Extending Graphical Representations for Compact Closed Categories with Applications to Symbolic Quantum Computation

Lucas Dixon[1] and Ross Duncan[2]

[1] l.dixon@ed.ac.uk, University of Edinburgh
[2] ross.duncan@comlab.ox.ac.uk, University of Oxford

Abstract. Graph-based formalisms of quantum computation provide an abstract and symbolic way to represent and simulate computations. However, manual manipulation of such graphs is slow and error prone. We present a formalism, based on compact closed categories, that supports mechanised reasoning about such graphs. This gives a compositional account of graph rewriting that preserves the underlying categorical semantics. Using this representation, we describe a generic system with a fixed logical kernel that supports reasoning about models of compact closed category. A salient feature of the system is that it provides a formal and declarative account of derived results that can include 'ellipses'-style notation. We illustrate the framework by instantiating it for a graphical language of quantum computation and show how this can be used to perform symbolic computation.

Key words: graph rewriting, quantum computing, categorical logic, interactive theorem proving, graphical calculi.

1 Introduction

Recent work in quantum computation has emphasised the use of graphical languages motivated by the underlying logical structure of quantum mechanics itself [1, 15, 3, 5, 6]. These techniques have a number of advantages over the conventional matrix-based approach to quantum mechanics:

- The visual representation abstracts over the values in the matrices. This removes detail that requires a lot of work for a human to interpret.
- Many properties have a natural graphical representation. For example, separability of quantum states can be inferred from disjoint subgraphs.
- The algebra of graphs generalises to domains other than vector spaces. In particular, it provides a representation for compact closed categories.

A major problem with these graphical representations is the lack of machinery for automating their manipulation. Unlike existing approaches to graph transformation, the graphs described here provide a representation of compact

closed categories; hence we require a new approach to rewriting, which is sound with respect to the underlying semantics.

The main contribution of this paper is a graph-based formalism that is suitable for representing and evaluating quantum computations. We start by reviewing a formalism for graphical representations of compact closed categories. We also revisit a model of quantum computation based on this calculus. We then extend the graphical calculus in two significant ways driven by the need to express rules that could not be accounted for in the initial formalism. By combining these extensions, we provide a representation that captures an interesting and useful set of graph patterns. Notably, it can express the Spider Theorem which is normally written using informal ellipses notation (see Figure 3). We provide a semantics for our graph-based formalism in terms of the initial representation of compact closed categories.

Using our graph-based formalism as the representational foundation, we develop a simple logical framework for manipulating models of compact closed categories. This has a suitable rewriting mechanism where the axioms of the underlying object-formalism are expressed as equations between graphs. We then present a short case study that illustrates the framework by instantiating it for the introduced model of quantum computation. This shows how the framework can be used to symbolically perform simplifications of quantum programs as well as simulate computations.

2 Graphs and Compact Closed Categories

Graphs A *directed graph*[3] consists of a 4-tuple (V, E, s, t) where V and E are sets, respectively of *vertices*[4] and *edges*, and s and t are maps

$$E \underset{t}{\overset{s}{\rightrightarrows}} V$$

which we call *source* and *target*. Let $\mathrm{in}(v) := t^{-1}(v)$ and $\mathrm{out}(v) := s^{-1}(v)$ denote the *incoming* and *outgoing* edges at a vertex v. The *degree* of a vertex v is $|\mathrm{in}(v)| + |\mathrm{out}(v)|$. To distinguish between elements of different graphs, we will use the subscript notation $G = (V_G, E_G, s_G, t_G)$.

Given graphs G and H, a *graph morphism* $f : G \to H$ consists of functions $f_E : E_G \to E_H$ and $f_V : V_G \to V_H$ such that:

$$s_H \circ f_E = f_V \circ s_G, \tag{1}$$

$$t_H \circ f_E = f_V \circ t_G. \tag{2}$$

These ensure that the structure of the graph is preserved by the morphism: an edge connected to a node gets mapped to a new edge that must be connected, in the same way, to the mapped node.

[3] Equivalently: a directed graph is a functor G from $\bullet \rightrightarrows \bullet$ to **Set**; a graph morphism is then a natural transformation $f : G \Rightarrow H$.

[4] We will use the words "vertex" and "node" interchangeably.

Definition 1. *Let $f : G \to H$ be a graph morphism and let $V' \subseteq V_G$. We say that f is* strict for V' *if $\forall e \in E_H$, if $s_H(e) \in f_V(V')$ or $t_H(e) \in f_V(V')$ then $\exists e' \in E_G$ such that $f_E(e') = e$.*

Strictness ensures that there are no additional edges connected to vertices in the image of V'.

Definition 2. *We call a graph morphism f an* open embedding which is strict for V' *if:*

1. *f_E is injective;*
2. *f_V restricted to V' is injective; and,*
3. *f is strict for V'.*

The intuition behind this definition is that the subgraph of G determined by V' should be preserved exactly by f; whereas other vertices may be identified and may contain additional incident edges.

Augmented by some additional structure, graphs form a *compact closed category*. In section 2.1 we will describe this structure, but first we review the basic properties of compact closed categories.

