Higher-order semantics for quantum programming languages with classical control

George Philip Atzemoglou
Wolfson College, Oxford

Department of Computer Science, University of Oxford

Submitted for the degree of Doctor of Philosophy
Michaelmas Term 2012

To my grandfathers,
George and Ermis.

Abstract

This thesis studies the categorical formalisation of quantum computing, through the prism of type theory, in a three-tier process. The first stage of our investigation involves the creation of the dagger lambda calculus; a lambda calculus for dagger compact categories [AC04]. Our second contribution lifts the expressive power of the dagger lambda calculus to that of a quantum programming language, by adding classical control in the form of complementary classical structures [CPP10, CD11] and dualisers [CPP08]. Finally, our third contribution demonstrates how our lambda calculus can be applied to various well known problems in quantum computation.

Our construction of the dagger lambda calculus extends the linear typed lambda calculus, by defining a higher-order language for quantum protocols which is an internal language for dagger compact categories. The resulting language includes a linear negation operator and redefines the notion of binding as a symmetric relation whose scope spans the entire sequent. Reduction works by means of an explicit substitution, in the spirit of the operational semantics of the linear chemical abstract machine. The rules for explicit substitution act globally on the entire typing judgement, instead of limiting their scope to a specific subterm. This particular implementation of reduction enables us to enrich our typing dynamics by allowing the binding not just of variables, but of arbitrary terms. An elimination procedure allows us to reconstruct application using Cut, hence removing it from our primitive rule set. The new rules allow for a fully symmetric language, where inputs and outputs are treated as elements of a symmetric relation, and give rise to a new structural rule called the *dagger-flip*. The resulting set of rules is minimal and simple to use, which allows us to prove with ease properties like

subject reduction, confluence, strong normalisation and consistency. Our analysis of the language's semantics is completed by a proof that the dagger lambda calculus is an internal language for dagger compact categories.

In our second contribution, we provide a controlled way of breaching the linearity constraints of the dagger lambda calculus. Instead of the exponential connective of linear logic, we enrich our language with a view of axiomatising the basis structures of Hilbert spaces. We achieve this by providing the type-theoretic equivalent of complementary classical structures, which allows us to implement a controlled form of copying and deleting of terms in our classical basis. In order to retain sensible types, our language is also equipped with dualisers in a non self-dual setting, allowing us to factor the familiar notion of Currying by making its rule admissible in the dagger lambda calculus.

Our final contribution is in the study of three well known applications of quantum computation. We first demonstrate the expressiveness of the dagger lambda calculus by using it to represent the formalism of *Quantum Key Distribution*; we then put it to use, by using the language to verify the protocol's procedures. Our second application is in the *quantum Fourier transform*; we demonstrate how our language can represent a controlled phase gate, we use that construction to build a *quantum Fourier transform*, and we then use the dagger lambda calculus to "run" a sample input through the *quantum Fourier transform*. Lastly, our third application provides an examination of the *teleportation protocol*, explaining how it is represented in our language, demonstrating corrections and the flow of measurement outcomes, and showing how a quantum term actually ends up teleporting from one party to the other.

Contents

Part I

Introduction

Chapter 1

Motivation

Quantum mechanics was initially developed in the first third of the previous century, with quantum computation being explicitly studied since the 1980's. Though a lot of work has been done on quantum programming languages, it is still a nascent branch of science, where most of the languages have not yet been developed to higher levels of abstraction. As a result, while we do have a handful of quantum algorithms which provide promising results in the area of computational efficiency, much of the work involved in designing such an algorithm seems to be still largely based on guesswork; there is no clear set of rules, or unifying principle, that would easily allow us to combine computational primitives into building a new and efficient quantum algorithm. Furthermore, even though quantum computation is probably a more powerful means of computation than classical computing, this remains an unproven conjecture to this day. Unless we develop the means of abstracting our computational primitives to higher notions, thereby setting the foundations for a solid theory of algorithms and figuring out what gives quantum computers their extra power, we will probably never be able to prove a computational speedup or write efficient algorithms en masse.

Since the turn of the century, we have seen increasing interest in the development of a quantum programming language. One very actively pursued approach has been from a programming languages perspective [Sel04a, Sel04b, vTD03, vT04, SV06, SV08, SV10]. The researchers in this field, whose work has been seminal in establishing a semantic approach to quantum programming language design, have focused in designing a higher order lambda calculus for quantum computation with classical control. More specifically, in [SV10], a quantum lambda calculus with a complicated set of rules is presented, whose structural equations nevertheless allow for higher-

order structures. The rest of the work towards constructing a concrete model for the language's semantics remains an open problem.

Another very successful approach has come from a category theoretic perspective, where researchers have proposed a variety of diagrammatic calculi: Starting with the work of [AC04] and [Sel07]; progressing into the classical structures of [CP06], [CP07] and [CPP10]; the dualisers of [CPP08]; up until the full axiomatisation of bases through complementary observables in [CD08] and [CD11]. Because of their design, these languages are capable of expressing both quantum and classical processes, which makes them ideal for representing measurement based quantum computation, the computational paradigm that is closest to being practically implemented. Furthermore, these languages possess an obvious visual appeal and are, as a result, easy to understand and work with. Despite all that, the diagrammatic calculus does not readily lend itself to higher order operations. Moreover, despite some recent results by [DP10] and [Kis11], which deal with the issue of automated rewriting in the diagrammatic calculus, the rewrite steps are not always immediately apparent or easy to follow.

The purpose of this dissertation is to bridge these two approaches, hence bringing the programming languages approach closer to the categorical approach of complementary observables, by casting the diagrammatic formalism into the rich and well established tradition of type theory. The language presented in part III of the dissertation attempts this bridge by providing a higher-order computational interpretation for the categorical semantics of [AC04], [CD11], [CPP08] and [CPP10]. The dagger lambda calculus is expressive enough to perform operations on arbitrary (black box) functions and, as such, it is capable of encoding many well known quantum algorithms.

The natural way in which the dagger lambda calculus represents higher-order operations, together with the simplicity of its sequent rewriting, make it an ideal tool to be used side by side with the diagrammatic calculus. The two calculi can then complement each other, by providing the best of both worlds; visual clarity, intuitive rewrite rules and higher-order expressibility.

Chapter 2

Outline of the Dissertation

Part I serves as an introduction to the semantics of quantum programming languages. It prepares the ground for the rest of the thesis by presenting the main motivation for research in this area. The current section will explain the structure of the remaining parts of the dissertation, by providing an outline of the various sections and subsections.

In order to make this dissertation more self-contained, part II covers all of the background material that will be used in the constructive portions of this work. This presentation starts with a quick overview of quantum computing in chapter 3. Chapter 4 provides some background on category theory, including definitions for a selection of topics from [Mac98], as well as the definitions for many of the categories that will be used in later chapters. Chapter 6 explains how the categorical structures of [AC04], together with the Frobenius algebras of [CD08], can be used to model quantum computation and classical control structures. Chapter 7 contains a brief presentation of linear logic, as well as a variant by [AD06] that is better suited for quantum computation. Finally, chapter 8, the last chapter in this part of the dissertation, prepares the ground for the later parts by presenting the linear typed lambda calculus of [AT10].

Part III of the dissertation forms the main constructive portion of this work. It begins by introducing in section 9.1 the dagger lambda calculus, a language for quantum protocols that corresponds to dagger compact categories. The next section, 9.2, provides proofs of the language's most important properties; namely of subject reduction, strong normalisation, confluence and consistency. Section 9.3 presents a proof of the language's correspondence to dagger compact categories by showing that the syntactic category is indeed a free dagger compact category. In section 10.1, the

dagger lambda calculus is enriched with the classical structures of [CP06], [CP07] and [CPP10]. The language is then further enriched by introducing the notion of a dualiser [CPP08] in section 10.2, which can be used as a primitive to factor the notion of Currying, hence making the language's Curry rule admissible. Sections 10.3 and 10.4 define the monoidal product and phase shift operations in the calculus and outline the requirements for the properties of unbiasedness and classical constants. Section 10.5 enriches the dagger lambda calculus with the complementary classical structures of [CD11], which lift its expressive power from quantum protocols to that of a quantum programming language.

Part IV focuses on the applications of the language designed in part III. Chapter 11 shows how the dagger lambda calculus can be used to perform the Quantum Key Distribution of [Eke91], using the formalism of [CWW$^+$11]. Chapter 12 then demonstrates how the language can represent and run a quantum Fourier transform, the most essential part of Shor's factoring algorithm [Sho97]. Completing the applications' part, chapter 13 uses the language to perform the teleportation protocol.

Finally, the dissertation comes to a close in part V. This part highlights the importance of the higher-order computational interpretation provided in the dissertation, by outlining the structural insights that were gained through the study of its semantics. A list of possible directions for future work is provided, along with some concluding remarks.

Part II

Background material

Chapter 3

Quantum computing

Quantum computing is a radically different paradigm for computation which relies on the laws of quantum mechanics, in the hopes of achieving a higher computational efficiency than its currently used classical counterpart. This section will cover some fundamental concepts of quantum computing [Mer07, NC00], reviewing all the material that is necessary for understanding this dissertation.

Classical computers operate on regular bits, whose value is either 0 or 1. When studying quantum computers, we view $|0\rangle$ and $|1\rangle$ as orthonormal vectors, using them as a basis to span a complex Hilbert space. We are free to pick a different set of orthonormal vectors as the basis of our Hilbert space, however, the one mentioned earlier is usually referred to as the *standard basis*. The length of a vector in this Hilbert space does not really matter; we therefore only keep its direction and group all like vectors up to a complex multiple into equivalence classes called rays. Qubits, the quantum analogue of a bit, can have any of these rays as their value. This means that the value or *state* of any qubit can be written as

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

where α and β are complex coefficients. The fact that a quantum state can be "a little bit of" $|0\rangle$ and "a little bit of" $|1\rangle$ at the same time, is called *superposition*. Transformations of a quantum system's state are described by unitary operations acting on the system's Hilbert space. Since the underlying field for our Hilbert spaces is the field of complex numbers, state vectors in $\mathcal{H}$ can be trivially shown to be isomorphic to the linear maps in $\mathbb{C} \to \mathcal{H}$. The isomorphism maps every state $|\psi\rangle$ to the linear map spanned by $1 \mapsto \psi$. By a slight abuse of notation, we sometimes use $|\psi\rangle$ to refer to

the linear map as such:

$$|\psi\rangle : \mathbb{C} \to \mathcal{H} :: c \mapsto c\psi$$

The state of composite quantum systems is represented by the tensor product of the Hilbert spaces that describe their constituent parts. This behaves like a regular Kronecker product; for a system composed of A and B, we write $A \otimes B$. Similarly, for two linear maps f and g running parallel to each other, each acting on a different state of a composite system, we would write $f \otimes g$. It seems natural that we could use $|\psi\rangle \otimes |\phi\rangle$ to describe the state of two qubits. We tend to write $|00\rangle$ for $|0\rangle \otimes |0\rangle$ and $|11\rangle$ for $|1\rangle \otimes |1\rangle$. Because of superposition, however, there are some cases where a state cannot be written as the tensor product of two or more states. Typical examples of this are the Bell states: $|00\rangle + |11\rangle$, $|00\rangle - |11\rangle$, $|01\rangle + |10\rangle$ and $|01\rangle - |10\rangle$.

Hilbert spaces come equipped with an inner-product which, as a convention in quantum computation, is usually defined as linear in the second argument:

$$\langle -|- \rangle : \mathcal{H} \times \mathcal{H} \to \mathbb{C}$$

In order to formally introduce Dirac notation in our work, we want to further refine the definition of an inner-product by breaking it down to a composition of a *bra* $\langle\phi|$ and a *ket* $|\psi\rangle$, yielding $\langle\phi|\psi\rangle = \langle\phi| \circ |\psi\rangle$. A *bra* $\langle\phi|$ stands for the dual vector of $|\phi\rangle$. At this point it is useful to introduce the notion of an *adjoint* or, more precisely, a Hermitian adjoint for linear maps:

$$(f : \mathcal{H}_1 \to \mathcal{H}_2) \mapsto (f^\dagger : \mathcal{H}_2 \to \mathcal{H}_1)$$

Building up on our convention to use $|\phi\rangle$ to represent linear maps, we will define the adjoint of that map to be equivalent to the dual of the corresponding state vector [NC00]:

$$(|\phi\rangle)^\dagger \equiv \langle\phi| : \mathcal{H} \to \mathbb{C}$$

Performing a measurement on a quantum system against some orthonormal basis destroys its state, by making it collapse into one of the basis vectors. To define this more formally, a measurement against some orthonormal basis consists of a set of projectors P_i, each sending the measured state to one of the basis vectors. At the time of measurement, the system's wave function collapses, causing a non-deterministic jump in the system's state by stochastically applying one of the projectors to the state vector. If we represent the state as a linear combination of basis vectors, the square of the complex coefficient of any basis vector gives us the probability of collapsing to that outcome during a measurement.

Chapter 4

Category theory

Category theory is an area of mathematics that provides us with a means of reasoning about common properties of abstract structures; it allows the use of diagrammatic reasoning while, at the same time, extending connections to mathematical logic. The rest of this dissertation makes extensive use of category theory [Mac98]. Therefore, for the sake of completeness, we will provide definitions for all of the notions used.

4.1 Basic notions

Since this chapter is about category theory, we will begin our exposition appropriately by providing a definition for categories. Following that, we will be able to define functors and natural transformations, all of which will be used extensively later on.

Definition 4.1.1 (Category)**.** A *category* is a collection of objects and arrows between objects such that the following conditions hold:

- There is a composition operator $\circ$, that can take any two arrows of the form $f : A \to B$ and $g : B \to C$ and produce a new arrow $g \circ f : A \to C$
- Composition is associative, so $h \circ (g \circ f) = (h \circ g) \circ f$
- For every object A in our category, there is an identity arrow $id_A : A \to A$
- The identity arrows satisfy the *unit law*, whereby, for any arrow $f : A \to B$ in our category, $id_B \circ f = f = f \circ id_A$

Definition 4.1.2 (Functor)**.** A *functor* is a morphism between categories, mapping objects to objects and arrows to arrows, in a way that preserves identities and composition.

Example. Consider two categories $\mathcal{C}$ and $\mathcal{D}$. A functor $F : \mathcal{C} \to \mathcal{D}$, assigns to each object $A \in \mathcal{C}$ an object $FA \in \mathcal{D}$ and to each arrow $f : A \to B$ of $\mathcal{C}$ an arrow $Ff : FA \to FB$ in $\mathcal{D}$. Since it preserves identities and composition, it will have to be the case that $F(id_A) = id_{FA}$ and $F(g \circ f) = Fg \circ Ff$.

Definition 4.1.3 (Natural transformation)**.** When F and G are functors such that $F, G : \mathcal{C} \to \mathcal{D}$, a *natural transformation* $\eta : F \Rightarrow G$ is a collection of arrows $\eta_A : FA \to GA$ in $\mathcal{D}$ for every object A in $\mathcal{C}$. These arrows have to be such that for any $f : A \to B$ in $\mathcal{C}$, the following diagram commutes:

$$\begin{array}{ccc} FA & \xrightarrow{\eta_A} & GA \\ {\scriptstyle Ff}\downarrow & & \downarrow{\scriptstyle Gf} \\ FB & \xrightarrow[\eta_B]{} & GB \end{array}$$

Note. When all the η_A of a natural transformation are isomorphisms, we call η a natural isomorphism.

4.2 More advanced notions

We now wish to define a way of describing how different categories are related to each other. In order to do that, consider categories $\mathcal{C}$ and $\mathcal{D}$, with functors F and G between them such that:

$$\mathcal{C} \underset{G}{\overset{F}{\rightleftarrows}} \mathcal{D}$$

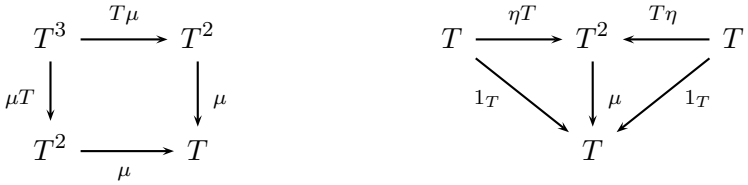
The strictest and perhaps most obvious type of relation occurs when both of these functors are identity endofunctors $F = G = Id$, which would mean that the categories are *equal*. An *isomorphism* is a weaker kind of relation where $G \circ F = Id_{\mathcal{C}}$ and $F \circ G = Id_{\mathcal{D}}$. An even weaker kind of relation is an *equivalence*, where instead of requiring the composition of functors to be equal to the identity functor, we ask that there be a natural isomorphism between them. In other words $G \circ F \cong Id_{\mathcal{C}}$ and $F \circ G \cong Id_{\mathcal{D}}$.

Continuing down that path leads us to a very important notion called an adjunction, which is another yet weaker kind of relation between categories [Che07]. To define this more rigorously:

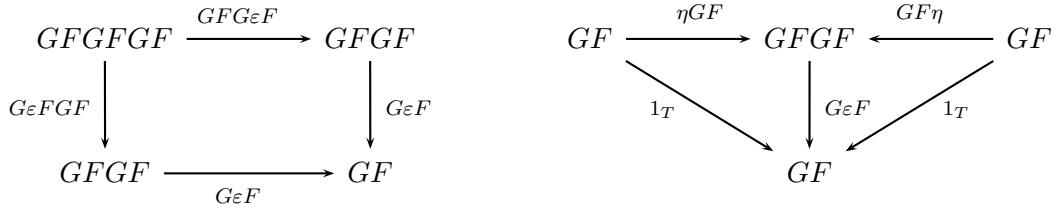
Definition 4.2.1 (Adjunction)**.** Let $\mathcal{C}$ and $\mathcal{D}$ be categories, with functors $F : \mathcal{C} \to \mathcal{D}$ and $G : \mathcal{D} \to \mathcal{C}$. We say there is an *adjunction* $\langle F, G, \eta, \varepsilon \rangle$ when there exist two natural transformations $\eta : Id_{\mathcal{C}} \Rightarrow GF$ and $\varepsilon : FG \Rightarrow Id_{\mathcal{D}}$, respectively called the *unit* and *counit* of the adjunction, such that the following diagrams commute:

Note. We say that F is *left adjoint* to G, writing this as $F \dashv G$. Similarly, G is right adjoint to F.

Definition 4.2.2 (Monad)**.** A *monad* $\langle T, \eta, \mu \rangle$ in a category $\mathcal{C}$ consists of an endofunctor $T : \mathcal{C} \to \mathcal{C}$, together with two natural transformations $\eta : Id_{\mathcal{C}} \Rightarrow T$ and $\mu : T^2 \Rightarrow T$, such that the following diagrams commute:



Corollary 4.2.1 (Adjunctions define a monad)**.** *Any adjunction $\langle F, G, \eta, \varepsilon \rangle$ defines a monad. This can be done by setting $T = GF$, which would make the unit of our adjunction a natural transformation of the form $\eta : Id_{\mathcal{C}} \Rightarrow T$ and our $\mu = G\varepsilon F : GFGF \Rightarrow GF$. The resulting monad would be of the form $\langle FG, \eta, G\varepsilon F \rangle$. This can be easily verified by checking that the following diagrams do indeed commute:*



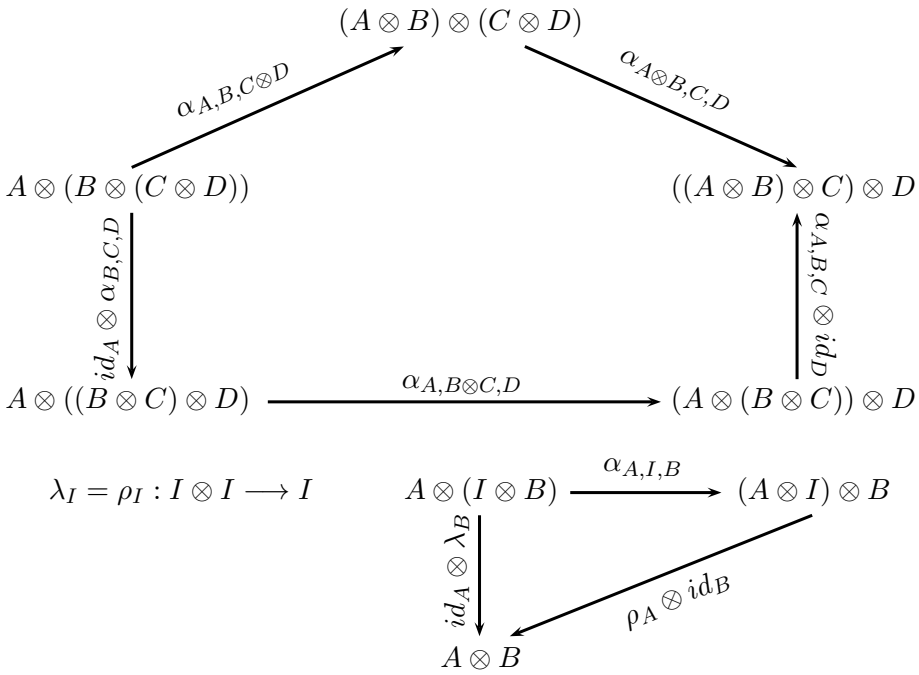
4.3 Specific constructions

Definition 4.3.1 (Monoidal category)**.** A *monoidal category* is a category that has been equipped with an bifunctor called tensor $\otimes : \mathcal{C} \times \mathcal{C} \longrightarrow \mathcal{C}$. Up to appropriate natural isomorphisms, the tensor is associative and features a special object I that acts as a left and right identity:

$$\alpha_{A,B,C} : A \otimes (B \otimes C) \xrightarrow{\cong} (A \otimes B) \otimes C$$

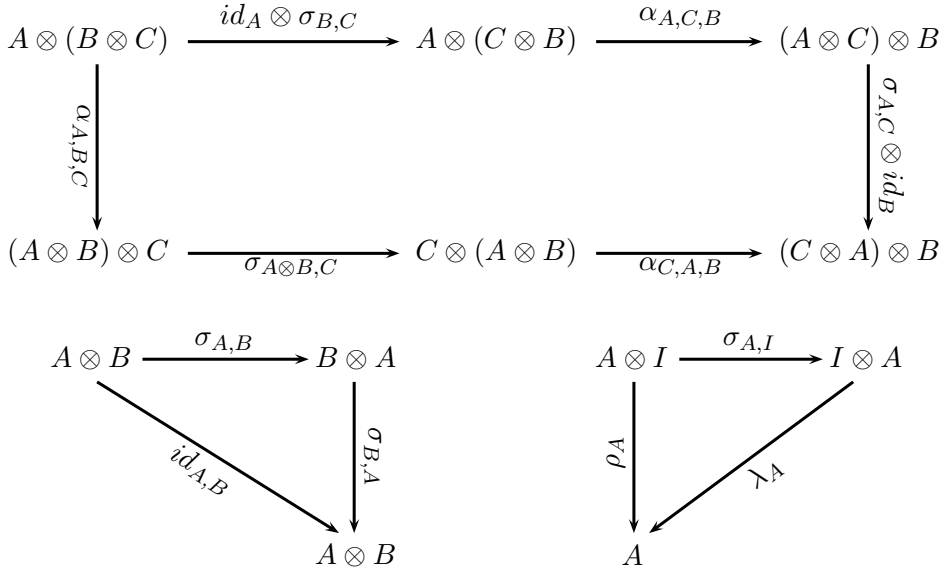
$$\lambda_A : I \otimes A \xrightarrow{\cong} A \qquad \rho_A : A \otimes I \xrightarrow{\cong} A$$

The tensor product thus forms a monoid, with I acting as the unit. These isomorphisms have to further satisfy some conditions called coherence conditions. These can be summarily represented by requiring that the following diagrams commute for all A, B, C and D:



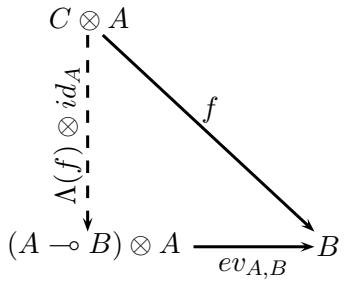
Definition 4.3.2 (Symmetric monoidal category)**.** A *symmetric* monoidal category is a monoidal category with an additional natural isomorphism called symmetry, $\sigma_{A,B} : A \otimes B \xrightarrow{\cong} B \otimes A$, such that the following diagrams

commute:



Definition 4.3.3 (Symmetric monoidal closed category)**.** A symmetric monoidal *closed* category is a symmetric monoidal category where, for any two objects A and B, there is an *exponential object*[1] $A \multimap B$, together with an evaluation morphism $ev_{A,B} : (A \multimap B) \otimes A \longrightarrow B$. In addition to that, on any arrow of the form $f : C \otimes A \longrightarrow B$, a process called *Currying* yields a unique morphism $\Lambda(f) : C \longrightarrow (A \multimap B)$ such that:

$$ev_{A,B} \circ (\Lambda(f) \otimes id_A) = f$$



Definition 4.3.4 (Compact closed category)**.** A *compact closed category* is a symmetric monoidal category where, for every object A, there is a dual

[1]Note that these are different from the exponential connectives of linear logic.

object A^* along with two morphisms $\eta_A : I \longrightarrow A^* \otimes A$ and $\varepsilon_A : A \otimes A^* \longrightarrow I$ such that:

$$(\varepsilon_A \otimes id_A) \circ (id_A \otimes \eta_A) = id_A \qquad \text{and} \qquad (id_{A^*} \otimes \eta_A) \circ (\varepsilon_A \otimes id_{A^*}) = id_{A^*}$$

The dual object is unique up to canonical isomorphism.

Corollary 4.3.1 (Closure of Compact Closed Categories)**.** *Every compact closed category is closed.*

Proof. All of the exponential structure can be recreated by setting $A \multimap B = A^* \otimes B$. The evaluation function can be simulated by:

$$ev_{A,B} = \rho_A \circ (id_B \otimes \varepsilon_A) \circ (id_B \otimes \sigma) \circ \alpha^{-1} \circ (\sigma \otimes id_A)$$

□

Definition 4.3.5 (Dagger compact category)**.** A †-*compact category* is a compact closed category that is equipped with an involutive, contravariant, identity-on-objects endofunctor. That functor, called dagger, reverses all arrows, leaves objects unchanged, and preserves the tensor structure. For any $f : A \to B$, it will be the case that $f^\dagger : B \to A$ and $f^{\dagger\dagger} = f$. Moreover, for any object A in our category, it must be the case that $\sigma_{A,A^*} \circ \varepsilon_A^\dagger = \eta_A$

Chapter 5

Frobenius algebras

This chapter presents a quick overview of *Frobenius algebras*, how they generalise over a category's monoidal structure, as well as some related algebraic properties that will be used extensively in the parts of this dissertation that deal with classical structures. We will provide diagrammatic representations to accompany many of the textual definitions for the required conditions and properties. It should be noted that, as a notational convention, all the diagrams should be read from bottom to top.

Frobenius algebras were studied since the 1930's but have become exceedingly popular, in the past three decades, in the mathematical representation of quantum physics [JS91, KL01, Koc03]. More recent developments in quantum information theory [CP07, CD08, CPP10, CD11] have used these algebras to axiomatise the notion of a classical basis in quantum computation.

We will start by presenting some of the most common definitions of a Frobenius algebra, adapted from [Str04]:

Definition 5.0.6 (Frobenius algebra)**.** A Frobenius algebra A is a finite dimensional, unital and associative algebra over a field k, that is equipped with a nondegenerate bilinear pairing $\sigma : A \otimes A \to k$. The bilinear form must be such that the following condition holds $\sigma((ab) \otimes c) = \sigma(a \otimes (bc))$, for $a, b, c \in A$.

