Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 336 (2018) 257–279

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Classical Control and Quantum Circuits in Enriched Category Theory

Mathys Rennela

Institute for Computing and Information Sciences
Radboud University
Nijmegen, The Netherlands

Sam Staton

Department of Computer Science
Oxford University
Oxford, United Kingdom

Abstract

We describe categorical models of a circuit-based (quantum) functional programming language. We show that enriched categories play a crucial role. Following earlier work on QWire by Paykin et al., we consider both a simple first-order linear language for circuits, and a more powerful host language, such that the circuit language is embedded inside the host language. Our categorical semantics for the host language is standard, and involves cartesian closed categories and monads. We interpret the circuit language not in an ordinary category, but in a category that is enriched in the host category. As an extended example, we recall an earlier result that the category of W*-algebras is dcpo-enriched, and we use this model to extend the circuit language with some recursive types.

Keywords: Enriched categories, categorical semantics, linear type theory, quantum circuits, relative monad, quantum domain theory

1 Introduction

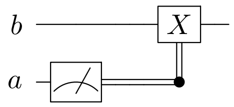
One of the subtle points about quantum computation is the interaction between classical control flow and quantum operations. One can measure a qubit, destroying the qubit but producing a classical bit; this classical bit can then be used to decide whether to apply quantum rotations to other qubits. This kind of classical control can be neatly described in quantum circuits, for example when one uses the measurement outcome of a qubit a to conditionally perform a gate X

$$\begin{tikzcd}\textbf{Classical} \arrow[r, bend left, "\text{control}"] & \textbf{Quantum} \arrow[l, bend left, "\text{measurements}"]\end{tikzcd}$$

https://doi.org/10.1016/j.entcs.2018.03.027

1571-0661/© 2018 The Author(s). Published by Elsevier B.V.

on a qubit b:



(1)

This can be understood semantically in terms of mixed states, density matrices, and completely positive maps. However, high level languages have more elaborate data structures than bits: they have higher order functions and mixed variance recursive types, and associated with these are elaborate control structures such as higher order recursive functions. These are important, paradigmatic ways of structuring programs.

How should these high level features be integrated with quantum computation? One option is to build a semantic domain that accommodates both quantum computation and higher order features. This is an aim of some categorical semantics of the quantum lambda calculus [21,17] and of prior work of the authors [24,26]. This is a fascinating direction, and sheds light, for example, on the structure of the quantum teleportation algorithm (e.g. [21, Ex. 6]). However, the general connection between physics and higher-order quantum functions is yet unclear. Although some recent progress has been made [14], it is still unclear whether higher-order quantum functions of this kind are useful for quantum algorithms.

Another approach is to understand a high level quantum programming language as an ordinary higher-order functional language with extra features for building and running quantum circuits. In this setting, quantum circuits form a first-order embedded domain specific language within a conventional higher order language. This fits the current state-of-the-art in interfaces to quantum hardware, and is the basis of the languages Quipper [10] and LiQUi$|\rangle$ [31]. This is the approach that we study in this paper.

1.1 *Embedded languages and enriched categories*

Our work revolves around a new calculus that we call ‘EWire’ (§2). It is a minor generalization of the QWire language [22]. QWire idealizes some aspects of the architecture of Quipper and LiQUi$|\rangle$. The idea is that we deal with a host language separated from an embedded circuit language.

- The circuit language is a first order typed language. The types, called ‘wire types’, include a type for qubits. The wire type system is *linear* to accommodate the fact that qubits cannot be duplicated.
- The host language is a higher order language. The types of the host language do not include the wire types, there is not a type of qubits, and it is not a linear type system. However, there is a special host type $\mathrm{Circ}(W_1, W_2)$ associated to any pair of wire types W_1 and W_2, whose inhabitants are the circuits with inputs of type W_1 and outputs of type W_2.

Let us describe the circuit language in a nutshell: the very simple circuit (1) corresponds to the instruction below (2) in the circuit language. Given two qubits a and b, it measures the qubit a, stores the result in a bit x which is later used in the application of the classical-controlled-X gate and discards the bit x, then outputs the resulting qubit y.