Compact Closed Categories

Definition 3. *A strict symmetric monoidal category [2] is called compact closed [10] when each object A has a chosen dual object A^*, and morphisms*

$$d_A : I \to A^* \otimes A \qquad e_A : A \otimes A^* \to I$$

where I is the tensor identity of the compact closed category, such that

$$A \cong A \otimes I \xrightarrow{\mathrm{id}_A \otimes d_A} A \otimes A^* \otimes A \xrightarrow{e_A \otimes \mathrm{id}_A} I \otimes A \cong A = \mathrm{id}_A \tag{3}$$

$$A^* \cong I \otimes A^* \xrightarrow{d_A \otimes \mathrm{id}_{A^*}} A^* \otimes A \otimes A^* \xrightarrow{\mathrm{id}_{A^*} \otimes e_A} A^* \otimes I \cong A^* = \mathrm{id}_{A^*} \tag{4}$$

Every arrow $f : A \to B$ in a compact closed category $\mathcal{C}$ has a *name* and *coname*:

$$\ulcorner f \urcorner : I \to A^* \otimes B, \qquad \llcorner f \lrcorner : A \otimes B^* \to I,$$

which are constructed as $\ulcorner f \urcorner = (\mathrm{id}_{A^*} \otimes f) \circ d_A$ and $\llcorner f \lrcorner = e_B \circ (f \otimes \mathrm{id}_{B^*})$. Hence there are natural isomorphisms $\mathcal{C}(A, B) \cong \mathcal{C}(I, A^* \otimes B) \cong \mathcal{C}(A \otimes B^*, I)$ making $\mathcal{C}$ monoidally closed[5]. Furthermore, f has a dual, $f^* : B^* \to A^*$, defined by

$$f^* = (\mathrm{id}_{A^*} \otimes e_B) \circ (\mathrm{id}_{A^*} \otimes f \otimes \mathrm{id}_{B^*}) \circ (d_A \otimes \mathrm{id}_{B^*})$$

By virtue of equations (3) and (4), $f^{**} = f$. Thus $(\cdot)^*$ lifts to an involutive functor $\mathcal{C}^{\mathrm{op}} \to \mathcal{C}$, making $\mathcal{C}$ equivalent to its opposite.

[5] In general compact closed categories are models of multiplicative linear logic where $A \multimap B$ is defined as $A^\perp \otimes B$.

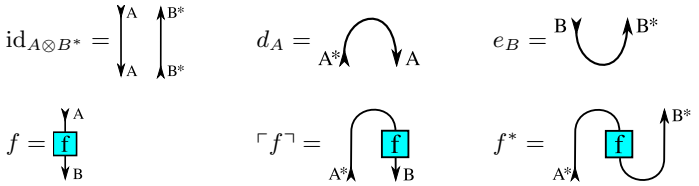


Fig. 1. Compact Closed Structure as Graphs.

2.1 Graph Representations for Compact Closed Categories

Graphs with certain additional structure give a representation for compact closed categories; we now give an overview of this construction. The details omitted here can be found in [8]. Pictorial representations are in Fig. 1. We make the convention that the domain of an arrow is at the top of the picture, and its codomain is at the bottom.

A *concrete graph* Γ is 5-tuple $(G, \operatorname{dom}\Gamma, \operatorname{cod}\Gamma, <_{\operatorname{in}(\cdot)}, <_{\operatorname{out}(\cdot)})$ where:

- $G = (V, E, s, t)$ is a graph;
- $\operatorname{dom}\Gamma$ and $\operatorname{cod}\Gamma$ are totally ordered disjoint sets of degree one vertices of G. The union of these sets is the *boundary* of Γ.
- $<_{\operatorname{in}(\cdot)}$ is a family of maps, indexed by V such that $<_{\operatorname{in}(v)}\colon \operatorname{in}(v) \xrightarrow{\cong} \mathbb{N}_k$ where $k = |\operatorname{in}(v)|$.
- $<_{\operatorname{out}(\cdot)}$ is a family of maps, indexed by V such that $<_{\operatorname{out}(v)}\colon \operatorname{out}(v) \xrightarrow{\cong} \mathbb{N}_{k'}$ where $k' = |\operatorname{out}(v)|$.

Since the sets $\operatorname{dom}\Gamma$ and $\operatorname{cod}\Gamma$ consist of vertices of degree one, we can assign a polarity to each one: $v \mapsto +$ if the edge incident at v is an incoming edge; $v \mapsto -$ otherwise. Hence $\operatorname{cod}\Gamma$ and $\operatorname{dom}\Gamma$ are *ordered signed sets*. Given any ordered signed set S we write S^* for the same ordered set with the opposite signing. Given two such sets we can define their disjoint union $R + S$ as the disjoint union of the underlying sets, inheriting the signing and the order from R and S, with the convention that $r < s$ for all $r \in R, s \in S$.

Proposition 1. *Concrete graphs form a compact closed category whose objects are ordered signed sets and whose arrows $f : A \to B$ are concrete graphs with* $\operatorname{cod} f = B$ *and* $\operatorname{dom} f = A^*$.