Alternatively, we could say that the algebra is Frobenius if it is equipped with a linear function $\varepsilon : A \to k$, such that:

$$\varepsilon(ab) = 0 \text{ for all } a \in A \text{ implies } b = 0.$$

In order to generalise this to the categorical setting, we will look at monoidal categories, and will define what it means for an object in that

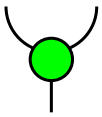
category to have a Frobenius structure associated with it.

Definition 5.0.7 (Frobenius structure)**.** We say that an object A in a monoidal category $\mathcal{C}$ has a *Frobenius structure* attached to it when it is equipped with four morphisms, $\mu : A \otimes A \to A$, $\eta : I \to A$, $\delta : A \to A \otimes A$ and $\varepsilon : A \to I$, such that the following conditions hold:

- (A, μ, η) forms a monoid,
- (A, δ, ε) forms a comonoid,
- and $(id_A \otimes \mu) \circ (\delta \otimes id_A) = \delta \circ \mu = (\mu \otimes id_A) \circ (id_A \otimes \delta)$

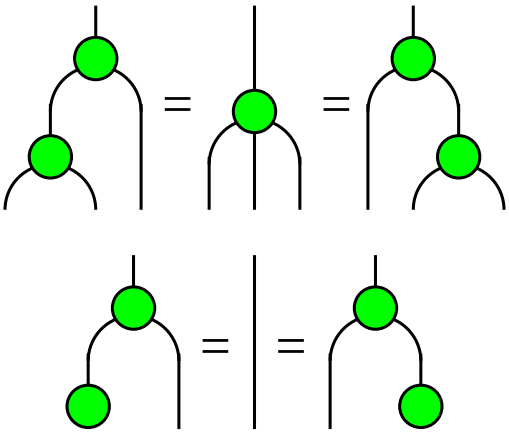
We denote the Frobenius structure as $(A, \mu, \eta, \delta, \varepsilon)$. In the case of dagger monoidal categories, the dagger functor can give us $\mu = \delta^\dagger$ and $\eta = \varepsilon^\dagger$, simplifying the Frobenius structure into (A, δ, ε). The conditions required by our definition are easier to visualise in diagrammatic form. In order to achieve this, we will represent our fundamental morphisms, δ and ε as follows:

$$\delta : A \longrightarrow A \otimes A :: a_i \mapsto a_i \otimes a_i \qquad \epsilon : A \longrightarrow I :: a_i \mapsto 1$$

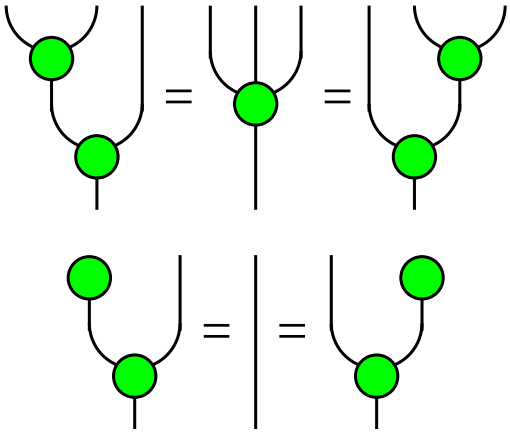


This allows us to represent the monoidal and comonoidal conditions diagrammatically as such:

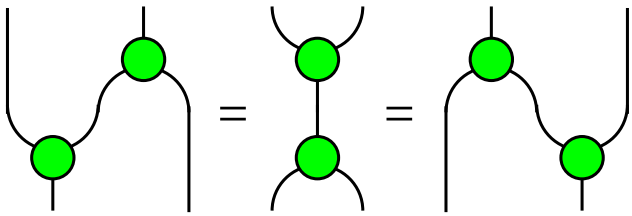
Monoid conditions



Comonoid conditions

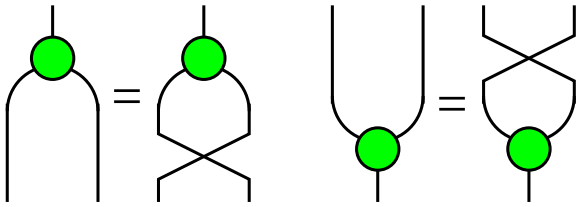


The last condition in the definition, also known as the *Frobenius condition*, thus becomes particularly easy to visualise:



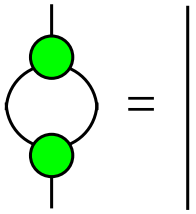
We will now define two important properties, *symmetry* and *isometry*, that will be used in later parts of our exposition. We will start with the definition of symmetry:

Definition 5.0.8 (Symmetric algebra)**.** Consider the symmetry isomorphism $\sigma_{A,B} : A \otimes B \to B \otimes A$ that is part of the definition of a symmetric monoidal category. A Frobenius algebra over an object of such a category is *symmetric* if its underlying monoid and comonoid are commutative and cocommutative, respectively. This means that δ must be $\delta = \sigma_{A,A} \circ \delta$ and μ must be $\mu = \mu \circ \sigma_{A,A}$.



The last property that we will define in this chapter is isometry:

Definition 5.0.9 (Isometric or special algebra)**.** A Frobenius algebra $(A, \mu, \eta, \delta, \varepsilon)$ over an object A is said to be *isometric* or *special* if $\mu \circ \delta = id_A$.



Chapter 6

Categorical model

This section presents the categories used to model quantum computation and classical operations in the rest of the dissertation. It begins by explaining how the Hilbert space formalism can be recast into the language of †-compact categories [AC04]. One of the biggest practical advantages of monoidal categories is that, on many occasions, they "formally justify their absence" [Coe06], meaning that they can be represented using a graphical calculus that greatly simplifies categorical reasoning. This section also demonstrates how every element of the initial quantum structure can be represented graphically in what resembles a two dimensional Dirac notation. Following our notational convention for the direction of the compositional flow of time, all the diagrams should be read from bottom to top. The next part of this section deals with classical operations, which are modelled in terms of internal spider monoids, as well as with the computational interplay inherent in introducing complementarity.

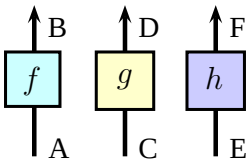
6.1 Categorical quantum computation

The category we will be using to model quantum computation is called *FDHilb* and is the category of finite dimensional complex Hilbert spaces. Its objects are finite dimensional Hilbert spaces and its arrows are linear maps. Monoidal multiplication is represented by the Kronecker tensor product, while the monoidal unit object corresponds to the set of complex numbers $I = \mathbb{C}$. Associativity of the tensor and tensor identities are up to equality, so $\alpha_{A,B,C}$, λ_A and ρ_A are reduced to identity arrows.

The adjoint is modelled using the dagger functor. In terms of the picture calculus, the dagger denotes flipping a picture upside down, while the arrows

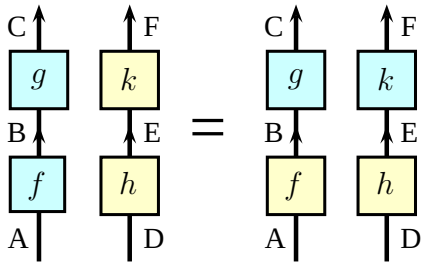
continue pointing the same way they were before (i.e. upwards).

For any three arrows $f : A \to B$, $g : C \to D$ and $h : E \to F$, associativity allows us to write their tensor product as:



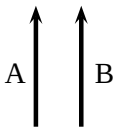
By bifunctoriality of the tensor, we know that it preserves composition. For any arrows of the form $f : A \to B$, $g : B \to C$, $h : D \to E$ and $k : E \to F$, once we add composition to our diagrams, the following property should become more evident:

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



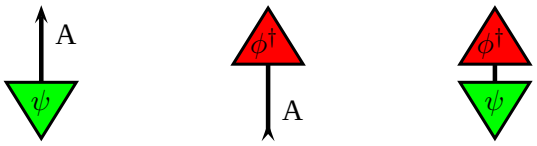
Moreover, it should also be evident that the tensor preserves identities:

$$id_{A \otimes B} = id_A \otimes id_B$$



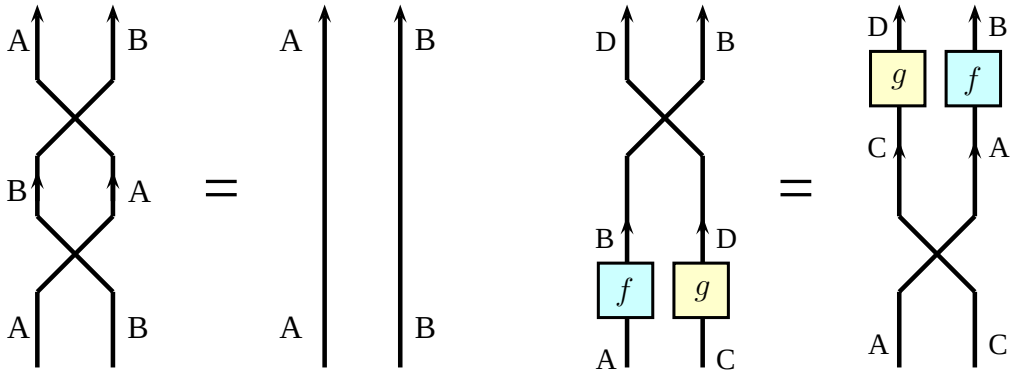
The proper graphical representation for the unit I is "no line", while arrows containing I as their domain or codomain are represented as follows:

$$\psi : I \to A \qquad \phi^\dagger : A \to I \qquad \phi^\dagger \circ \psi : I \to I$$



A special case of arrows called scalars consists of all arrows of the form $c : I \to I$. In these specific cases, our categorical structure collapses to the point where tensor is equal to composition. In other words $c_1 \otimes c_2 = c_1 \circ c_2 = c_2 \otimes c_1$. Scalars can be moved freely around in the category's graphical representation.

Symmetry corresponds to a well known quantum operation called *swap*; it is graphically represented by a pair of crossing lines. The following properties are more easily understood graphically:



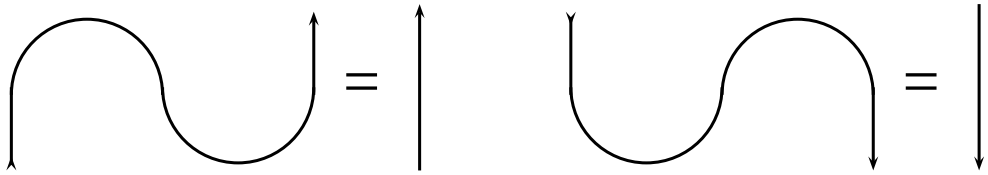
$\sigma_{A,B} \circ \sigma_{A,B} = id_A \otimes id_B$ $\sigma_{B,D} \circ (f \otimes g) = (g \otimes f) \circ \sigma_{A,C}$

Compact closure is used to model entangled states. These are the only cases where we see arrows pointing downwards, as the $*$ in A^* reverses the arrow's direction. The graphical representation looks like this:

$d_A : I \longrightarrow A^* \otimes A$ $e_A : A \otimes A^* \longrightarrow I$

These have to adhere to a property, fundamental in proving teleporta-

tion, whose graphical representation is reminiscent of *yanking* a wire:



6.2 Representation of classical structures

One of the fundamental known distinctions between quantum and classical computation is derived from no-go theorems. Classical computers routinely copy and delete data; it is such a commonplace thing to do that we hardly ever notice how entwined it is to the classical computational paradigm itself. Quantum computers, on the other hand, cannot perform either of these operations. In this part of this section, we will see how to turn this problem into a very important feature, which will in turn enable us to account for classical operations within the, already defined, quantum categorical framework. The following result [WZ82] is referred to as the *no-cloning* theorem:

Theorem 6.2.1 (No-cloning theorem)**.** *There is no quantum operation D, such that*

$$D : |0\rangle \otimes |\psi\rangle \mapsto |\psi\rangle \otimes |\psi\rangle$$
$$D : |0\rangle \otimes |\phi\rangle \mapsto |\phi\rangle \otimes |\phi\rangle$$

unless $|\psi\rangle$ and $|\phi\rangle$ are orthogonal.

The ancilla qubit $|0\rangle$ is sometimes not included in information-theoretic notation [CP07, Abr10], reducing D to $D : |\psi\rangle \mapsto |\psi\rangle \otimes |\psi\rangle$. Another result [PB00], complementary to the no-cloning theorem, is commonly referred to as the *no-deleting* theorem. In the original formulation of this result, Pati and Braunstein used an ancilla qubit $|A\rangle$ and included a qubit state in a standard state $|\Sigma\rangle$, making E look more like $E : |\psi\rangle \otimes |\psi\rangle \otimes |A\rangle \mapsto |\psi\rangle \otimes |\Sigma\rangle \otimes |A_\psi\rangle$. The formulation we will use is more common in the information-theoretic literature [CP07, Abr10]:

Theorem 6.2.2 (No-deleting theorem)**.** *There is no quantum operation E, such that*

$$E : |\psi\rangle \mapsto 1$$
$$E : |\phi\rangle \mapsto 1$$

unless $|\psi\rangle$ and $|\phi\rangle$ are orthogonal.

From these two theorems, we can deduce that the only cases of quantum states that we could treat as classical are those pertaining to orthogonal vectors. This seems to reinforce the notion of a classical basis spanning a complex Hilbert space.

Definition 6.2.1 (Classical structure (categorical))**.** A *classical structure*[2] [CP07, CD08, CPP10] is defined in terms of special †-Frobenius cocommutative comonoids, also referred to as spider monoids. These are represented by a triplet (A, δ, ϵ), where A is an object and δ and ϵ are two morphisms of the form:

$$\delta : A \longrightarrow A \otimes A :: a_i \mapsto a_i \otimes a_i \qquad \epsilon : A \longrightarrow I :: a_i \mapsto 1$$

where the a_i are the orthogonal vectors to which copying and deleting take place. These two maps, also called *copying* and *deleting* maps, have to satisfy a number of conditions to ensure that they are well behaved and that their addition does not cause a collapse of the compact structure. More specifically, they must satisfy all of the conditions of a cocommutative comonoid, as well as the isometry and Frobenius conditions.

Example. *To help the reader become more accustomed to our new concepts, we will present a linear algebraic example with matrices. Suppose that our chosen basis consisted of* $|0\rangle$ *and* $|1\rangle$*, the computational basis. The appropriate* δ *to copy the basis vectors would be:*

$$\delta = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \text{ and } \varepsilon = \begin{pmatrix} 1 & 1 \end{pmatrix}$$

It should be easy to verify that $\delta|0\rangle = |00\rangle$*,* $\delta|1\rangle = |11\rangle$*,* $\varepsilon|0\rangle = 1$ *and* $\varepsilon|1\rangle = 1$*. The Hermitian adjoints of those states,* $\delta^\dagger$ *and* $\varepsilon^\dagger$*, would be:*

$$\delta^\dagger = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \text{ and } \varepsilon^\dagger = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

[2]N.B. Thanks to [CPV08], we now know that classical structures are in bijective correspondence to bases.

Once again we can verify that $\delta^\dagger$, also known as fusion, merges two basis states when they are the same: $\delta^\dagger|00\rangle = |0\rangle$ and $\delta^\dagger|11\rangle = |1\rangle$. The $\varepsilon^\dagger$ operation "creates" a state by being itself equivalent up to a scalar multiple to the constant state $|+\rangle$.

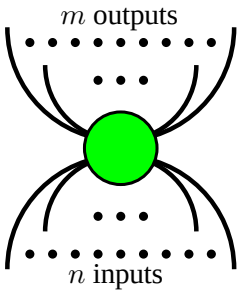
When expressed more rigorously, the comonoid conditions require that the copying and deleting maps form an internal cocommutative comonoid:

$$(\delta \otimes id_A) \circ \delta = (id_A \otimes \delta) \circ \delta$$
$$(\epsilon \otimes id_A) \circ \delta = (id_A \otimes \epsilon) \circ \delta = id_A$$
$$\delta = \sigma_{A,A} \circ \delta$$

Because of the dagger functor, all of the comonoidal conditions can be reversed to give us the conditions required of a commutative monoid:

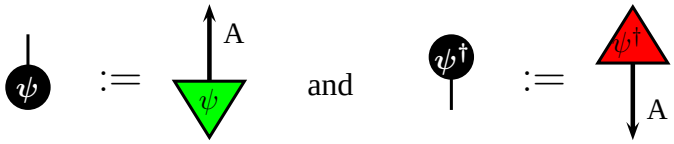
$$\delta^\dagger \circ (\delta^\dagger \otimes id_A) = \delta^\dagger \circ (id_A \otimes \delta^\dagger)$$
$$\delta^\dagger \circ (\epsilon^\dagger \otimes id_A) = \delta^\dagger \circ (id_A \otimes \epsilon^\dagger) = id_A$$
$$\delta^\dagger = \delta^\dagger \circ \sigma_{A,A}$$

The graphical representation for all of these conditions corresponds to connected graphs. Furthermore, the inputs and outputs on all of the required equations match, so, as proved in [CP06, CPP10, CD08], we can equivalently define classical structures using the *spider theorem*. This theorem states that if a graph generated by δ and ϵ is connected, then it is completely characterized by its domain and codomain. If the domain is $\underbrace{A \otimes \ldots \otimes A}_{n}$ and the codomain is $\underbrace{A \otimes \ldots \otimes A}_{m}$, then it can be reduced to a *"spider"* with n input and m output wires.

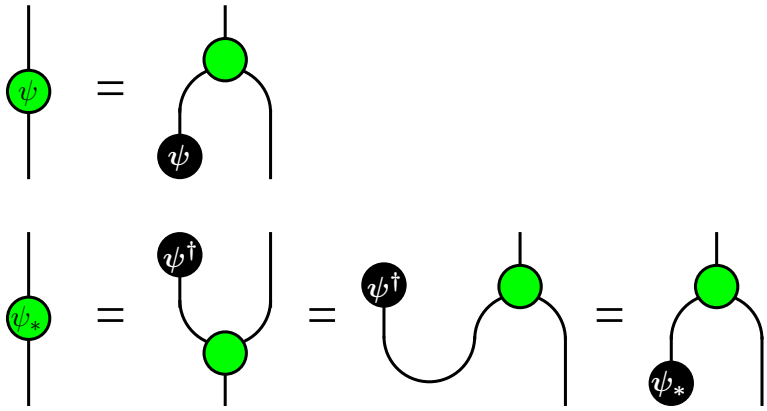


Arbitrary states that are points of A (i.e. of the form $|\psi\rangle : I \longrightarrow A$) are

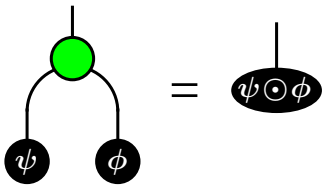
denoted by "black dots":



On any classical structure (A, δ, ϵ), we define a map K that lifts[3] any state $|\psi\rangle$ of A to the endomorphism $K^\delta(\psi) := \delta^\dagger \circ (|\psi\rangle \otimes id_A) : A \longrightarrow A$. Similarly, we can lift any bra $\langle\psi|$ by setting $K^\delta(\psi)^\dagger = K^\delta(\psi_*) = (\psi^\dagger \otimes id_A) \circ \delta : A \longrightarrow A$. We denote these graphically as:



The monoid operation $\delta^\dagger : A \otimes A \longrightarrow A$, also known as *fusion*, can be used to combine arbitrary states. For two states $|\psi\rangle$ and $|\phi\rangle$, their fusion is written as $\psi \odot \phi := \delta^\dagger \circ (\psi \otimes \phi)$. The same operation can be used to merge pairs of lifted states $K^\delta(\psi \odot \phi) := \delta^\dagger(K^\delta(\psi) \otimes K^\delta(\phi)) = \delta^\dagger(K^\delta(\phi) \otimes K^\delta(\psi))$. As per the definition of $\delta^\dagger$, the fusion operation is associative and commutative. This concept of fusion can be represented diagrammatically as follows:



Example. *Let* $|+_\alpha\rangle$ *and* $|+_\beta\rangle$ *be states such that:*

[3]The notation used for this lifting by [CD08] is $\Lambda^\mu(\psi)$. In our case, we avoid using the letter Λ, to prevent any confusion with the letter used for Currying.

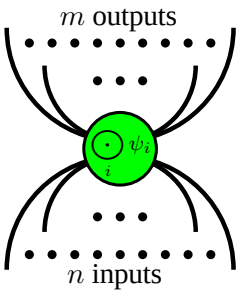
$$|+_\alpha\rangle = |0\rangle + e^{i\alpha}|1\rangle = \begin{pmatrix} 1 \\ e^{i\alpha} \end{pmatrix} \text{ and } |+_\beta\rangle = |0\rangle + e^{i\beta}|1\rangle = \begin{pmatrix} 1 \\ e^{i\beta} \end{pmatrix}$$

The fusion of those two states will be:

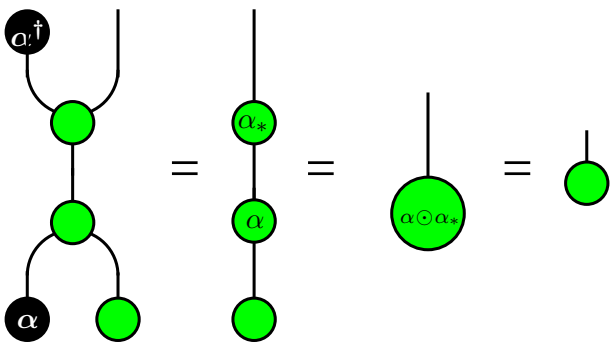
$$\delta^\dagger \circ (|+_\alpha\rangle \otimes |+_\beta\rangle) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ e^{i\beta} \\ e^{i\alpha} \\ e^{i(\alpha+\beta)} \end{pmatrix} = \begin{pmatrix} 1 \\ e^{i(\alpha+\beta)} \end{pmatrix} = |+_{\alpha+\beta}\rangle$$

After combining all of these definitions, we can prove [CD08] that classical structures follow what is known as the generalized spider theorem:

Theorem 6.2.3 (Spider theorem)**.** *Any connected graph generated by the operations of the classical structure (A, δ, ϵ), states $|\psi_i\rangle : I \longrightarrow A$ and the $\dagger$-compact structure, is completely characterized by its domain, codomain and $K^\delta(\bigodot_i \psi_i)$. The graphical representation is that of a "decorated spider"*

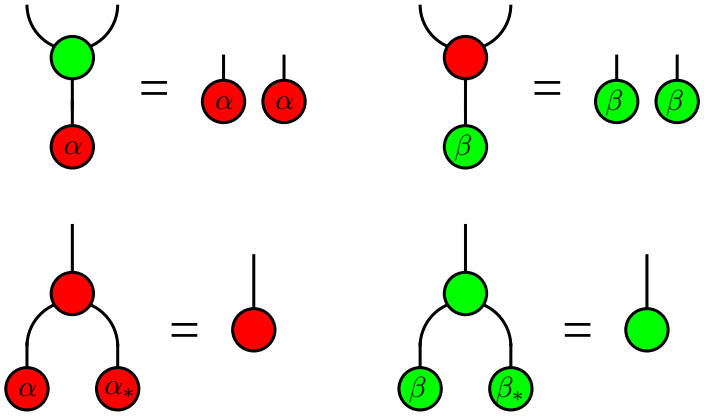


Definition 6.2.2 (Unbiasedness (categorical))**.** A point $\alpha : I \longrightarrow A$ is *unbiased* relative to (A, δ, ϵ) iff $K^\delta(\alpha)$ is unitary. In other words, there needs to be a scalar $s : I \longrightarrow I$ such that $s \cdot \alpha \odot \alpha^\dagger = \epsilon^\dagger$, or graphically:



Example. *Recall the state* $|+_\alpha\rangle$ *from our previous example. States of the form* $|+_\theta\rangle$ *will always be unbiased with respect to the classical basis structure we defined for copying* $|0\rangle$ *and* $|1\rangle$*. That is because* $\delta^\dagger \circ (|+_\theta\rangle \otimes |+_{-\theta}\rangle)$ *will always be equal to* $|+_0\rangle = \varepsilon^\dagger$*.*

Definition 6.2.3 (Complementarity (categorical))**.** Two classical structures $(A, \delta_G, \epsilon_G)$ and $(A, \delta_R, \epsilon_R)$ in a †-compact category are called *complementary* if the points that are classical for one are unbiased for the other and vice versa. δ_G and ϵ_G are depicted using green dots; the points they copy and delete are drawn in red and are unbiased for the second classical structure. δ_R and ϵ_R are depicted using red dots; they copy and delete green points, which are unbiased for the green classical structure. Graphically this condition is depicted as:



Example. *Let* $(A, \delta_G, \epsilon_G)$ *be the classical structure that we described earlier for copying and deleting* $|0\rangle$ *and* $|1\rangle$*. Let* $(A, \delta_R, \epsilon_R)$ *be a similarly constructed classical structure that can copy and delete* $|+\rangle$ *and* $|-\rangle$ *(i.e. state vectors orthogonal to the computational basis), such as:*

$$\delta_R = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix} \text{ and } \varepsilon_R = \begin{pmatrix} 1 & 0 \end{pmatrix}$$

As one can easily verify ,the two classical structures are an example of structures that are complementary to one another.

Chapter 7

Linear logic

This section provides an overview of the structures found in some flavours of linear logic, building up to the corresponding logic for compact closed categories, while illustrating how some of these concepts relate to category theoretic notions and properties.

