hat{x}$ and $\hat{y}$ in a superposition of orders in accordance with Equation 5.1. We provide the process in $\hat{z}$ as the control system. Let ρ_{switch} be the Choi-Jamiołkovski operator that results. We assume that the quantum switches are provided as physical resources, so they can only be applied to physical inputs. Therefore, we must build any descriptive value we reach along the way.

$$
\begin{aligned}
[\rho, [\ldots, \hat{x} : \hat{A} \multimap \hat{A}, \hat{y} : \hat{A} \multimap \hat{A}, \hat{z} : \hat{B} \oplus \hat{C}], & \\
\mathit{switch}\ \hat{x}\ \mathit{and}\ \hat{y}\ \mathit{by}\ \hat{z}] &\to [\rho_{switch}, [\ldots, \hat{x} : (\hat{A} \multimap \hat{A}) \parr (\hat{B} \oplus \hat{C})], \hat{x}] \\
[\rho, L, \mathit{switch}\ M\ \mathit{and}\ N\ \mathit{by}\ V^*] &\to [\rho, L, \mathit{switch}\ M\ \mathit{and}\ N\ \mathit{by}\ \mathit{build}(V^*)] \\
[\rho, L, \mathit{switch}\ M\ \mathit{and}\ V^*\ \mathit{by}\ \hat{z}] &\to [\rho, L, \mathit{switch}\ M\ \mathit{and}\ \mathit{build}(V^*)\ \mathit{by}\ \hat{z}] \\
[\rho, L, \mathit{switch}\ V^*\ \mathit{and}\ \hat{y}\ \mathit{by}\ \hat{z}] &\to [\rho, L, \mathit{switch}\ \mathit{build}(V^*)\ \mathit{and}\ \hat{y}\ \mathit{by}\ \hat{z}]
\end{aligned}
$$

$$
\frac{[\rho, L, P] \to [\rho', L', P']}{[\rho, L, \mathit{switch}\ M\ \mathit{and}\ N\ \mathit{by}\ P] \to [\rho', L', \mathit{switch}\ M\ \mathit{and}\ N\ \mathit{by}\ P']}
$$

$$
\frac{[\rho, L, N] \to [\rho', L', N']}{[\rho, L, \mathit{switch}\ M\ \mathit{and}\ N\ \mathit{by}\ \hat{z}] \to [\rho', L', \mathit{switch}\ M\ \mathit{and}\ N'\ \mathit{by}\ \hat{z}]}
$$

$$
\frac{[\rho, L, M] \to [\rho', L', M']}{[\rho, L, \mathit{switch}\ M\ \mathit{and}\ \hat{y}\ \mathit{by}\ \hat{z}] \to [\rho', L', \mathit{switch}\ M'\ \mathit{and}\ \hat{y}\ \mathit{by}\ \hat{z}]}
$$

⅋ Operations

Separation is only defined for $\tilde{A} \parr \tilde{B}$ with first-order $\tilde{A}$ and $\tilde{B}$. We note that the only form that any value of the type $\tilde{A} \parr \tilde{B}$ must be a register variable, so we do not need to handle descriptive values. Since $\tilde{A} \parr \tilde{B}$ and $\tilde{A} \otimes \tilde{B}$ are considered equivalent types, there is no need to perform any real computation on separation, just modifying the interface to allow operations over the joint system. Mixing works similarly.

For local application, we take a quantum register and split it in two. We then apply functions locally on each side before combining them back into a single register representing the connected system.

$$
\begin{aligned}
[\rho, [\ldots, \hat{x} : \tilde{A} \parr \tilde{B}, \ldots], \mathit{separate}(\hat{x})] &\to [\rho, [\ldots, \hat{x} : \tilde{A} \otimes \tilde{B}, \ldots], \hat{x}] \\
[\rho, [\ldots, \hat{x} : \hat{A} \otimes \hat{B}], \mathit{mix}(\hat{x})] &\to [\rho, [\ldots, \hat{x} : \hat{A} \parr \hat{B}], \hat{x}] \\
[\rho, L, \mathit{mix}(\langle V, W \rangle)] &\to [\rho, L, \mathit{mix}(\mathit{build}(\langle V, W \rangle))] \\
[\rho, [\ldots, \hat{z} : \hat{A} \parr \hat{B}], & \\
\mathit{local}\ \hat{z}\ \mathit{in}\ (x \mapsto M \mid y \mapsto N)] &\to [\rho, [\ldots, \hat{x} : \hat{A}, \hat{y} : \hat{B}], \\
& \qquad \mathit{mix}(\langle M[\hat{x}/x], N[\hat{y}/y] \rangle)]
\end{aligned}
$$

$$
\frac{[\rho, L, M] \to [\rho', L', M']}{[\rho, L, \mathit{separate}(M)] \to [\rho', L', \mathit{separate}(M')]}
$$

$$
\frac{[\rho, L, M] \to [\rho', L', M']}{[\rho, L, \mathit{mix}(M)] \to [\rho', L', \mathit{mix}(M')]}
$$

$$
\frac{[\rho, L, P] \to [\rho', L', P']}{[\rho, L, \mathit{local}\ P\ \mathit{in}\ (x \mapsto M \mid y \mapsto N)] \to [\rho', L', \mathit{local}\ P'\ \mathit{in}\ (x \mapsto M \mid y \mapsto N)]}
$$

5.6 Probability

When reducing $\mathit{build}(\lambda x.M)$, we simulate running M on an unspecified input value. If M contains a measurement, the physical interpretation would be to connect a measurement device which triggers one of the two optional processes to be run. However, our semantics define the effect of the measurement process as opposed to a physical measurement device. Consequently, we can get multiple reduction sequences during the building procedure, resulting in several Choi-Jamiołkovski operators. Each operator will correspond to a specific sequence of measurement outcomes. We may not know the probability of any of these outcomes until the circuit is provided with all inputs, preventing us from knowing the probability of a reduction sequence at all points. This is why we defined our reduction relation to be non-deterministic as opposed to the probabilistic relations of other languages such as the quantum lambda calculus of [45].

The call-by-value reduction scheme ensures that, once we start reducing a particular subterm, we only reduce it completely to a value before reducing any other part of the term. A special case of this is that, once we start building a circuit, the context of this circuit in the overall program will not change until the circuit is built. As a result, all branches from measurement during the building procedure

will eventually reduce to a register variable in the same context (assuming each branch terminates). At this point, the only difference between the possible quantum closures will be the Choi-Jamiołkovski operator. Taking the sum/probabilistic mixture of these operators corresponds to eliminating any classical knowledge of the measurement outcomes and results in the true Choi-Jamiołkovski operator of the circuit.

The Born rule (Equation 2.5) tells us that, given the state obtained from the result of a non-deterministic process, the probability of that result occurring is given by the trace. Using this fact, we can determine the probability of a reduction sequence whenever the quantum array in the final closure consists only of first-order systems.

5.7 Examples

Here we will consider a couple of example programs and their reduction sequences to examine how this calculus uses the Choi-Jamiołkovski isomorphism. Recall the shorthand types $qbit \coloneqq \top \oplus \top$ and $chan \coloneqq qbit \multimap qbit$. We can define a discarding map for a qubit by measuring the qubit and forgetting the measurement result:

$$\mathbf{discard} \coloneqq \lambda z : qbit.match\ z\ with\ (x \mapsto x \mid y \mapsto y) : qbit \multimap \top \tag{5.3}$$

The reduction sequence of building this in a quantum register is given below. The type is omitted since it remains the same throughout, and $\rightarrow^i$ refer to the possible non-deterministic branches. We start with an empty quantum array.

$$[1, [\,], build(\mathbf{discard})]$$

The first operation is to construct the Choi-Jamiołkovski of the identity process, i.e. the maximally entangled state. We use $\hat{z}$ to refer to the output handle, to which we apply the discarding process, and we set aside the input handle $\hat{z}'$.

$$\rightarrow \left[\begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}, [\hat{z} : qbit, \hat{z}' : qbit], Choi\ \hat{z}'\ into\ match\ z\ with\ (x \mapsto x \mid y \mapsto y) \right]$$

The measurement causes non-deterministic branching in the reduction.

$$\rightarrow^1 \left[\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, [\hat{x} : \top, \hat{z}' : qbit], Choi\ \hat{z}'\ into\ \hat{x} \right] \quad \rightarrow^2 \left[\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}, [\hat{y} : \top, \hat{z}' : qbit], Choi\ \hat{z}'\ into\ \hat{x} \right]$$

Each of these now corresponds to a completed circuit, so we can package them up into a single register.

$$\rightarrow^1 \left[\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, [\hat{f} : qbit \multimap \top], \hat{f} \right] \qquad \rightarrow^2 \left[\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}, [\hat{f} : qbit \multimap \top], \hat{f} \right]$$

These two irreducible closures have an identical form, and we find that the sum of the matrices gives the identity matrix, which is exactly the Choi-Jamiołkovski

operator of the trace/discarding process [16]. In the following, we will assume that **discard** operates deterministically as a discarding map for clarity.