For each ordered signed set A, the identity map id_A has $\operatorname{dom}\operatorname{id}_A = A^*$ and $\operatorname{cod}\operatorname{id}_A = A$; its underlying graph has $E = A$ and $V = A^* + A$ with $t(a) = a$ and $s(a) = a^*$. Given a pair of concrete graphs $f : A \to B$ and $g : B \to C$ their composition $g \circ f : A \to C$ is constructed by merging the two graphs, erasing the vertices of $\operatorname{cod} f$ and $\operatorname{dom} g$ (called the *boundary vertices*), and identifying the edges previously incident at the deleted vertices. (Due to the opposite polarity of the domain and codomain the edges have compatible direction.) The tensor

product on objects A, B is simply $A+B$; given $f : A \to B$, $g : C \to D$, the graph of $f \otimes g$ is the disjoint union of the graphs of f and g. The unit for the tensor is the empty set. The morphisms $d_A : I \to A^* \otimes A$, $e_A : A \otimes A^* \to I$ have the same underlying graph as id_A, but $\mathrm{dom}\, d = \emptyset$, $\mathrm{cod}\, d = A^* + A$, $\mathrm{dom}\, e = A + A^*$ and $\mathrm{cod}\, e = \emptyset$.

Remark 1. Given concrete graphs $f : A \to B$ and $g : B \to C$ there exists exactly one graph morphism $\tau : f \to (g \circ f)$ such that τ is an open embedding which is strict for the non-boundary nodes of f. Indeed the intuition behind an open embedding, is that any such map picks out a subgraph which forms a well defined arrow in its own right.

This category captures exactly the axioms for compact closed structure, in the sense that any freely generated compact closed category can be represented by concrete graphs. We will consider a collection of basic terms[6] F whose types are vectors of some set of basic types T. Then:

Definition 4. *A T, F-*labelling *θ for a concrete graph Γ is a pair of maps $\theta_T : E \to T$ and $\theta_F : (V - \mathrm{cod}\, \Gamma - \mathrm{dom}\, \Gamma) \to F$ such that for each vertex v, if $in(v) = \langle a_1, \ldots, a_n \rangle$ and $out(v) = \langle b_1, \ldots, b_m \rangle$ then*

$$\theta v : \langle \theta a_1, \ldots, \theta a_n \rangle \to \langle \theta b_1, \ldots, \theta b_m \rangle$$

*We say a concrete graph Γ is T, F-*labellable *if there exists an T, F-labelling for it; and if θ is a labelling for Γ, then the pair (Γ, θ) is called a T, F-*labelled graph.

The T, F-labelled graphs form a compact closed category in the same way as the concrete graphs, subject to the further restriction that arrows are composable only when their labellings agree.

Theorem 1. *Let $\mathcal{C}$ be a compact closed category, freely generated by some set of arrows F and ground types T; then $\mathcal{C}$ is equivalent to the category of T, F-labelled graphs.*

Given a compact closed category $\mathcal{C}$ generated by some basic set of operations, the arrows of $\mathcal{C}$ have a canonical representation as labelled graphs. A consequence of the theorem is then that two arrows are equal by the equations of the compact closed structure if and only if their graph representations are equal.

As a final remark before moving on, note that the external structure of a vertex in a concrete graph is essentially the same as that of a complete graph; hence one can consistently view subgraphs as vertices, and abstract over the their internal structure.

[6] See [8] for a more thorough description of the nature of the terms.

3 Quantum Computations as Graphs

A substantial strand of work in quantum computation has involved the development of high-level models of quantum processes based on compact closed categories. In these formalisms, initiated in [1], quantum processes—such as quantum logic gates, or the measurement of a qubit—correspond to arrows in a category, while the different quantum data types, usually just arrays of qubits, are the objects. The graphical representation described in the preceding section thus provides a very expressive notation for quantum processes.

While compact closed categories provide a suitable setting for reasoning about quantum computation, freely generated structure will not suffice: we need additional equations. These equations will be expressed as rewrites rules for graphs. In this section we will describe a set of generators and equations used to reason about quantum computation, and show how some of its formal properties lead to particular issues for the rewriting machinery.

Coecke and Duncan [4] propose a formal algebraic system for quantum computation built from the following collection of generators:

Objects: a single object, written Q.
Arrows: there are families of arrows X and Z:

$$\begin{aligned} &\epsilon_Z : Q \to I, && \epsilon_X : Q \to I, \\ &\delta_Z : Q \to Q \otimes Q, && \delta_X : Q \to Q \otimes Q, \\ &\alpha_Z : Q \to Q && \alpha_X : Q \to Q \end{aligned}$$

where $\alpha \in [0, 2\pi)$, and in addition $H : Q \to Q$.

To each each arrow $f : A \to B$ we assign a formal adjoint $f^\dagger : B \to A$. Each arrow is represented as a small graph; its adjoint is the same graph written upside down by reflection in the x-axis. We use colours (light green and a darker red) to denote the two families:

$\epsilon_Z =$  $\delta_Z =$  $\epsilon_Z^\dagger =$  $\delta_Z^\dagger =$  $\alpha_Z =$ 

$\epsilon_X =$  $\delta_X =$  $\epsilon_X^\dagger =$  $\delta_X^\dagger =$  $\alpha_X =$ 

The H arrow is denoted by  and represents a Hadamard gate. The adjoints for H, α_X and α_Z will be defined equationally. The free compact closed category is then given by all graphs formed by composing and tensoring these basic graphs.