Linear logic is a resource sensitive logic, first introduced in [Gir87]. Whereas other logics provide the structural rules of weakening and contraction in order to facilitate predicate re-use or non-use in proving theorems, linear logic drops the indiscriminate use of these rules and treats predicates as resources that need to be expended in order to produce proofs. In terms of the Gentzen sequent calculus, the rules of weakening and contraction would be represented as:

$$\text{Weakening } \frac{\vdash \Gamma}{\vdash \Gamma, A} \qquad \text{Contraction } \frac{\vdash \Gamma, A, A}{\vdash \Gamma, A}$$

Due to its resource sensitivity, linear logic finds many applications in computer science, such as in type theory, the semantics of programming languages, and the study of concurrency. The definitions and presentation of this section are largely based on [Abr93] and [AT10]. The sequent rules for linear logic are as follows:

$$\text{Axiom } \frac{}{\vdash A^{\perp}, A} \qquad \text{Exchange } \frac{\vdash \Gamma, A, B, \Delta}{\vdash \Gamma, B, A, \Delta} \qquad \text{Cut } \frac{\vdash \Gamma, A \quad \vdash \Delta, A^{\perp}}{\vdash \Gamma, \Delta}$$

$$\text{Unit } \frac{}{\vdash 1} \qquad \text{Perp } \frac{\vdash \Gamma}{\vdash \Gamma, -}$$

$$\text{Times } \frac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B} \qquad \text{Par } \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}$$

$$\text{With } \frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \& B} \qquad \text{Plus (i) } \frac{\vdash \Gamma, A}{\vdash \Gamma, A \oplus B} \qquad \text{Plus (ii) } \frac{\vdash \Gamma, B}{\vdash \Gamma, A \oplus B}$$

In order to recover the structural rules of weakening and contraction, we can introduce an exponential operator called *bang*, which is denoted by !

$$\text{Dereliction } \frac{\vdash \Gamma, A}{\vdash \Gamma, !A} \qquad \text{Of Course } \frac{\vdash !\Gamma, A}{\vdash !\Gamma, !A}$$

$$\text{Weakening } \frac{\vdash \Gamma}{\vdash \Gamma, !A} \qquad \text{Contraction } \frac{\vdash \Gamma, !A, !A}{\vdash \Gamma, !A}$$

7.1 Multiplicative quantum logic

We will now focus on and extend the multiplicative fragment of linear logic by choosing to ignore the rules for *additives* and *exponentials*. The reader is referred to [AT10] for a comprehensive, yet still accessible, overview of multiplicative linear logic. That logic was extended by [AD06], to yield a multiplicative quantum logic that simulates the compact structure found in compact closed categories. The key idea lies in the definition of linear negation, whereby multiplicative conjunction (tensor) is equated with multiplicative disjunction (par), by trivializing the notion of De Morgan duality. This is the most natural way of introducing compactness to linear logic and uses $A^\perp$ to represent the dual object A^* in our category. Linear negation in this case is characterized by the following laws:

$$\begin{aligned} A^{\perp\perp} &= A \\ 1^\perp &= 1 \\ (A \otimes B)^\perp &= A^\perp \otimes B^\perp \\ A \multimap B &= A^\perp \otimes B \end{aligned}$$

A categorical interpretation will be given for each of the rules presented, in order to better illustrate the Curry-Howard parallelism. The proof rules for this logic are as follows:

	Logic	Categories
Id	$\dfrac{}{A \vdash A}$	$\dfrac{}{id_A : A \longrightarrow A}$
$\bigotimes R$	$\dfrac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B}$	$\dfrac{f : \Gamma \longrightarrow A \quad g : \Delta \longrightarrow B}{f \otimes g : \Gamma \otimes \Delta \longrightarrow A \otimes B}$
$\bigotimes L$	$\dfrac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C}$	$\dfrac{f : (\Gamma \otimes A) \otimes B \longrightarrow C}{f \circ a_{A,B,\Gamma} : \Gamma \otimes (A \otimes B) \longrightarrow C}$
Cut	$\dfrac{\Gamma \vdash A \quad A, \Delta \vdash B}{\Gamma, \Delta \vdash B}$	$\dfrac{f : \Gamma \longrightarrow A \quad g : A \otimes \Delta \longrightarrow B}{g \circ (f \otimes id_\Delta) : \Gamma \otimes \Delta \longrightarrow B}$
$\multimap E$	$\dfrac{\Gamma \vdash A \multimap B \quad \Delta \vdash A}{\Gamma, \Delta \vdash B}$	$\dfrac{f : \Gamma \longrightarrow (A \multimap B) \quad g : \Delta \longrightarrow A}{ev_{A,B} \circ (f \otimes g) : \Gamma \otimes \Delta \longrightarrow B}$
$\multimap R$	$\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}$	$\dfrac{f : \Gamma \otimes A \longrightarrow B}{\Lambda(f) : \Gamma \longrightarrow (A \multimap B)}$

At this point, it is interesting to compare the two worlds and see how some notions translate from one to the other. The identity rule corresponds to identity arrows in our categories. The Cut rule defines function composition. The right tensor rule ($\bigotimes R$) defines tensoring, while the left tensor rule ($\bigotimes L$) defines the associativity of the tensor. Linear implication ($\multimap$) is a notion equivalent to a category's exponential objects, so naturally, implication elimination ($\multimap E$) and the right implication rule ($\multimap R$) respectively define the category's Evaluation and Currying functions. The only structural rule in this logic is the rule known as *exchange*:

$$\frac{\Gamma, A, B, \Delta \vdash C}{\Gamma, B, A, \Delta \vdash C} \qquad \frac{f : \Gamma \otimes A \otimes B \otimes \Delta \longrightarrow C}{f \circ (id_\Gamma \otimes s_{A,B} \otimes id_\Delta) : \Gamma \otimes B \otimes A \otimes \Delta \longrightarrow C}$$

This rule corresponds to the symmetry isomorphism for the tensor. All we need in order to have a fully fledged representation of symmetric monoidal closed categories is a monoidal unit. That is denoted by 1 and comes with the simple rule of $\dfrac{}{\vdash 1}$. We use the right implication rule to transform our identity rule to $\vdash A \multimap A$ which, when translated via the linear negation laws, becomes $\vdash A^\perp \otimes A$, or the equivalent of $d_A : I \longrightarrow A^* \otimes A$ in our

category. Thus, we can now represent compact structure in this flavour of linear logic.

Chapter 8

The linear typed lambda calculus

This section will provide an overview of the linear typed lambda calculus of [AT10], to prepare the ground for our extension to *dagger compact categories* in the next section. We will define well formed formulas for terms, types and sequents and provide Gentzen-style inference rules for deriving these formulas. This lambda calculus provides a computational interpretation for symmetric monoidal closed categories and is in direct correspondence with the multiplicative fragment of intuitionistic linear logic.

Definition 8.0.1 (Variables and terms in the lambda calculus)**.** The fundamental building blocks of our language are *variables*. They are denoted by single letters and are traditionally represented using the later letters of the alphabet (i.e. x, y, z). These variables can then be combined with each other to form composite *terms*, denoted by different combinations of the following forms:

$$t \quad ::= \quad x \mid t_1 \otimes t_2 \mid \text{let } \psi \text{ be } x \otimes y \text{ in } c \mid \lambda x.t \mid fa$$

Definition 8.0.2 (Types in the lambda calculus)**.** Every term in our language, regardless of whether it is a variable or composite, has a *type*. Types can appear as any combination of the following forms:

$$\text{type} \quad ::= \quad A \mid A \otimes B \mid A \multimap B$$

Definition 8.0.3 (Typing judgements in the lambda calculus)**.** The typing judgements, or sequents, of our language are composed of terms and their respective types. They are always of the form:

$$x_1 : A_1,\ x_2 : A_2,\ \ldots,\ x_n : A_n \vdash t : B$$

A set of typing rules is used to produce typing judgements. Now that we know which formulas are well formed, we can continue our language exposition by providing these rules in the form of a Gentzen-style Sequent Calculus. The set of inference rules corresponds to the rules that were presented in the section on linear logic. The rules and their respective correspondences in category theory are as follows:

	Lambda Calculus	Categories
Id	$\dfrac{}{x : A \vdash x : A}$	$\dfrac{}{id_A : A \longrightarrow A}$
$\bigotimes R$	$\dfrac{\Gamma \vdash a : A \quad \Delta \vdash b : B}{\Gamma, \Delta \vdash a \otimes b : A \otimes B}$	$\dfrac{f : \Gamma \longrightarrow A \quad g : \Delta \longrightarrow B}{f \otimes g : \Gamma \otimes \Delta \longrightarrow A \otimes B}$
$\bigotimes L$	$\dfrac{\Gamma, x : A, y : B \vdash c : C}{\Gamma, \psi : A \otimes B \vdash \text{ let } \psi \text{ be } x \otimes y \text{ in } c : C}$	$\dfrac{f : (\Gamma \otimes A) \otimes B \longrightarrow C}{f \circ a_{A,B,\Gamma} : \Gamma \otimes (A \otimes B) \longrightarrow C}$
Cut	$\dfrac{\Gamma \vdash c : A \quad x : A, \Delta \vdash b : B}{\Gamma, \Delta \vdash b[c/x] : B}$	$\dfrac{f : \Gamma \longrightarrow A \quad g : A \otimes \Delta \longrightarrow B}{g \circ (f \otimes id_\Delta) : \Gamma \otimes \Delta \longrightarrow B}$
$\multimap R$	$\dfrac{\Gamma, x : A \vdash b : B}{\Gamma \vdash \lambda x.b : A \multimap B}$	$\dfrac{f : \Gamma \otimes A \longrightarrow B}{\Lambda(f) : \Gamma \longrightarrow (A \multimap B)}$
$\multimap E$	$\dfrac{\Gamma \vdash g : A \multimap B \quad \Delta \vdash a : A}{\Gamma, \Delta \vdash ga : B}$	$\dfrac{f : \Gamma \longrightarrow (A \multimap B) \quad g : \Delta \longrightarrow A}{ev_{A,B} \circ (f \otimes g) : \Gamma \otimes \Delta \longrightarrow B}$

Applying these rules a number of times can lead us to composite but reducible forms. These are reduced using a process called β-reduction as follows:

$$\begin{aligned} (\lambda x.b)a &\xrightarrow{\beta} b[a/x] \\ \text{let } a \otimes b \text{ be } x \otimes y \text{ in } c &\xrightarrow{\beta} c[a/x, b/y] \end{aligned}$$

The notation $b[c/x]$ is referred to as substitution and means "take b and replace all free occurrences of x in it with c". Substitution is a meta operation that takes place outside of the language. In defining an operational semantics for substitution, care has to be taken to prevent us from violating the capture of free variables[4] of terms. A more rigorous way of defining substitution is given by an induction on the structure of b:

$$\begin{aligned} y\,[c/x] &:= \begin{cases} c & \text{for } x = y, \\ y & \text{for } x \neq y \end{cases} \\ (pq)[c/x] &:= (p[c/x])(q[c/x]) \\ (\lambda z.q)[c/x] &:= \lambda z.(q[c/x]) \end{aligned}$$

A number of interesting translations take place between category theoretic and lambda calculus notions. One of them occurs in $\bigotimes L$, the rule that defines tensor associativity in our category, whereby we gain a rule on how to type a term when, instead of using two variables for its derivation, we plug in a composite term in their place. Another interesting case is Cut, the rule that defines composition of functions, which in the lambda calculus corresponds to a type rule for substitution. Implication elimination $(\multimap E)$, the rule responsible for the evaluation operation in closed categories, turns out to be function application. Consequently, the related notion of Currying that comes with the right implication rule $(\multimap R)$, corresponds to lambda abstraction.

Similarly to the way we defined rules in the section on Logic, we need to add the structural rule of Exchange, in order to account for the symmetry isomorphism of the tensor:

$$\frac{\Gamma, x : A, y : B, \Delta \vdash c : C}{\Gamma, y : B, x : A, \Delta \vdash c : C} \qquad \frac{f : \Gamma \otimes A \otimes B \otimes \Delta \longrightarrow C}{f \circ (id_\Gamma \otimes s_{A,B} \otimes id_\Delta) : \Gamma \otimes B \otimes A \otimes \Delta \longrightarrow C}$$

[4]In the cases where b is a λ-abstraction, certain provisions have to be in place to prevent us from violating the capture of free variables. These provisions are that $x \neq z$ and $z \notin FV(c)$.

Part III

Quantum programming and classical control

Chapter 9

The dagger lambda calculus

Dagger compact categories were first introduced in [ABP99], albeit under a different name, using some of the terminology of [DR89]. They were later proposed by [AC04] and [Sel07] as an axiomatic framework for the study of quantum protocols. Though a lot of work has been done on categorically driven quantum programming languages [SV06], [SV08] and [SV10], these lambda calculi did not provide a way of modelling the dagger functor of dagger compact categories. The work of [BS10] highlighted the importance of dagger compact categories for the semantics of quantum computation; it presented a rough correspondence between quantum computation, logic and the lambda calculus, yet its type theory fell short of providing a correspondence to the entire structure of dagger compact categories. This section fills this gap by presenting the *dagger lambda calculus*: a computational interpretation for dagger compact categories.

9.1 Language construction

We will now construct a language for *dagger compact categories* by defining well formed formulas for terms, types and sequents. The rules for deriving these formulas will be given in the form of Gentzen-style inference rules. In order to give computational meaning to our language, we will begin our presentation of the typing dynamics by reformalising the linear typed lambda calculus of [AT10] with the explicit substitution used by the linear chemical abstract machine of [Abr93]. The linear negation we will be using causes a significant collapse between conjunction and disjunction, extends tensor to a (potentially) binding operator, and provides us with a semantics similar to that of the proof nets in [AD06]. The set of rules we use to define

this language is kept at a minimum, allowing for clean proofs of the various desired properties. Many familiar computational notions do not appear as primitives, but they do arise as constructed notions in good time.

Definition 9.1.1 (Variables, constants and terms in the dagger lambda calculus)**.** The fundamental building blocks of our language are *variables*; they are denoted by single letters and are traditionally represented using the later letters of the alphabet (i.e. x, y, z). We also allow for the use of *constant terms* (i.e. c_1, c_2, c_3); these are terms with an inherent value, that cannot serve as placeholders for substitution. These primitives can then be combined with each other to form composite *terms*, denoted by different combinations of the following forms:

$$\langle term \rangle \quad ::= \quad variable \mid \langle term \rangle_* \mid \langle term \rangle \otimes \langle term \rangle \mid constant$$

Definition 9.1.2 (Types in the dagger lambda calculus)**.** Every term in our language, regardless of whether it is a variable, a constant or composite, has a *type*. We will first start by defining a set of *atomic types*; these are traditionally represented using capital letters (i.e. A, B, C). Atomic types can then be combined to give us types of the following forms:

$$\langle type \rangle \quad ::= \quad atomic \mid \langle type \rangle^* \mid \langle type \rangle \otimes \langle type \rangle$$

The star operator that we use is not a repetition operator; instead, it corresponds to a particular form of *linear negation*. As one would expect from a negation operation, the star operator is involutive $(a_*)_* \equiv a$ and $(A^*)^* \equiv A$. Abramsky [Abr93] proposed using linear negation as the passageway between Intuitionistic Linear Logic and Classical Linear Logic. The linear negation used in [AD06] "trivialized" the notion of De Morgan duality of [Abr93] by setting $(A \otimes B)^* := A^* \otimes B^*$. The linear negation that we use is similar to the one used in [CPP08]; it distributes differently over tensor by performing a swap of the terms/types at hand and allows for a more "planar" representation.

Definition 9.1.3 (Linear negation)**.** The star operator is a form of linear negation whose De Morgan duality is defined by:

$$\begin{aligned} (a \otimes b)_* &:= b_* \otimes a_* && \text{on terms and} \\ (A \otimes B)^* &:= B^* \otimes A^* && \text{on types.} \end{aligned}$$

Definition 9.1.4 (Scalars)**.** One of the language's atomic types, denoted by I, acts as the tensor unit. One of the very important properties of the type I is *negation invariance*, whereby $I \equiv I^*$. We say that a term i is a *scalar* iff it is of type I.

Definition 9.1.5 (Dimensions)**.** For every type A, we will define a scalar constant $D_A : I$, referring to it as the *dimension* of type A. The dimension of I is defined to be $D_I = 1 : I$, where $1 = 1_* : I \equiv I^*$.

Definition 9.1.6 (Soup)**.** All of the computation in our language is performed inside a relational *soup*. The soup is a set, consisting of pairs of equityped terms, connecting them to each other in a form of explicit substitution. A *soup connection* between two terms of type A is written as $t_1 :_A t_2$ or $t_1 : t_2$; to simplify our notation in the soup, we omit writing the type whenever there is no ambiguity about the type of the connected terms. The resulting soup is of the form $S = \{v_1 : v_2, \ldots, v_{m-1} : v_m\}$. We use the following property $\{a_1 : a_2\} \equiv \{a_{2*} : a_{1*}\}$ to equate some soup terms by collapsing them into the same congruence class. We also define *soup negation* as $(S \cup S')_* := S_* \cup S'_*$, where $\{t : u\}_* := \{t_* : u_*\}$.

Definition 9.1.7 (Typing judgements in the dagger lambda calculus)**.** The *typing judgements*, or *sequents*, of our language are composed of terms, their respective types and a relational soup. A typing judgement is thus represented by:

$$t_1 : A_1,\ t_2 : A_2,\ \ldots,\ t_n : A_n \vdash_S t : B$$

Example. *In the following typing judgement, the types of t_1 and t_2 are both known to be A. Similarly, we know that both D_C and 1 are scalars, so their type is I. We omit writing the types for soup connections $t_1 : t_2$ and $D_C : 1$ but, to prevent ambiguity, we have to write it for $x :_B x$, because we have no other way of deducing it from the sequent:*

$$t_1 : A \vdash_{\{t_1:t_2, x:_B x, D_C:1\}} t_2 : A$$

Now that we know which formulas are well formed in our language, we can proceed by defining a notion of binding. Contrary to what we are used to from the lambda calculus, where the notion of binding is restricted in scope to the confines of a single term, the dagger lambda calculus supports a binding that is global and whose scope spans the entire typing judgement. The computational interpretation of classical linear logic, which was provided by [Abr93] in his linear chemical abstract machine, views two occurrences of the same variable as two ends of a communication channel. Adhering to the spirit of that definition, we define binding as follows:

Definition 9.1.8 (Bound variables and terms in the dagger lambda calculus)**.** For any variable x, we say that it is a *bound variable* when it appears twice within a given sequent. As such, variable capture is not limited to the scope of a single term but spans the entire sequent. For any term t that does not contain any occurrences of constants, we say that term is captured when it consists entirely of variables that are captured within the scope of the current sequent. We use the phrases *bound term* and *bundle of bound variables* interchangeably when referring to captured terms. Trivially, a bound variable is also a bound term.

Example. *In the following sequent, x_1, x_2, y_1, y_2 and f are all bound variables. The individual variables may be free when looking at subterms $x_{1*} \otimes y_1$ and $x_{2*} \otimes y_2$ but, when considering the scope of the entire sequent, they are captured by other occurrences of themselves in the soup. Moreover, the terms $x_{1*} \otimes y_1$ and $x_{2*} \otimes y_2$ are both bound terms because they contain no constants and they consist solely of variables that are captured by variables in the soup:*

$$x_{1*} \otimes y_1 : A^* \otimes B \vdash_{\{x_{1*} \otimes y_1 : f, f : x_{2*} \otimes y_2\}} x_{2*} \otimes y_2 : A^* \otimes B$$

In the following sequent, f, y, x_1 and x_2 are bound variables; they can also be viewed as bound terms since a single variable is a term and since they do not contain any constants. The term $x_1 * \otimes x_1$ is a bundle of bound variables because it contains no constants and consists solely of bound variables. The term $c_* \otimes x_2$, however, is not a bundle of bound variables because it contains a constant called c:

$$f : A^* \otimes B \vdash_{\{x_1 * \otimes x_1 : c_* \otimes x_2, f : x_{2*} \otimes y\}} y : B$$

Remark. As will become obvious from our language's sequent rules, which will impose linearity constraints on the introduction of variables, the nature of linearity in our language mandates that all of the variables within a given sequent occur exactly twice. This means that all of the free variables in a given term will occur once more in the sequent within which they reside, hence becoming captured in the scope of that sequent. Within that scope, all terms will essentially consist of captured variables and constants.

Definition 9.1.9 (α-renaming on variables in the dagger lambda calculus)**.** A bound variable x can be α-renamed by replacing all of its instances, in a given sequent, with a bundle of bound variables t. The term t has to be of the same type as x, must not contain any constants (since it will be a bundle of bound variables), and it must consist of variables that do not already appear in the sequent.

We can now extend the operation of α-renaming to operate on captured terms:

Definition 9.1.10 (α-renaming on terms in the dagger lambda calculus)**.** A bound term t can be α-renamed by either α-renaming its constituent variables or, in cases where t appears twice in a given sequent, by replacing all of its instances with a variable x. The variable x has to be of the same type as t and it must not already appear in the sequent.

Definition 9.1.11 (α-equivalence in the dagger lambda calculus)**.** We define a notion of *α-equivalence* as the reflexive, symmetric and transitive closure of α-renaming. In other words, we say that two sequents are *α-equivalent*, or *equivalent up to α-renaming*, when one can be transformed to the other by α-renaming zero or more terms.

Example. *Going back to the examples we used earlier, the sequent*

$$x_{1*} \otimes y_1 : A^* \otimes B \vdash_{\{x_{1*} \otimes y_1 : f, f : x_{2*} \otimes y_2\}} x_{2*} \otimes y_2 : A^* \otimes B$$

is α-equivalent to

$$g : A^* \otimes B \vdash_{\{g : f, f : x_{2*} \otimes y_2\}} x_{2*} \otimes y_2 : A^* \otimes B$$

because we can α-rename the bound term $x_{1*} \otimes y_1$ into the variable g. Similarly, the sequent

$$f : A^* \otimes B \vdash_{\{x_{1*} \otimes x_1 : c_* \otimes x_2, f : x_{2*} \otimes y\}} y : B$$

is α-equivalent to

$$x_{3*} \otimes y_2 : A^* \otimes B \vdash_{\{x_{1*} \otimes x_1 : c_* \otimes x_2, x_{3*} \otimes y_2 : x_{2*} \otimes y_1\}} y_1 : B$$

because we can α-rename the bound variable y into y_1 and also α-rename the bound variable f into the term $x_{3*} \otimes y_2$.

Definition 9.1.12 (Typing contexts in the dagger lambda calculus)**.** The left-hand-side of a typing judgement is actually a list of typed terms. We use the letters Γ and Δ as shorthand for arbitrary (possibly empty) lists of such terms. Let Δ be the list $t_1 : T_1, t_2 : T_2, \ldots, t_n : T_n$. We define $\bigotimes \Delta$ to be the term $(((t_1 \otimes t_2) \otimes \ldots) \otimes t_n) : (((T_1 \otimes T_2) \otimes \ldots) \otimes T_n)$, referring to it as Δ *in tensor form.*

Our language exposition features a Gentzen-style Sequent Calculus, which provides us with the inference rules used to produce judgements. Rules with a double line are bidirectional; sequents matching the top of the rule can be used to derive sequents matching the bottom and vice versa. The rules are formed in a way that allows composite terms to appear to the left of the turnstile. The set of sequent rules is:

$$\frac{}{x : A \vdash x : A}\ \text{Id},$$

$$\frac{a : A \vdash_S b : B}{a_* : A^* \vdash_{S_*} b_* : B^*}\ \text{Negation},$$

$$\frac{\Gamma \vdash_{S_1} a : A \qquad \Delta \vdash_{S_2} b : B}{\Gamma, \bigotimes \Delta \vdash_{S_1 \cup S_2} a \otimes b : A \otimes B}\ \otimes R,$$

$$\frac{\Gamma, a : A, b : B \vdash_S c : C}{\Gamma, a \otimes b : A \otimes B \vdash_S c : C}\ \otimes L,$$

$$\frac{\Gamma \vdash_{S_1} a : A \qquad a' : A, \Delta \vdash_{S_2} b : B}{\Gamma, \Delta \vdash_{S_1 \cup S_2 \cup \{a:a'\}} b : B}\ \text{Cut},$$

$$\frac{a : A, \Gamma \vdash_S b : B}{\Gamma \vdash_S a_* \otimes b : A^* \otimes B}\ \text{Curry}.$$

Linearity constraints: The identity axiom (Id) is the only inference rule we have for introducing variables into our expressions. Consequently, variables are always introduced as bound pairs. We should note at this point that $\otimes R$ and Cut are the only two rules that can be used to merge two typing judgements. In order to preserve linearity, we have to impose a very important condition on such a merge; in doing so, we will prevent the appearance of more than two instances of a variable in a given sequent. The condition required, in order to merge two sequents, is that they do not share any common variables. Whenever we want to use the $\otimes R$ and Cut rules to merge two sequents whose variables overlap, we have to α-rename them first to ensure that the linearity condition is satisfied.

We sometimes use sequents with an empty right-hand-side, like $a : A, \Gamma \vdash$, as shorthand for $a : A, \Gamma \vdash 1 : I$. Such sequents are easy to produce by using *Uncurrying*, the inverse of the *Curry* rule, together with the constant $1 : I$:

$$\dfrac{\dfrac{\Gamma \vdash a_* : A^* \qquad \vdash 1 : I}{\Gamma \vdash a_* \otimes 1 : A^* \otimes I}\otimes R}{a : A, \Gamma \vdash 1 : I}\ \text{Uncurry}$$

The language has a structural exchange rule that can be used to swap terms on the left hand side of a sequent. When navigating through a proof tree, instances of the exchange rule can be used to keep track of which terms were swapped and at which points during a derivation:

$$\dfrac{\Gamma, a : A, b : B, \Delta \vdash c : C}{\Gamma, b : B, a : A, \Delta \vdash c : C}\ \text{Exchange.}$$

Our language also has two unit rules, λ_Γ and ρ_Γ, that are used to more accurately represent scalars:

$$\dfrac{\Gamma \vdash_{S \cup \{i_* : 1\}} b : B}{i : I, \Gamma \vdash_S b : B}\,\lambda_\Gamma,$$

$$\dfrac{\Gamma \vdash_{S \cup \{i_* : 1\}} b : B}{\Gamma, i : I \vdash_S b : B}\,\rho_\Gamma.$$

Our language dynamics are defined through soup rules. These rules explain how the relational connections propagate within the soup, giving rise to an operational semantics for a form of "global substitution" that resembles pattern matching on terms. The soup propagation rules, called *bifunctoriality*, *trace* and *cancellation* respectively, are:

$$\begin{aligned} S \cup \{a \otimes b : c \otimes d\} &\longrightarrow S \cup \{a : c, b : d\} \\ S \cup \{x :_A x\} &\longrightarrow S \cup \{D_A : 1\} \\ S \cup \{1 : 1\} &\longrightarrow S \end{aligned}$$

where ψ is a constant and x is a variable. Our soup rules also contain a *consumption rule*. This rule uses up a relational connection between $\{t : u\}$ to perform a substitution in the typing judgement. Note, however, that the term we are substituting for has to be one that was captured in the scope of the sequent:

$$\Gamma \vdash_{S \cup \{t:u\}} b : B \longrightarrow \Big(\Gamma \vdash_S b : B\Big)\Big[t/u\Big] \quad \text{if } u \text{ does not contain constants,}$$

$$\Gamma \vdash_{S \cup \{t:u\}} b : B \longrightarrow \Big(\Gamma \vdash_S b : B\Big)\Big[u/t\Big] \quad \text{if } t \text{ does not contain constants.}$$

If t and u are both without constants, linearity implies that their constituent variables were all captured in the scope of the original sequent. In such a case, we can choose the way in which we want to substitute. This gives us a symmetric notion of substitution, where our choice of substitution does not affect the typing judgement, as the sequents will be equivalent up to alpha renaming.

Example. *Consider the following sequent:*

$$f : A^* \otimes B \vdash_{\{f:c_* \otimes y\}} y : B$$

The variable f is captured within the scope of the sequent. As such, we can use the consumption rule to consume the connection in the soup and substitute $c_* \otimes y$ for f in the rest of the sequent. This will change the sequent to:

$$c_* \otimes y : A^* \otimes B \vdash y : B$$

Alternatively, if we had α-renamed the original sequent to:

$$x_{1*} \otimes y_1 : A^* \otimes B \vdash_{\{x_{1*} \otimes y_1 : c_* \otimes y_2\}} y_2 : B$$

we could have then used the bifunctoriality rule to split the soup connection:

$$x_{1*} \otimes y_1 : A^* \otimes B \vdash_{\{x_{1*}:c_*, y_1:y_2\}} y_2 : B$$

The first connection of the resulting soup is only consumable in one way, since c is a constant, by substituting c_* for x_{1*}. The second soup connection, however, presents us with a choice, since both y_1 and y_2 are captured in the sequent. One choice will give us

$$c_* \otimes y_2 : A^* \otimes B \vdash y_2 : B$$

while the other choice will give us

$$c_* \otimes y_1 : A^* \otimes B \vdash y_1 : B$$

Upon closer inspection, one will notice that all three of the resulting sequents are α-equivalent.

Definition 9.1.13 (Soup reduction)**.** We use the term *soup reduction* to refer to the binary relation that extends α-equivalence with the sequent transformations that are caused by applying one of the soup rules. Thus, for two sequents $\Gamma \vdash_{S_1} t : T$ and $\Gamma \vdash_{S_2} t : T$, if the soup S_1 is transformed into S_2 through the application of one of the soup propagation rules, $S_1 \to S_2$, then we say that one sequent reduces to the other via *soup reduction*. Similarly, if a sequent J_1 is transformed into J_2 by using the consumption rule to perform a substitution, we say that J_1 reduces to J_2 via *soup reduction*.

Definition 9.1.14 (Soup equivalence)**.** We define a notion of *soup equivalence* as the reflexive, symmetric and transitive closure of soup reduction. In other words, we say that two sequents J_1 and J_2 are *soup-equivalent*, or *equivalent up to soup-reduction*, when we can convert one to the other by using zero or more instances of α-renaming and soup reduction.