We can use a discarding map to construct the completely depolarising qubit channel. For this, we assume that our set of elementary circuits contains/permits the construction of a controlled-not gate $\triangleright\mathbf{CNOT} : \mathit{qbit} \otimes \mathit{qbit} \multimap \mathit{qbit} \otimes \mathit{qbit}$, a Hadamard gate $\triangleright\mathbf{H} : \mathit{chan}$, and a left unitor $\triangleright\mathbf{LUnit} : \top \otimes \mathit{qbit} \multimap \mathit{qbit}$. The construction $\mathbf{CNOT}\ \langle \mathbf{H}\mathit{inj}_{-\oplus\top}(*), \mathit{inj}_{-\oplus\top}(*)\rangle$ builds a (normalised) maximally entangled state, so discarding one side of this results in the maximally mixed state.

$$
\begin{aligned}
\mathbf{depolarise} \coloneqq \lambda x : \mathit{qbit}.\mathbf{LUnit}\ \langle \mathbf{discard}\ x, \mathbf{LUnit}(& \\
& \mathit{let}\ \langle y, z\rangle = \mathbf{CNOT}\ \langle \mathbf{H}\mathit{inj}_{-\oplus\top}(*), \mathit{inj}_{-\oplus\top}(*)\rangle \\
& \mathit{in}\ \langle \mathbf{discard}\ y, z\rangle)\rangle : \mathit{chan}
\end{aligned}
\tag{5.4}
$$

In the following, we adopt the notation $\rightarrow^*$ to refer to the reflexive and transitive closure of the reduction relation, i.e. where $\rightarrow$ represents a single step of reduction, $\rightarrow^*$ means reduction in zero or more steps.

$$
\begin{aligned}
&[1, [\,], \mathit{build}(\mathbf{depolarise})] \\
&\to \left[\begin{pmatrix} 1&0&0&1\\0&0&0&0\\0&0&0&0\\1&0&0&1 \end{pmatrix}, [\hat{x} : \mathit{qbit}, \hat{x}' : \mathit{qbit}], \right. \\
&\quad \mathit{Choi}\ \hat{x}'\ \mathit{into}\ \mathbf{LUnit}\ \langle \mathbf{discard}\ \hat{x}, \mathbf{LUnit}(\\
&\quad\quad \mathit{let}\ \langle y, z\rangle = \mathbf{CNOT}\ \langle \mathbf{H}\mathit{inj}_{-\oplus\top}(*), \mathit{inj}_{-\oplus\top}(*)\rangle \\
&\quad\quad \mathit{in}\ \langle \mathbf{discard}\ y, z\rangle)\rangle \left. \right] \\
&\to^* \left[\begin{pmatrix} 1&0&0&1\\0&0&0&0\\0&0&0&0\\1&0&0&1 \end{pmatrix} \otimes \frac{1}{2} \begin{pmatrix} 1&0&1&0\\0&0&0&0\\1&0&1&0\\0&0&0&0 \end{pmatrix}, [\hat{x} : \mathit{qbit}, \hat{x}' : \mathit{qbit}, \hat{w} : \mathit{qbit} \otimes \mathit{qbit}], \right. \\
&\quad \mathit{Choi}\ \hat{x}'\ \mathit{into}\ \mathbf{LUnit}\ \langle \mathbf{discard}\ \hat{x}, \mathbf{LUnit}(\\
&\quad\quad \mathit{let}\ \langle y, z\rangle = \mathbf{CNOT}\hat{w}\ \mathit{in}\ \langle \mathbf{discard}\ y, z\rangle)\rangle \left. \right] \\
&\to^* \left[\begin{pmatrix} 1&0&0&1\\0&0&0&0\\0&0&0&0\\1&0&0&1 \end{pmatrix} \otimes \frac{1}{2} \begin{pmatrix} 1&0&0&1\\0&0&0&0\\0&0&0&0\\1&0&0&1 \end{pmatrix}, [\hat{x} : \mathit{qbit}, \hat{x}' : \mathit{qbit}, \hat{y} : \mathit{qbit}, \hat{z} : \mathit{qbit}], \right. \\
&\quad \mathit{Choi}\ \hat{x}'\ \mathit{into}\ \mathbf{LUnit}\ \langle \mathbf{discard}\ \hat{x}, \mathbf{LUnit}\langle \mathbf{discard}\ \hat{y}, \hat{z}\rangle\rangle \left. \right] \\
&\to^* \left[\begin{pmatrix} 1&0&0&1\\0&0&0&0\\0&0&0&0\\1&0&0&1 \end{pmatrix} \otimes \frac{1}{2} \begin{pmatrix} 1&0\\0&1 \end{pmatrix}, [\hat{x} : \mathit{qbit}, \hat{x}' : \mathit{qbit}, \hat{z} : \mathit{qbit}], \mathit{Choi}\ \hat{x}'\ \mathit{into}\ \mathbf{LUnit}\ \langle \mathbf{discard}\ \hat{x}, \hat{z}\rangle \right] \\
&\to^* \left[\begin{pmatrix} 1&0\\0&1 \end{pmatrix} \otimes \frac{1}{2} \begin{pmatrix} 1&0\\0&1 \end{pmatrix}, [\hat{x} : \top, \hat{x}' : \mathit{qbit}, \hat{z} : \mathit{qbit}], \mathit{Choi}\ \hat{x}'\ \mathit{into}\ \mathbf{LUnit}\langle \hat{x}, \hat{z}\rangle \right] \\
&\to^* \left[\frac{1}{2} \begin{pmatrix} 1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1 \end{pmatrix}, [\hat{f} : \mathit{chan}], \hat{f} \right]
\end{aligned}
$$

The final example describes the construction from [22] which enables the "Causal Activation" of depolarising channels, producing a channel with non-zero capacity.

$$
\mathbf{activated} \coloneqq \mathit{switch}\ \mathbf{depolarise}\ \mathit{and}\ \mathbf{depolarise}\ \mathit{by}\ \mathbf{H}\mathit{inj}_{-\oplus\top}(*) : \mathit{chan} \parr \mathit{qbit} \tag{5.5}
$$

$$
\begin{aligned}
&[1, [\,], \textbf{activated}] \\
\to^* &\left[\frac{1}{2}\begin{pmatrix}1 & 1\\ 1 & 1\end{pmatrix}, [\hat{x} : qbit], \mathit{switch}\ \textbf{depolarise}\ \mathit{and}\ \textbf{depolarise}\ \mathit{by}\ \hat{x}\right] \\
\to^* &\left[\begin{array}{c}\frac{1}{2}\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix} \otimes \frac{1}{2}\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix} \otimes \frac{1}{2}\begin{pmatrix}1 & 1\\ 1 & 1\end{pmatrix}, \\ \left[\hat{f} : \mathit{chan}, \hat{g} : \mathit{chan}, \hat{x} : \mathit{qbit}\right], \mathit{switch}\ \hat{f}\ \mathit{and}\ \hat{g}\ \mathit{by}\ \hat{x}\end{array}\right]
\end{aligned}
$$

After some calculation on the matrices, we find that this reduces to the following final result, where σ is the permutation matrix reversing the order of three qubits (meaning the main qubit in focus in the matrix shown is the control qubit).

$$
\to \left[\frac{1}{8}\sigma \begin{pmatrix}2 & 0 & 0 & 0 & 1 & 0 & 0 & 1\\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 2 & 1 & 0 & 0 & 1\\ 1 & 0 & 0 & 1 & 2 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0\\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 2\end{pmatrix} \sigma^\dagger, [\hat{y} : \mathit{chan} \parr \mathit{qbit}], \hat{y}\right]
$$

Decomposing the output based on the control qubit gives the identity matrix (the Choi-Jamiołkovski operator of the completely depolarising channel) up to some number on its diagonal elements, but on the off-diagonals we find the maximally entangled vector (corresponding to the identity channel). It is clear that by utilising these off-diagonal elements, we can use the channel to successfully transmit information which is not possible when combining completely depolarising channels in either classical order. To convert the type from $(\mathit{qbit} \multimap \mathit{qbit}) \parr \mathit{qbit}$ to the more typical channel form, $\mathit{qbit} \multimap (\mathit{qbit} \otimes \mathit{qbit})$, we can apply the transformation $\lambda c.\lambda z.\mathit{separate}(\mathit{local}\ c\ \mathit{in}\ (x \mapsto xz \mid y \mapsto y))$.

Chapter 6

Evaluation of the Language

The previous chapter presented the programming language with some informal reasoning behind the design choices. Here, we will formally justify that our grammar of first-order types actually correspond to first-order systems, the choice for the type given to the quantum switch. We also prove the main functional properties of the language.

6.1 First-Order Types

In this section, we provide a categorical interpretation for the types of the programming language and show the equivalence of the notions of first-order types.

We would like to discuss direct sums of processes in the context of Caus[CP(**FHilb**)], the category of causal, completely positive maps between finite-dimensional Hilbert spaces. Whilst direct sums $\oplus$ are coproducts for **FHilb**, they are not coproducts for CP(**FHilb**). This is because the fail to satisfy the universal coproduct property that a map out from the direct sum can be completely defined by its compositions with the injectors. For example, the identity operator on qubits and the Pauli Z gate both agree on the inputs $|0\rangle\langle 0|$ and $\langle 1|\langle 1|$ (the states corresponding to the injectors from the trivial system) but disagree on other states.

Nonetheless, we can lift the structure of $\oplus$ into Caus[CP(**FHilb**)].

$$\mathbf{A} \oplus \mathbf{B} \coloneqq (A \oplus B, \{\pi_A^* \oplus \pi_B^* | \pi_A \in c_{\mathbf{A}}^*, \pi_B \in c_{\mathbf{B}}^*\}^*)^1 \tag{6.1}$$

We note that the discarding maps in CP(**FHilb**) satisfy the following property for all A, B:

$$\bar{\top}_{A\oplus B} = \bar{\top}_A \oplus \bar{\top}_B \tag{6.2}$$

In Section 5.2, we defined first-order types in the programming language by the grammar $\tilde{A}, \tilde{B} ::= \top \mid \tilde{A} \otimes \tilde{B} \mid \tilde{A} \oplus \tilde{B}$. We can now show that the objects in Caus[$\mathcal{C}$] to which these correspond are actually first-order systems in terms of Definition 3.4.6.

[1] One could imagine an alternative definition of the causal set via injections as $(\{\iota_A \circ \rho_A | \rho_A \in c_{\mathbf{A}}\} \cup \{\iota_B \circ \rho_B | \rho_B \in c_{\mathbf{B}}\})^{**}$. If $\oplus$ were a true categorical coproduct, then these two definitions are equal. However, for CP(**FHilb**), the definition from injections only contains the probabilistic mixtures of states from **A** and **B**, whereas the definition above from the sum of effects captures coherent superpositions of states.