In terms of quantum processes, each edge in a graph represents a qubit, although several edges may represent the same physical qubit at different times. An edge may even represent a "virtual" qubit which stands for a correlation between different parts of the system. The maps δ_Z and ϵ_Z represent quantum operations which respectively copy and delete the eigenstates of the Pauli Z operator.[7] Notice that δ_Z has one edge in its domain representing the qubit

[7] Uniform copying operations are forbidden by the no-cloning theorem [18], but such operations are possible if we demand only the eigenstates of some self-adjoint operator to be copied. Other states will not not copied. The same remarks hold true for erasing [11].

to be copied, and two edges in its codomain for the two copies it produces. Similarly, ϵ_Z has one qubit as input and no outputs. The adjoints $\delta_Z^\dagger$ and $\epsilon_Z^\dagger$ correspond to an operation known as *fusion*, and to the operation of preparing a fresh qubit in a certain state. The α_Z corresponds to phase shift of angle α in the Z direction. The family of maps indexed by X are defined in exactly the same way, but relative to the Pauli X operator rather than the Z. All quantum operations may be defined by combining these simple operations—which are essentially classical—on two complementary observables.

We emphasise that this is a notation for representing quantum processes, rather than quantum states. In this setting a state is simply a process with no inputs; that is, a concrete graph with empty domain. Since our formalism is based on the underlying mathematical structure rather than any particular model of quantum computation, it is capable of representing quantum circuits, and measurement-based quantum computations, among other models. Indeed, an important application of this work is to show that states or computations implemented differently are equivalent.

To model the behaviour of quantum systems certain additional equations must be satisfied. At the level of objects, we ask that Q is self dual, i.e. that is $Q = Q^*$. Hence we use *undirected* graphs. The equations between arrows are discussed in detail in [4]; we present them in graphical form in Figure 2.

Consider the equations from Figure 2 which involve only one colour: these allow the remarkable *spider theorem*, first noted in [6], to be proved:

Theorem 2 (Spider Theorem). *Let G be a connected graph generated from δ_Z, ϵ_Z, α_Z and their adjoints; then G is totally determined by the number of inputs, the number of outputs, and the sum modulo 2π of the αs which occur in it.*

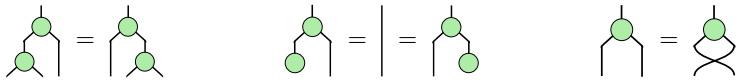
Hence any connected subgraph involving nodes of only one colour may be collapsed to a single vertex, with a single value α, giving a "spider". Informally, this can be depicted graphically as the equation in Figure 3-left. Conversely, a spider may be arbitrarily divided into sub-spiders, provided the total in- and out-degree is preserved, along with the sum of the αs. Furthermore, one can derive, from the Spider Theorem, n-fold versions of many of the other equations.

Spiders offer a very intuitive way to manipulate graphs, and are far more compact and convenient in calculations than the graphs built up naively from the generators. However, formalising spiders requires moving from finite graphs, where each vertex has bounded degree, and which are subject to a finite number of rewrite rules, to a system where nodes may have arbitrarily many edges, and there are infinitely many concrete rewrite rules. The desire to retain intuitive reasoning methods for these infinite families of rewrites motivates the extension from concrete graphs to *graph patterns*, the main subject of this paper.

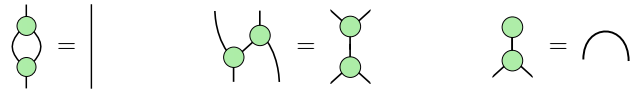
4 Graphs with Variable Nodes

In the concrete representation, the following graphs represent different computations:

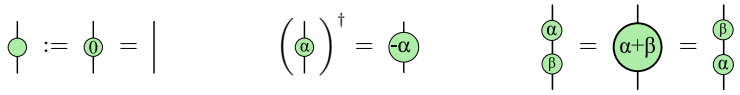
Comonoid Laws



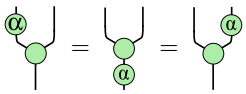
Isometry, Frobenius, and Compact Structure



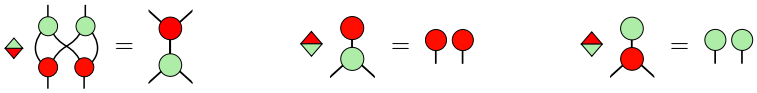
Abelian Unitary Group



Bilinearity



Bialgebra Laws Let  ; then:



Group Actions

H **Properties**

Colour Duality

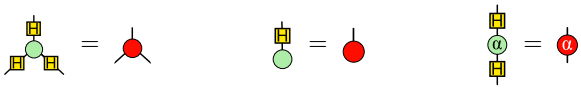


Fig. 2. Graphical Equations for Quantum Systems. The $(\cdot)^\dagger$ functor gives a vertical symmetry to the category, hence for every equation we have a second equation obtained by flipping the diagram upside down. In addition, we have a "colour duality": each equation shown here gives rise to second, which is obtained by exchanging the two colours. The colour duality is derivable from the equations involving H.