We can now use the rules that we have defined so far in order to express the computational notion of application:

Definition 9.1.15 (Application in the dagger lambda calculus)**.** Let t and f be terms such that $t : A$ and $f : A^* \otimes B$ for some types A and B. We define the *application* ft as representing a variable $x : B$, along with a connection in our soup. The origins of the application affect the structure of its corresponding soup connection:

$$\begin{aligned} ft : B, \Gamma \vdash c : C &:= x : B, \Gamma \vdash_{\{f:t_* \otimes x\}_*} c : C \\ \Gamma \vdash ft : B &:= \Gamma \vdash_{\{f:t_* \otimes x\}} x : B \end{aligned}$$

For an application originating inside our soup, we have:

$$\begin{aligned} \{ft : c\} &:= \{x : c\} \cup \{f : t_* \otimes x\} \\ \{c : ft\} &:= \{c : x\} \cup \{f : t_* \otimes x\}_* \end{aligned}$$

Corollary 9.1.1 (Beta reduction)**.** *This immediately allows us to represent a form of beta reduction. Instead of relying on an implicit meta concept of substitution, our beta reduction is going to express the binding and reduction of terms by connecting them in the context soup by setting* $(a_* \otimes b)t \xrightarrow{\beta} b$ *while causing* $\{t : a\}$ *or* $\{t : a\}_*$ *to be added to the relational soup.*

Proof. This is derived from our definition of application because $(a_* \otimes b)t$ represents a variable x along with one of two possible connections in our soup. The soup connection can be manipulated into:

$$\{a_* \otimes b : t_* \otimes x\} \to \{a_* : t_*, b : x\} \to \{t : a\} \cup \{b : x\}$$

$$\{a_* \otimes b : t_* \otimes x\}_* \to \{a_* : t_*, b : x\}_* \to \{t : a\}_* \cup \{x : b\}$$

The connection between b and x can then be consumed to change the variable x into a b. All that remains is $\{t : a\}$ or $\{t : a\}_*$. □

Now that all of the language's rules are in place, we can make up for its apparent lack of a λ operator by defining it to be a notational shorthand:

Definition 9.1.16 (Lambda abstraction in the dagger lambda calculus)**.** $\lambda a.b := a_* \otimes b$ and $A \multimap B := A^* \otimes B$

The following combinators are used in the rest of this dissertation:

$$\begin{aligned} id_A &:= \lambda a.a \text{ (where } a : A) \\ \bar{b} &:= \lambda g.\lambda f.\lambda a.g(fa) \\ \bar{s} &:= \lambda(a \otimes b).(b \otimes a) \\ \bar{t} &:= \lambda f.\lambda g.\lambda(x_1 \otimes x_2).(fx_1 \otimes gx_2) \end{aligned}$$

Theorem 9.1.1 (Admissibility of $\multimap E$)**.** *We can also use the definition of application to demonstrate that an implication elimination rule ($\multimap E$) is admissible within our set of rules:*

$$\dfrac{\Gamma \vdash_{S_1} t : A \qquad \dfrac{\Delta \vdash_{S_2} f : A^* \otimes B \qquad \dfrac{\dfrac{\dfrac{}{a : A \vdash a : A}}{a_* : A^* \vdash a_* : A^*} \qquad \dfrac{}{b : B \vdash b : B}}{a_* : A^*, b : B \vdash a_* \otimes b : A^* \otimes B}}{a_* \otimes b : A^* \otimes B \vdash a_* \otimes b : A^* \otimes B}\ \text{Cut}}{\dfrac{\Delta \vdash_{S_2 \cup \{f : a_* \otimes b\}} a_* \otimes b : A^* \otimes B}{a : A, \Delta \vdash_{S_2 \cup \{f : a_* \otimes b\}} b : B}\ \text{Uncurry}}}{\dfrac{\Gamma, \Delta \vdash_{S_1 \cup S_2 \cup \{t : a, f : a_* \otimes b\}} b : B}{\dfrac{\Gamma, \Delta \vdash_{S_1 \cup S_2 \cup \{f : t_* \otimes b\}} b : B}{\Gamma, \Delta \vdash_{S_1 \cup S_2} ft : B}}}\ \text{Cut}$$

We define some additional notational conventions, so that we can more easily describe the reversal in the causal order of computation:

Definition 9.1.17 (Complex conjugation)**.** Let $f : A^* \otimes B$ be an arbitrary function. As a notational convention, we set $f^* := \bar{s}f : B \otimes A^*$.

Theorem 9.1.2 (Admissibility of †-flip)**.** *We can use the language's rules and definitions in order to admit a new structural rule called the †-flip. This rule contains all the computational symmetry that we will later need in order to model the dagger functor:*

$$\dfrac{\dfrac{\dfrac{\dfrac{a : A \vdash_S b : B}{a_* : A^* \vdash_{S_*} b_* : B^*}\ \text{Negation}}{b : B, a_* : A^* \vdash_{S_*}}\ \text{Uncurry}}{a_* : A^*, b : B \vdash_{S_*}}\ \text{Exchange}}{b : B \vdash_{S_*} a : A}\ \text{Curry}$$

Theorem 9.1.3 (Interchangeability of †-flip and Negation)**.** *Alternatively, we could have defined the language by including †-flip in our initial set of sequent rules. That would have allowed us to admit the Negation rule as a derived rule:*

$$\dfrac{\dfrac{\dfrac{\dfrac{a : A \vdash_S b : B}{b : B \vdash_{S_*} a : A}\,\dagger\text{-flip}}{a_* : A^*, b : B \vdash_{S_*}}\,\text{Uncurry}}{b : B, a_* : A^* \vdash_{S_*}}\,\text{Exchange}}{a_* : A^* \vdash_{S_*} b_* : B^*}\,\text{Curry}$$

9.1.1 Scalars

Similarly to the attachable monoid that is described in [Abr05] for multiplying scalars, we can optionally define a multiplication operation for the scalars in the dagger lambda calculus. This is not part of the structure that is necessary to model dagger compact categories computationally, hence the designation *optional*, but it does provide a good example of how connections propagate in the soup:

Definition 9.1.18 (Scalar multiplication)**.** For any two scalars $m : I$ and $n : I$, we define a multiplication operation $m \cdot n : I$ such that:

$$m \cdot 1 = 1 \cdot m = m$$

and

$$\{m \cdot p : n \cdot q\} := \{m : n, p : q\}$$

The operation features a number of properties. To help the reader get more accustomed to the way things propagate in the soup, we will demonstrate some of them as an example. First of all, scalar multiplication is *associative*:

Lemma 9.1.1 (Associativity of multiplication)**.** $(a \cdot b) \cdot c = a \cdot (b \cdot c)$

Proof.

$$\begin{aligned}\{(a \cdot b) \cdot c : 1\} &= \{(a \cdot b) \cdot c : (1 \cdot 1) \cdot 1\} \\ &= \{a : 1, b : 1, c : 1\} \\ &= \{a \cdot (b \cdot c) : 1 \cdot (1 \cdot 1)\} \\ &= \{a \cdot (b \cdot c) : 1\}\end{aligned}$$

□

The multiplication operation is also *commutative*:

Lemma 9.1.2 (Commutativity of multiplication)**.** $m \cdot n = n \cdot m$

Proof.

$$\begin{aligned}\{m \cdot n : 1\} &= \{m \cdot n : 1 \cdot 1\} \\ &= \{m : 1, n : 1\} \\ &= \{n : 1, m : 1\} \\ &= \{n \cdot m : 1 \cdot 1\} \\ &= \{n \cdot m : 1\}\end{aligned}$$

□

It is *sesquilinear*:

Lemma 9.1.3 (Sesquilinearity of scalar connections)**.** $\{m : n\} = \{m \cdot n_* : 1\}$

Proof.

$$\begin{aligned}\{m : n\} &= \{m \cdot 1 : 1 \cdot n\} \\ &= \{m : 1, 1 : n\} \\ &= \{m : 1, n_* : 1\} \\ &= \{m \cdot n_* : 1 \cdot 1\} \\ &= \{m \cdot n_* : 1\}\end{aligned}$$

□

Finally, it is easy to deduce that the dimension of a tensor of types distributes into a product of dimensions:

Corollary 9.1.2 (Dimension multiplication)**.** $\{D_A \cdot D_B : 1\} = \{D_{A \otimes B} : 1\}$

Proof.

$$\begin{aligned}\{D_A \cdot D_B : 1\} &= \{D_A : 1, D_B : 1\} & &= \{a :_A a, b :_B b\} \\ &= \{a \otimes b :_{A \otimes B} a \otimes b\} & &= \{D_{A \otimes B} : 1\}\end{aligned}$$

□

9.2 Proofs of properties

Many lambda calculi suffer from being complicated, which makes it hard to prepare and follow proofs about their properties. Our language is tractable and consists of a minimal set of rules. As a result, most of the language's properties are easy to prove by structural inductions. Throughout the rest of this section, we prove that our lambda calculus satisfies most of the really important properties a calculus can have, namely subject reduction, confluence, strong normalisation and consistency.

9.2.1 Subject reduction

The first thing we have to prove, in order to demonstrate that our typing system is well defined, is the consistency of our typing dynamics. In other words, we have to verify that the way in which relational connections propagate through our soup preserves type assignments. This is easy to observe because our soup only connects *equityped* terms. Pair consumption substitutes a term for another of the same type, thus preserving types.

Theorem 9.2.1 (Subject reduction)**.** *Let J_1 and J_2 be two typing judgements such that $J_1 = \Gamma \vdash_S t_1 : A_1$ and $J_2 = \Delta \vdash_{S'} t_2 : A_2$. Suppose that these two judgements are such that we can use a soup reduction rule $S \longrightarrow S'$ to reduce one to the other: $J_1 \longrightarrow J_2$. Then, the reduction will not alter type assignments in any way: $types(\Gamma) = types(\Delta)$ and $A_1 \equiv A_2$.*

Proof. Due to the way the dagger lambda calculus was designed, the proof of subject reduction will be trivial. We will prove this by induction on the rules of the soup reduction. There are four different rules that could be used when performing a soup reduction:

- $S \cup \{a \otimes b : c \otimes d\} \longrightarrow S \cup \{a : c, b : d\}$ (the *bifunctoriality* rule);
- $S \cup \{x :_A x\} \longrightarrow S \cup \{D_A : 1\}$ (the *trace* rule);
- $S \cup \{1 : 1\} \longrightarrow S$ (the *cancellation* rule);
- The *consumption* rule.

If the reduction is an instance of one of the first three rules, then the theorem holds trivially; $types(\Gamma) = types(\Delta)$ and $A_1 \equiv A_2$ since the bifunctoriality, trace and cancellation rules do not alter anything outside of the soup. If the soup reduction is an instance of the consumption rule, then a soup connection will be consumed to substitute a term at the other end of a

bounded pair of variables. The substitution may be global in scope, but it does not affect the sequent's typing, since it is substituting one term for another one of the same type. The act of consumption itself does not affect the typing of the sequent either, since it removes a connection from the soup without affecting the terms outside. Therefore, regardless of the soup rule used, soup reduction has no effect on the typing of terms outside of the soup. This ensures that the typing dynamics of the dagger lambda calculus will be consistent. □

9.2.2 Normalisation

Strong normalisation is a highly sought after property for lambda calculi, primarily because of the implications it has on the practical implementation of the language. A reduction that is strongly normalising implies that every sequent has a normal form. Furthermore, it requires that the normal form is attained after a finite number of steps, without any chance of running into an infinite reduction loop. We now prove that the dagger lambda calculus has this property:

Theorem 9.2.2 (Strong normalisation)**.** *Every sequence of soup reduction steps is finite and ends with a typing judgement that is in normal form.*

Proof. We begin by proving a simplified form of the theorem, where the terms connected in our soup are all of atomic type. Obviously, in a setting like this, the first reduction rule for our soup would never be used since there would be no tensored terms. The simplified theorem can be proved by induction on the length of the soup. In judgements where the soup is empty, or only contains pairs of constants that are not usable, it is obvious that reduction cannot proceed any further and that we have already reached a normal form in a finite (actually zero) number of steps. For a reduction soup with at least one usable connection, there are three possible ways this could go:

(a) A pair of the form $\{x :_A x\}$ can be transformed into $\{D_A : 1\}$, a scalar reference of the dimension of A. This will either give us an unusable soup connection or, if the type A is I, it will give us $\{1 : 1\}$, which can later be thrown away by using the cancellation rule. Either way, since our soup has a finite length, we will be left with a smaller usable soup. Hence, by the induction hypothesis, a normal form is attainable after finitely many steps.

(b) A pair of the form $\{1 : 1\}$ can be thrown out of the soup as it does not contribute anything to our sequent. In doing so, since our soup has a finite length, we are left with a smaller soup. Hence, by the induction hypothesis, a normal form is attainable after finitely many steps.

(c) A pair of the form $\{t : u\}$, where t or u is bound, can be consumed to perform a substitution. In doing so, since our soup has a finite length, we are again left with a smaller soup. By the induction hypothesis, we can attain a normal form after finitely many steps.

This completes our proof of the simplified form of the theorem. In order to prove the full theorem, we perform an induction on the structure of the relational soup:

(i) A relational soup that does not contain connections between tensored terms leads to a normal form after finitely many steps. This was already proved in the simplified version of this theorem.

(ii) A connection between two tensors $\{a \otimes b : c \otimes d\}$ can be reduced into two distinct connections $\{a : c, b : d\}$. The resulting pairs consist of subterms of the original pair. Hence, by the induction hypothesis, we can attain a normal form after finitely many steps.

□

9.2.3 Confluence

Another very important property for our language is the Church-Rosser property. It ensures that we can end up with the same sequent regardless of the reduction path we choose to follow. A careful observation of our rewrite rules will reveal that the rules are all left-linear.

Lemma 9.2.1 (Left-linearity)**.** *All of our soup rewrite rules are left-linear.*

Proof. In accordance with the linearity constraints of our language, no variable appears more than twice on the left hand side of any of our soup reduction rules:

$$\begin{aligned} S \cup \{a \otimes b : c \otimes d\} &\longrightarrow S \cup \{a : c, b : d\} \\ S \cup \{x :_A x\} &\longrightarrow S \cup \{D_A : 1\} \\ S \cup \{1 : 1\} &\longrightarrow S \\ S \cup \{t : u\} &\longrightarrow S \quad \text{(if } t \text{ or } u \text{ is a bound variable)} \end{aligned}$$

□

One should note, at this point, that our soup rules do exhibit a form of "harmless" overlap. More specifically, the consumption rule ($S \cup \{t : u\} \longrightarrow S$) forms a critical pair with itself in cases where t and u are both bound. Fortunately, as we will see in the next lemma, these pairs can be proved to be *trivial* as they correspond to sequents that are equivalent up to α-renaming.

Lemma 9.2.2 (Symmetry of substitution)**.** *Let J be a typing judgement of the form $J := \Gamma \vdash_{S \cup \{t:u\}} a : A$, where t and u are both bound. The connection $\{t : u\}$ can be consumed in either of two ways; one substitutes t for u and the other substitutes u for t in the typing judgement. Let's call these J_1 and J_2 respectively. J_1 will then be α-equivalent to J_2.*

Proof. We know that

$$J \longrightarrow J_1 := \Big(\Gamma \vdash_S a : A\Big)\Big[t/u\Big]$$

$$J \longrightarrow J_2 := \Big(\Gamma \vdash_S a : A\Big)\Big[u/t\Big]$$

Since t and u are both bound, by linearity, we know that they appear exactly once in $\Gamma \vdash_S a : A$. After substitution is performed, J_1 will have two occurrences of t where t and u used to be, so t will be a bound term in that judgement. Similarly, J_2 will have two occurrences of u where t and u used to be, so u will be a bound term in that judgement. These bound terms occur in the exact same spots, so we can *alpha*-rename J_1 to J_2 and vice versa. □

Corollary 9.2.1 (No overlap)**.** *The rewrite rules have no overlap up to α-equality of typing judgements.*

Theorem 9.2.3 (Confluence)**.** *Our reduction rules have the Church-Rosser property.*

Proof. Our set of rewrite rules is *left-linear* and has no significant overlap, since it only gives rise to critical pairs that are *trivial* up to α-equivalence. Therefore, our rewrite rules constitute a *weakly orthogonal* rewrite system, which is *weakly confluent* according to [Klo92]. Since the rewrite system is both strongly normalising and weakly confluent, we can use Newman's lemma to conclude that it also possesses the Church-Rosser property. See [Klo92] for a more detailed explanation of the properties of orthogonal rewriting systems. □

9.2.4 Consistency

In order to show that our type theory is consistent, we have to show that our soup dynamics do not collapse all equityped terms to the same element.

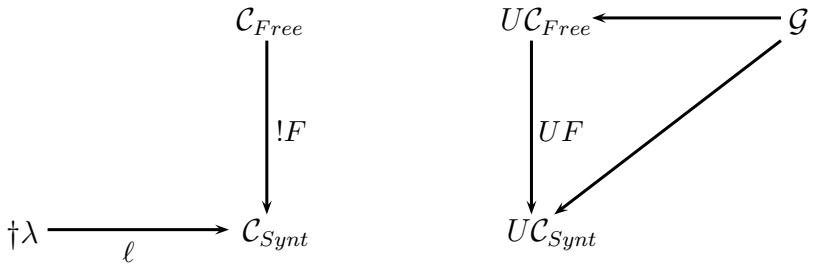
Theorem 9.2.4 (Consistency)**.** *There exist two terms of the same type, henceforth referred to as t_1 and t_2, such that $\Gamma \vdash_{S_1} t_1 : A$ and $\Gamma \vdash_{S_2} t_2 : A$ could never reduce to the same typing judgement.*

Proof. Consider two combinators of the same type, $t_1 = id_{A\otimes A}$ and $t_2 = \bar{s}_{A\otimes A}$. Both terms are closed, containing no free variables or constants. The sequents $\vdash id_{A\otimes A} : (A \otimes A) \multimap (A \otimes A)$ and $\vdash \bar{s}_{A\otimes A} : (A \otimes A) \multimap (A \otimes A)$ are distinct normal forms: They are clearly distinct from one another and cannot be further reduced using any of our rules, thereby proving that they could never reduce to the same typing judgement. □

9.3 Correspondence to dagger compact categories

The purpose of this section is to provide a full Curry-Howard-Lambek correspondence between the dagger lambda calculus and dagger compact categories. We start by defining a directed graph $\mathcal{G}$, representing a signature for dagger compact categories. We then show how that graph can be interpreted to define the free dagger compact category $\mathcal{C}_{Free}$ and the dagger lambda calculus $\dagger\lambda$. An appropriate Cut-elimination procedure is defined to partition the sequents of the dagger lambda calculus into equivalence classes up to soup equivalence. The resulting equivalence classes are modular proof invariants represented by denotations. We show that the types and denotations can be used to form a syntactic category, $\mathcal{C}_{Synt}$, and prove that the category is dagger compact. The diagram below, fashioned to resemble the diagram at the bottom of page 49 in [Mac98], is provided to help visualise the Curry-Howard-Lambek correspondence. In this diagram, $U\mathcal{C}_{Free}$ and $U\mathcal{C}_{Synt}$ are the underlying graphs of their respective categories, where identities, composition, natural isomorphisms and other structural elements of the parent categories have been "forgotten" by applying the forgetful functor U. F is the unique functor between the free and the syntactic category,

that satisfies the rest of the conditions in the diagram.



We will prove an equivalence between the free category and the syntactic category. We should note at this point that our typing conventions of an involutive negation ($A \equiv (A^*)^*$) and negation invariance of the tensor unit ($I \equiv I^*$) implicitly introduce equivalence classes on types. Our proof of equivalence will be achieved by fully exhibiting the correspondence in objects and arrows between the two categories, up to the equivalence classes that are induced by our typing conventions.

9.3.1 A signature for dagger compact categories

The notion of signature we will be using combines that of the algebraic signature of [Sel10] with the directed graph used by [Mac98]. Consider a set of object variables Σ_0. Using the tensor operation, an associated tensor identity and the duality operator star, we can construct the free $(\otimes, I, \Box^*)$-algebra over Σ_0. This corresponds to the set of all object terms or vertices in a compact closed category and will be denoted by $Dagger(\Sigma_0)$. Now consider a set Σ_1 of morphism variables or edges between those vertices. Let dom, cod be a pair of functions such that $dom, cod : \Sigma_1 \longrightarrow Dagger(\Sigma_0)$. Throughout the rest of this section, we will be referring to the graph $\mathcal{G}$ as the directed graph whose vertices and edges are defined by $Dagger(\Sigma_0)$ and Σ_1. This graph forms the signature upon which we will base both the dagger lambda calculus and our description of the free dagger compact category; it includes all of the symbols but none of the logic of the languages that we want to describe.

9.3.2 The free dagger compact category

We will now show how to define the free dagger compact category $\mathcal{C}_{Free}$ as an interpretation of the graph $\mathcal{G}$. A highly intuitive introduction to free categories and how they can be generated from directed graphs can be found

in [Mac98]. Furthermore, a more extensive presentation of the process of constructing of various kinds of free categories can be found in [Sel10]. A more detailed presentation of the incremental buildup to the construction of free dagger compact categories can also be found in [Abr05].

The set of objects for the free category in this section will be the same as the set of vertices $Dagger(\Sigma_0)$ in the graph $\mathcal{G}$. The set of edges Σ_1 in the graph is used to generate morphisms for the free category. Thus, an edge of the form $f : A \to B$ generates an arrow in $\mathcal{C}_{Free}$ which we will denote as $\langle A, f, B\rangle$. The free category over a directed graph, also referred to as a path category, includes morphisms that correspond to the paths generated by combining adjoining edges in $\mathcal{G}$. These morphisms are formed using the free category's composition operation. Given two morphisms $\langle A, f, B\rangle$ and $\langle B, g, C\rangle$, we write their composition in $\mathcal{C}_{Free}$ as $\langle A, f, B, g, C\rangle$.

Since the free category is a monoidal category, it allows us to consider two of the graph's edges concurrently by bringing together their corresponding categorical morphisms using a monoidal tensor product. Given two morphisms $\langle A, f, B\rangle$ and $\langle C, h, D\rangle$, we write their tensor product as $\langle A \otimes C, f \otimes h, B \otimes D\rangle$.

The free category generated by the graph $\mathcal{G}$ also includes a number of morphisms that are part of the dagger compact logical structure. The identities are represented by:

$$\langle A\rangle, \langle B\rangle, \langle C\rangle, \ldots$$

The monoidal natural isomorphisms are written as:

$$\langle A \otimes (B \otimes C), \alpha_{A,B,C}, (A \otimes B) \otimes C\rangle$$

$$\langle I \otimes A, \lambda_A, A\rangle$$

$$\langle A \otimes I, \rho_A, A\rangle$$

The symmetry isomorphism is written as:

$$\langle A \otimes B, \sigma_{A,B}, B \otimes A\rangle$$

And the units and counits are written as:

$$\langle I, \eta_A, A^* \otimes A\rangle$$

$$\langle A \otimes A^*, \varepsilon_A, I\rangle$$

For every map $\langle A, f, B\rangle$ in the free category, the dagger compact logical structure contains maps f_* and $f^\dagger$, represented by $\langle A^*, f_*, B^*\rangle$ and

$\langle B, f^\dagger, A\rangle$ respectively. When acting on compositions of paths, such as $\langle A, f, B, g, C, \ldots, X, h, Y, t, Z\rangle$, the dagger operator reverses the order of operations, yielding:

$$\langle Z, t^\dagger, Y, h^\dagger, X, \ldots, C, g^\dagger, B, f^\dagger, A\rangle$$

9.3.3 The dagger lambda calculus

This section demonstrates how the graph signature $\mathcal{G}$ can be interpreted to derive the dagger lambda calculus. The set of types used by $\dagger\lambda$ is precisely the set of vertices $Dagger(\Sigma_0)$ used in graph $\mathcal{G}$. Every edge

$$f : A \to B$$

in Σ_1 is interpreted as a sequent

$$a : A \vdash_{\{f:a_*\otimes b\}} b : B$$

up to alpha-equivalence. These interpretations essentially introduce constants, in our case $f : A^* \otimes B$, written as sequents that are reminiscent of η-expanded forms. The rest of the rules of the dagger lambda calculus can be used to process and combine sequents, yielding a richer logical structure.

9.3.4 The syntactic category

Following a method that is similar to [Mel09], we will define a process of Cut-elimination by using the soup reduction relation to partition the sequents of the dagger lambda calculus into equivalence classes. The resulting equivalence classes are modular proof invariants called *denotations*. This section demonstrates how these denotations give rise to a dagger compact categorical structure $\mathcal{C}_{Synt}$ called the *syntactic category*.

Definition 9.3.1 (Denotations)**.** We will use the term *denotations* to refer to the equivalence classes that are formed by partitioning the sequents of the lambda calculus according to soup equivalence. Hence, two sequents will correspond to the same denotation if and only if they are equivalent up to soup reduction.

Theorem 9.3.1 (The syntactic category)**.** *The types of the lambda calculus and the denotations generated by soup equivalence form a category whose objects are types and whose arrows are denotations.*

Proof. As we noticed during the proof of the subject reduction property, soup reduction rules do not affect our language's type assignments. Consequently, the type of the premises used by a sequent will be the same across all sequents in a given denotation. Similarly, the type of the conclusion produced by a sequent will be the same across all sequents in a given denotation. For any sequent $\Gamma \vdash_S b : B$, corresponding to a denotation $[\pi_1]$, we will say that its *domain* is Γ and its *codomain* is B, writing this as $[\pi_1] : \Gamma \to B$.

Let $[f] : A \to B$ and $[g] : B \to C$ be denotations representing the soup equivalent forms of some sequents $a : A \vdash_{S_1} b : B$ and $b' : B \vdash_{S_2} c : C$ respectively. For any two such denotations, where the codomain of the first matches the domain of the second, we will define a *composition* operator $\circ$ that can combine them into $[g] \circ [f] : A \to C$. The new denotation will represent all the soup equivalent forms of the sequent that is generated by combining the two sequents using the Cut rule:

$$\frac{a : A \vdash_{S_1} b : B \qquad b' : B \vdash_{S_2} c : C}{a : A \vdash_{S_1 \cup S_2 \cup \{b:b'\}} c : C}\ \text{Cut}$$

The composition operation we just defined inherits associativity from the Cut rule; the order in which Cuts are performed does not matter since the connected terms are allowed to "float" freely within the soup. Therefore, $[h] \circ ([g] \circ [f]) = ([h] \circ [g]) \circ [f]$. Moreover, for every type A, there is a denotation $[id_A]$ that represents the sequent generated by the Identity axiom (Id): $x : A \vdash x : A$.

Composing a denotation $[f] : A \to B$ with an identity yields $[f] \circ [id_A]$ or $[id_B] \circ [f]$ depending on whether we compose with an identity on the right or on the left. The two resulting denotations represent

$$\frac{x : A \vdash x : A \qquad a : A \vdash_S b : B}{x : A \vdash_{S \cup \{x:a\}} b : B} \quad \text{and} \quad \frac{a : A \vdash_S b : B \qquad x : B \vdash x : B}{a : A \vdash_{S \cup \{b:x\}} x : B}$$

both of which are soup equivalent to $a : A \vdash_S b : B$ and the rest of the sequents represented by $[f]$. Hence $[id_B] \circ [f] = [f] = [f] \circ [id_A]$ □

We call this category the *syntactic category*. We will now incrementally check that it satisfies the criteria needed to be a dagger compact category.