Definition 6.1.1 (Categorical Interpretation)**.** *The interpretation* $[\![\hat{A}]\!]$ *of a quantum type* $\hat{A}$ *is the object in* Caus[CP(**FHilb**)] *defined recursively as:*

$$
\begin{aligned}
[\![\top]\!] &:= \mathbf{I} \\
[\![\hat{A} \otimes \hat{B}]\!] &:= [\![\hat{A}]\!] \otimes [\![\hat{B}]\!] \\
[\![\hat{A} \oplus \hat{B}]\!] &:= [\![\hat{A}]\!] \oplus [\![\hat{B}]\!] \\
[\![\hat{A} \parr \hat{B}]\!] &:= [\![\hat{A}]\!] \parr [\![\hat{B}]\!] \\
[\![\hat{A} \multimap \hat{B}]\!] &:= [\![\hat{A}]\!] \multimap [\![\hat{B}]\!]
\end{aligned}
$$

Theorem 6.1.2. *For any first-order type* $\tilde{A}$*, its interpretation* $[\![\tilde{A}]\!]$ *is a first-order system in* Caus[CP(**FHilb**)].

Proof. By structural induction over the grammar of first-order types. For $\top$, $[\![\top]\!] = \mathbf{I}$ and $c^*_{\mathbf{I}} = c_{\mathbf{I}} = \{1\} = \{\bar{\top}_I\}$, so it is first-order. The case for $\tilde{A} \otimes \tilde{B}$ follows from Proposition 5.3 in [29] which shows that the collection of first-order systems is closed under tensor product.

For $\tilde{A} \oplus \tilde{B}$, we assume by the induction hypothesis that $c^*_{[\![\tilde{A}]\!]} = \{\bar{\top}_{[\![\tilde{A}]\!]}\}$ and $c^*_{[\![\tilde{B}]\!]} = \{\bar{\top}_{[\![\tilde{B}]\!]}\}$. It immediately follows that $c^*_{[\![\tilde{A}\oplus\tilde{B}]\!]} = \{\pi^*_A \oplus \pi^*_B | \pi_A \in c^*_{[\![\tilde{A}]\!]}, \pi_B \in c^*_{[\![\tilde{B}]\!]}\}^{**} = \{\bar{\top}_{[\![\tilde{A}]\!]} \oplus \bar{\top}_{[\![\tilde{B}]\!]}\}^{**}$. Using Equation 6.2, this is equal to $\{\bar{\top}_{[\![\tilde{A}]\!]\oplus[\![\tilde{B}]\!]}\}^{**} = \{\bar{\top}_{[\![\tilde{A}\oplus\tilde{B}]\!]}\}^{**}$. From the definition of causality (Equation 3.1), $\{\bar{\top}_{[\![\tilde{A}\oplus\tilde{B}]\!]}\}^{*}$ is exactly the set of causal states for $[\![\tilde{A} \oplus \tilde{B}]\!]$, so it follows that any effect in $c^*_{[\![\tilde{A}\oplus\tilde{B}]\!]} = \{\bar{\top}_{[\![\tilde{A}\oplus\tilde{B}]\!]}\}^{**}$ is equal to $\bar{\top}_{[\![\tilde{A}\oplus\tilde{B}]\!]}$ by enough causal states. We can now conclude that $[\![\tilde{A} \oplus \tilde{B}]\!]$ is a first-order system. □

The combination of this and Lemma 3.4.7 means that there is an isomorphism between the interpretations of processes of type $\tilde{A} \parr \tilde{B}$ and $\tilde{A} \otimes \tilde{B}$. One direction of this isomorphism is given by *mix*. The inclusion of the *separate* construct and its rule in the type system allows us to realise the other direction from $\tilde{A} \parr \tilde{B}$ to $\tilde{A} \otimes \tilde{B}$.

6.2 Induced Signalling and the Quantum Switch

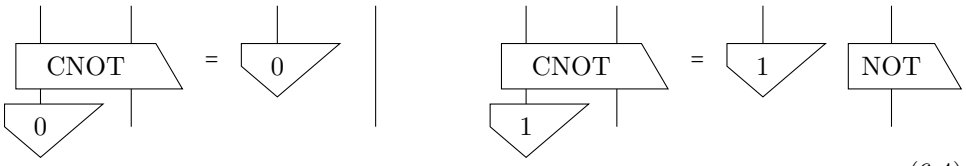
From the definition of the quantum switch, the control system affects the order of the switched channels, and consequently the final channel produced. We also know that the channels can feed back into the control system as achieved in the construction in [12] for discrimination between a pair of commuting or anticommuting channels.

In Section 4.3, we discussed how controlled-unitaries and the quantum switch can be expressed as instances of quantum alternation. Furthermore, alternation itself was a special case of the quantum control flow offered by structures such as QML's **case**$^\circ$ statement [3]. If we adopt a higher-order version of the **case**$^\circ$ statement, the signalling between the control and affected systems is not obvious.

Let us consider the simpler example of a controlled-not gate. We can describe this the following program:

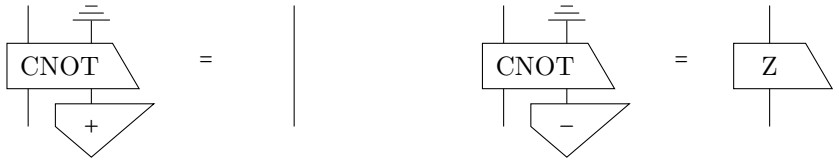
$$
\begin{aligned}
\mathit{cnot}\ c \ =\ & \mathbf{case}^\circ\ c\ \mathbf{of} \\
& \{\mathbf{inl}\ x \Rightarrow (\mathbf{inl}\ x, \mathit{id}) \\
& \mid \mathbf{inr}\ y \Rightarrow (\mathbf{inr}\ y, \mathit{not})\}
\end{aligned}
\tag{6.3}
$$

This specifies the action of the CNOT gate on the computational basis states:



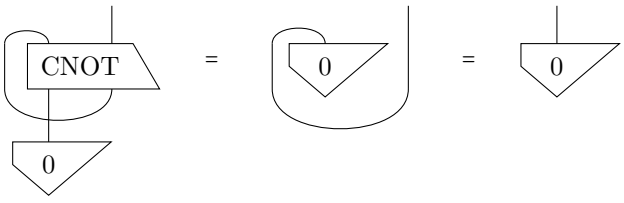
(6.4)

In each case, there is no signalling occurring between the channel on the right and the state on the left, but we know that the actual CNOT gate can permit signalling:

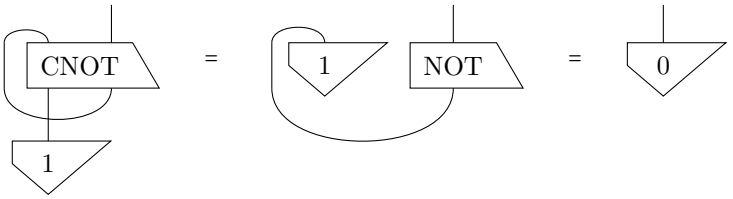


(6.5)

The implications of this are that operations possible on the individual programs combined by the alternation may not be valid on the combination. For example, in each of the results in Equation 6.4 we are able to connect the control system output to the input of the channel and produce $|0\rangle\langle 0|$.

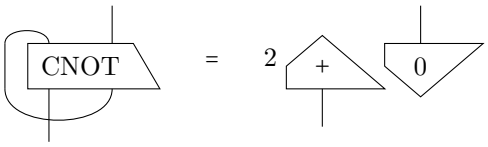


(6.6)



(6.7)

However, the result of performing the same connection on the CNOT gate itself does not satisfy causality:



(6.8)

In definitions of the quantum switch, this signalling constraint has been enforced by requiring the control system to be completely external to the affected system (i.e. a system cannot control itself). We find similar constraints for quantum alternation across the literature [9, 53, 52].

Each of the branches of the CNOT program (6.3) could reasonably be typed as $qbit \otimes (qbit \multimap qbit)$. However, naively typing the CNOT construction as $qbit \multimap qbit \otimes (qbit \multimap qbit)$ would permit this invalid operation. We must alter the type to account for the signalling induced by the quantum alternation. This is done by the following theorem.

Theorem 6.2.1. *Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be some first-order systems in* Caus[CP(**FHilb**)]. *Given two pure processes* $\hat{\psi_0}, \hat{\psi_1} \in c_{\mathbf{C}}$, *the map* $\mathrm{Alt}_{\mathbf{A}\oplus\mathbf{B}}[\hat{\psi_0}, \hat{\psi_1}]$ *defined by the alternation of* $\hat{\psi_0}$ *and* $\hat{\psi_1}$ *with* $\mathbf{A}\oplus\mathbf{B}$ *control (*$|\phi_{\mathbf{A}}\rangle \oplus 0 \mapsto (|\phi_{\mathbf{A}}\rangle \oplus 0) \otimes \psi_0, 0 \oplus |\phi_{\mathbf{B}}\rangle \mapsto (0 \oplus |\phi_{\mathbf{B}}\rangle) \otimes \psi_1$*) is in* $c_{(\mathbf{A}\oplus\mathbf{B})\multimap((\mathbf{A}\oplus\mathbf{B})\parr\mathbf{C})}$.

Proof. It is sufficient to show that applying any effect in the dual set yields 1. We note the existence of the following isomorphism:

$$\begin{aligned}((\mathbf{A}\oplus\mathbf{B}) \multimap ((\mathbf{A}\oplus\mathbf{B}) \parr \mathbf{C}))^* &\cong ((\mathbf{A}\oplus\mathbf{B})^* \parr ((\mathbf{A}\oplus\mathbf{B}) \parr \mathbf{C}))^* \\ &\cong (\mathbf{A}\oplus\mathbf{B}) \otimes ((\mathbf{A}\oplus\mathbf{B}) \parr \mathbf{C})^* \\ &\cong (\mathbf{A}\oplus\mathbf{B}) \otimes (\mathbf{A}\oplus\mathbf{B})^* \otimes \mathbf{C}^*\end{aligned}$$

This means it is enough to consider an arbitrary state $\rho \in c_{\mathbf{A}\oplus\mathbf{B}}$ and arbitrary effects $\pi \in c_{\mathbf{C}^*}$ and $\pi' \in c_{(\mathbf{A}\oplus\mathbf{B})^*}$. Since $\mathbf{A}$ and $\mathbf{B}$ are first-order, so is $\mathbf{A}\oplus\mathbf{B}$, hence $\pi' = \bar{\top}_{A\oplus B}$.