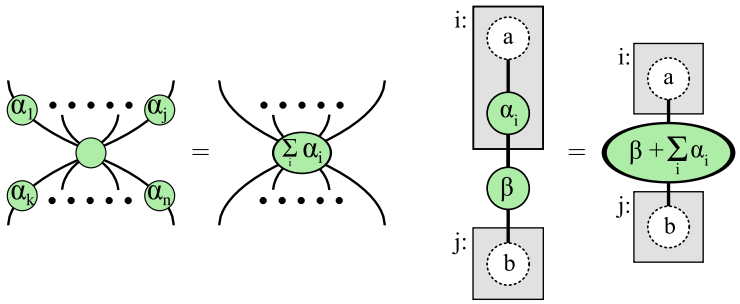
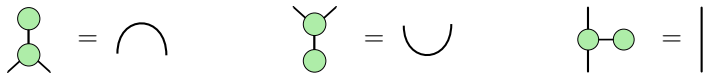


Fig. 3. [Left] An informal equation on graphs that expresses the Spider Theorem. [Right] The Spider Theorem expressed formally using graph patterns. The !-boxes are named i and j. The variable nodes are white and named a and b. The non-variable node data (the angle) is written inside the node when non-zero.



However, composition with semi-circles (d_Q and e_Q from Definition 3), on the co-domain or domain, allows an equation involving one of the above to easily lead to a derivation corresponding to either of the other two: given any one of the allows a trivial derivation of the others.

To address this, we formalise a representation that abstracts over the boundary nodes membership in the domain or co-domain. This gives rise to a *variable-node graphs*, in which boundary nodes have been generalised to *variable nodes*. The intuition of variable nodes is that they can replaced by concrete nodes in some graph in a process analogous to composition.

To formalise the semantics of variable-node graphs, we define *matching*, which captures the intuitive idea of a graph with variable nodes occurring within another graph:

Definition 5. *A variable-node (resp., concrete) graph G* matches *another graph H if there exists an open embedding $G \to H$ which is strict on the non-variable (resp., non-boundary) nodes. This open embedding is called a* matching*. The notation $G \leq_v H$ is used for G matches H.*

A graph with variable nodes, G, can be given a formal semantics by being interpreted as a set of concrete graphs, denoted by $[\![G]\!]_v$. The interpretation is simply the set of concrete graphs which the variable node graph matches.

Proposition 2. $G \leq_v H \Leftrightarrow [\![G]\!]_v \supseteq [\![H]\!]_v$.

Proof. The proof of the implication from left to right is a consequence of the fact that a composition of open embeddings is an open embedding. In particular, the embedding of $G \leq_v H$ composed with $[\![H]\!]_v$ is thus an open embedding of G into $[\![H]\!]_v$. Hence $[\![G]\!]_v \supseteq [\![H]\!]_v$. From right to left, we compose the open embedding $[\![G]\!]_v$, restricted to its subset $[\![H]\!]_v$, with the inverse of $[\![H]\!]_v$ to get $G \leq_v H$.

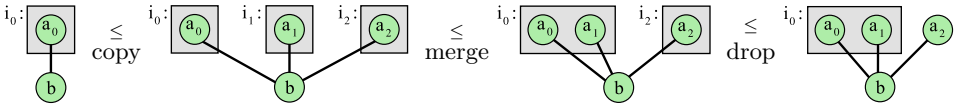


Fig. 4. An illustration of !-box graph matching using the !-box operations. This involves first copying !-box i_0 twice, then merging i_0 and i_1 and finally dropping i_3.

5 !-Boxes

If we wish formalise reasoning via spiders then an arbitrary number of repetitions of a subgraph need to be matched. To support this we introduce an operation, !-boxing (pronounced bang-boxing), on graph representations. Given a graph representation, this introduces a new notation by outlining a set of nodes. These nodes are said to be in a !-box. Intuitively, the resulting !-boxed graph can be thought of as representing a set of graphs with an arbitrary number of copies of the !-boxed nodes, where every copy connects, in the same way, to the nodes outside the !-box. The !-boxes can also be instantiated with zero copies. This erases all edges to and from the !-box.

More formally, a *!-box graph* is a pair $(G, \mathcal{B})$ where G is a graph and $\mathcal{B}$ is a set of disjoint subsets of V i.e. $b_1, b_2 \in \mathcal{B}$ then $b_1 \cap b_2 = \emptyset$.[8] To formalise the intuitive notion that a !-box represents arbitrary number of copies of the subgraph made from its nodes, we introduce *!-box matching*. This binary relation, written infix as $\leq_!$, is defined such that $(G, \mathcal{B}) \leq_! (H, \mathcal{C})$ whenever $(H, \mathcal{C})$ can be obtained from $(G, \mathcal{B})$ by a sequence the following operations:

copy : copies a !-box, b in a graph to produce a new graph with two copies of the !-boxed subgraph, b is the old one and b' is the new one. The set of !-boxes in the copied graph now also contains the new !-box b'. Any edges between a node, n, inside the !-box b, and a node, m, outside it, get copied so that there is a new edge from m to the new copy of n in b'.

drop : simply removes the !-box, but leaves its contents in the graph.

kill : removes from the graph all nodes in the !-box as well as any incident edges.

merge : combines two !-boxes, B_1 and B_2 into a single larger !-box $B_1 \cup B_2$. To ensure that copying after merging commutes with copying before, merging is restricted to !-boxes which do not have an edge between their nodes.