Definition 9.3.2 (Syntactic category notational conventions)**.** For notational convenience, we define the following combinators:

$$\alpha_{A,B,C} := \lambda\,(a \otimes (b \otimes c))\,.\,((a \otimes b) \otimes c) : (A \otimes (B \otimes C)) \multimap ((A \otimes B) \otimes C)$$

$$\alpha^{-1}_{A,B,C} := \lambda\,((a \otimes b) \otimes c)\,.\,(a \otimes (b \otimes c)) : ((A \otimes B) \otimes C) \multimap (A \otimes (B \otimes C))$$

$$\lambda_A := \lambda(1 \otimes a).a : (I \otimes A) \multimap A$$
$$\lambda_A^{-1} := \lambda a.(1 \otimes a) : A \multimap (I \otimes A)$$
$$\rho_A := \lambda(a \otimes 1).a : (A \otimes I) \multimap A$$
$$\rho_A^{-1} := \lambda a.(a \otimes 1) : A \multimap (A \otimes I)$$
$$\sigma_{A,B} := \lambda(a \otimes b).(b \otimes a) : (A \otimes B) \multimap (B \otimes A)$$
$$\sigma_{A,B}^{-1} := \sigma_{B,A} = \lambda(b \otimes a).(a \otimes b) : (B \otimes A) \multimap (A \otimes B)$$
$$\eta_A := \lambda 1.(x_* \otimes x) : I \multimap (A^* \otimes A)$$
$$\varepsilon_A := \lambda(x \otimes x_*).1 : (A \otimes A^*) \multimap I$$

Theorem 9.3.2 (Monoidal category)**.** *The syntactic category is a monoidal category*

Proof. Let $[f] : A \to B$ and $[g] : C \to D$ be denotations representing the soup equivalent forms of some sequents $a : A \vdash_{S_1} b : B$ and $c : C \vdash_{S_2} d : D$ respectively. We define a monoidal product $\otimes$ that can combine them into $[f] \otimes [g] : A \otimes B \to C \otimes D$. The new denotation will represent all the soup equivalent forms of the sequent that is generated by combining the two sequents using the right tensor rule:

$$\frac{a : A \vdash_{S_1} b : B \qquad c : C \vdash_{S_2} d : D}{a : A, c : C \vdash_{S_1 \cup S_2} b \otimes d : B \otimes D} \otimes R$$

Let $[f] : A \to B$, $[g] : B \to P$, $[h] : C \to D$ and $[t] : D \to Q$ be denotations in the syntactic category. Using composition and tensor, we can combine these denotations to form $([g] \circ [f]) \otimes ([t] \circ [h])$, which represents the soup equivalent sequents of:

$$\frac{\dfrac{a : A \vdash_{S_1} b : B \qquad b' : B \vdash_{S_3} p : P}{a : A \vdash_{S_1 \cup S_3 \cup \{b:b'\}} p : P}\,\text{Cut} \qquad \dfrac{c : C \vdash_{S_2} d : D \qquad d' : D \vdash_{S_4} q : Q}{c : C \vdash_{S_2 \cup S_4 \cup \{d:d'\}} q : Q}\,\text{Cut}}{a : A, c : C \vdash_{S_1 \cup S_2 \cup S_3 \cup S_4 \cup \{b:b', d:d'\}} p \otimes q : P \otimes Q} \otimes R$$

We can also combine the same denotations to form $([g] \otimes [t]) \circ ([f] \otimes [h])$, which represents the soup equivalent sequents of:

$$\frac{\dfrac{a : A \vdash_{S_1} b : B \qquad c : C \vdash_{S_2} d : D}{a : A, c : C \vdash_{S_1 \cup S_2} b \otimes d : B \otimes D} \otimes R \qquad \dfrac{\dfrac{b' : B \vdash_{S_3} p : P \qquad d' : D \vdash_{S_4} q : Q}{b' : B, d' : D \vdash_{S_3 \cup S_4} p \otimes q : P \otimes Q} \otimes R}{b' \otimes d' : B \otimes D \vdash_{S_3 \cup S_4} p \otimes q : P \otimes Q} \otimes L}{a : A, c : C \vdash_{S_1 \cup S_2 \cup S_3 \cup S_4 \cup \{b \otimes d : b' \otimes d'\}} p \otimes q : P \otimes Q}\,\text{Cut}$$

Applying our soup's bifunctoriality rule, $\{b \otimes d : b' \otimes d'\} \to \{b : b', d : d'\}$, reduces one of these sequents to the other, which means that they belong to the same equivalence class. Hence, the two sequents are represented by the same denotation:

$$([g] \circ [f]) \otimes ([t] \circ [h]) = ([g] \otimes [t]) \circ ([f] \otimes [h])$$

The tensor product also preserves identities since $a \otimes b : A \otimes B \vdash a \otimes b : A \otimes B$ is α-equivalent to $x : A \otimes B \vdash x : A \otimes B$.

The denotation $[\alpha_{A\otimes B,C,D}] \circ [\alpha_{A,B,C\otimes D}]$ represents the soup equivalent sequents of:

$$a : A, b \otimes (c \otimes d) : B \otimes (C \otimes D) \vdash_{S_1} ((a_3 \otimes b_3) \otimes c_3) \otimes d_3 : ((A \otimes B) \otimes C) \otimes D$$

where the soup S_1 is:

$$\left\{ \begin{array}{ll} \alpha_{A,B,C\otimes D} & : \lambda\,(a \otimes (b \otimes (c \otimes d)))\,.\,((a_2 \otimes b_2) \otimes (c_2 \otimes d_2))\,, \\ \alpha_{A\otimes B,C,D} & : \lambda\,((a_2 \otimes b_2) \otimes (c_2 \otimes d_2))\,.\,(((a_3 \otimes b_3) \otimes c_3) \otimes d_3) \end{array} \right\}$$

Similarly, the denotation $([\alpha_{A,B,C}] \otimes [id_D]) \circ [\alpha_{A,B\otimes C,D}] \circ ([id_A] \otimes [\alpha_{B,C,D}])$ represents the soup equivalent sequents of:

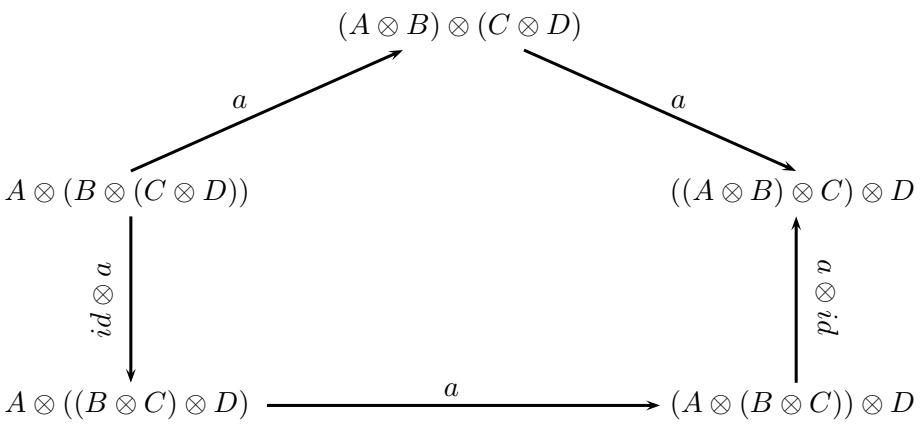
$$a : A, b \otimes (c \otimes d) : B \otimes (C \otimes D) \vdash_{S_2} ((a_3 \otimes b_3) \otimes c_3) \otimes d_3 : ((A \otimes B) \otimes C) \otimes D$$

where the soup S_2 is:

$$\left\{ \begin{array}{ll} \alpha_{B,C,D} & : \lambda\,(b \otimes (c \otimes d))\,.\,((b_1 \otimes c_1) \otimes d_1)\,, \\ \alpha_{A,B\otimes C,D} & : \lambda\,(a \otimes ((b_1 \otimes c_1) \otimes d_1))\,.\,((a_2 \otimes (b_2 \otimes c_2)) \otimes d_3)\,, \\ \alpha_{A,B,C} & : \lambda\,(a_2 \otimes (b_2 \otimes c_2))\,.\,((a_3 \otimes b_3) \otimes c_3) \end{array} \right\}$$

We can use our soup's bifunctoriality and substitution rules to show that S_1 and S_2 are equivalent. Hence, the two denotations we constructed are equal, which corresponds to the associativity pentagon for monoidal categories:

$$[\alpha_{A\otimes B,C,D}] \circ [\alpha_{A,B,C\otimes D}] = ([\alpha_{A,B,C}] \otimes [id_D]) \circ [\alpha_{A,B\otimes C,D}] \circ ([id_A] \otimes [\alpha_{B,C,D}])$$



Now consider the denotation $([\rho_A] \otimes [id_B]) \circ [\alpha_{A,I,B}]$, which represents the soup equivalent sequents of:

$$a : A, i \otimes b : I \otimes B \vdash_{S_3} a_3 \otimes b_2 : A \otimes B$$

where the soup S_3 is:

$$\left\{ \begin{array}{ll} \alpha_{A,I,B} & : \lambda(a \otimes (i \otimes b)).((a_2 \otimes i_2) \otimes b_2), \\ \rho_A & : \lambda(a_2 \otimes i_2).a_3 \end{array} \right\}$$

and the denotation $[id_A] \otimes [\lambda_B]$, which corresponds to the soup equivalent sequents of:

$$a : A, i \otimes b : I \otimes B \vdash_{S_4} a_3 \otimes b_2 : A \otimes B$$

where the soup S_4 is:

$$\left\{ \begin{array}{ll} \lambda_B & : \lambda(i \otimes b).b_2, \\ a & : a_3 \end{array} \right\}$$

The two soups, S_3 and S_4, are equivalent up to soup reduction, which means that the two denotations representing them are equal:

$$([\rho_A] \otimes [id_B]) \circ [\alpha_{A,I,B}] = [id_A] \otimes [\lambda_B]$$

The syntactic category, therefore, satisfies all of the requirements and coherence conditions of a monoidal category. □

Theorem 9.3.3 (Symmetric monoidal category)**.** *The syntactic category is a symmetric monoidal category*

Proof. Consider the denotation $[\sigma_{B,A}] \circ [\sigma_{A,B}]$, representing the soup equivalent sequents of:

$$a : A, b : B \vdash_{\{\sigma_{A,B}:\lambda(a\otimes b).(b_2\otimes a_2),\ \sigma_{B,A}:\lambda(b_2\otimes a_2).(a_3\otimes b_3)\}} a_3 \otimes b_3 : A \otimes B$$

These sequents are soup equivalent to $x : A \otimes B \vdash x : A \otimes B$, which means that the denotation used as a symmetry isomorphism is involutive:

$$[\sigma_{B,A}] \circ [\sigma_{A,B}] = [id_{A\otimes B}]$$

Moreover, if we consider $[\lambda_A] \circ [\sigma_{A,I}]$, the denotation corresponding to the sequent:

$$a : A, i : I \vdash_{\{\sigma_{A,I}:\lambda(a\otimes i).(i_2\otimes a_2),\lambda_A:\lambda(i_2\otimes a_2).a_3\}} a_3 : A$$

and $[\rho_A]$, the denotation corresponding to:

$$a : A, i : I \vdash_{\{\rho_A:\lambda(a\otimes i).a_3\}} a_3 : A$$

Since the two sequents are soup equivalent, we can say that:

$$[\rho_A] = [\lambda_A] \circ [\sigma_{A,I}]$$

Finally, consider the denotation $[\alpha_{C,A,B}] \circ [\sigma_{A\otimes B,C}] \circ [\alpha_{A,B,C}]$, which represents the soup equivalent sequents of:

$$a : A, b \otimes c : B \otimes C \vdash_{S_5} (c_3 \otimes a_3) \otimes b_3 : (C \otimes A) \otimes B$$

where the soup S_5 is:

$$\left\{ \begin{array}{ll} \alpha_{A,B,C} & : \lambda\left(a \otimes (b \otimes c)\right).\left((a_1 \otimes b_1) \otimes c_1\right), \\ \sigma_{A\otimes B,C} & : \lambda\left((a_1 \otimes b_1) \otimes c_1\right).\left(c_2 \otimes (a_2 \otimes b_2)\right), \\ \alpha_{C,A,B} & : \lambda\left(c_2 \otimes (a_2 \otimes b_2)\right).\left((c_3 \otimes a_3) \otimes b_3\right) \end{array} \right\}$$

and the denotation $([\sigma_{A,C}]\otimes[id_B])\circ[\alpha_{A,C,B}]\circ([id_A]\otimes[\sigma_{B,C}])$, which represents the soup equivalent sequents of:

$$a : A, b \otimes c : B \otimes C \vdash_{S_6} (c_3 \otimes a_3) \otimes b_3 : (C \otimes A) \otimes B$$

where the soup S_6 is:

$$\left\{ \begin{array}{ll} \sigma_{B,C} & : \lambda(b \otimes c).(c_1 \otimes b_1), \\ \alpha_{A,C,B} & : \lambda\left(a \otimes (c_1 \otimes b_1)\right).\left((a_2 \otimes c_2) \otimes b_3\right), \\ \sigma_{A,C} & : \lambda(a_2 \otimes c_2).(c_3 \otimes a_3) \end{array} \right\}$$

The two sequents are soup equivalent, so the denotations representing them are equal:

$$[\alpha_{C,A,B}] \circ [\sigma_{A\otimes B,C}] \circ [\alpha_{A,B,C}] = ([\sigma_{A,C}] \otimes [id_B]) \circ [\alpha_{A,C,B}] \circ ([id_A] \otimes [\sigma_{B,C}])$$

The syntactic category, therefore, satisfies all of the requirements and coherence conditions of a symmetric monoidal category. □

Theorem 9.3.4 (Compact closure)**.** *The syntactic category is a compact closed category*

Proof. Consider the denotation $[\lambda_A]\circ([\varepsilon_A]\otimes[id_A])\circ[\alpha_{A,A^*,A}]\circ([id_A]\otimes[\eta_A])\circ [\rho_A]^{-1}$, which represents the soup equivalent sequents of $x_1 : A \vdash_{S_7} x_8 : A$ where the soup S_7 is:

$$\left\{ \begin{array}{ll} \rho_A^{-1} & : \lambda x_1.(x_2 \otimes i_1), \\ \eta_A & : \lambda i_1.(x_{3*} \otimes x_4), \\ \alpha_{A,A^*,A} & : \lambda\left(x_2 \otimes (x_3 \otimes x_4)\right).\left((x_5 \otimes x_{6*}) \otimes x_7\right), \\ \varepsilon_A & : \lambda(x_5 \otimes x_{6*}).i_2, \\ \lambda_A & : \lambda(i_2 \otimes x_7).x_8 \end{array} \right\}$$

Using our soup reduction rules, we can reduce the above sequent to $x : A \vdash x : A$, hence showing that:

$$[\lambda_A] \circ ([\varepsilon_A] \otimes [id_A]) \circ [\alpha_{A,A^*,A}] \circ ([id_A] \otimes [\eta_A]) \circ [\rho_A]^{-1} = [id_A]$$

Now, let us consider $[\rho_{A^*}] \circ ([id_{A^*}] \otimes [\varepsilon_A]) \circ [\alpha_{A^*,A,A^*}]^{-1} \circ ([\eta_A] \otimes [id_{A^*}]) \circ [\lambda_{A^*}]^{-1}$; the denotation that corresponds to the soup equivalent sequents of $x_{1*} : A^* \vdash_{S_8} x_{8*} : A^*$, where the soup S_8 is:

$$\left\{ \begin{array}{ll} \lambda_{A^*}^{-1} & : \lambda x_{1*}.(i_1 \otimes x_{2*}), \\ \eta_A & : \lambda i_1.(x_{3*} \otimes x_4), \\ \alpha_{A^*,A,A^*}^{-1} & : \lambda\left((x_{3*} \otimes x_4) \otimes x_{2*}\right).\left(x_{5*} \otimes (x_6 \otimes x_7)\right), \\ \varepsilon_A & : \lambda(x_6 \otimes x_{7*}).i_2, \\ \rho_{A*} & : \lambda(x_{5*} \otimes i_2).x_{8*} \end{array} \right\}$$

Using our soup reduction rules, we can reduce the above sequent to $x_* : A^* \vdash x_* : A^*$, hence showing that:

$$[\rho_{A^*}] \circ ([id_{A^*}] \otimes [\varepsilon_A]) \circ [\alpha_{A^*,A,A^*}]^{-1} \circ ([\eta_A] \otimes [id_{A^*}]) \circ [\lambda_{A^*}]^{-1} = [id_{A^*}]$$

The syntactic category thus satisfies both of the yanking conditions that are required of a compact closed category. □

Theorem 9.3.5 (Dagger compact closure)**.** *The syntactic category is a dagger compact category*

Proof. Let $[f] : A \to B$ be a denotation representing the soup equivalent sequents of $a : A \vdash_{S_9} b : B$. For every such denotation $[f]$, we define its dagger $[f]^\dagger : B \to A$ such that it represents the soup equivalent sequents of the †-flipped version of the original sequent: $b : B \vdash_{S_{9*}} a : A$. The †-flip rule, however, is involutive since $(S_{9*})_* = S_9$, hence $([f]^\dagger)^\dagger = f$.

Now consider the denotation $[\sigma_{A,A^*}] \circ [\varepsilon_A]^\dagger$, which represents the soup equivalent sequents of:

$$i : I \vdash_{S_{10}} x_{3*} \otimes x_4 : A^* \otimes A$$

where the soup S_{10} is:

$$\left\{ \begin{array}{ll} \varepsilon_{A*} & : (\lambda(x_1 \otimes x_{2*}).1)_*, \\ \sigma_{A,A^*} & : \lambda(x_1 \otimes x_{2*}).(x_{3*} \otimes x_4) \end{array} \right\}$$

By using soup reduction, we get $S_{10} \to \{\eta_A : \lambda i.(x_{3*} \otimes x_4)\}$. But the sequent $i : I \vdash_{\{\eta_A : \lambda i.(x_{3*} \otimes x_4)\}} x_{3*} \otimes x_4 : A^* \otimes A$ is represented by the denotation $[\eta_A]$, which means that

$$[\sigma_{A,A^*}] \circ [\varepsilon_A]^\dagger = [\eta_A]$$

The syntactic category, therefore, satisfies all of the requirements of a dagger compact category. □

9.3.5 Proof of equivalence

We will now prove that the free dagger compact category $\mathcal{C}_{Free}$ is equivalent to the syntactic category $\mathcal{C}_{Synt}$.

Lemma 9.3.1 (Essentially surjective on objects)**.** *The set of objects in the free category and the set of objects in the syntactic category are surjective, up to isomorphism.*

Proof. Recall $Dagger(\Sigma_0)$; the free $(\otimes, I, \Box^*)$-algebra over the set of object variables Σ_0. The sets of objects in $\mathcal{C}_{Free}$ and $\mathcal{C}_{Synt}$ both correspond to $Dagger(\Sigma_0)$, up to the equivalence classes induced by $(A^*)^* \equiv A$ and $I^* \equiv I$. □

Lemma 9.3.2 (Equal arrows correspond to equal denotations)**.** *If two arrows, $\langle A, f, B\rangle$ and $\langle A, f', B\rangle$ are equal in the free category, then they will also be equal in the syntactic category: $[f] = [f'] : A \to B$.*

Proof. The structure of the free category $\mathcal{C}_{Free}$ imposes the minimum number of equalities for a category to be dagger compact. Moreover, both the free category and the syntactic category derive their symbols from the same signature graph $\mathcal{G}$. Since we have already shown that $\mathcal{C}_{Synt}$ is dagger compact, the same steps can be used to show that any arrows $\langle A, f, B\rangle$ and $\langle A, f', B\rangle$ that are equal in the free category, correspond to equal denotations $[f] = [g]$ in the syntactic category. □

Lemma 9.3.3 (Equal denotations correspond to equal arrows)**.** *Any denotations that are equal in the syntactic category, correspond to equal arrows in the free category.*

Proof. Let $[f] : \Gamma \to B$ and $[g] : \Gamma \to B$ be denotations in the syntactic category such that $[f] = [g]$. Since the two denotations are equal, the sequents they represent in the dagger lambda calculus must be equivalent up to soup reduction. Without loss of generality, lets assume that $[f]$ represents a sequent J_1 and that $[g]$ represents a sequent J_2, where $J_1 \to J_2$. The soup reduction relation consists of four soup rules: *bifunctoriality*, *trace*, *cancellation* and *consumption*. We prove this theorem by induction on the structure of the soup reduction linking J_1 and J_2:

- If we use a *bifunctoriality* rule, then we will be reducing a sequent of the form $\Gamma \vdash_{S \cup \{x_1 \otimes x_2 : x_3 \otimes x_4\}} b : B$ to one of the form $\Gamma \vdash_{S \cup \{x_1 : x_3, x_2 : x_4\}} b : B$. By carefully separating the appropriate connections, we can break down $[g]$ into $([\pi_2] \circ [\pi_1]) \otimes ([\pi_4] \circ [\pi_3])$ where: $[\pi_1]$ represents $\Gamma_1 \vdash_{S_1} x_1 :$

T_1; $[\pi_2]$ represents $x_3 : T_1 \vdash_{S_2} b_1 : B_1$; $[\pi_3]$ represents $\Gamma_2 \vdash_{S_3} x_2 : T_2$; $[\pi_4]$ represents $x_4 : T_2 \vdash_{S_4} b_2 : B_2$; and $\Gamma = \Gamma_1, \Gamma_2$, $S = S_1 \cup S_2 \cup S_3 \cup S_4$, $b = b_1 \otimes b_2$ and $B = B_1 \otimes B_2$. The individual $[\pi_i]$ denotations can be reconstructed in a different way to form $([\pi_2] \otimes [\pi_4]) \circ ([\pi_1] \otimes [\pi_3])$, which actually forms $[f]$. But $\langle \Gamma_1, \pi_1, T_1, \pi_2, B_1 \rangle \otimes \langle \Gamma_2, \pi_3, T_2, \pi_4, B_2 \rangle = \langle \Gamma, \pi_1 \otimes \pi_3, T_1 \otimes T_2, \pi_2 \otimes \pi_4, B \rangle$ because $\mathcal{C}_{Free}$ is a *dagger compact category*, so $\langle \Gamma, f, B \rangle = \langle \Gamma, g, B \rangle$.

- If we use a *trace* rule on J_1, we do not in any way affect the information that is contained in the soup connection, we are simply rewriting it using different notation, so we are in no way affecting the derivation of J_2 from sequents represented by smaller denotations.

- If we use a *cancellation* rule on J_1, the information contained on the connection we are striking out is nil, so we are in no way affecting the derivation of J_2 from sequents represented by smaller denotations. In this case, $[f]$ represents $\langle \Gamma, f, B \rangle = \langle \Gamma, g, B \rangle \otimes 1$ in the free category, which is the same as $\langle \Gamma, g, B \rangle$.

- If we use a *consumption* rule on J_1, then the bound variable we are substituting for will either appear to the left or to the right of the turnstile, or it will appear elsewhere in the soup.

  - If the bound variable appears to the *left* of the turnstile then the general form of the sequent J_1 can be written as $t_1 : T_1, x : T_2, t_3 : T_3 \vdash_{S \cup \{x:t_2\}} b : B$. Let $\Gamma = T_1 \otimes T_2 \otimes T_3$. Since J_2 is the result of consuming a soup connection that was created by performing a Cut with the identity, the sequent can be written as $t_1 : T_1, t_2 : T_2, t_3 : T_3 \vdash_S b : B$. This means that $[f] = [g] \circ [id_\Gamma]$, which causes $\langle \Gamma, f, B \rangle = \langle \Gamma, g, B \rangle \circ \langle \Gamma \rangle = \langle \Gamma, g, B \rangle$.
  - If the bound variable appears to the *right* of the turnstile then the general form of the sequent J_1 can be written as $\Gamma \vdash_{S \cup \{t_2:x\}} t_1 \otimes x \otimes t_3 : B$. Since J_2 is the result of consuming a soup connection that was created by performing a Cut with the identity, the sequent can be written as $\Gamma \vdash_S t_1 \otimes t_2 \otimes t_3 : B$. This means that $[f] = [id_B] \circ [g]$, which causes $\langle \Gamma, f, B \rangle = \langle B \rangle \circ \langle \Gamma, g, B \rangle = \langle \Gamma, g, B \rangle$.
  - If the bound variable appears *elsewhere in the soup*, then J_1 will be the result of applying the Cut rule twice on J_2, each time with an instance of the identity. The denotation representing J_1 will

either be $[f] = ([id_B] \circ [id_B]) \circ [g]$ or $[f] = [g] \circ ([id_\Gamma] \circ [id_\Gamma])$, both of which cause $\langle \Gamma, f, B \rangle = \langle \Gamma, g, B \rangle$.

We have, therefore, shown that in all cases, $\langle \Gamma, f, B \rangle = \langle \Gamma, g, B \rangle$. □

Theorem 9.3.6 (Equivalence between the free category and the syntactic category)**.** *The free dagger compact category $\mathcal{C}_{Free}$ and the syntactic category $\mathcal{C}_{Synt}$ are equivalent.*

Proof. The two categories derive their symbols from a common signature graph $\mathcal{G}$. As we have already shown, bearing in mind the equivalence classes that we have induced on types, the categories are essentially surjective on objects. Moreover, arrows that are equal in the free category are equal in the syntactic category and vice versa. This means that the functor F is *full* and *faithful*, causing the notions of equality between arrows overlap in these two categories. Consequently, the categories are equivalent. □

Corollary 9.3.1 (Internal language)**.** *The dagger lambda calculus is an internal language for dagger compact categories.*

Chapter 10

Classical control in the $\dagger\lambda$-calculus

In this section we will see how the dagger lambda calculus can be imbued with classical control structures, similar to the ones in [CD11]. We will begin by defining classical structures within the dagger lambda calculus, hence allowing it to support the Frobenius algebras of [CP06], [CP07] and [CPP10]. We will then demonstrate how our extension of the lambda calculus can be used to represent the notion of dualiser that was introduced in [CPP08], as well as how this notion now makes the Currying rule admissible in our language. Finally, we will further extend the dagger lambda calculus with complementary classical structures, by defining a notion of complementarity that extends that of [CD11], with the dualisers of [CPP08], to allow for non self-dual †-compact structures.

Throughout this entire section, we will be using a version of the dagger lambda calculus that has been modified to simplify our notation. In order to do this, we will be restricting our attention to *strict dagger compact categories*; categories whose monoidal natural isomorphisms, $\alpha_{A,B,C}$, λ_Γ and ρ_Γ, are identities. As a result, instances of the $\otimes L$, λ_Γ and ρ_Γ rules will be equated with the identity, allowing us to be more relaxed when it comes to tensor identities and parenthesising terms.

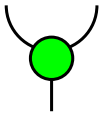
10.1 Classical structures

As we have seen in chapter 3 and section 6.2, of part II of this dissertation, the classical world is a lot less "restrictive" than the quantum world in that it allows us to freely copy and delete data. Classical states can be thought

of as a basis that spans a vector space of quantum states and it is in this subset of the quantum world that the linearity restrictions can be relaxed.

Linear logic [Gir87] achieves its resource sensitivity by dropping the rules of weakening and contraction. In place of those rules, Girard introduced the exponential connective $!A$, denoting an infinite supply of the type A, as a more tightly controlled way of breaching linearity. In designing the dagger lambda calculus, we have also dropped the rules of weakening and contraction. Instead of replacing them, however, with an exponential connective, we will extend our language by providing a stricter and yet more controlled way of relaxing resource sensitivity. This will be done by supporting the Frobenius algebras of [CP06], [CP07] and [CPP10] with copying and deleting maps and will allow us to model the behaviour of a classical basis.