Consider the case where ρ is a pure state $\sum_{i,j=0}^{1} \alpha_i\overline{\alpha_j} |\phi_i\rangle\langle\phi_j|$ where $|\phi_0\rangle = |\xi_A\rangle\oplus 0$, $|\phi_1\rangle = 0 \oplus |\xi_B\rangle$ (for some $|\xi_A\rangle\langle\xi_A| \in c_{\mathbf{A}}, |\xi_B\rangle\langle\xi_B| \in c_{\mathbf{B}}$) and $|\alpha_0|^2 + |\alpha_1|^2 = 1$. In the following, we use the fact that $|\phi_0\rangle$ and $|\phi_1\rangle$ are orthogonal ($\bar{\top}_{\mathbf{A}\oplus\mathbf{B}}(|\phi_0\rangle\langle\phi_1|) = \bar{\top}_{\mathbf{A}\oplus\mathbf{B}}(|\phi_1\rangle\langle\phi_0|) = 0$) and π is a causal effect ($\pi \circ |\hat{\psi_i}\rangle\langle\hat{\psi_i}| = 1$).

$$\begin{aligned}(\bar{\top}_{\mathbf{A}\oplus\mathbf{B}} \otimes \pi) \circ \mathrm{Alt}_{\mathbf{A}\oplus\mathbf{B}}[\hat{\psi_0}, \hat{\psi_1}] \circ \rho &= \sum_{i,j=0}^{1} \alpha_i\overline{\alpha_j}\, \bar{\top}_{\mathbf{A}\oplus\mathbf{B}} (|\phi_i\rangle\langle\phi_j|) \otimes (\pi \circ |\hat{\psi_i}\rangle\langle\hat{\psi_j}|) \\ &= \sum_{i=0}^{1} |\alpha_i|^2\, \bar{\top}_{\mathbf{A}\oplus\mathbf{B}} (|\phi_i\rangle\langle\phi_i|) \otimes (\pi \circ |\hat{\psi_i}\rangle\langle\hat{\psi_i}|) \\ &= \sum_{i=0}^{1} |\alpha_i|^2 \\ &= 1\end{aligned}$$

Since this is the case for all pure ρ, it must hold for all mixed states by linearity. □

The importance of this result is in the preservation of the $\mathbf{C}$ subsystem; quantum alternation can only introduce signalling between the control and affected systems and not within the affected system itself. For example, if we alternate over a pair of non-signalling channels, they still remain locally non-signalling in the sense that, whilst they can both signal with the control system, it is not possible to signal from one to the other.

We can also see that $(\mathbf{A}\oplus\mathbf{B}) \multimap ((\mathbf{A}\oplus\mathbf{B}) \parr \mathbf{C})$ is, in general, the most specific type which can be given. We can recover each of the $\mathbf{C}$ processes, so we cannot refine $\mathbf{C}$ to a more specific causal set. Furthermore, the CNOT example shows that the result may not live in $(\mathbf{A}\oplus\mathbf{B}) \multimap ((\mathbf{A}\oplus\mathbf{B}) \otimes \mathbf{C})$.

Corollary 6.2.2. *In* Caus[CP(**FHilb**)], *the quantum switch on channels over* $\mathbf{C}$ *with a* $\mathbf{A}\oplus\mathbf{B}$ *control system (where* $\mathbf{A}$ *and* $\mathbf{B}$ *are first-order systems) is a process in* $(\mathbf{A}\oplus\mathbf{B}) \multimap (\mathbf{A}\oplus\mathbf{B}) \parr ((\mathbf{C} \multimap \mathbf{C}) \otimes (\mathbf{C} \multimap \mathbf{C}) \multimap (\mathbf{C} \multimap \mathbf{C}))$.

Proof. Recall that the quantum switch can be defined as the alternation of the programs $\lambda x.\lambda y.\lambda z.x(yz)$ and $\lambda x.\lambda y.\lambda z.y(xz)$. These are the programs taking two channels and combining them in the two possible definite orders. Both correspond to

processes in $(\mathbf{C} \multimap \mathbf{C}) \multimap (\mathbf{C} \multimap \mathbf{C}) \multimap (\mathbf{C} \multimap \mathbf{C}) \cong (\mathbf{C} \multimap \mathbf{C}) \otimes (\mathbf{C} \multimap \mathbf{C}) \multimap (\mathbf{C} \multimap \mathbf{C})$. The result then follows directly from Theorem 6.2.1. □

In Section 3.5, we claimed that the type of the quantum switch with a qubit control system is $\mathbb{C}^2 \multimap \mathbb{C}^2 \parr ((\mathbf{A} \multimap \mathbf{A}) \otimes (\mathbf{A} \multimap \mathbf{A}) \multimap (\mathbf{A} \multimap \mathbf{A}))$, which follows as a special case of Corollary 6.2.2.

The type rule for the quantum switch in our programming language is as follows:

$$\frac{!\Delta, \Gamma_0 \rhd M : \hat{A} \multimap \hat{A} \qquad !\Delta, \Gamma_1 \rhd N : \hat{A} \multimap \hat{A} \qquad !\Delta, \Gamma_2 \rhd P : \tilde{B} \oplus \tilde{C}}{!\Delta, \Gamma_1, \Gamma_2 \rhd \textit{switch } M \textit{ and } N \textit{ by } P : (\hat{A} \multimap \hat{A}) \parr (\tilde{B} \oplus \tilde{C})} \ (\textit{switch})$$

The processes M and N being switched are required to be transformations over a quantum type. Using Definition 6.1.1, we can associate these types to objects in Caus[CP(**FHilb**)]. Moreover, the control system P is of a first-order type, so by Theorem 6.1.2 it is associated with a first-order system in Caus[CP(**FHilb**)]. Using these associations, the type of the quantum switch according to Corollary 6.2.2 is associated to $\tilde{B} \oplus \tilde{C} \multimap ((\hat{A} \multimap \hat{A}) \otimes (\hat{A} \multimap \hat{A})) \multimap (\hat{A} \multimap \hat{A}) \parr (\tilde{B} \oplus \tilde{C})$. If we apply this to the inputs M, N, and P, the result will have type $(\hat{A} \multimap \hat{A}) \parr (\tilde{B} \oplus \tilde{C})$, which is exactly the type assigned to *switch M and N by P*. In this sense, the typing rule for the quantum switch in our programming language is consistent with its interpretation in Caus[CP(**FHilb**)].

6.3 Properties of the Language

A well-designed functional programming language should be sound in the sense that well-typed programs do not reach error states - in this case, a quantum closure is not well formed when the term is not a value but it also cannot be reduced. This is the principle behind the property of type safety. We can build up to a proof of this by breaking it down into a handful of lemmas. A number of the lemmas are properties that are intuitively natural for quantum systems, but we must nonetheless show that the language we defined satisfies them.

In this section, we assume that a variables x range over both term variables and register variables.

Lemma 6.3.1. *If $x \notin \mathrm{FV}(M)$, then $\Delta, x :!A \rhd M : B$ iff $\Delta \rhd M : B$.*

Proof. By structural induction on the derivation of the typing judgement. □

This first lemma shows that making the weakening rule of ! implicit does not affect how a term can be typed. In essence, having extra classical data at the start of the program on top of what we need will not affect the ability to execute the program.

Lemma 6.3.2. *If $\Delta \rhd M : B$, then $\forall x \in |\Delta| \setminus \mathrm{FV}(M). \exists A. \Delta(x) = !A$.*

Proof. By structural induction on the derivation of the typing judgement. □

The consequence of this lemma is that the only resources we are able to discard implicitly are classical. Every quantum system mentioned in the typing context must be mentioned in M. In order to discard a quantum system, we must do so explicitly, specifying the exact discarding operation to be applied.

Lemma 6.3.3. *For any value V, if $\Delta \rhd V : !A$ then $\forall x \in |\Delta|.\exists B.\Delta(x) = !B$.*

Proof. The only cases in the grammar of values that can be given a !-type are variables or !-suspensions. In either case, the typing rule given for these requires the typing context to be of the form $!\Delta$.

□

The only point at which a term can be copied is on the applying a λ-abstraction to something of !-type, and since we have a call-by-value reduction scheme only values can be copied. This lemma shows that there is no hidden quantum system within values of !-type which could be copied by this procedure, in line with the no-cloning theorem.

We can now prove the properties required for type safety.

Lemma 6.3.4 (Substitution)**.** *If $!\Delta, \Gamma_1, x : A \rhd M : B$ and $!\Delta, \Gamma_2 \rhd V : A$, then $!\Delta, \Gamma_1, \Gamma_2 \rhd M[V/x] : B$*

Proof. By structural induction on the derivation of $!\Delta, \Gamma_1, x : A \rhd M : B$, using Lemmas 6.3.1 and 6.3.3 as appropriate. The $(\oplus.E)$ case demonstrates how the non-trivial cases can be handled:

Suppose that the instance of the $(\oplus.E)$ rule in the derivation is:

$$\frac{!\Delta, \Gamma_1 \rhd P : A \oplus B \qquad !\Delta, \Gamma_2, x : A \rhd M : C \qquad !\Delta, \Gamma_2, y : B \rhd N : C}{!\Delta, \Gamma_1, \Gamma_2 \rhd \mathit{match}\ P\ \mathit{with}\ (x \mapsto M \mid y \mapsto N) : C}$$

We are aiming to substitute $!\Delta', \Gamma_0 \rhd V : D$ for $z : D \in !\Delta, \Gamma_1, \Gamma_2$. As we are working up to α-equivalence, we can assume that $x, y \notin \mathrm{FV}(V)$, in accordance with notions of capture-avoiding substitution.