An example of matching with these operations is illustrated in Figure 5.

We give a formal semantics to !-box graphs by defining them in terms of a set of graphs in the underlying representation. In particular, we denote the interpretation of a !-box graph $(G, \mathcal{B})$ by $[\![(G, \mathcal{B})]\!]_!$ and say that its members are instances.

[8] One could consider more expressive notions of nested, or overlapping, node sets in the !-boxes. While such expressivity is interesting, it is not required for the system we formalise here.

Definition 6. *The* interpretation $[\![(G, \mathcal{B})]\!]_!$ *is a set of concrete graphs defined by*

$$[\![(G, \mathcal{B})]\!]_! = \{H \mid (G, \mathcal{B}) \leq_! (H, \emptyset)\}$$

i.e. the those graphs matched by the !-box graph that have no !-boxes.

Observe that every concrete instance of a !-box graph can be defined by pairing each !-box with the natural number that defines how many copies are made of it. Thus $[\![G]\!]_!$ is isomorphic to the set of k-tuples of natural numbers, where k is the number of !-boxes. The need for the !-box matching operation, rather than using a direct k-tuple interpretation, is to allow matching between !-box graphs, and thus to provide a mechanism for derived rules.

Proposition 3. !-Matching respects !-box semantics*:* $G \leq_! H \Leftrightarrow [\![G]\!]_! \supseteq [\![H]\!]_!$. *The proof is a simple consequence from the definition of* $[\![G]\!]_!$ *being a subset of the graphs that match* G*.*

For the purposes of this paper, the underlying graph representation of !-boxed graphs is variable-node graphs. This gives rise to *graph patterns* which we now discuss in more detail.

6 Graph Patterns

The representation of Compact Closed Categories as graphs, discussed in §2.1, is too restrictive for reasoning about quantum computation. In particular, graphical rules such the Spider Theorem, from Figure 3-left, are frequently needed, but not expressible. In this section we combine the *!-boxes* and *variable-node* extensions to graphs. We call this representation *graph patterns*. This forms a representation that allows us to express, in a finite way, certain infinite families of equations between concrete graphs. In particular, the Spider Theorem can now be represented as shown in Figure 3-right. We also extend the notion of matching for graph pattens. This provides the foundations for the rewriting machinery in §7 which can then be used to reason about quantum computation.

The semantics for a graph pattern G is a set of concrete graphs denoted by $[\![G]\!]$ and define it as:

$$[\![G]\!] = \{[\![G']\!]_v \,.\, G' \in [\![G]\!]_!\}$$

this simply considers every interpretation of the !-boxes to give variable-node graphs for which we then appeal to their own semantics.

The specification for one pattern, G, to match another one, H is that it is more general with respect to the interpretation: $[\![G]\!] \supseteq [\![H]\!]$. However, graph patterns can correspond to a countably infinite number of concrete graphs. Thus matching between graph patterns cannot be implemented by simply unfolding all interpretations as concrete graphs and checking the membership relation.

Fortunately, it is quite easy to provide decidable matching: the size of the unfolding that needs to be considered can be bounded. The key observation

is that a graph G_1 will never match a graph with fewer non-variable nodes. Thus unfolding of G_1 can be bounded by the number of non-variable nodes in G_2. While this gives a generate and test style algorithm, it is not efficient. The intuition for an efficient algorithm is to search through one graph incrementally increasing the matched part.

7 Reasoning with Graph Patterns

In this section we describe how the graph pattern formalism can provide a *meta-level* framework for reasoning about models of compact closed categories. Following the terminology of logical frameworks such as Isabelle [12], we call the specification provided by the underlying model an *object-level* graph formalism. An object-level formalisation defines a set of rules which are treated as the axioms for the system; for instance, the equations from Figure 2. It also defines the data at the nodes and edges as well as corresponding data-matching behaviour. For its part, the meta-level provides generic machinery to manipulate graphs and derive new rules. We now describe the meta-level framework, noting the conditions for a rule to be valid, and prove the systems adequacy for rewriting. The resulting system forms the basis for an interactive proof assistant that supports reasoning about compact closed categories.

7.1 Equational Rules

In our framework, the axioms defined by an object-level model, as well as derived rules, are pairs of graph patterns. Such a pair represents the left and right hand sides of an equation. Rules are declarative in that they denote a set of concrete equational rules.