For every type A in the dagger lambda calculus, we will be introducing two constants; $G_1^2 : A \multimap A \otimes A$ and $G_1^0 : A \multimap I$. This allows us to represent the copying and deleting operations as sequents in our language:



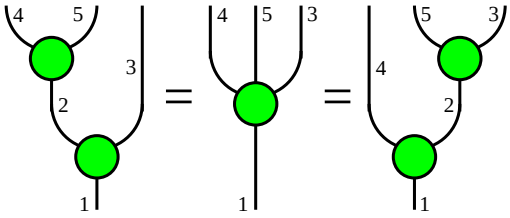
$$x_1 : A \vdash_{\{G_1^2 : x_{1*} \otimes x_2 \otimes x_3\}} x_2 \otimes x_3 : A \otimes A$$

$$\vdash G_1^0 : A^*$$

Definition 10.1.1 (Notational conventions for classical structures in the dagger lambda calculus)**.** For notational convenience, we will use $A^{\otimes n}$ as shorthand for $\underbrace{A \otimes \ldots \otimes A}_{n \text{ times}}$. We also define G_1^1, G_2^1 and G_0^1 as follows:

$$G_1^1 := id_A, \qquad G_2^1 := (G_1^2)_*, \qquad G_0^1 := (G_1^0)_*$$

We will define the copying and deleting maps by describing all of the conditions that we require them to satisfy in the dagger lambda calculus. The first such condition is the comonoidal coassociativity condition:



which requires the following two soups, S_1 and S_2, to be equal in the lambda calculus:

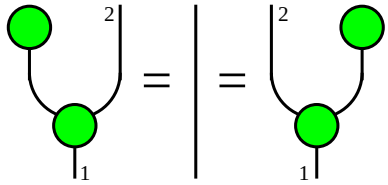
$$S_1 = \{G_1^2 : x_{1*} \otimes x_2 \otimes x_3, G_1^2 : x_{2*} \otimes x_4 \otimes x_5\}$$

$$S_2 = \{G_1^2 : x_{1*} \otimes x_4 \otimes x_2, G_1^2 : x_{2*} \otimes x_5 \otimes x_3\}$$

This allows us to relate the sequents for $x_1 : A \vdash_{S_1} x_4 \otimes x_5 \otimes x_3 : A^{\otimes 3}$ and $x_1 : A \vdash_{S_2} x_4 \otimes x_5 \otimes x_3 : A^{\otimes 3}$, matching the equality of arrows that is required in the categorical setting. Since it does not matter whether we copy one or the other part of a copied pair, we can write both cases as:

$$x_1 : A \vdash_{\{G_1^3 : x_{1*} \otimes x_4 \otimes x_5 \otimes x_3\}} x_4 \otimes x_5 \otimes x_3 : A^{\otimes 3}$$

The next condition we require of the dagger lambda calculus is the comonoidal identity condition:



which requires the following soups, S_3, S_4 and S_5, to be equal to each other:

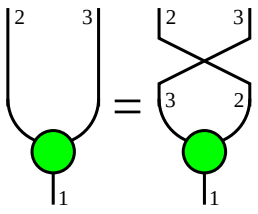
$$S_3 = \{G_1^2 : x_{1*} \otimes G_0^1 \otimes x_2\}$$

$$S_4 = \{G_1^1 : x_{1*} \otimes x_2\} \longrightarrow \{x_1 : x_2\}$$

$$S_5 = \{G_1^2 : x_{1*} \otimes x_2 \otimes G_0^1\}$$

This allows us to relate the sequents for $x_1 : A \vdash_{S_3} x_2 : A$, $x_1 : A \vdash_{S_4} x_2 : A$ and $x_1 : A \vdash_{S_5} x_2 : A$, matching the equality of arrows that is required in the categorical setting, to get an identity sequent $x : A \vdash x : A$.

We will also require the copy map to satisfy a cocommutativity requirement:



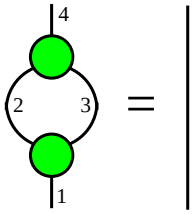
which requires the following soups, S_6 and S_7, to be equal in the dagger lambda calculus:

$$S_6 = \{G_1^2 : x_{1*} \otimes x_2 \otimes x_3\}$$

$$S_7 = \{G_1^2 : x_{1*} \otimes x_3 \otimes x_2\}$$

This allows us to relate the sequents for $x_1 : A \vdash_{S_6} x_2 \otimes x_3 : A \otimes A$ and $x_1 : A \vdash_{S_7} x_2 \otimes x_3 : A \otimes A$, matching the equality of arrows that is required in the categorical setting.

Since all spider monoids have to be *special*, we require the copying map to satisfy the isometry condition:



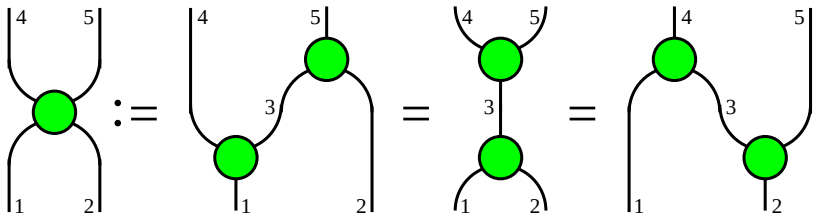
which requires the following soups to be equal:

$$S_8 = \quad \left\{G_1^2 : x_{1*} \otimes x_2 \otimes x_3,\ G_2^1 : (x_2 \otimes x_3)_* \otimes x_4\right\}$$

$$S_9 = \quad \{G_1^1 : x_{1*} \otimes x_4\} \longrightarrow \{x_1 : x_4\}$$

and allows us to match $x_1 : A \vdash_{S_8} x_4 : A$ with $x_1 : A \vdash_{S_9} x_4 : A$ and the identity sequent $x : A \vdash x : A$.

Finally, we require the copying and deleting maps to satisfy the Frobenius condition:



which requires the following soups to be equal to each other in the dagger lambda calculus:

$$S_{10} = \{G_1^2 : x_{1*} \otimes x_4 \otimes x_3, G_2^1 : (x_3 \otimes x_2)_* \otimes x_5\}$$

$$S_{11} = \{G_2^1 : (x_1 \otimes x_2)_* \otimes x_3, G_1^2 : x_{3*} \otimes x_4 \otimes x_5\}$$

$$S_{12} = \{G_1^2 : x_{2*} \otimes x_3 \otimes x_5, G_2^1 : (x_1 \otimes x_3)_* \otimes x_4\}$$

and allows us to relate the sequents for

$$x_1 : A, x_2 : A \vdash_{S_{10}} x_4 \otimes x_5 : A \otimes A,$$
$$x_1 : A, x_2 : A \vdash_{S_{11}} x_4 \otimes x_5 : A \otimes A$$
$$\text{and } x_1 : A, x_2 : A \vdash_{S_{12}} x_4 \otimes x_5 : A \otimes A$$

matching the equality of arrows that is required in the categorical setting.

All of these soup equality conditions can be rewritten as equalities between terms to make them more readable. The following table lists the term equalities corresponding to each of the conditions:

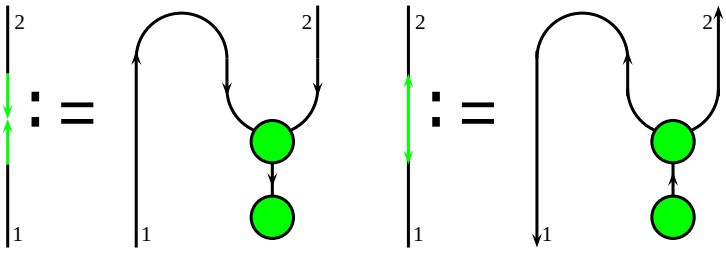
Comonoidal associativity	$\bar{b}(\bar{t}G_1^2 id_A)G_1^2 =$ $\bar{b}(\bar{t}id_A G_1^2)G_1^2$
Comonoidal identity	$\bar{b}(\bar{t}G_1^0 id_A)G_1^2 =$ $G_1^1 =$ $\bar{b}(\bar{t}id_A G_1^0)G_1^2$
Cocommutativity	$G_1^2 =$ $\bar{b}\sigma_{A,A}G_1^2$
Isometry	$\bar{b}G_2^1 G_1^2 =$ G_1^1
Frobenius	$\bar{b}(\bar{t}id_A G_2^1)(\bar{t}G_1^2 id_A) =$ $\bar{b}G_1^2 G_2^1 =$ $\bar{b}(\bar{t}G_2^1 id_A)(\bar{t}id_A G_1^2)$

10.2 Dualisers

In this subsection, we will demonstrate how the *dagger lambda calculus* with *classical structures* can be used to present the notion of a dualiser [CPP08]; an explicit witness of the passage from one object to its dual or, in our case, from one type to its linear negation. While [CPP10] and [CD11] do not use this form of dualiser, we consider them very important from a programming language perspective as they allow us to differentiate inputs from outputs and make the flow of information explicit in the language. After defining the dualiser, the rest of the subsection will show how this notion makes the Currying rule admissible in our language.

Definition 10.2.1 (Dualiser)**.** For every type A, we define a constant $d_A : A \multimap A^*$, the *dualiser*, as a shorthand for $G_2^0 : A^* \otimes A^*$. In other words, the dualiser will be $d_A := (G_0^2)_* = G_2^0 : A^* \otimes A^*$. The sequents $[d_A]$ and $[d_A]^\dagger$ that represent the η-expanded form of the dualiser and its dual are:

$$x_1 : A \vdash_{\{d_A : x_{1*} \otimes x_{2*}\}} x_{2*} : A^* \qquad x_{2*} : A^* \vdash_{\{d_{A*} : x_2 \otimes x_1\}} x_1 : A$$



Theorem 10.2.1 (Unitarity of the dualiser)**.** *The sequent representing the η-expanded form of the dualiser is unitary. In other words, we can compose a dualiser sequent $[d_A]$ with its dagger $[d_A]^\dagger$, via a Cut, and the result will be an identity sequent:*

$$\dfrac{x_1 : A \vdash_{\{d_A : x_{1*} \otimes x_{2*}\}} x_{2*} : A^* \qquad \dfrac{x_3 : A \vdash_{\{d_A : x_{3*} \otimes x_{2*}\}} x_{2*} : A^*}{x_{2*} : A^* \vdash_{\{d_{A*} : x_2 \otimes x_3\}} x_3 : A}\ \dagger\text{-}\mathit{flip}}{x_1 : A \vdash_{\{d_A : x_{1*} \otimes x_{2*}, d_{A*} : x_2 \otimes x_3\}} x_3 : A}\ \mathit{Cut}$$

whose soup is equal to $\{G_1^1 : x_{1*} \otimes x_3\}$.

Proof. Consider the following soup reduction steps:

$$\begin{aligned}
\{d_A : x_{1*} \otimes x_{2*}, d_{A*} : x_2 \otimes x_3\} &\to \\
\{G_2^0 : x_{1*} \otimes x_{2*}, G_0^2 : x_2 \otimes x_3\} &\to \\
\{G_2^1 : (x_1 \otimes x_2)_* \otimes G_0^1, G_1^2 : (G_0^1)_* \otimes x_2 \otimes x_3\} &\to \text{(Frobenius)} \\
\{G_2^1 : (x_1 \otimes G_0^1)_* \otimes x_2, G_1^2 : x_{2*} \otimes G_0^1 \otimes x_3\} &\to \text{(Identity)} \\
\{G_1^1 : x_{1*} \otimes x_2, G_1^1 : x_{2*} \otimes x_3\} &\to \\
\{x_1 : x_2, x_2 : x_3\} &\to \\
\{x_1 : x_3\}
\end{aligned}$$

□

Having properly defined dualisers in the dagger lambda calculus, we proceed to show how classical structures and dualisers can be used to reconstruct the Currying rule.

Theorem 10.2.2 (Admissibility of the Currying rule)**.** *The Currying rule is admissible in a dagger lambda calculus with classical structures and dualisers.*

Proof. We can construct a big sequent following the steps outlined in this proof tree:

$$\dfrac{\dfrac{\vdash_{\{G_0^2:x_2\otimes x_3\}} x_2\otimes x_3 : A\otimes A}{\vdash_{\{G_1^2:(G_0^1)_*\otimes x_2\otimes x_3\}} x_2\otimes x_3 : A\otimes A} \qquad \dfrac{\dfrac{\dfrac{x_2 : A \vdash_{\{d_A:x_{2*}\otimes x_{1*}\}} x_{1*} : A^*}{x_2 : A \vdash_{\{G_2^0:x_{2*}\otimes x_{1*}\}} x_{1*} : A^*}}{x_2 : A \vdash_{\{G_2^1:x_{2*}\otimes x_{1*}\otimes G_0^1\}} x_{1*} : A^*} \qquad x_3 : A \vdash x_3 : A}{x_2 : A, x_3 : A \vdash_{\{G_2^1:x_{2*}\otimes x_{1*}\otimes G_0^1\}} x_{1*}\otimes x_3 : A^*\otimes A}}{\vdash_{\{G_1^2:(G_0^1)_*\otimes x_2\otimes x_3, G_2^1:(x_1\otimes x_2)_*\otimes G_0^1\}} x_{1*}\otimes x_3 : A^*\otimes A}$$

We can then use the following reduction steps on the resulting sequent's soup:

$$\begin{aligned}\{G_1^2 : (G_0^1)_*\otimes x_2\otimes x_3, G_2^1 : (x_1\otimes x_2)_*\otimes G_0^1\} &\to (Frobenius)\\ \{G_2^1 : (x_1\otimes G_0^1)_*\otimes x_2, G_1^2 : x_{2*}\otimes G_0^1\otimes x_3\} &\to (Identity)\\ \{G_1^1 : x_{1*}\otimes x_2, G_1^1 : x_{2*}\otimes x_3\} &\to\\ \{x_1 : x_2, x_2 : x_3\} &\to\\ \{x_1 : x_3\}&\end{aligned}$$

This gives us $\vdash x_*\otimes x : A^*\otimes A$, which is called a *cup*. Once we have this term, we can Cut it with any sequent of the form $a : A \vdash_S b : B$ to reconstruct the Currying rule:

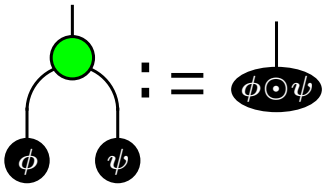
$$\dfrac{\dfrac{\vdash x_*\otimes x : A^*\otimes A \qquad \dfrac{x_{1*} : A^* \vdash x_{1*} : A^* \qquad a : A \vdash_S b : B}{x_{1*} : A^*, a : A \vdash_S x_{1*}\otimes b : A^*\otimes B}\otimes R}{\vdash_{S\cup\{x_*\otimes x:x_{1*}\otimes a\}} x_{1*}\otimes b : A^*\otimes B}}{\vdash_S a_*\otimes b : A^*\otimes B}$$

□

10.3 Monoidal product of terms and phase shifts

This section shows how the monoidal operation defined by G_2^1 can be used to fuse together the terms of the dagger lambda calculus, or lift them into *phase shifts* in a sequent. This usage of the monoidal operation, described in [CD08] and [CD11], makes our language more expressive by allowing us to introduce rotations on quantum states.

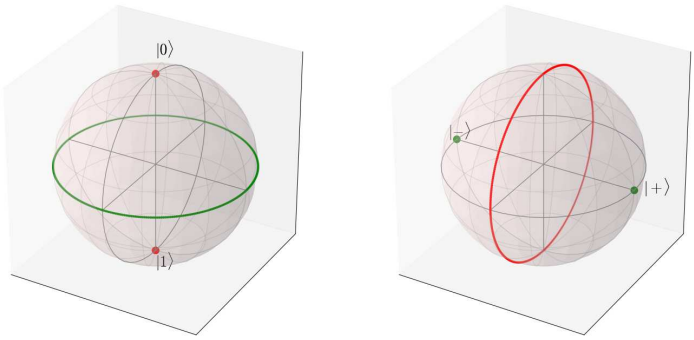
Definition 10.3.1 (Monoidal product)**.** We define the notation $G[\phi \odot \psi]_0^1 : A$ to represent the product generated by the action of the monoidal operation $G_2^1 : A \otimes A \multimap A$ on two terms ϕ and ψ. More specifically, we set $G[\phi \odot \psi]_0^1 := G_2^1(\phi \otimes \psi)$.



Corollary 10.3.1 (Associativity and commutativity of $\odot$)**.** *The $\odot$ operator inherits associativity and commutativity from the monoidal associativity and commutativity conditions of G_2^1. This allows us to introduce the following notation for the fusion of multiple terms:*

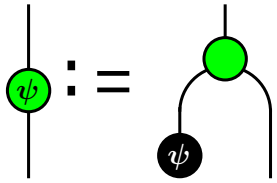
$$\bigodot \psi_i := \psi_1 \odot \ldots \odot \psi_n$$

Given a classical structure, we now define a way of performing rotations against the axis defined by that observable structure. These rotations, also known as *phase shifts*, are more easily visualised as rotations of a qubit's vector in the Bloch sphere [JNN12]:



Definition 10.3.2 (Phase shift)**.** The *phase shift* generated by the action of G_2^1 on a term ψ is represented by the η-expanded form of the term $G[\psi]_1^1 := \lambda x.G_2^1(\psi \otimes x)$:

$$x_1 : A \vdash_{\{G[\psi]_1^1 : x_{1*} \otimes x_2\}} x_2 : A \quad \to x : A \vdash G[\psi \otimes x]_0^1 : A$$



The composition, under Cut, of the phase shifts generated by two terms ϕ and ψ is:

$$\dfrac{\dfrac{x_1 : A \vdash_{\{G[\phi]^1_1 : x_{1*} \otimes x_2\}} x_2 : A \qquad x_2 : A \vdash_{\{G[\psi]^1_1 : x_{2*} \otimes x_3\}} x_3 : A}{x_1 : A \vdash_{\{G^1_2 : (\phi \otimes x_1)_* \otimes x_2, G^1_2 : (\psi \otimes x_2)_* \otimes x_3\}} x_3 : A}\,\text{Cut}}{x_1 : A \vdash_{\{G^1_2 : (\phi \otimes \psi)_* \otimes x_2, G^1_2 : (x_2 \otimes x_1)_* \otimes x_3\}} x_3 : A}\,\text{Monoidal associativity}$$

The resulting sequent can be rewritten as

$$x_1 : A \vdash_{\{G^1_2 : (G[\phi \odot \psi]^1_0 \otimes x_1)_* \otimes x_3\}} x_3 : A$$

Which is actually a lifting of the monoidal product of those terms

$$x_1 : A \vdash_{\{G[G[\phi \odot \psi]^1_0]^1_1 : x_{1*} \otimes x_3\}} x_3 : A$$

Definition 10.3.3 (Lifting of the monoidal product of terms)**.** We define a shorthand for the lifting of the monoidal product of terms:

$$G[\phi \odot \psi]^1_1 := G[G[\phi \odot \psi]^1_0]^1_1$$

Corollary 10.3.2 (Phase shift commutativity)**.** *Phase shifts generated by the action of G^1_2 are commutative under Cut:*

$$G[\phi \odot \psi]^1_1 = G[\psi \odot \phi]^1_1$$

Proof. The composition of the phase shift corresponding to a term ϕ, with that of a term ψ, produces:

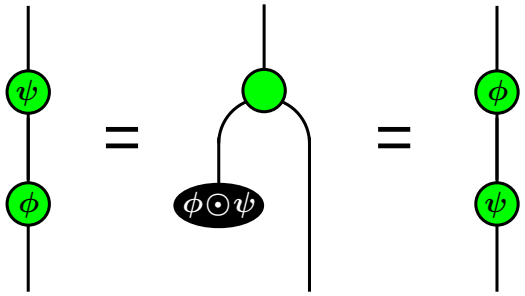
$$x_1 : A \vdash_{\{G^1_2 : (\phi \otimes \psi)_* \otimes x_2, G^1_2 : (x_2 \otimes x_1)_* \otimes x_3\}} x_3 : A$$

Composing the phase shifts the other way around results in:

$$x_1 : A \vdash_{\{G^1_2 : (\psi \otimes \phi)_* \otimes x_2, G^1_2 : (x_2 \otimes x_1)_* \otimes x_3\}} x_3 : A$$

The monoidal commutativity condition allows us to transform $G^1_2 : (\phi \otimes \psi)_* \otimes x_2$ into $G^1_2 : (\psi \otimes \phi)_* \otimes x_2$, in our soup, which makes the two sequents equal.

It therefore follows that the resulting sequents, $G[\phi \odot \psi]^1_1$ and $G[\psi \odot \phi]^1_1$, are also equal.



□

10.4 Unbiased and classical constants

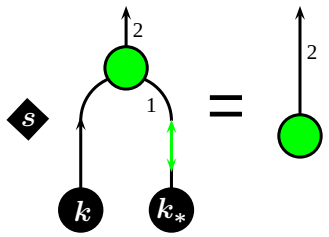
This section establishes the properties of unbiasedness and classicality for constants, with respect to a given observable structure. The notion of unbiasedness differs slightly from the one presented in [CD08] and [CD11], as it uses the dualisers of [CPP08] to allow for non self-dual structures. The properties of unbiasedness and classicality will later be used in defining the interaction between complementary observables.

Definition 10.4.1 (Unbiasedness (dagger lambda calculus))**.** We say that a constant $k : A$ is *unbiased* with respect to an observable structure (A, G^2_1, G^0_1) when there exists a scalar $s : I$ such that we can match the sequents:

$$\vdash_{\{d_{A*}:k\otimes x_1, G^1_2:(k\otimes x_1)_*\otimes x_2, s:1\}} x_2 : A \text{ and } \vdash_{\{G^1_0:x_2\}} x_2 : A$$

by equating their soups:

$$\{d_{A*} : k \otimes x_1, G^1_2 : (k \otimes x_1)_* \otimes x_2, s : 1\} = \{G^1_0 : x_2\}$$



Intuitively, a constant is said to be *classical*, with respect to a given observable structure, when it is copied and deleted by that structure. More formally, this is defined as:

Definition 10.4.2 (Classicality (dagger lambda calculus))**.** We say that a constant $k : A$ is *classical*, with respect to an observable structure (A, G_1^2, G_1^0), when we can match the following pairs of sequents:

$$\vdash_{\{G_1^2 : k_* \otimes x_1 \otimes x_2\}} x_1 \otimes x_2 : A \otimes A \text{ with } \vdash_{\{k:x_1, k:x_2\}} x_1 \otimes x_2 : A \otimes A$$

and

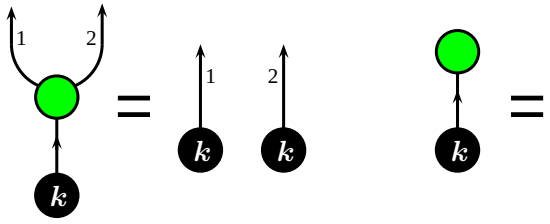
$$\vdash_{\{G_1^0 : k_*\}} \text{ with } \vdash$$

by equating their respective soups:

$$\{G_1^2 : k_* \otimes x_1 \otimes x_2\} = \{k : x_1, k : x_2\}$$

and

$$\{G_1^0 : k_*\} = \emptyset$$

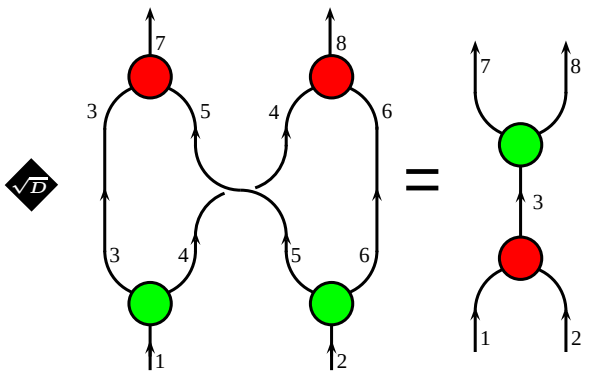


10.5 Complementary observables

This section introduces two interacting observable structures to our lambda calculus, a Green one and a Red one, as the final step towards axiomatising basis structures in the dagger lambda calculus. The two observable structures are similar to the ones described in the later parts of [CD11], with the added support for non self-dual †-compact structures and the dualisers of [CPP08]. The interaction between the two observable structures is defined by requiring that they satisfy a *Bi-algebra* and a *Hopf law* condition. We then show that the interaction between Green and Red makes them *complementary*, as they possess certain properties with regards to classical and unbiased constants. Finally, we define a function that can be used to transform the Green sequents into Red ones and vice versa.

Definition 10.5.1 (Complementary observable structures (dagger lambda calculus))**.** For every type A in the dagger lambda calculus, we define two observable structures: A *Green* structure (A, G_1^2, G_1^0) and a *Red* structure (A, R_1^2, R_1^0). Each of these observable structures comes with a dualiser; we will use $d_A^Z : A \multimap A^*$ to refer to the dualiser generated by the Green observable structure and $d_A^X : A \multimap A^*$ to refer to the one generated by Red.

We define the interaction between these two observable structures by describing the conditions that we require them to satisfy in the dagger lambda calculus. The first such condition is the *Bi-algebra* condition:



which requires the following two soups, S_{13} and S_{14}, to be equal in the lambda calculus:

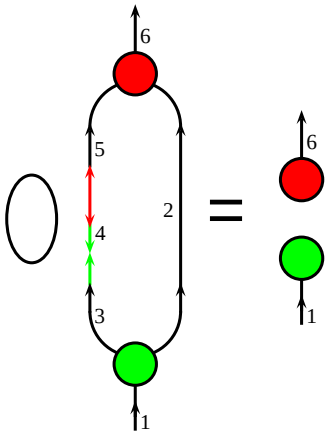
$$S_{13} = \left\{ \begin{array}{ll} G_1^2 : x_{1*} \otimes x_3 \otimes x_4, & G_1^2 : x_{2*} \otimes x_5 \otimes x_6, \\ R_2^1 : (x_3 \otimes x_5)_* \otimes x_7, & R_2^1 : (x_4 \otimes x_6)_* \otimes x_8, \\ \sqrt{D} : 1 & \end{array} \right\}$$

$$S_{14} = \left\{ R_2^1 : (x_1 \otimes x_2)_* \otimes x_3, G_1^2 : x_{3*} \otimes x_7 \otimes x_8 \right\}$$

This allows us to relate the sequents for $x_1 : A, x_2 : A \vdash_{S_{13}} x_7 \otimes x_8 : A \otimes A$ and $x_1 : A, x_2 : A \vdash_{S_{14}} x_7 \otimes x_8 : A \otimes A$, matching the equality of arrows that is required in the categorical setting.

The next condition that we require of the interaction between the Green and Red observable structures in the dagger lambda calculus is the *Hopf law*

condition:



which requires the following two soups, S_{15} and S_{16}, to be equal in the lambda calculus:

$$S_{15} = \left\{ \begin{array}{ll} G_1^2 : x_{1*} \otimes x_3 \otimes x_2, & R_2^1 : (x_5 \otimes x_2)_* \otimes x_6, \\ d_A^Z : x_{3*} \otimes x_{4*}, & d_{A^*}^X : x_4 \otimes x_5, \\ D : 1 & \end{array} \right\}$$

$$S_{16} = \left\{ G_1^0 : x_{1*}, R_0^1 : x_6 \right\}$$

This allows us to relate the sequents for $x_1 : A \vdash_{S_{15}} x_6 : A$ and $x_1 : A \vdash_{S_{16}} x_6 : A$, or in other words $x_1 : A \vdash_{S_{15}} x_6 : A$ with $G_0^1 : A \vdash R_0^1 : A$, matching the equality of arrows that is required in the categorical setting.

We will now show an interesting property about the interaction that we have defined against the two observable structures. When a constant is classical against the Green structure, it is unbiased against the Red one. Conversely, when a constant is classical against the Red structure, it is unbiased against the Green one.