If $z : D \in !\Delta$, then $D = !D'$ for some type D'. By Lemma 6.3.3, Γ_0 is of the form $!\Gamma_0$. By Lemma 6.3.1, we have that $!(\Delta \setminus z : D), !\Sigma \rhd V : D$ where $!\Delta', !\Gamma_0 \subseteq !(\Delta \setminus z : D), !\Sigma$. Using the induction hypothesis, we have that $!(\Delta \setminus z : D), !\Sigma, \Gamma_1 \rhd P[V/z] : A \oplus B$, and similar for $M[V/z]$ and $N[V/z]$. From here, we can use the $(\oplus.E)$ rule:

$$\frac{\begin{array}{cc} & !(\Delta \setminus z : D), !\Sigma, \Gamma_2, x : A \rhd M[V/z] : C \\ !(\Delta \setminus z : D), !\Sigma, \Gamma_1 \rhd P[V/z] : A \oplus B & !(\Delta \setminus z : D), !\Sigma, \Gamma_2, y : B \rhd N[V/z] : C \end{array}}{!(\Delta \setminus z : D), !\Sigma, \Gamma_1, \Gamma_2 \rhd (\mathit{match}\ P\ \mathit{with}\ (x \mapsto M \mid y \mapsto N))[V/z] : C}$$

If $z : D$ is instead in Γ_1 or Γ_2, the proof follows similarly by applying the induction hypothesis on the relevant subterm(s). □

Lemma 6.3.5 (Type Preservation)**.** *For any well-formed quantum closure $[\rho, L, M] : A$, if $[\rho, L, M] \to [\rho', L', M']$ then $[\rho', L', M'] : B$ is well-formed with $A = B$.*

Proof. By structural induction on the derivation of the reduction, using the Substitution Lemma (6.3.4) for the cases involving substitution of values. □

Lemma 6.3.6 (Progress)**.** *For any well-formed quantum closure $[\rho, L, M] : A$, either M is a value or there exists some $[\rho', L', M']$ such that $[\rho, L, M] \to [\rho', L', M']$.*

Proof. By structural induction over the type derivation in $[\rho, L, M]$. □

Corollary 6.3.7 (Type Safety)**.** *Any reduction sequence starting from a well-formed quantum closure is either infinite or terminates with a value after a finite number of reduction steps.*

Proof. This follows straightforwardly from Type Preservation (6.3.5) and Progress (6.3.6). □

The reduction of a program is determined entirely by the term, thanks to embedded concrete type information which fixes the dimensions of the Choi-Jamiołkovski operator (e.g. in $\lambda x : A.M$ or $inj_{A \oplus -}(M)$). Type Preservation prevents this syntax-bound type information from becoming invalid after a successful check. As a result, it is sufficient to check the type once and for all before we execute a program and, furthermore, we do not need to carry around any additional type information during reduction.

Chapter 7

Extensions

This chapter provides a discussion of future extensions for this programming language

7.1 Quantum Alternation

As discussed in Section 4.3, the quantum switch is an instance of higher-order quantum alternation. Providing a mechanism for alternation (or the more general coherent control, such as in QML's **case**$^\circ$ statements [3]) could break the switch primitive down into finer elements. The ability to describe processes such as the CNOT gate with the explicit constructs of the programming language could also reduce the dependence on the provision of the elementary operations U, making the language more concrete and architecture-independent.

Quantum alternation is only valid when it combines pure, white-box processes. For our programming language, we made the design choice to permit mixed states and processes by allowing measurements. A prerequisite to adding alternation would be the tracking of purity of programs. QML achieves this via a separate "strict" type system: any constant is given a strict type; we can freely remove the strictness; measurement uses non-strict types and coherent operations use strict types. From these rules, any program with a strict type corresponds to a pure process. A similar technique could be applied here.

7.2 Infinite Data Types

Quantum algorithms are generally defined to be independent of the size of the input system - for instance, Shor's algorithm for integer factorisation works for any input consisting of a finite number of qubits. In the field of quantum complexity theory, a quantum algorithm is defined to be an infinite family of circuits, with one for each possible input size [19]. The most conventional method of describing such families using a single program is by using a recursively defined data type representing a list of identically-typed objects, $\mathit{list}(A) \coloneqq \top \oplus (A \otimes \mathit{list}(A))$, as achieved in [45]. Unravelling this recursive definition, we find that it is equivalent to $\top \oplus A \oplus (A \otimes A) \oplus \cdots \oplus A^{\otimes n} \oplus \cdots$. As the type itself is recursively defined, they are naturally handled by recursive functions.

Given a finite collection of systems, each of which living in $\mathcal{H}$, the state of the overall collection lies in the Fock space $\bigoplus_{i=0}^{\infty} \mathcal{H}^{\otimes i}$ (see [52] for a good explanation of Fock spaces and some of their uses). The elementary operations on lists correspond to simple operations in a Fock space:

- Generating an empty list takes an empty system and applies the injection $\mathcal{H}^{\otimes 0} \to \bigoplus_{i=0}^{\infty} \mathcal{H}^{\otimes i}$.
- Adding an element to the head of the list can be achieved by combining the new element and the existing list to a state in $\mathcal{H} \otimes \bigoplus_{i=0}^{\infty} \mathcal{H}^{\otimes i}$. By distributivity, this space is equal to $\bigoplus_{i=1}^{\infty} \mathcal{H}^{\otimes i}$. We then apply the injection $\bigoplus_{i=1}^{\infty} \mathcal{H}^{\otimes i} \to \bigoplus_{i=0}^{\infty} \mathcal{H}^{\otimes i}$ to obtain another list.
- Distinguishing between an empty and non-empty list corresponds to the projective measurement from $\bigoplus_{i=0}^{\infty} \mathcal{H}^{\otimes i} = \mathcal{H}^{\otimes 0} \oplus \bigoplus_{i=1}^{\infty} \mathcal{H}^{\otimes i}$ into either $\mathcal{H}^{\otimes 0}$ or $\bigoplus_{i=1}^{\infty} \mathcal{H}^{\otimes i}$. Once we have determined that a list is non-empty, we can apply distributivity to separate the head of the list from the rest.

If we adopted the use of Fock spaces to represent lists, simulating the execution of programs would require us to represent the Choi-Jamiołkovski operator symbolically as we may no longer describe it by a finite matrix.

7.3 Inclusion of Full Linear Logic

With the introduction of the $⅋$ operator in this programming language, we now have representations of most of the operators of classical linear logic available as types of quantum systems. This begs the question of whether or not it would be useful to develop theories behind quantum interpretations of the missing operators to allow more expressiveness or flexibility of programs. In particular, we are yet to see uses of additive conjunction &, exponential disjunction ?, and the quantifiers $\forall$ and $\exists$ in the literature.

7.3.1 Additive Conjunction &

The interpretation of a unit of $A\&B$ in linear logic is a single resource unit that can be used as either a unit of A or a unit of B, and we have the freedom to choose which of these we want at the point of using the resource. In Categorical interpretations, this corresponds to the categorical product of objects, with the freedom of choice given by the presence of the projection morphisms. It is also the De Morgan Dual of $\oplus$ as $A\&B \cong (A^* \oplus B^*)^*$.

If we assume that we can make the choice of projector coherently with some state from a $\oplus$-system, the links between this and quantum alternation are very clear. We recall from Section 4.3 that we can define alternation via the matrix direct sum of Kraus operators. We can use the matrix direct sum of the Choi-Jamiołkovski operators to embed processes of type $\hat{A}\&\hat{B}$ in the space $\mathcal{H}_{\hat{A}\oplus\hat{B}}$. However, rather than alternation, this corresponds to the Choi-Jamiołkovski operator of a decoherent choice (a control system is measured, and the outcome is used to select either the $\hat{A}$ or the $\hat{B}$ process).

7.3.2 Exponential Disjunction ?

The resource-theoretic interpretation of the ? exponential is to indicate collections of resources of a given type. Such a collection can be empty, finite, or infinite. The elementary operations from the rules in linear logic available are construction of an empty collection ($W?$), construction of a singleton collection from a unit of the resource ($R?$), mapping some transformation over every element of a collection ($L?$), and merging two collections ($C?$).

One can note close similarities between this and the *list* data type, but they are fundamentally different constructions. Given a $?A$ collection, one can only perform operations locally on each A element - there is no way to accumulate information from separate units within the collection - whereas $list(A)$ permits processing the units in any order or combination. In this sense, we can view $list(A)$ as a collection of completely separate resources ($\bigoplus_{i=0}^{\infty} A^{\otimes n}$) and $?A$ as a collection of connected resources, possibly via signalling ($\bigoplus_{i=0}^{\infty} A^{⅋ n}$). We can obtain the definition of $list(A)$ from $\bigoplus_{i=0}^{\infty} A^{\otimes n}$ by applying distributivity, but this method fails for $?A$ since the distributivity is only one-sided for ⅋ - $(A ⅋ C) \oplus (B ⅋ C) \vdash (A \oplus B) ⅋ C$ but $(A \oplus B) ⅋ C \not\vdash (A ⅋ C) \oplus (B ⅋ C)$. As a result, one can prove the sequent $0 \oplus (A ⅋ ?A) \vdash ?A$ in linear logic, though the converse does not hold.

7.3.3 Polymorphism and Quantifiers

Polymorphism is the ability for one program to be applied to systems of many different types. Several forms of polymorphism exist in the study of programming languages:

- Parametric polymorphism is where a program only assumes some basic structure over its inputs but does not require a concrete type, hence can be applied to any input whose type fits some pattern. For example, given any process of type $A \otimes B$, we can always mix this to give $A ⅋ B$ regardless of what A and B actually are.

  In classical programming, we can model this by adding type variables and the universal quantifier $\forall$ to the syntax of types. One could imagine a similar method could be applied to quantum programming languages, allowing the generic mixing example above to be typed as $\forall\alpha.\forall\beta.(\alpha \otimes \beta) \multimap (\alpha ⅋ \beta)$.

  The De Morgan dual of $\forall$ is the existential quantifier $\exists$ which is useful in classical programming for encapsulation and information hiding. In a quantum context, we could type a pair of devices for encoding and decoding information as $\exists\alpha.(A \multimap \alpha) \otimes (\alpha \multimap B)$. The ability to hide the type of the encoded data via α can be useful in preventing a programmer from tampering with the encoded information.