The intuitive idea of substitution with a rule is to replace a subgraph that matches the left hand side with the right hand side. However, not all pairs of rules make sense with respect to the underlying semantics. For an equation to be well defined with respect to the compact closed structure it must not be possible to change the type (the boundary nodes in the domain and co-domain) of a concrete graph by rewriting. Mapping this restriction back to pairs of graph patterns results in the following conditions on rules:

- There has to be a isomorphism between variable nodes in the left and right hand subgraphs. Given a matching against the left hand side of a rule, the target subgraph is replaced with the right hand side while keeping the same instantiations for the isomorphic variable nodes of the right hand side.
- Rules must also define a partial injective mapping between !-boxes on the left and right hand sides. The intuition for this mapping is that the unfolding used when matching a !-box on the left, is applied to the mapped !-box on the right before replacement.
- The interplay between !-boxes and variable nodes means that when a variable node appears within a !-box on one side of a rule, it must also appear under a mapped !-box on the other side.

Notationally, and implementationally, we annotate !-boxes and variable nodes in a graph with unique names. For example, Figure 3 shows the Spider Theorem. In this figure, the mapping between !-boxes is represented by !-boxes having the same name on the left and right of a rule. Similarly, the isomorphism between variable nodes is captured by the set of variable node names on the left and right hand side being equal.

7.2 Lifting Axioms and Adequacy

The axioms of an object formalism come from the semantics of the underlying system. For instance, the equations given in Figure 2 can be proved by matrix calculations in the underlying model. When such rules are expressed as graph patterns, we replace the concrete representation's boundary nodes with variable nodes. This operation is called *lifting*. An equation on graph patterns corresponds to an infinite (when their are variable nodes) set of equations between concrete graphs. Thus we might worry that the lifted equations express too much: they may allow rewrites which are not true.

We call the property that the lifted representation is a conservative extension of the initial theory *adequacy*. For models of compact closed categories, the proof of adequacy is quite simple: given an equation between concrete graphs, $G = H$, we observe that every instance of the lifted equation corresponds to the original equations composed with some graph. The graph is given by the unmapped subgraph using the open embedding from the lifted equations onto the considered instance. Thus if $G = H$ is true, then so is every instance of its lifting, and thus lifting produces an adequate representation.

7.3 Meta-Level Logic and Derived Rules

Having defined what makes a valid rule, we now present the meta-logic of the framework. This is quite simple as it only involves dealing with object-level equations:

$$\frac{}{\Gamma \vdash A = A}\,\text{refl} \qquad \frac{\Gamma \vdash A = B}{\Gamma \vdash B = A}\,\text{sym} \qquad \frac{\Gamma \vdash A = B \qquad \Gamma \vdash C = D}{\Gamma \vdash C = (D[A/B])}\,\text{subst}$$

where Γ is the set of object-level axioms.

We assume that the axioms in Γ meet the validity conditions described earlier. These rules all preserve the validity conditions on equations and thus the system as a whole ensures only valid rules are derived. For the reflexivity rule (*refl*), we assumes that A is a well-formed pattern graph. This rule allows a new graph to be introduced. By then applying the *subst* rule, intermediate results are derived which can themselves be used to rewrite other rules and conjectures. In this way, the system allows derived rules to provide an abbreviation for a combination of steps.

A sets of rules can be applied automatically to simplify a graph or simulate computation in the object domain. For such rewriting to terminate, a suitable

left-to-right ordering on rules needs to be observed, such as a decrease in the size of the subgraph. In §8 we illustrate simulating a quantum computation.

8 A Case Study in Quantum Computation

The model of quantum computation introduced in §3 provides an object formalism for our meta-level framework. In particular, the object level axioms come from lifting the equations in Figure 2 and from the encoding of the Spider Theorem as shown in Figure 3.

Our model of quantum computation requires no data for the edges. The nodes on the other hand are either H (a Hadamard gate), with no additional data, or have an *angle* and a *colour*, expressing that they are defined in the X or Z basis. For their part, angles are expressed as rational numbers which correspond to the coefficient of π in the underlying matrix.

To allow composition of rules to compute the resulting angles we give the X and Z nodes an *angle expression*. When a node is within a !-box, the expression is a single *angle-variable* which gets instantiated to a new angle-variable in each of the unfoldings of the !-box. When a node is not within a !-box, the angle-expression is a mapping from a set of angle-variables to the corresponding rational coefficient. When an angle-expression contains an angle-variable within a !-box, this is interpreted as a sum of the variables that result from its unfolding. This rather simple expression language has a normal form by ordering the angle-variable by name. Matching then results in angle-variables being instantiated and the expressions in all affected nodes are then (re)normalised. An additional implementation detail must also be observed for the substitution rule: it must ensure that angle-variables in the rule being applied are distinct from those in the expression being rewritten.

The quantum Fourier transform is among the most important quantum algorithms, forming an essential part of Shor's algorithm [16], famous for providing polynomial factoring. In our graph pattern calculus this circuit becomes the top-left graph in Figure 8. This figure shows how computation can be symbolically performed by rewriting with the lifted equations from Figure 2 and the graph pattern version of the Spider Theorem.

9 Related Work

There are several foundational approaches to graph transformation, including algebraic approaches [7], node-label controlled [9], matrix based [17], and programmed graph replacement [14]. These provide general ways of understanding graph transformations which can then be implemented to provide machinery for a specific application. However, systems based on these theories do not provide machinery for the semantics of compact closed categories. Thus their notion of matching and replacement do not guarantee well-typed results. Furthermore, they do not provide machinery for rewriting of graphs with ellipses notation which is needed to represent the Spider Theorem.