Theorem 10.5.1 (Complementarity)**.** *Let $k : A$ be a constant such that $\{G_1^2 : k_* \otimes x_1 \otimes x_2\} = \{k : x_1, k : x_2\}$ and $\{G_1^0 : k_*\}$. (1) The interaction between the two observable structures causes $\{d_{A^*}^X : k \otimes x_5, R_2^1 : (x_5 \otimes k)_* \otimes x_6, D : 1\}$ to be equal to $\{R_0^1 : x_6\}$. Similarly, let $\ell : A$ be a constant such that $\{R_1^2 : \ell_* \otimes x_1 \otimes x_2\} = \{\ell : x_1, \ell : x_2\}$ and $\{R_1^0 : \ell_*\}$. (2) The interaction causes $\{d_{A^*}^Z : \ell \otimes x_5, G_2^1 : (x_5 \otimes \ell)_* \otimes x_6, D : 1\}$ to be equal to $\{G_0^1 : x_6\}$.*

Proof. (1) We begin by trying to *fuse* together k_* and k, in a manner that is similar to the definition of unbiasedness. We, therefore, start with the

following set of connections in our soup:

$$\{d^X_{A^*} : k \otimes x_5, R^1_2 : (x_5 \otimes k)_* \otimes x_6, D : 1\}$$

By the definition of dualisers and since k is classical under the Green observable structure, we can use the explicit witness of the passage between k and k_* to produce:

$$\{d^Z_A : k_* \otimes x_{4*}, d^X_{A^*} : x_4 \otimes x_5, R^1_2 : (x_5 \otimes k)_* \otimes x_6, D : 1\}$$

Since k is classical under the Green observable structure, we can replace the two k's with a Green copy map, operating on a single k:

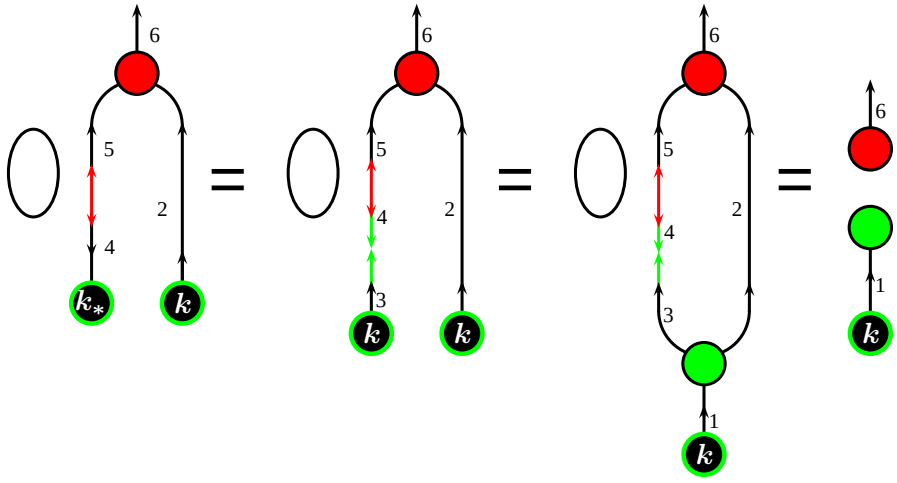
$$\{G^2_1 : k_* \otimes x_3 \otimes x_2, d^Z_A : x_{3*} \otimes x_{4*}, d^X_{A^*} : x_4 \otimes x_5, R^1_2 : (x_5 \otimes x_2)_* \otimes x_6, D : 1\}$$

We can now use the *Hopf law* to separate some of the connections so that we get:

$$\{G^0_1 : k_*, R^0_1 : x_6\}$$

Since k is classical under the Green observable structure, we can safely remove the first soup connection as it deletes a k, which simplifies our soup to what we have been trying to prove:

$$\{R^1_0 : x_6\}$$



(2) We begin by trying to *fuse* together ℓ_* and ℓ, in a manner that is similar to the definition of unbiasedness. We, therefore, start with the following set of connections in our soup:

$$\{d^Z_{A^*} : \ell \otimes x_5, G^1_2 : (x_5 \otimes \ell)_* \otimes x_6, D : 1\}$$

By the definition of dualisers and since ℓ is classical under the Red observable structure, we can use the explicit witness of the passage between ℓ and ℓ_* to produce:

$$\{d_A^X : \ell_* \otimes x_{4*}, d_{A^*}^Z : x_4 \otimes x_5, G_2^1 : (x_5 \otimes \ell)_* \otimes x_6, D : 1\}$$

Since ℓ is classical under the Red observable structure, we can replace the two ℓ's with a Red copy map, operating on a single ℓ:

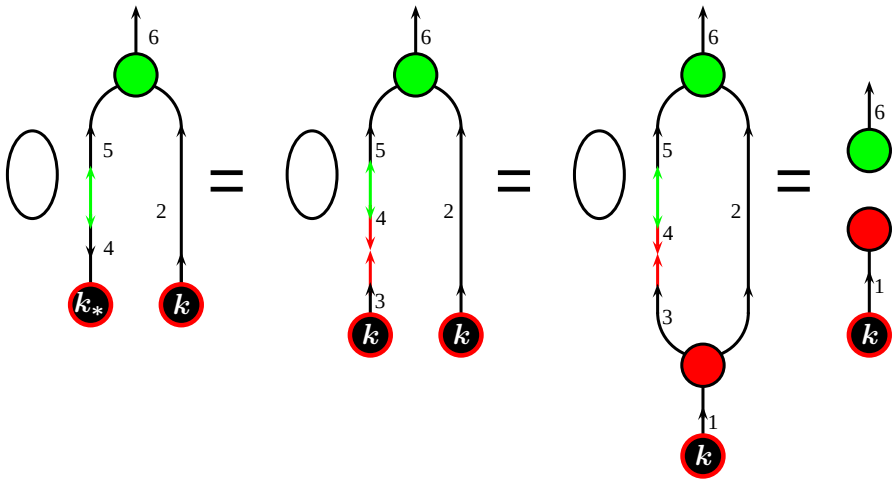
$$\{R_1^2 : \ell_* \otimes x_3 \otimes x_2, d_A^X : x_{3*} \otimes x_{4*}, d_{A^*}^Z : x_4 \otimes x_5, G_2^1 : (x_5 \otimes x_2)_* \otimes x_6, D : 1\}$$

We can now use the *Hopf law* to separate some of the connections so that we get:

$$\{R_1^0 : \ell_*, G_0^1 : x_6\}$$

Since ℓ is classical under the Red observable structure, we can safely remove the first soup connection as it deletes a ℓ, which simplifies our soup to what we have been trying to prove:

$$\{G_0^1 : x_6\}$$



□

We can now define the notion of *complementarity* to describe the property of the interaction that was proved in the previous theorem.

Definition 10.5.2 (Complementarity (dagger lambda calculus))**.** We say that two observable structures are *complementary* when their interaction causes the constants that are *classical* under one structure to be *unbiased* against the other.

Having defined the interaction between complementary observables, we will now define a function that can be used to transform Green sequents into Red ones and vice versa.

Definition 10.5.3 (Hadamard (dagger lambda calculus))**.** For every type A in the dagger lambda calculus, we define a constant $H : A \multimap A$, called the *Hadamard*. The sequent that represents the η-expansion of the *Hadamard*, also known as the *Hadamard gate*, is:

$$x_1 : A \vdash_{\{H:x_{1*} \otimes x_2\}} x_2 : A$$

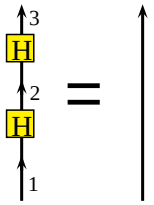
The *Hadamard* must satisfy certain conditions; namely, the *Hadamard gate* must be involutive under Cut, and the *Hadamard* must allow us to transform Green sequents into Red ones.

The first condition, *involution* under Cut, means that we must be able to relate the following sequent:

$$x_1 : A \vdash_{\{H:x_{1*} \otimes x_2, H:x_{2*} \otimes x_3\}} x_3 : A$$

to the identity sequent $x_1 : A \vdash_{\{x_1:x_3\}} x_3 : A$, by equating their soups:

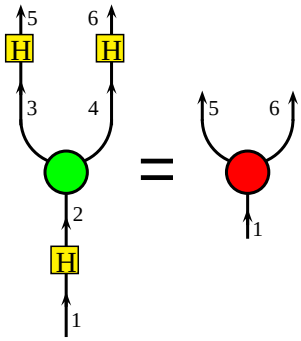
$$\{H : x_{1*} \otimes x_2, H : x_{2*} \otimes x_3\} = \{x_1 : x_3\}$$



Moreover, the *Hadamard* can be capable of transforming a Green copying sequent to a Red one. This is achieved by requiring that the following two soups, S_{17} and S_{18}, be equal in our lambda calculus:

$$S_{17} = \left\{ \begin{array}{ll} H : x_{1*} \otimes x_2, & G_1^2 : x_{2*} \otimes x_3 \otimes x_4, \\ H : x_{3*} \otimes x_5, & H : x_{4*} \otimes x_6 \end{array} \right\}$$

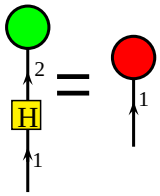
$$S_{18} = \left\{ R_1^2 : x_{1*} \otimes x_5 \otimes x_6 \right\}$$



Finally, the *Hadamard* must be capable of transforming a Green deleting sequent to a Red one. This is achieved by requiring that the following two soups, S_{19} and S_{20}, be equal in our lambda calculus:

$$S_{19} = \quad \left\{ H : x_{1*} \otimes x_2,\ G_1^0 : x_{2*} \right\}$$

$$S_{20} = \quad \left\{ R_1^0 : x_{1*} \right\}$$



Part IV

Applications

Chapter 11

Quantum Key Distribution

Key distribution has traditionally been a very important area of cryptography. While the need for securely distributing a symmetric key has waned with the advent of public key cryptography, key distribution is still required in cases where processing speed is paramount as well as in cases where the level of secrecy does not allow us to rely on complexity assumptions. Because of the nature of the algorithms involved, encrypting messages with one-time pads is inherently faster than encrypting with a public key. Moreover, public key encryption relies on complexity assumptions that, though conjectured correct, have never been proven to be so. Elaborate schemes have been designed for publicising public keys, yet these are usually vulnerable to man-in-the-middle attacks or rely on a pre-existing secure communication channel. The big advantage of Quantum Key Distribution, as it was presented in [BB84] and [Eke91], is that it allows us to securely distribute a symmetric key in a tamper-proof manner. This chapter explains how the dagger lambda calculus can be used to perform Quantum Key Distribution, by expressing the formalism that was used in [CWW$^+$11].
We will begin by presenting a high level description of the steps involved in the protocol:

1. Alice chooses two random strings of bits; $a = a_1, a_2, \ldots, a_{4n}$ and $b = b_1, b_2, \ldots, b_{4n}$.

2. She then uses those strings to generate a string of qubits $|\psi\rangle = |\psi_1\rangle, |\psi_2\rangle, \ldots |\psi_{4n}\rangle$ by using a function $m : A \otimes B \to B$ such that $|\psi_i\rangle = m \circ (a_i \otimes b_i)$, where m acts as an identity on b_i if a_i is $|0\rangle$ and as a Hadamard on b_i if a_i is $|1\rangle$.

3. Alice transfers the string of quantum bits ψ via a quantum channel to

Bob.

4. Bob chooses a random string of bits $c = c_1, c_2, \ldots, c_{4n}$ and measures each qubit $|\psi_i\rangle$ in the Z-basis if $c_i = 0$ and in the X-basis if $c_i = 1$, yielding $d = d_1, d_2, \ldots, d_{4n}$.

5. Bob sends c to Alice via a classical channel.

6. Alice sends $a \oplus c = a_1 \oplus c_1, a_2 \oplus c_2, \ldots, a_{4n} \oplus c_{4n}$ to Bob via a classical channel.

7. Alice and Bob check for which i, $a_i \oplus c_i = 0$. They maintain the corresponding b_i and d_i respectively and they throw away the rest.

8. Alice and Bob should on average each be left with $2n$ bits, which should coincide if there has been no attack.

9. Alice and Bob agree on a subset of roughly half of the remaining bits and compare them to ensure that they haven't been tampered with.

10. If they coincide, they should on average be left with n bits, with which they can engage in one-time-pad cryptographic communication.

The first step in expressing Quantum Key Distribution, is defining the function $m : A \otimes B \to B$. When considering this from a linear algebraic perspective, m is defined by its action on the standard basis, whereby $m(|0\rangle \otimes id_B) = id_B$ and $m(|1\rangle \otimes id_B) = H_B$. Consequently, in order to represent the function in the dagger lambda calculus, we will use a term $m : (A \otimes B) \multimap B$ such that the following equalities hold in the soup:

$$\{m : (R_0^1 \otimes b)_* \otimes b'\} = \{id_B : b_* \otimes b'\}$$

$$\{m : (R[\pi]_0^1 \otimes b)_* \otimes b'\} = \{H : b_* \otimes b'\}$$

Lemma 11.0.1 (Controlled unitary 1)**.** *If copies of the same classical input, in the dagger lambda calculus, are used as control terms for both m and $(m_*)^*$, then the resulting maps will cancel each other out of the soup when composed:*

$$\left\{ \begin{array}{ll} m : (a_2 \otimes b_1)_* \otimes b_2, & (m_*)^* : (a_4 \otimes b_1) \otimes b_{3*}, \\ G_1^2 : a_{1*} \otimes a_2 \otimes a_3, & d_A : a_{3*} \otimes a_{4*} \end{array} \right\} = \left\{ G_1^0 : a_{1*}, b_3 : b_2 \right\}$$

Proof. If a_1 is R^1_0, then it is copied by G^2_1 into a_2 and a_3:

$$\left\{\ m : (R^1_0 \otimes b_1)_* \otimes b_2, \quad (m_*)^* : (R^1_0 \otimes b_1) \otimes b_{3*}\ \right\}$$

which, by the definition of m, results in two identities:

$$\left\{\ id_B : b_{1*} \otimes b_2, \quad id_{B^*} : b_1 \otimes b_{3*}\ \right\} = \left\{\ b_1 : b_2, \quad b_{1*} : b_{3*}\ \right\} = \{b_3 : b_2\}$$

If, on the other hand, a_1 is $R[\pi]^2_1$, then it will again be copied by G^2_1 into a_2 and a_3:

$$\left\{\ m : (R[\pi]^1_0 \otimes b_1)_* \otimes b_2, \quad (m_*)^* : (R[\pi]^1_0 \otimes b_1) \otimes b_{3*}\ \right\}$$

which, by the definition of m, results in two applications of the Hadamard gate:

$$\left\{\ H : b_{1*} \otimes b_2, \quad (H_*)^* : b_1 \otimes b_{3*}\ \right\} = \left\{\ H : b_{1*} \otimes b_2, \quad H : b_{3*} \otimes b_1\ \right\} = \{b_3 : b_2\}$$

□

Lemma 11.0.2 (Controlled unitary 2)**.** *If the classical values that are used in the dagger lambda calculus as control terms for m and $(m_*)^*$ do not coincide, then the resulting maps will compose into a Hadamard transform in the soup:*

$$\left\{\begin{array}{ll} m : (a_1 \otimes b_1)_* \otimes b_2, & (m_*)^* : (a_4 \otimes b_1) \otimes b_{3*}, \\ & d_A : a_{3*} \otimes a_{4*}, \\ G^2_1 : a_{0*} \otimes a_1 \otimes a_2, & R[\pi]^1_1 : a_{2*} \otimes a_3* \end{array}\right\} = \left\{G^0_1 : a_{0*}, H : b_{3*} \otimes b_2\right\}$$

Proof. If a_0 is R^1_0, then it will be copied by G^2_1 into a_1 and a_2:

$$\left\{\begin{array}{ll} m : (R^1_0 \otimes b_1)_* \otimes b_2, & (m_*)^* : (a_4 \otimes b_1) \otimes b_{3*}, \\ & d_A : (R[\pi]^1_0)^* \otimes a_{4*} \end{array}\right\}$$

By the definition of m, the soup reduces to:

$$\left\{\ id_B : b_{1*} \otimes b_2, \quad (H_*)^* : b_1 \otimes b_{3*}\ \right\} = \left\{\ id_B : b_{1*} \otimes b_2, \quad H : b_{3*} \otimes b_1\ \right\} = \{H : b_{3*} \otimes b_2\}$$

If, on the other hand, a_0 is $R[\pi]^1_0$, then it will again be copied by G^2_1 into a_1 and a_2:

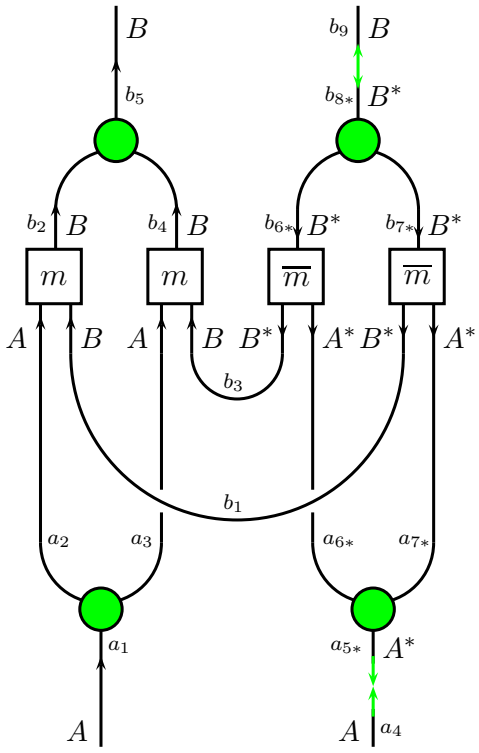
$$\left\{\begin{array}{ll} m : (R[\pi]^1_0 \otimes b_1)_* \otimes b_2, & (m_*)^* : (a_4 \otimes b_1) \otimes b_{3*}, \\ & d_A : (R^1_0)^* \otimes a_{4*} \end{array}\right\}$$

Which, by the definition of m, reduces to:

$$\left\{\ H : b_{1*} \otimes b_2, \quad id_{B^*} : b_1 \otimes b_{3*}\ \right\} = \{H : b_{3*} \otimes b_2\}$$

□

Switching back to category-theoretic notation, we will now present an adaptation of the diagram that was derived in [CWW+11], which will be used to represent the protocol's procedures:



This can translated to the dagger lambda calculus, where the corresponding sequent would be represented by $a_1 : A, a_4 : A \vdash_{S_1} b_5 \otimes b_9 : B \otimes B$ and where the soup S_1 would be of the form:

$$S_1 = \left\{ \begin{array}{ll} & d_{B^*} : b_8 \otimes b_9, \\ G_2^1 : (b_2 \otimes b_4)_* \otimes b_5, & G_2^1 : (b_6 \otimes b_7) \otimes b_{8*}, \\ (m_*)^* : (b_3 \otimes a_6) \otimes b_{6*}, & (m_*)^* : (b_1 \otimes a_7) \otimes b_{7*}, \\ m : (a_2 \otimes b_1)_* \otimes b_2, & m : (a_3 \otimes b_3)_* \otimes b_4, \\ G_1^2 : a_{1*} \otimes a_2 \otimes a_3, & G_1^2 : a_5 \otimes a_{6*} \otimes a_{7*}, \\ & d_A : a_{4*} \otimes a_{5*} \end{array} \right\}$$

In both the categorical diagram and the sequent in the dagger lambda calculus, Alice's control input is denoted by a_1 and Bob's control input is denoted by a_4. We will use the soup derivations of the dagger lambda calculus to verify the protocol's behaviour, in a way that resembles the

verification what was performed in the categorical setting by [CWW+11]. In order to verify the protocol, we will first examine the case where a_1 and a_4 coincide by "hardwiring" them to two different copies of the same original state. This will be done by performing a Cut on $a_0 : A \vdash_{\{G_1^2 : a_{0*} \otimes a_1 \otimes a_4\}} a_1 \otimes a_4 : A \otimes A$ with $a_1 : A, a_4 : A \vdash_{S_1} b_5 \otimes b_9 : B \otimes B$. The resulting sequent can be written as $a_0 : A \vdash_{S_2} b_5 \otimes b_9 : B \otimes B$, where the soup S_2 is:

$$S_2 = \left\{ \begin{array}{ll} & d_{B^*} : b_8 \otimes b_9, \\ G_2^1 : (b_2 \otimes b_4)_* \otimes b_5, & G_2^1 : (b_6 \otimes b_7) \otimes b_{8*}, \\ (m_*)^* : (a_6 \otimes b_3) \otimes b_{6*}, & (m_*)^* : (a_7 \otimes b_1) \otimes b_{7*}, \\ m : (a_2 \otimes b_1)_* \otimes b_2, & m : (a_3 \otimes b_3)_* \otimes b_4, \\ & d_A : a_{5*} \otimes a_{7*}, \\ & d_A : a_{4*} \otimes a_{6*}, \\ G_1^4 \ : \ a_{0*} \ \otimes \ a_2 \ \otimes \ a_3 \ \otimes a_4 \ \otimes \ a_5 & \end{array} \right\}$$

At this point, we can simplify the soup by using Lemma 12.0.2 twice:

$$\left\{ \begin{array}{ll} & d_{B^*} : b_8 \otimes b_9, \\ G_2^1 : (b_2 \otimes b_4)_* \otimes b_5, & G_2^1 : (b_6 \otimes b_7) \otimes b_{8*}, \\ b_7 : b_2, & b_6 : b_4, \\ & G_1^0 : a_{0*} \end{array} \right\}$$

We can use soup reduction to rewrite this as:

$$\left\{ \begin{array}{ll} & d_{B^*} : b_8 \otimes b_9, \\ G_2^1 : (b_2 \otimes b_4)_* \otimes b_5, & G_2^1 : (b_4 \otimes b_2) \otimes b_{8*}, \\ & G_1^0 : a_{0*} \end{array} \right\}$$

Which is essentially:

$$\left\{ \ G_1^0 : a_{0*}, \quad G_0^2 : b_5 \otimes b_9 \ \right\}$$

The resulting sequent reduces to:

$$a_0 : A \vdash_{\{G_1^0 : a_{0*}\}} b \otimes b : B \otimes B$$

Similarly, to examine the case where the control inputs do not coincide, we can perform a Cut on $a_0 : A \vdash_{\{G_1^2 : a_{0*} \otimes a_1 \otimes a, R_1^1[\pi] : a_* \otimes a_4\}} a_1 \otimes a_4 : A \otimes A$ with $a_1 : A, a_4 : A \vdash_S b_5 \otimes b_9 : B \otimes B$. The resulting sequent can be written

as $a_0 : A \vdash_{S_3} b_5 \otimes b_9 : B \otimes B$, where the soup S_3 is:

$$S_3 = \left\{ \begin{array}{ll} & d_{B^*} : b_8 \otimes b_9, \\ G_2^1 : (b_2 \otimes b_4)_* \otimes b_5, & G_2^1 : (b_6 \otimes b_7) \otimes b_{8*}, \\ (m_*)^* : (a_6 \otimes b_3) \otimes b_{6*}, & (m_*)^* : (a_7 \otimes b_1) \otimes b_{7*}, \\ m : (a_2 \otimes b_1)_* \otimes b_2, & m : (a_3 \otimes b_3)_* \otimes b_4, \\ & d_A : a_{5*} \otimes a_{7*}, \\ & d_A : a_{4*} \otimes a_{6*}, \\ G_1^4 \ : \ a_{0*} \ \otimes \ a_2 \ \otimes \ a_3 \ \otimes a_4 \ \otimes \ a_5 & \end{array} \right\}$$

The resulting sequent reduces to:

$$a_0 : A \vdash_{\{G_1^0 : a_{0*}\}} G_0^1 \otimes G_0^1 : B \otimes B$$

This completes our functional verification of the protocol from within the sequents of our dagger lambda calculus. We have proved that in both the case when the control inputs over lap and when they don't, the protocol's behaviour is as expected.

Chapter 12

Quantum Fourier Transform

The quantum Fourier transform is a calculation that lies at the heart of many quantum algorithms, including Shor's factoring algorithm [Sho97]. Functionally, when considering its action on basis states,the transformation is exactly the same transformation as the discrete Fourier transform. Because the transform is a linear operator, its action on arbitrary quantum states is completely determined by its discreet counterpart.

In this chapter we will explain how the QFT is constructed, present its corresponding diagrammatic representation in the picture calculus, and demonstrate that the dagger lambda calculus is expressive enough by using it to represent the transform. We will then proceed by "running" a calculation through, while observing how connections propagate in the soup. In order to keep our exposition simple, we will limit our description to the behaviour of the two-qubit QFT, though it should not be hard to generalise our QFT to a larger numbers of qubits.

As we can see from [NC00] and [CD11], the only gates that are required to construct the quantum Fourier transform are the Hadamard gate and a gate called $\wedge Z_\alpha$ that performs a controlled Z rotation on an arbitrary angle α. In Hilbert spaces, the two-qubit QFT can be expressed as $(H \otimes id_A) \circ \wedge Z_{\pi/2} \circ (id_A \otimes H)$. We can verify its behaviour on a given input (for example $|10\rangle$) by applying it to that state. This would give us:

$$\begin{aligned}(H \otimes id_A) \circ \wedge Z_{\pi/2} \circ (|1\rangle \otimes |0\rangle) &= (H \otimes id_A) \circ (|1\rangle \otimes |+_{\pi/2}\rangle) \\ &= |-\rangle \otimes |+_{\pi/2}\rangle\end{aligned}$$

The dagger lambda calculus is best used alongside the diagrammatic calculus, so that one can complement the other. We will proceed by explaining how the quantum Fourier transform can be represented in the calculus for

complementary observables, as well as how these diagrams translate to sequents and soup connections in the dagger lambda calculus. The two notations will be presented side by side: a set of soup connections on the left and a picture on the right. For a more complete exposition of the diagrammatic representation, the reader is referred to [CD11].