- Ad hoc polymorphism is where multiple instances of a program are defined, each with a different type. On application, the appropriate definition is selected (if one exists) and is executed.

  Since this corresponds to a choice over a finite collection of predefined processes, we could encode this using &, where application projects out the appropriate version for the given input.

- Subtype polymorphism defines a relation between types to say where one is sufficient to be used for the other. If A is a subtype of B ($A <: B$), then an object of type A can be used as an input to any function $B \multimap C$. $A <: B$ means that A is at least as specific as B. Programs are typically defined with generic inputs and specific outputs in order to be more flexible in where it can be used. This is most typically found in object oriented programming situations.

  This has already been seen in quantum programming languages including Selinger and Valiron's quantum lambda calculus [45] where it was used to remove the need for explicit dereliction of !. As the *mix* and *separate* commands perform no actual manipulation of data, one could argue that they serve no place in quantum programs, so making them implicit in a similar fashion would benefit programmers through making programs more concise and clear.

Chapter 8

Conclusion

Indefinite causal orders provide useful computational advantages to programmers, yet they cannot be constructed in programming languages limited to classical control flow. The higher-order programming language we have presented addresses this by extending existing models with the quantum switch as a new primitive construction. The new computational model builds in higher-order structure from the ground up with each register in the quantum array capable of storing a higher-order process over finite-dimensional Hilbert spaces, modelling such processes by their Choi-Jamiołkovski operator. This is complemented in our switch primitive by generalising the definition of the quantum switch itself to allow higher-order processes to be combined in a superposition of orders.

The other novel feature of this programming language is the addition of a new type operator $\parr$ for compound systems with potential for internal signalling. This idea originated in linear logic as the De Morgan dual of the tensor product $\otimes$, with the signalling interpretation given by Kissinger and Uijlen [29]. This not only improves the safety of the language by preventing backwards-in-time signalling, but allows higher-order processes to be combined in new ways through local applications.

Whilst full categorical semantics for the programming language are beyond the scope of this thesis, we made a case for a correspondence between programs of the language and processes in Caus[CP(**FHilb**)]. With this in mind, we lifted the direct sum $\oplus$ operator from CP(**FHilb**) into Caus[CP(**FHilb**)]. The main result following this showed that quantum alternation induces signalling between the control system and the affected system but not within the affected system itself. By describing the quantum switch as an instance of quantum alternation, we showed that it exists in Caus[CP(**FHilb**)], exhibiting a concrete type for it and justifying the rule for the switch in our type system.

Our language is the first higher-order programming language to permit indefinite causal orders over black-box processes. Furthermore, whilst many other languages assume the use of qubit computers, our model can be used with implementations based on any discrete quantum system.

Future directions for this programming language could see it extended to allow for quantum alternation, infinite data types, or automated handling of the $\parr$ operator via subtyping. These features are already available in existing programming languages [3, 45] and similar techniques could be used to introduce them here.

Providing access to indefinite causal structures makes this language incredibly expressive and flexible. A programming language like the one presented here would

be useful to demonstrate the correctness of programs and aid the discovery and expression of new algorithms using indefinite causal structures.

Bibliography

[1] Samson Abramsky. "Computational interpretations of linear logic". In: *Theor. Comput. Sci.* 111.1&2 (1993), pp. 3–57.

[2] Samson Abramsky and Bob Coecke. "A categorical semantics of quantum protocols". In: *Logic in computer science, 2004. Proceedings of the 19th Annual IEEE Symposium on*. IEEE. 2004, pp. 415–425.

[3] Thorsten Altenkirch and Jonathan Grattage. "A functional quantum programming language". In: *Logic in Computer Science, 2005. LICS 2005. Proceedings. 20th Annual IEEE Symposium on*. IEEE. 2005, pp. 249–258.

[4] Thorsten Altenkirch and Alexander S Green. "The quantum IO monad". In: *Semantic Techniques in Quantum Computation* (2010), pp. 173–205.

[5] Mateus Araújo, Fabio Costa, and Časlav Brukner. "Computational advantage from quantum-controlled ordering of gates". In: *Physical review letters* 113.25 (2014), p. 250402.

[6] Mateus Araújo et al. "Quantum circuits cannot control unknown operations". In: *New Journal of Physics* 16.9 (2014), p. 093026.

[7] Pablo Arrighi and Gilles Dowek. "Linear-algebraic λ-calculus: higher-order, encodings, and confluence." In: *International Conference on Rewriting Techniques and Applications*. Springer. 2008, pp. 17–31.

[8] Miriam Backens. "The ZX-calculus is complete for stabilizer quantum mechanics". In: *New Journal of Physics* 16.9 (2014), p. 093021.

[9] Costin Bădescu and Prakash Panangaden. "Quantum Alternation: Prospects and Problems". In: *Proceedings 12th International Workshop on Quantum Physics and Logic, QPL 2015*. 2015, pp. 33–42.

[10] Howard Barnum et al. "Noncommuting mixed states cannot be broadcast". In: *Physical Review Letters* 76.15 (1996), p. 2818.

[11] David Beckman et al. "Causal and localizable quantum operations". In: *Physical Review A* 64.5 (2001), p. 052309.

[12] Giulio Chiribella. "Perfect discrimination of no-signalling channels via quantum superposition of causal structures". In: *Physical Review A* 86.4 (2012), p. 040301.

[13] Giulio Chiribella, G Mauro DAriano, and Paolo Perinotti. "Quantum circuit architecture". In: *Physical review letters* 101.6 (2008), p. 060401.

[14] Giulio Chiribella, Giacomo Mauro DAriano, and Paolo Perinotti. "Optimal cloning of unitary transformation". In: *Physical review letters* 101.18 (2008), p. 180504.

[15] Giulio Chiribella, Giacomo Mauro DAriano, and Paolo Perinotti. "Probabilistic theories with purification". In: *Physical Review A* 81.6 (2010), p. 062348.

[16] Giulio Chiribella, Giacomo Mauro DAriano, and Paolo Perinotti. "Theoretical framework for quantum networks". In: *Physical Review A* 80.2 (2009), p. 022339.

[17] Giulio Chiribella, Giacomo Mauro D'Ariano, and Paolo Perinotti. "Transforming quantum operations: quantum supermaps". In: *EPL (Europhysics Letters)* 83.3 (2008), p. 30004.

[18] Giulio Chiribella et al. "Quantum computations without definite causal structure". In: *Physical Review A* 88.2 (2013), p. 022318.

[19] Richard Cleve. "An introduction to quantum complexity theory". In: *Quantum Computation And Quantum Information Theory: Reprint Volume with Introductory Notes for ISI TMR Network School*. World Scientific, 2000, pp. 103–127.

[20] Bob Coecke and Ross Duncan. "Interacting quantum observables: categorical algebra and diagrammatics". In: *New Journal of Physics* 13.4 (2011), p. 043016.

[21] Bob Coecke and Aleks Kissinger. *Picturing quantum processes*. Cambridge University Press, 2017.

[22] Daniel Ebler, Sina Salek, and Giulio Chiribella. "Enhanced Communication with the Assistance of Indefinite Causal Order". In: *Physical Review Letters* 120.12 (2018), p. 120502.

[23] Yuan Feng, Nengkun Yu, and Mingsheng Ying. "Model checking quantum Markov chains". In: *Journal of Computer and System Sciences* 79.7 (2013), pp. 1181–1198.

[24] Yuan Feng et al. "QPMC: a model checker for quantum programs and protocols". In: *International Symposium on Formal Methods*. Springer. 2015, pp. 265–272.

[25] Nicolai Friis et al. "Implementing quantum control for unknown subroutines". In: *Physical Review A* 89.3 (2014), p. 030303.

[26] Simon J Gay. "Quantum programming languages: Survey and bibliography". In: *Mathematical Structures in Computer Science* 16.4 (2006), pp. 581–600.

[27] Simon J Gay and Rajagopal Nagarajan. "Communicating quantum processes". In: *ACM SIGPLAN Notices*. Vol. 40. 1. ACM. 2005, pp. 145–157.

[28] Alexander S Green et al. "Quipper: a scalable quantum programming language". In: *ACM SIGPLAN Notices*. Vol. 48. 6. ACM. 2013, pp. 333–342.

[29] Aleks Kissinger and Sander Uijlen. "A categorical semantics for causal structure". In: *Logic in Computer Science (LICS), 2017 32nd Annual ACM/IEEE Symposium on*. IEEE. 2017, pp. 1–12.

[30] Aleks Kissinger and Sander Uijlen. "Picturing Indefinite Causal Structure". In: *Proceedings 13th International Conference on Quantum Physics and Logic, QPL 2016*. 2016, pp. 87–94.

[31] Emmanuel Knill. *Conventions for quantum pseudocode*. Tech. rep. Citeseer, 1996.

[32] Joachim Lambek and Philip J Scott. *Introduction to higher-order categorical logic*. Vol. 7. Cambridge University Press, 1988.

[33] Kang Feng Ng and Quanlong Wang. *A universal completion of the ZX-calculus*. arXiv:1706.09877. 2017.

[34] Michael A Nielsen and Isaac L Chuang. "Programmable quantum gate arrays". In: *Physical Review Letters* 79.2 (1997), p. 321.

[35] Bernhard Ömer. *Structured quantum programming*. na, 2003.

[36] Michał Oszmaniec et al. "Creating a superposition of unknown quantum states". In: *Physical review letters* 116.11 (2016), p. 110403.

[37] Simon Perdrix. "Quantum patterns and types for entanglement and separability". In: *Electronic Notes in Theoretical Computer Science* 170 (2007), pp. 125–138.

[38] Lorenzo M Procopio et al. "Experimental superposition of orders of quantum gates". In: *Nature communications* 6 (2015), p. 7913.

[39] John C Reynolds. "Towards a theory of type structure". In: *Programming Symposium*. Springer. 1974, pp. 408–425.

[40] Giulia Rubino et al. *Experimental Entanglement of Temporal Orders*. arXiv:1712.06884. 2017.

[41] Giulia Rubino et al. "Experimental verification of an indefinite causal order". In: *Science advances* 3.3 (2017), e1602589.