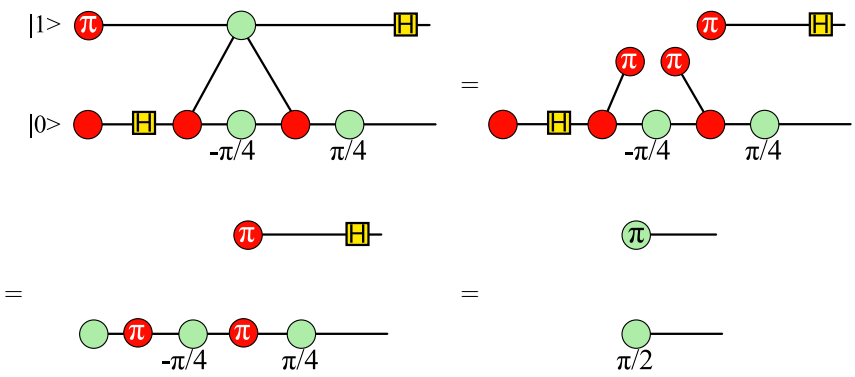


Fig. 5. An example computation of the Quantum Fourier Transform with inputs 1 and 0, performed symbolically by rewriting.

The distinctive feature of our form of graph rewriting is that the graphs capture the structural properties of compact closed categories and rewriting is compositional: it preserves the type of the rewritten subgraph. However, our system can also be seen as an instantiation of a general graph rewriting system: matching provides the embedding information and the object-level node matching defines the application conditions and the attribute transfer function.

We note that our graphical notation has little connection to *graph states* as used in various approaches to measurement-based quantum computation [13]. In that approach the graph structure is used to provide a description of the entanglement in a state: it does not provide a complete description of a computation.

10 Conclusions and Further Work

We have introduced a representation for graphs which can formally characterise the ellipses notation used informally to represent certain infinite families of graph rewrites, such as the Spider Theorem. This representation provides the foundation for a simple meta-logic for reasoning about models of compact closed categories. We illustrated this by providing an account of quantum computation and showing how computation can be performed. Having developed the basic representational machinery and shown matching to be decidable, we are left with several exciting avenues for further research. The most immediate direction we are pursuing is to provide a full implementation - only a partial one is currently available[9]. Other areas of further work include considering confluence results for sets of rewrite rules, increasing the expressiveness of the representation for graph-patterns, and finding a complete set of rewrite rules for the considered model of quantum computation.

[9] `http://dream.inf.ed.ac.uk/projects/quantomatic`

References

1. S. Abramsky and B. Coecke. A categorical semantics of quantum protocols. In *LICS 2004*, pages 415–425. IEEE Computer Society, 2004.
2. A. Asperti and G. Longo. *Categories, Types and Structures*. MIT Press, 1991.
3. B. Coecke. Kindergarten quantum mechanics. Lecture Notes, 2005.
4. B. Coecke and R. Duncan. Interacting quantum observables. In *ICALP 2008*. LNCS, 2008.
5. B. Coecke and E. O. Paquette. POVMs and Naimark's theorem without sums. In *Proc. of the 4th International Workshop on Quantum Programming Languages*, 2006.
6. B. Coecke and D. Pavlovic. Quantum measurements without sums. In *The Mathematics of Quantum Computation and Technology*, CRC Applied Mathematics & Nonlinear Science. Taylor and Francis, 2007.
7. A. Corradini, H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, and A. Wagner. Algebraic approaches to graph transformation - part I: Single pushout approach and comparison with double pushout approach. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation. Vol. I: Foundations*, pages 247–312. World Scientific, 1997.
8. R Duncan. *Types for Quantum Computation*. PhD thesis, Oxford University, 2006.
9. D. Janssens and G. Rozenberg. Graph grammars with node-label controlled rewriting and embedding. In *Proc. of the 2nd International Workshop on Graph-Grammars and Their Application to Computer Science*, pages 186–205. Springer-Verlag, 1983.
10. G.M. Kelly and M.L. Laplaza. Coherence for compact closed categories. *Journal of Pure and Applied Algebra*, 19:193–213, 1980.
11. A.K. Pati and S. L. Braunstein. Impossibility of deleting an unknown quantum state. *Nature*, 404:164–165, 2000.
12. L. C. Paulson. *Isabelle: A generic theorem prover*. Springer-Verlag, 1994.
13. R. Raussendorf and H. J. Briegel. A one-way quantum computer. *Phys. Rev. Lett.*, 86:5188–5191, 2001.
14. A. Schfürr. *Programmed graph replacement systems*, pages 479–546. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1997.
15. P. Selinger. Dagger compact closed categories and completely positive maps. In *Proc. of the 3rd International Workshop on Quantum Programming Languages*, 2005.
16. P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J.Sci.Statist.Comput.*, 26(5), 1997.
17. P. P. P. Velasco and J. de Lara. Matrix approach to graph transformation: Matching and sequences. In *ICGT*, volume 4178 of *LNCS*, pages 122–137. Springer, 2006.
18. W. Wootters and W. Zurek. A single quantum cannot be cloned. *Nature*, 299:802–803, 1982.