The first step towards constructing a QFT in the dagger lambda calculus, consists of representing the controlled phase gate $\wedge Z_\alpha$. The sequent for this gate can be written as $a_1 : A, a_2 : A \vdash_S a_5 \otimes a_9 : A \otimes A$ where the soup S is:

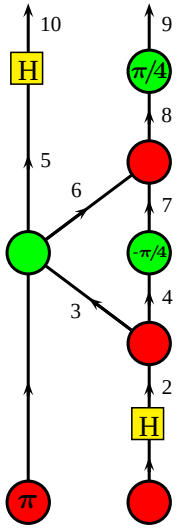
$$S = \left\{ \begin{array}{ll} & G[\alpha/2]^1_1 : a_{8*} \otimes a_9, \\ & R^1_2 : (a_6 \otimes a_7)_* \otimes a_8, \\ G^2_2 : (a_1 \otimes a_3)_* \otimes a_5 \otimes a_6, & G[-\alpha/2]^1_1 : a_{4*} \otimes a_7, \\ & R^2_1 : a_{2*} \otimes a_3 \otimes a_4 \end{array} \right\}$$

We will use S_α in the dagger lambda calculus to denote the soup S of the controlled phase gate, along with its associated angle of rotation α. In the context of Hilbert spaces, we presented a way of generating the two-qubit quantum Fourier transform by using $\wedge Z_{\pi/2}$, the controlled Z rotation with angle $\pi/2$, composed with some Hadamard transforms. Therefore, in the dagger lambda calculus, it can be written as:

$$a_1 : A, a_0 : A \vdash_{S_{\pi/2} \cup \{H:a_{0*} \otimes a_2, H:a_{5*} \otimes a_{10}\}} a_{10} \otimes a_9 : A \otimes A$$

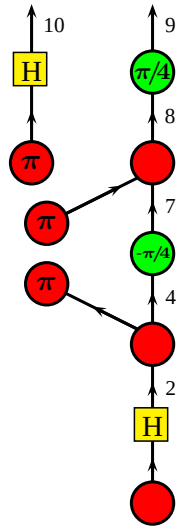
Similarly to the diagrammatic "execution" that was presented in [CD11], we can now "run" the two-qubit quantum Fourier transform in the dagger lambda calculus by plugging in an input state in place of a_1 and a_0 in our sequent. We will be using $|10\rangle = R[\pi]^1_0 \otimes R^1_0$, by plugging in $R[\pi]^1_0$ for a_1 and R^1_0 for a_0. The soup thus becomes:

$$\left\{ \begin{array}{ll} H : a_{5*} \otimes a_{10}, & G[\pi/4]^1_1 : a_{8*} \otimes a_9, \\ & R^1_2 : (a_6 \otimes a_7)_* \otimes a_8, \\ G^2_2 : (R[\pi]^1_0 \otimes a_3)_* \otimes a_5 \otimes a_6, & G[-\pi/4]^1_1 : a_{4*} \otimes a_7, \\ & R^2_1 : a_{2*} \otimes a_3 \otimes a_4, \\ & H : (R^1_0)_* \otimes a_2 \end{array} \right\}$$



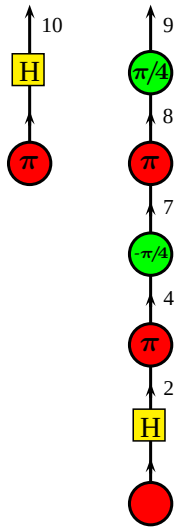
We can then use G_2^2 to copy $R[\pi]_0^1$ across three terms, a_3, a_5 and a_6, giving us:

$$\left\{ \begin{array}{ll} H : (R[\pi]_0^1)_* \otimes a_{10}, & G[\pi/4]_1^1 : a_{8*} \otimes a_9, \\ & R_2^1 : (R[\pi]_0^1 \otimes a_7)_* \otimes a_8, \\ & G[-\pi/4]_1^1 : a_{4*} \otimes a_7, \\ & R_1^2 : a_{2*} \otimes R[\pi]_0^1 \otimes a_4, \\ & H : (R_0^1)_* \otimes a_2 \end{array} \right\}$$



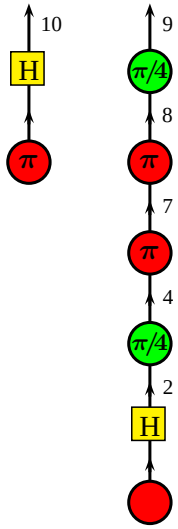
The red phase terms, $R[\pi]_1^1$, can be written in a simpler way:

$$\left\{ \begin{array}{ll} H : (R[\pi]_0^1)_* \otimes a_{10}, & G[\pi/4]_1^1 : a_{8*} \otimes a_9, \\ & R[\pi]_1^1 : a_{7*} \otimes a_8, \\ & G[-\pi/4]_1^1 : a_{4*} \otimes a_7, \\ & R[\pi]_1^1 : a_{2*} \otimes a_4, \\ & H : (R_0^1)_* \otimes a_2 \end{array} \right\}$$



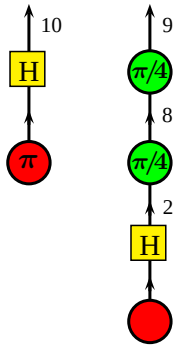
We can then commute red and green phases as follows:

$$\left\{ \begin{array}{ll} H : (R[\pi]^1_0)_* \otimes a_{10}, & G[\pi/4]^1_1 : a_{8*} \otimes a_9, \\ & R[\pi]^1_1 : a_{7*} \otimes a_8, \\ & R[\pi]^1_1 : a_{4*} \otimes a_7, \\ & G[\pi/4]^1_1 : a_{2*} \otimes a_4, \\ & H : (R^1_0)_* \otimes a_2 \end{array} \right\}$$



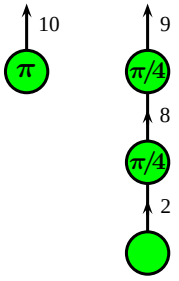
The red phases cancel each other out, since two consecutive π rotations bring us back where we started:

$$\left\{ \begin{array}{ll} H : (R[\pi]^1_0)_* \otimes a_{10}, & G[\pi/4]^1_1 : a_{8*} \otimes a_9, \\ & G[\pi/4]^1_1 : a_{2*} \otimes a_8, \\ & H : (R^1_0)_* \otimes a_2 \end{array} \right\}$$



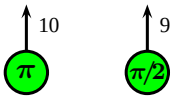
We can now use the Hadamards to transform the remaining red terms into green ones:

$$\left\{ \begin{array}{ll} G[\pi]^1_0 : a_{10}, & G[\pi/4]^1_1 : a_{8*} \otimes a_9, \\ & G[\pi/4]^1_1 : a_{2*} \otimes a_8, \\ & G^1_0 : a_2 \end{array} \right\}$$



Which in the end reduces to the two-qubit quantum Fourier transform's known output for our choice of input:

$$\left\{G[\pi]_0^1 : a_{10}, G[\pi/2]_0^1 : a_9\right\}$$



The quantum Fourier Transform's behaviour on other possible inputs can be verified by following a similar process for the rest of the basis states. The fact that the dagger lambda calculus can represent the quantum Fourier transform and, by extension, Shor's factoring algorithm speaks volumes about the language's expressive power.

Chapter 13

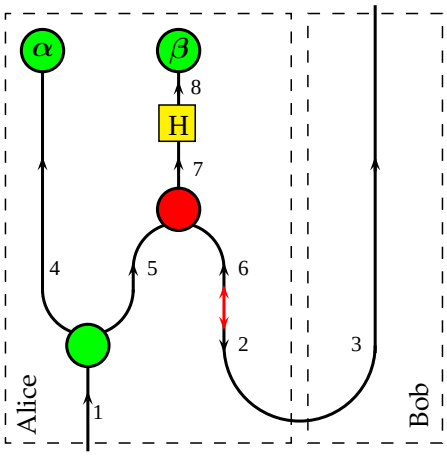
Teleportation Protocol

The teleportation protocol, one of the most famous quantum protocols, uses entanglement to teleport an arbitrary quantum state. In the most common presentation of the protocol, Alice is in possession of a qubit with a quantum state that she wants to teleport to Bob. Alice and Bob share a classical two-bit communication channel but, since the qubit has not been measured, its state carries much more information than two classical bits. In order for Alice to teleport the data, she makes use of an entangled pair of qubits that she shares with Bob. Alice performs a Bell basis measurement on her qubit and her part of the entangled pair of qubits; she then uses the classical communication channel to communicate the result of that measurement to Bob, who uses it to perform a unitary correction on his qubit.

Remark. Similarly to the remark of [CD11] about their ZX-calculus, the diagrammatic forefather of the dagger lambda calculus, our language does not represent the non-deterministic aspect of measurements. Every sequent in this lambda calculus, like every diagram in the ZX-calculus, corresponds to one run of the experiment, as far as measurements are concerned. For this reason, measurements are replaced by the projections into which they will resolve in this run of the experiment. To avoid having to consider sequents for every possible outcome of an experiment, the dagger lambda calculus could be extended to support the conditional diagrams of [DP10] and [CD11]. This would be a significant departure from the simplified logic of the lambda calculus and thereby one that merits to be studied in its own right. Consequently, we include this later on, in the *Further work* section of this dissertation.

Back to our description of the teleportation protocol, Alice's Bell basis measurement will resolve to a projection on one of the Bell basis states:

$\langle\Psi_+|$, $\langle\Psi_-|$, $\langle\Phi_+|$ or $\langle\Phi_-|$. We therefore use $\langle+_\alpha|$ and $\langle+_\beta|$ to represent the outcome of the measurement, where the four possible pairs of $\alpha, \beta \in \{0, \pi\}$ range over all the outcomes of a measurement against the Bell basis. We will now present a diagram for part of the teleportation protocol, like the one used by [CD11], but adapted to support dualisers. Note that this diagram is from before the unitary corrections are performed. The full diagram for teleportation, with all the corrections, will be presented shortly afterwards:

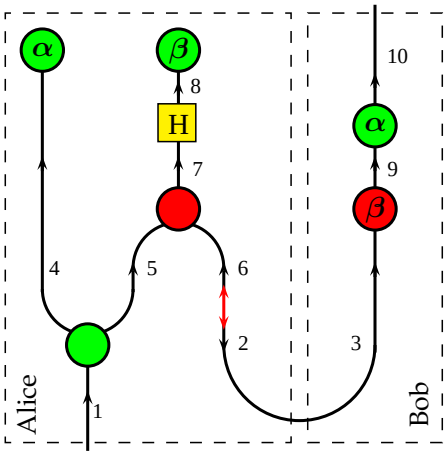


This diagram can easily be represented as a sequent in the lambda calculus. The sequent will be $a_1 : A \vdash_S a_3 : A$, where the soup S is:

$$\left\{ \begin{array}{ll} G[\alpha]^0_1 : x_{4*} & G[\beta]^0_1 : a_{8*}, \\ & H : a_{7*} \otimes a_8, \\ & R^1_2 : (a_5 \otimes a_6)_* \otimes a_7, \\ G^2_1 : x_{1*} \otimes x_4 \otimes x_5 & d_{A*} : a_2 \otimes a_6, \\ & x_* \otimes x : a_{2*} \otimes a_3 \end{array} \right\}$$

Note that, in this sequent, the terms a_{2*} and a_3 represent the entangled pair of qubits that are shared between Alice and Bob. Alice possesses a_{2*} and Bob possesses a_3. Bob receives the results of Alice's projection, α and β, from the classical communication channel. We will now present the full diagram, which includes the unitary corrections that will be performed by Bob; the classical communication channel is not represented in the diagram with any wires but is instead reflected by the fact that Alice's α and β measurements are matched by corresponding corrections by Bob. Having received Alice's measurement values classically, Bob can perform a red phase rotation by an angle of β, followed by a green phase rotation by an angle

of α, on his term a_3. This completes Bob's unitary correction, who should now possess Alice's original input state:



Going back to the lambda calculus, the sequent for the full diagram is expressed by performing a Cut of $a_1 : A \vdash_S a_3 : A$ with $a_3 : A \vdash_{\{R[\beta]^1_1 : a_{3*} \otimes a_9, G[\alpha]^1_1 : a_{9*} \otimes a_{10}\}} a_{10} : A$, which gives us:

$$a_1 : A \vdash_{S \cup \{R[\beta]^1_1 : a_{3*} \otimes a_9, G[\alpha]^1_1 : a_{9*} \otimes a_{10}\}} a_{10} : A$$

We will now start performing operations in our soup, to simplify the existing connections, and prove that the teleportation sequent does actually produce Alice's original input state when Bob runs his corrections at the other end. We will use the soup rules for lifting terms, to lift $G[\alpha]^0_1$ and $G[\beta]^0_1$ into phase shifts. We will also use the rule for the Hadamard gate, to transform the resulting $G[\beta]^1_1$ to a $R[\beta]^1_1$. The resulting soup will be of the form:

$$\left\{ \begin{array}{rr} & G[\alpha]^1_1 : a_{9*} \otimes a_{10}, \\ & R[\beta]^1_1 : a_{3*} \otimes a_9, \\ R[\beta]^1_1 : a_{5*} \otimes a_3, & \\ G[\alpha]^1_1 : a_{1*} \otimes a_5 & \end{array} \right\}$$

But recall that $\alpha, \beta \in \{0, \pi\}$, so $2\alpha = 2\beta = 0$. The consecutive red β rotations hence cancel out and so do the remaining green α rotations after them. This leaves us with the following soup, thus proving that Alice's original input state was teleported to Bob:

$$\{a_1 : a_{10}\}$$

Part V

Conclusion

Chapter 14

Concluding remarks

We have now completed our study of the dagger lambda calculus; a higher-order language that was initially defined with dagger compact categories in mind, but which was later imbued with classical control by enriching it with complementary control structures. The dagger lambda calculus has proven to be a powerful and expressive language, capable of running quantum protocols and programs. Our main motivation behind this language has been to bridge the gap between two different approaches towards quantum computation, by reconciling the diagrammatic formalism with the type theoretic research of higher-order quantum programming languages. Our study of the language's semantics has revealed a number of insights about the building blocks of quantum computation, such as the symmetry of substitution, or the admissibility of some of the rules, giving us the ability to decompose classical notions of computation into finer primitives.

14.1 Future work

As per our remark in chapter 13, the dagger lambda calculus in its current form does not have a way of representing the non-deterministic aspect of measurements. Every sequent in the lambda calculus corresponds to one run of the experiment, where measurements are, as a result, replaced by the projections into which they will resolve. A way of binding measurement results, using *conditional diagrams*, was recently proposed by [DP10] as a modification to the diagrammatic calculus. One area of future work is the modification of the dagger lambda calculus, so that it supports *conditional sequents*. This would require a significant departure from the language's original structure, but it would be interesting to examine whether these

new connections between measurement results could be incorporated into a richer type of soup.

Another area for potential future work is that of Measurement Based Quantum Computation [RB01, RB02, RBB03]. Since the dagger lambda calculus supports both quantum and classical data, it would be ideally suited for such a computational paradigm. The lambda calculus could be extended to support a language like the *Measurement Calculus* of [DKP07]. It would then be interesting to investigate whether the dagger lambda calculus can exhibit an equational correspondence to the Measurement Calculus, as well as to analyse the language's properties in a way that is similar to the analysis we performed in section 9.2.

Once we have defined a programming language that fully supports Measurement Based Quantum Computation and the non-deterministic behaviour of measurements, we will be able to use it to represent all of the elements of quantum algorithms. There are not many efficient quantum algorithms out there. By representing the ones that are available, however, we will be able to spot their common elements and distil the structural elements that are responsible for the quantum mechanical speedup.

14.2 Acknowledgements

I would like to thank my supervisors, Samson Abramsky and Bob Coecke, for their guidance and support throughout the course of my DPhil. Samson is an inexhaustible source of information in the areas of type theory, logic and category theory. In addition to being a supporting, fatherly figure for his students, he could always see many steps ahead in my research and steer me clear of problems. Moreover, his advice has been instrumental in helping me shape the dagger lambda calculus, which serves as the cornerstone of this dissertation. Bob introduced me to categorical quantum computing and his guidance in the early stages of my DPhil inspired my interest in diagrammatic calculi and measurement based quantum computation.

I also want to thank my examiners, Prakash Panangaden and Jonathan Barrett, for their very detailed and helpful comments and the corrections they suggested. I am also thankful towards the anonymous reviewers at a premature conference submission of mine (if you are reading this, you know who you are); their feedback has proved invaluable in shaping the exposition of some of my chapters.

I particularly want to thank my friend and colleague Nikos Tzevelekos for teaching me category theory and for the many in depth discussions and

insights he offered during the development of my calculus. I also want to thank Rick Blute, Andrzej Murawski and Andreas Doering for their helpful feedback on early drafts of my research, and Peter Selinger, Benoît Valiron, John Baez and Mike Stay for our discussions on quantum programming languages. Thanks are also due to Ross Duncan, Mehrnoosh Sadrzadeh, Simon Perdrix, Éric Paquette, Chris Heunen, Jaime Vicary, Aleks Kissinger, Bill Edwards, Alejandro Díaz-Caro and Duško Pavlović for answering questions, sharing their ideas and offering advice on topics ranging from Frobenius algebras and logic, to free categories and quantum programming. I am also thankful to Bob Harper, Benjamin Piece and Frank Pfenning for their lectures at the 2010 Oregon Programming Languages Summer School and for the engaging discussions we had in between sessions.

I am particularly grateful to Joel Spencer, for transmitting to me his love of mathematical logic and theoretical computer science during my undergraduate years. I am also grateful to Erasmia Kiriazi and Elias Kamouzis for lighting up in me a passion for mathematics.

I am grateful to the US Office of Naval Research, FQXi, and the Levelhulme Foundation for supporting my studies financially; this research would not have been possible without them. I also want to thank my college, Wolfson College, and the Department of Computer Science for supporting my trip to attend the 2010 Oregon Programming Languages Summer School.

An acknowledgements section, like this one, would not be complete without mentioning those whose love, friendship, and support have accompanied me throughout my DPhil. A doctoral degree is as much a struggle with one's inner self as it is a struggle with uncovering the secrets of science and, in that sense, personal relationships play a pivotal role in helping the human achieve his true potential. I am thankful to my fiancée, Chara Tzanetaki, for her unwavering love and support throughout the best and the worst times in my doctoral research. Chara has always been there to push me forward, lift my spirits in times I could not do so myself, and even proofread drafts of my research. I am also thankful to my parents, my brother, my friends from Oxford: Yiannis Hadjimichael, Konstantinos Stamatis, Thomas Papadopoulos, Illektra Apostolidou, Mary Kopsacheili and Kyriaki Michailidou; as well as my friends from Greece: Tasos Katechis, Nicholas Tsiroyiannis, Konstantinos Nikolaras, George Pallis. Finally, a big thanks to my friends in Cambridge: Napoleon Katsos, Nausica Smith, and little George Jason Katsos who are now a lot more than just friends and a lot more than just family.

Bibliography

[ABP99] Samson Abramsky, Rick Blute, and Prakash Panangaden. Nuclear and trace ideals in tensored $*$-categories. *Journal of Pure and Applied Algebra*, 143:3–47, 1999.

[Abr93] Samson Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 111:3–57, 1993. (DOI:10.1.1.16.2984).

[Abr05] Samson Abramsky. Abstract scalars, loops, and free traced and strongly compact closed categories. In *In Proceedings of the First Conference on Algebra and Coalgebra in Computer Science (CALCO 2005)*, volume 3629, pages 1–31. Springer Lecture Notes in Computer Science, 2005. (arXiv:0910.2931v1 [quant-ph]).

[Abr10] Samson Abramsky. No-cloning in categorical quantum mechanics. In S. Gay and I. Mackie, editors, *Semantic Techniques in Quantum Computation*. Cambridge University Press, 2010. (arXiv:0910.2401v2 [quant-ph]).

[AC04] Samson Abramsky and Bob Coecke. A categorical semantics of quantum protocols. In *Proceedings of the 19th IEEE conference on Logic in Computer Science (LiCS'04)*. IEEE Computer Science Press, 2004. (arXiv:quant-ph/0402130v5).

[AD06] Samson Abramsky and Ross Duncan. A categorical quantum logic. *Mathematical Structures in Computer Science*, 16:469–489, 2006. (arXiv:quant-ph/0512114v1).

[AT10] Samson Abramsky and Nikos Tzevelekos. Introduction to categories and categorical logic. In Bob Coecke, editor, *New Structures for Physics*. Springer Lecture Notes in Physics, 2010. (arXiv:1102.1313v1 [math.CT]).

[BB84] Charles H. Bennett and Gilles Brassard. Quantum cryptography: Public key distribution and coin tossing. In *Proceedings of the IEEE International Conference on Computers, Systems, and Signal Processing*, pages 175–179. IEEE Press, 1984.

[BS10] John Baez and Michael Stay. Physics, topology, logic and computation: A rosetta stone. In Bob Coecke, editor, *New Structures for Physics*. Springer Lecture Notes in Physics, 2010. (arXiv:0903.0340v3 [quant-ph]).

[CD08] Bob Coecke and Ross Duncan. Interacting quantum observables. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming*, pages 298–310. Lecture Notes in Computer Science 5126, Springer-Verlag, 2008. (arXiv:0906.4725v1 [quant-ph]).

[CD11] Bob Coecke and Ross Duncan. Interacting quantum observables: Categorical algebra and diagrammatics. *New Journal of Physics*, 13:043016, 2011. (arXiv:0906.4725v3 [quant-ph]).

[Che07] Eugenia Cheng. Adjunctions 1. TheCatsters Channel, YouTube, September 13 2007. (`http://www.youtube.com/watch?v=loOJxIOmShE`).

[Coe06] Bob Coecke. Introducing categories to the practicing physicist. *Advanced Studies in Mathematics and Logic, Polimetrica Publishing*, 30:45–74, 2006. (arXiv:0808.1032v1 [quant-ph]).

[CP06] Bob Coecke and Éric Oliver Paquette. POVMs and Naimark's theorem without sums. *Electronic Notes in Theoretical Computer Science*, 2006. (arXiv:quant-ph/0608072).

[CP07] Bob Coecke and Duško Pavlović. Quantum measurements without sums. In G. Chen, L. Kauffman, and S. Lamonaco, editors, *Mathematics of Quantum Computing and Technology*, pages 567–604. Taylor and Francis, 2007. (arXiv:quant-ph/0608035).

[CPP08] Bob Coecke, Éric Oliver Paquette, and Simon Perdrix. Bases in diagrammatic quantum protocols. *Electronic Notes in Theoretical Computer Science*, 218:131–152, 2008. (arXiv:0808.1029v1 [quant-ph]).

[CPP10] Bob Coecke, Éric Oliver Paquette, and Duško Pavlović. Classical and quantum structuralism. In S. Gay and I. Mackie, editors, *Semantic Techniques in Quantum Computation*. Cambridge University Press, 2010. (arXiv:0904.1997v2 [quant-ph]).

[CPV08] Bob Coecke, Duško Pavlović, and Jamie Vicary. A new description of orthogonal bases. *Mathematical Structures in Computer Science*, page 13, 2008. (arXiv:0810.0812v1 [quant-ph]).

[CWW+11] Bob Coecke, Quanlong Wang, Baoshan Wang, Yongjun Wang, and Qiye Zhang. Graphical calculus for quantum key distribution (extended abstract). *Electronic Notes in Theoretical Computer Science*, 270(2):231–249, 2011. ¡ce:title¿Proceedings of the 6th International Workshop on Quantum Physics and Logic (QPL 2009)¡/ce:title¿.

[DKP07] Vincent Danos, Elham Kashefi, and Prakash Panangaden. The measurement calculus. *Journal of the ACM (JACM)*, 54(2), 2007. (arXiv:0704.1263v1 [quant-ph]).

[DP10] Ross Duncan and Simon Perdrix. Rewriting measurement-based quantum computations with generalised flow. In *Proceedings of the 37th international colloquium conference on Automata, languages and programming: Part II*, ICALP'10, pages 285–296, Berlin, Heidelberg, 2010. Springer-Verlag.

[DR89] Sergio Doplicher and John E. Roberts. A new duality theory for compact groups. *Inventiones mathematicae*, 98(1):157–218, 1989.

[Eke91] Arthur K. Ekert. Quantum cryptography based on bells theorem. *Physical review letters*, 67(6):661–663, 1991.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

[JNN12] J. R. Johansson, P. D. Nation, and Franco Nori. Qutip: An open-source python framework for the dynamics of open quantum systems. *Computer Physics Communications*, 183(8):1760 – 1772, 2012. (arXiv:1110.0573 [quant-ph]).

[JS91] André Joyal and Ross Street. An introduction to tannaka duality and quantum groups. In Aurelio Carboni, Maria Pedicchio, and Guiseppe Rosolini, editors, *Category Theory*, volume

1488 of *Lecture Notes in Mathematics*, pages 411–492. Springer Berlin / Heidelberg, 1991.

[Kis11] Aleks Kissinger. *Pictures of processes: Automated graph rewriting for monoidal categories and applications to quantum computing*. PhD thesis, Department of Computer Science, University of Oxford, 2011. (arXiv:1203.0202 [math.CT]).

[KL01] Thomas Kerler and Volodymyr Vasyliovych Lyubashenko. *Non-Semisimple Topological Quantum Field Theories for 3-Manifolds with Corners*. Springer, 2001.

[Klo92] Jan Willem Klop. Term rewriting systems. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 1–116. Oxford University Press, 1992. (DOI:10.1.1.35.425).

[Koc03] Joachim Kock. *Frobenius Algebras and 2-D Topological Quantum Field Theories*, volume 59 of *London Mathematical Society Student Texts*. Cambridge University Press, 2003.

[Mac98] Saunders Mac Lane. *Categories for the Working Mathematician*. Springer, second edition, 1998.

[Mel09] Paul-André Melliès. Categorical semantics of linear logic. *Panoramas et synthèses - Société mathématique de France*, (27):1–196, 2009. (DOI:10.1.1.62.5117).

[Mer07] N. David Mermin. *Quantum Computer Science: An Introduction*. Cambridge University Press, first edition, 2007.

[NC00] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[PB00] Arun Kumar Pati and Samuel L. Braunstein. Impossibility of deleting an unknown quantum state. *Nature*, 404:164–165, 2000. (arXiv:quant-ph/9911090v2).

[RB01] Robert Raussendorf and Hans J. Briegel. A one-way quantum computer. *Physical Review Letters*, 86(22):5188, 2001. DOI:10.1103/PhysRevLett.86.5188.

[RB02] Robert Raussendorf and Hans J. Briegel. Computational model underlying the one-way quantum computer. *Quantum Information and Computation*, 2, 2002. (arXiv:quant-ph/0108067v2).

[RBB03] Robert Raussendorf, Dan E. Browne, and Hans J. Briegel. Measurement-based quantum computation on cluster states. *Physical Review A*, 68, 2003. (arXiv:quant-ph/0301052v2, DOI:10.1103/PhysRevA.68.022312).

[Sel04a] Peter Selinger. A brief survey of quantum programming languages. In *Proceedings of the 7th International Symposium on Functional and Logic Programming*, volume 2998, pages 1–6, Nara, Japan, 2004. Springer Lecture Notes in Computer Science. (DOI:10.1.1.94.463).

[Sel04b] Peter Selinger. Towards a quantum programming language. *Mathematical Structures in Computer Science*, 14(4):527–586, 2004. (DOI:10.1.1.144.6380).

[Sel07] Peter Selinger. Dagger compact closed categories and completely positive maps. In *Proceedings of the 3rd International Workshop on Quantum Programming Languages (QPL 2005)*, volume 170, pages 139–163, Chicago, 2007. Electronic Notes in Theoretical Computer Science. (DOI:10.1.1.134.2476).

[Sel10] Peter Selinger. A survey of graphical languages for monoidal categories. In Bob Coecke, editor, *New Structures for Physics*. Springer Lecture Notes in Physics, 2010. (arXiv:0908.3347v1 [math.CT]).

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. In *SIAM Journal on Scientific and Statistical Computing*, volume 26, page 14841509, 1997. arXiv:quant-ph/9508027v2.

[Str04] Ross Street. Frobenius monads and pseudomonoids. *Journal of Mathematical Physics*, 45(10.III):3930–3948, 2004. (DOI:10.1063/1.1788852).

[SV06] Peter Selinger and Benoît Valiron. A lambda calculus for quantum computation with classical control. *Mathematical Structures in Computer Science*, 16(3):527–552, 2006. (arXiv:cs/0404056v2 [cs.LO]).

[SV08] Peter Selinger and Benoît Valiron. A linear-non-linear model for a computational call-by-value lambda calculus (extended abstract). In *Proceedings of the Eleventh International Conference on Foundations of Software Science and Computation Structures (FOSSACS 2008)*, volume 4962, pages 81–96, Budapest, 2008. Springer Lecture Notes in Computer Science. (arXiv:0801.0813v1 [cs.LO]).

[SV10] Peter Selinger and Benoît Valiron. Quantum lambda calculus. In S. Gay and I. Mackie, editors, *Semantic Techniques in Quantum Computation*. Cambridge University Press, 2010. (`http://www.mscs.dal.ca/~selinger/papers.html#qlambdabook`).

[vT04] André van Tonder. A lambda calculus for quantum computation. *SIAM Journal on Computing*, 33(5):1109–1135, 2004. (arXiv:quant-ph/0307150v5).

[vTD03] André van Tonder and Miquel Dorca. Quantum computation, categorical semantics and linear logic. Archive, 2003. (arXiv:quant-ph/0312174v4).

[WZ82] William K. Wootters and Wojciech H. Zurek. A single quantum cannot be cloned. *Nature*, 299:802–803, 1982. (DOI:10.1038/299802a0).

List of Definitions

List of Theorems, Lemmas and Corollaries