[42] Amr Sabry, Benoît Valiron, and Juliana Kaizer Vizzotto. "From Symmetric Pattern-Matching to Quantum Control". In: *International Conference on Foundations of Software Science and Computation Structures*. Springer. 2018, pp. 348–364.

[43] Peter Selinger. "Dagger compact closed categories and completely positive maps". In: *Electronic Notes in Theoretical computer science* 170 (2007), pp. 139–163.

[44] Peter Selinger. "Towards a quantum programming language". In: *Mathematical Structures in Computer Science* 14.4 (2004), pp. 527–586.

[45] Peter Selinger, Benoıt Valiron, et al. "Quantum lambda calculus". In: *Semantic Techniques in Quantum Computation* (2009), pp. 135–172.

[46] Mehdi Soleimanifar and Vahid Karimipour. "No-go theorem for iterations of unknown quantum gates". In: *Physical Review A* 93.1 (2016), p. 012344.

[47] Jean-Pierre Talpin and Pierre Jouvelot. "The type and effect discipline". In: *Information and computation* 111.2 (1994), pp. 245–296.

[48] Naoyuki Tamura. "Users guide of a linear logic theorem prover (llprover)". In: *Department of Computer and Systems Engineering, Faculty of Engineering, Kobe University, Japan* (1998).

[49] André Van Tonder. "A lambda calculus for quantum computation". In: *SIAM Journal on Computing* 33.5 (2004), pp. 1109–1135.

[50] Dave Wecker and Krysta M Svore. *LIQUi|⟩: A Software Design Architecture and Domain-Specific Language for Quantum Computing*. arXiv:1402.4467. 2014.

[51] William K Wootters and Wojciech H Zurek. “A single quantum cannot be cloned”. In: *Nature* 299.5886 (1982), pp. 802–803.

[52] Mingsheng Ying. *Quantum Recursion and Second Quantisation*. arXiv:1405.4443. 2014.

[53] Mingsheng Ying, Nengkun Yu, and Yuan Feng. *Alternation in quantum programming: from superposition of data to superposition of programs*. arXiv:1402.5172. 2014.

[54] Magdalena Zych et al. *Bell's Theorem for Temporal Order*. arXiv:1708.00248. 2017.

Appendix A

Lambda Calculus

The lambda calculus is a popular model of classical computing which, using an incredibly minimal set of rules, is as computationally powerful as Turing Machines or, in fact, any other known classical model. It consists of a grammar for terms (representing both programs and state) and some equivalence relations defining identical programs (i.e. those which will yield the same result). This equivalence is defined based on β-reduction, which specifies how the term changes as a result of a single elementary computation step, akin to operational semantics of a programming language. Some celebrated results from the study of lambda calculus include a response to the Halting problem and the introduction of lambda expressions into programming languages.

Many interesting variants of lambda calculus exist for capturing specific behaviours of interest or allowing certain forms of logical reasoning. The simply-typed lambda calculus has long been known to have correspondence with a simple predicate logic and cartesian-closed categories [32]. Polymorphic lambda calculus [39] allows for quantification over type variables, extending the correspondence to first-order logic. A linear lambda calculus [1] exists as a model of intuitionistic linear logic. Additionally, the effect calculus [47] provides a flexible way of keeping track of the possible side-effects of computation, such as IO interaction or non-determinism.

A.1 Terms and Reduction

The grammar of terms of the lambda calculus is given as:

$$M ::= x \mid \lambda x.M \mid MN \tag{A.1}$$

At some level, every term in the lambda calculus is defined to be (or refer to) a function. Here, we let x range over some set of variable names. Lambda abstractions $\lambda x.M$ resemble functions where x is a name given to the input and M is the computation executed on applying the function. The compound term MN states that the function M is applied to input N. Variables x are placeholders for objects such as inputs or elements of the environment. The simplest examples include the identity function $\lambda x.x$ and an encapsulation of application $\lambda x.\lambda y.xy$.

A variable x is bounded if it is within some $\lambda x.M$, otherwise it is free. Since bound variables are just placeholder names for something as-so-far undefined, the name of the variable should not make any difference to the interpretation of the

lambda term - for example, $\lambda x.xz$ and $\lambda y.yz$ perform the same operation on any input. We formalise this to give a simple notion of equivalent terms:

Definition A.1.1 (α-Equivalence)**.** *Two terms are α-equivalent ($M \equiv_\alpha N$) when we can transform M into N only by renaming all instances of some bound variables.*

Given that the only constructs we have in this language correspond to functions and applications, the only computational actions we can take are performing said applications. β-reduction formally captures this to define a single step of computation as the substitution of an input in place of all occurrences of a bound variable (possibly applying some α-equivalence as needed).

Definition A.1.2 (β-Reduction)**.** *The relation $\to_\beta$ between lambda terms is defined inductively by the following rules:*

$$\frac{}{(\lambda x.M)N \to_\beta M[N/x]} \qquad \frac{N \equiv_\alpha M \qquad M \to_\beta M' \qquad M' \equiv_\alpha N'}{N \to_\beta N'}$$

$$\frac{M \to_\beta M'}{\lambda x.M \to_\beta \lambda x.M'} \qquad \frac{M \to_\beta M'}{MN \to_\beta M'N} \qquad \frac{N \to_\beta N'}{MN \to_\beta MN'}$$

This is the extension of the relation between β-redexes $(\lambda x.M)N$ and their β-reducts $M[N/x]$ (up to α-equivalence) to a congruence relation.

In most cases, when discussing reduction, we refer to the reflexive, transitive closure $\to^*$ of β-reduction, with β-equivalence $\equiv_\beta$ referring to the symmetric closure of the reduction relation.

This reduction is clearly non-deterministic since any term of the form MN could potentially reduce in either the M or N branches of the syntax tree. In general, we make the assumption that we can freely choose which reduction to take when multiple options are available. This claim is backed up by the confluence property: if $M \to^* M_0$ and $M \to^* M_1$, then there is some M' such that both $M_0 \to^* M'$ and $M_1 \to^* M'$. So with respect to long-term reachability it does not matter which path we take.

Reduction schemes specify a fixed method for choosing which subterm to reduce. The most significant of these are "call-by-name" (left-most), prioritising reducing M up to a λ-abstraction in MN, or "call-by-value" (right-most), prioritising reducing N in MN.

Values in the lambda calculus are represented by terms in β-normal form.

Definition A.1.3 (β-Normal Form)**.** *A term is in β-normal form (β-NF) if it contains no subterms of the form $(\lambda x.M)N$.*

From this condition, we can see that any term in β-NF cannot be reduced. If a term has a β-NF term (i.e. it can reduce to some term in β-NF), then this β-NF term is unique up to α-equivalence.

However, not every term has a β-normal form, with such examples corresponding to non-terminating computation. For example, $\Omega \coloneqq (\lambda x.xx)(\lambda x.xx)$ only has one β-redex, and reducing it produces Ω again (up to α-equivalence). The normal-order scheme reduces the leftmost of the β-redexes which are not contained within any other β-redex. For the term MN, if $M = \lambda x.M'$ then we reduce it to $M'[N/x]$; if M is in some other β-normal form, it will reduce within N; otherwise it will prioritise

reducing M. It is possible to show that normal-order reduction will always reach a β-normal form if the original term possesses one. The set of β-normal forms can be described by a regular language.

Despite the incredibly minimal syntax, it is possible to encode a number of more significant structures within lambda calculus. Church numerals represent natural numbers by the terms $\underline{n} := \lambda f.\lambda x.f^n x$ (where $f^0 x = x$ and $f^{n+1} x = f(f^n x)$), from which we can define the successor function, addition, and other arithmetic operations. Other notable constructions include pairs of terms, Boolean values, conditional expressions, and primitive recursion, which can all be found in any text book on the topic.

A.2 Type Systems

The grammar of lambda terms is incredibly flexible and expressive, but we often find a lot of terms will have no sensible interpretation as a physical process. Type systems provide a way to restrict the set of valid terms to those with a "nice" interpretation by specifying the interface of a program and enforcing that only terms with complementary interfaces can be composed. At the level of programming languages, types can provide a sanity check that the program matches the programmer's intentions and help to build clear documentation by describing the structure of the system.

A typing judgement is a logical statement that a term is sound under some set of typing rules and assumptions about the environment. We should read $\Gamma \vdash M : A$ as stating that a proof exists in our type system to show that the term M has type A in a context Γ (a map from free variables to their types).

The simply-typed lambda calculus uses the same grammar of terms as the untyped lambda calculus of the previous section but introduces the type system defined by the following rules:

$$\frac{}{\Gamma, x : A \vdash x : A}$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B}$$

$$\frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

The restrictions put in place by this type system grant the Strong Normalisation property: every well-typed term will reduce to β-normal form in a finite number of steps of β-reduction, regardless of which path is taken. This clearly limits the language to a strict subset of the computable functions, which may seem like a disadvantage, but it guarantees the elimination of poorly-constructed or nonsensical lambda terms which, in most applications, is more preferred than completeness.

The "simply-typed lambda calculus" also refers to another form with explicit syntax and types for handling operations on pairs of objects. In literature, the Curry-Howard-Lambek correspondence refers to the link between the rules of the type system for this form with explicit pairs, the inference rules of a natural deduction system for propositional logic with conjunction and implication, and the constructions available in cartesian-closed categories [32]. By altering the type system, we can build calculi that correspond with other forms of logic or classes of categories.

Appendix B

Linear Logic

B.1 Introduction to Liner Logic

The propositional logic which corresponds to the simply-typed lambda calculus is concerned with mathematical proofs, where assumptions and any lemmas we prove can be used arbitrarily many times in the proof of the goal. Linear logic, on the other hand, models the management and transformation of resources, allowing the expression of a finite number of available resources or transformers and enforcing that every resource is tracked (i.e. if some resource from the environment is not actively used, then it persists and is carried on through to the output). This has been particularly significant in modelling quantum systems as this "resource-aware" paradigm can be used to enforce compliance with the no-cloning theorem, as well as ensuring any quantum process defined is linear in its inputs.

The operators in linear logic can be divided into categories:

- Multiplicatives capture the idea of a pair of objects. $A \otimes B$ means we have one unit of resource A and one of B, both of which can be used freely. $A \parr B$ is also a pair of A and B, but restricted to only allow transformations to be applied locally. The units of these are 1 and 0 which represent systems we can freely created or destroyed.

- Additives allow the representation of a single unit adopting behaviour from multiple different resource types. The difference between the two operators is whether or not we are able to choose which resource type we interpret the unit as - $A \& B$ allows us to choose between a unit of A or one of B, whereas the choice for $A \oplus B$ is either predetermined or determined by something out of our control. The units $\top$ and $\bot$ represent impossible systems. A unit of $\bot$ can never be created, so if the environment provides it to us, we have no restriction on how we can use it (similar to assuming logical false in predicate logic).

- Implications represent resource transformations; combining a transformer with its corresponding input resource gives the output resource. Linear implication $A \multimap B$ is exactly this, where one unit is consumed and one is produced. The dual A^* of a resource type A is some process which eliminates a unit of A. Each of these can be used to define the other as $A \multimap B \cong A^* \parr B$

$$\frac{}{A \Rightarrow A}\ (\text{Ax}) \qquad \frac{\Gamma \Rightarrow A, \Delta \qquad \Gamma', A \Rightarrow \Delta'}{\Gamma, \Gamma' \Rightarrow \Delta, \Delta'}\ (\text{Cut})$$

$$\frac{\Gamma, A, B \Rightarrow \Delta}{\Gamma, A \otimes B \Rightarrow \Delta}\ (\text{L}\otimes) \qquad \frac{\Gamma \Rightarrow A, \Delta \qquad \Gamma' \Rightarrow B, \Delta'}{\Gamma, \Gamma' \Rightarrow A \otimes B, \Delta, \Delta'}\ (\text{R}\otimes)$$

$$\frac{\Gamma, A \Rightarrow \Delta \qquad \Gamma', B \Rightarrow \Delta'}{\Gamma, \Gamma', A \parr B \Rightarrow \Delta, \Delta'}\ (\text{L}\parr) \qquad \frac{\Gamma \Rightarrow A, B, \Delta}{\Gamma \Rightarrow A \parr B, \Delta}\ (\text{R}\parr)$$

$$\frac{\Gamma \Rightarrow \Delta}{\Gamma, 1 \Rightarrow \Delta}\ (\text{L1}) \qquad \frac{}{\Rightarrow 1}\ (\text{R1})$$

$$\frac{}{0 \Rightarrow}\ (\text{L0}) \qquad \frac{\Gamma \Rightarrow \Delta}{\Gamma \Rightarrow 0, \Delta}\ (\text{R0})$$

$$\frac{\Gamma, A \Rightarrow \Delta}{\Gamma, A\&B \Rightarrow \Delta}\ (\text{L}\&_1) \quad \frac{\Gamma, B \Rightarrow \Delta}{\Gamma, A\&B \Rightarrow \Delta}\ (\text{L}\&_2) \qquad \frac{\Gamma \Rightarrow A, \Delta \qquad \Gamma \Rightarrow B, \Delta}{\Gamma \Rightarrow A\&B, \Delta}\ (\text{R}\&)$$

$$\frac{\Gamma, A \Rightarrow \Delta \qquad \Gamma, B \Rightarrow \Delta}{\Gamma, A \oplus B \Rightarrow \Delta}\ (\text{L}\oplus) \qquad \frac{\Gamma \Rightarrow A, \Delta}{\Gamma \Rightarrow A \oplus B, \Delta}\ (\text{R}\oplus_1) \quad \frac{\Gamma \Rightarrow B, \Delta}{\Gamma \Rightarrow A \oplus B, \Delta}\ (\text{R}\oplus_2)$$

$$\frac{}{\Gamma, \bot \Rightarrow \Delta}\ (\text{L}\bot) \qquad \frac{}{\Gamma \Rightarrow \top, \Delta}\ (\text{R}\top)$$

$$\frac{\Gamma \Rightarrow A, \Delta}{\Gamma, A^* \Rightarrow \Delta}\ (\text{L}\ ^*) \qquad \frac{\Gamma, A \Rightarrow \Delta}{\Gamma \Rightarrow A^*, \Delta}\ (\text{R}\ ^*)$$

$$\frac{\Gamma \Rightarrow A, \Delta \qquad \Gamma', B \Rightarrow \Delta'}{\Gamma, \Gamma', A \multimap B \Rightarrow \Delta, \Delta'}\ (\text{L}\multimap) \qquad \frac{\Gamma, A \Rightarrow B, \Delta}{\Gamma \Rightarrow A \multimap B, \Delta}\ (\text{R}\multimap)$$

$$\frac{\Gamma, A[x/t] \Rightarrow \Delta}{\Gamma, \forall x.A \Rightarrow \Delta}\ (\text{L}\forall) \qquad \frac{\Gamma \Rightarrow A[x/y], \Delta}{\Gamma \Rightarrow \forall x.A, \Delta}\ (\text{R}\forall)$$

$$\frac{\Gamma, A[x/y] \Rightarrow \Delta}{\Gamma, \exists x.A \Rightarrow \Delta}\ (\text{L}\exists) \qquad \frac{\Gamma \Rightarrow A[x/t], \Delta}{\Gamma \Rightarrow \exists x.A, \Delta}\ (\text{R}\exists)$$

$$\frac{\Gamma \Rightarrow \Delta}{\Gamma, !A \Rightarrow \Delta}\ (\text{W}!) \qquad \frac{\Gamma, !A, !A \Rightarrow \Delta}{\Gamma, !A \Rightarrow \Delta}\ (\text{C}!)$$

$$\frac{\Gamma, A \Rightarrow \Delta}{\Gamma, !A \Rightarrow \Delta}\ (\text{L}!) \qquad \frac{!\Gamma \Rightarrow A, ?\Delta}{!\Gamma \Rightarrow !A, ?\Delta}\ (\text{R}!)$$

$$\frac{\Gamma \Rightarrow \Delta}{\Gamma \Rightarrow ?A, \Delta}\ (\text{W}?) \qquad \frac{\Gamma \Rightarrow ?A, ?A, \Delta}{\Gamma \Rightarrow ?A, \Delta}\ (\text{C}?)$$

$$\frac{!\Gamma, A \Rightarrow ?\Delta}{!\Gamma, ?A \Rightarrow ?\Delta}\ (\text{L}?) \qquad \frac{\Gamma \Rightarrow A, \Delta}{\Gamma \Rightarrow ?A, \Delta}\ (\text{R}?)$$

Figure B.1: Rules of linear logic. $!\Gamma$ and $?\Gamma$ refer to contexts where every formula is of the form $!A$ or $?A$ respectively. For the quantifier rules, we assume y is not free in Γ or Δ.

$$\frac{\Gamma \Rightarrow \Delta \qquad \Gamma' \Rightarrow \Delta'}{\Gamma, \Gamma' \Rightarrow \Delta, \Delta'} \qquad\qquad \frac{}{1 \Rightarrow 0}$$

Figure B.2: Mix rules for linear logic.

and $A^* \cong A \multimap 0$. We also find De Morgan duals between associated pairs of operators, such as $(A \otimes B)^* \cong A^* \parr B^*$.

- Quantifiers introduce a notion of polymorphism where some parameter of the resource type is not provided, and either we can choose ($\forall$) or it is otherwise determined ($\exists$).

- Exponentials act similarly to quantifiers, except rather than the form of the resource being unspecified, it is the quantity that is not given. A $!A$ resource refers to any number of units of A. Interpretations of this include the idea that we have a unit of A which is freely copyable and destructible, or that we have access to a factory which can build units of A at the point of needing them. The other case, $?A$, is where there exists some (possibly zero) units of A.

At first glance, one may imagine that, due to the tight restrictions imposed by the rules for each operator, linear logic admits a strict subset of the proofs available in non-linear logic. However, it is possible to encode proofs from classical logic in linear logic using the ! modality, mapping the sequent $\Gamma \vdash A \to B$ to $!\Gamma \vdash !A \multimap ?B$.

The rule specification of linear logic permits substantial automation of theorem proving, with tools such as `llprover` [48] readily available.

B.2 Links to the Caus Construction

Rather than cartesian-closed categories, models of linear logic are found in $*$-autonomous categories.

Definition B.2.1 ($*$-Autonomous Category)**.** *A $*$-autonomous category is a symmetric monoidal category $\mathcal{C}$ equipped with a full and faithful "dual" functor $(-)^* : \mathcal{C}^{op} \to \mathcal{C}$ which yields a natural isomorphism $\mathcal{C}(A \otimes B, C) \cong \mathcal{C}(A, (B \otimes C^*)^*)$.*

In terms of $\multimap$, this natural isomorphism is $\mathcal{C}(A \otimes B, C) \cong \mathcal{C}(A, B \multimap C)$. This is very similar to the currying natural isomorphism in cartesian-closed categories $\mathcal{C}(A \times B, C) \cong \mathcal{C}(A, C^B)$ but using the monoidal product and linear implication as opposed to the categorical product and exponential objects. The monoidal product and dual functor of a $*$-autonomous category correspond to multiplicative conjunction ($\otimes$) and dual in linear logic. The additives can be used when the category has products ($\&$) and coproducts ($\otimes$). The quantifiers and exponentials are often viewed as infinitary cases of the additives and multiplicatives respectively, and are typically represented categorically as such.

The Caus construction on precausal categories, as defined in [29], yields a $*$-autonomous category, giving a correspondence with the multiplicative fragment of linear logic. In fact, we gain additional structure from the existence of a coherent isomorphism $\mathbf{I} \cong \mathbf{I}^*$ (such categories are called isomix categories in the literature). This corresponds to an extension of linear logic which adds in the mix rules in

Figure B.2. These mix rules are what allow the inclusion map from $\mathbf{A} \otimes \mathbf{B}$ into $\mathbf{A} \parr \mathbf{B}$ in Caus[$\mathcal{C}$].