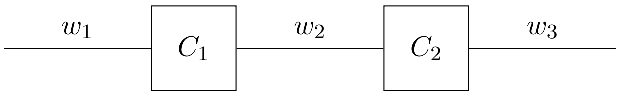
$$\begin{aligned} -; a, b : \text{qubit} \vdash C \overset{\text{def}}{=} \; & x \leftarrow \textbf{gate meas } a; (x, y) \leftarrow \textbf{gate } (\textbf{bit-control } X)(x, b); \\ & () \leftarrow \textbf{gate discard } x; \textbf{output } y \quad : \text{ qubit} \end{aligned} \tag{2}$$

The interface between the host language and the circuit language is set up in terms of boxing and unboxing. For example, the instruction

$$t \overset{\text{def}}{=} \textbf{box } (a, b) \Rightarrow C(a, b) \qquad (\text{where } C \text{ is as in } (2)) \tag{3}$$

creates a closed term of type Circ(qubit⊗qubit,qubit) in the host language. We recover the instruction C in the circuit language (2) from the boxed expression t in the host language (3) by using the instruction **unbox** $t\,w$ for some fresh wire w of type qubit.

Also, it is possible to write a program that composes two circuits C_1 and C_2 with the right input/output types, for example:



This is a program

$$\text{comp} \overset{\text{def}}{=} \lambda(C_1, C_2).\, \textbf{box } w_1 \Rightarrow \big(w_2 \leftarrow \textbf{unbox } C_1 w_1; w_3 \leftarrow \textbf{unbox } C_2 w_2; \textbf{output } w_3\big)$$

in the host language, associated with the type

$$\text{comp} : \text{Circ}(W_1, W_2) \times \text{Circ}(W_2, W_3) \to \text{Circ}(W_1, W_3) \tag{4}$$

where W_i is the type of the wire w_i for $i \in \{1, 2, 3\}$.

Now, recall the idea of an enriched category, which is informally a category such that the morphisms from A to B form an object of another category. In Section 3, once we conceptualize types as objects and terms as morphisms, we show that ***the embedding of the circuit language in the host language is an instance of enriched category theory***: the circuits (morphisms) between wire types (objects) form a type (object) of the host language. The host composition term in (4) is precisely composition in the sense of enriched categories. (See §3 for details.)

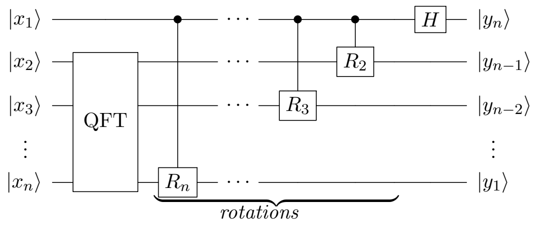
For a simple version of the model, wire types are understood as finite-dimensional C*-algebras, and circuits are completely positive unital maps – the accepted model of quantum computation. Host types are interpreted as sets, and the type of all circuits is interpreted simply as the set of all circuits. The category of sets supports higher order functions, which shows that it is consistent for the host language to have higher order functions.

As with any higher order language, the set-theoretic model is not sufficient to fully understand the nature of higher order functions. We envisage that other semantic models (e.g. based on game semantics or realizability) will also fit the same framework of enriched categories, so that our categorical framework provides a sound description of the basic program equivalences that should hold in all models. These equivalences play the same role that the β and η equivalences play in the pure lambda calculus. In other words, we are developing a basic type theory for quantum computation.

1.2 Recursive types and recursive terms

Within this semantic model, based on enriched categories, we can freely accommodate various additional features in the host language, while keeping the circuit language the same. For example, we could add recursion to the host language, to model the idea of repeatedly trying quantum experiments, or recursive types, to model arbitrary data types. This can be shown to be consistent by modifying the simple model so that host types are interpreted as directed complete partial orders (dcpo's).

Many quantum algorithms are actually parameterized in the number of qubits that they operate on. For example, the Quantum Fourier Transform (QFT) has a uniform definition for any number of qubits, where H is the Hadamard gate $\frac{1}{\sqrt{2}}\left(\begin{smallmatrix}1 & 1\\ 1 & -1\end{smallmatrix}\right)$ and R_n is the Z rotation gate $\left(\begin{smallmatrix}1 & 0\\ 0 & e^{2\pi i/2^n}\end{smallmatrix}\right)$.



We formalize this by extending the circuit language with a wire type QList of qubit-lists for which the following equivalence of types holds:

$$\mathrm{QList} \;\cong\; (\mathrm{qubit} \otimes \mathrm{QList}) \oplus 1$$

so that we can define a function

$$\mathrm{fourier} : \mathrm{Circ}(\mathrm{QList}, \mathrm{QList})$$

In Section 4, we part away from the canonical intuition of circuits by considering them as first-order and linear (in the linear logic sense of the term) instructions. In practice, it will be useful for a circuit layout engine to know the number of qubits in the lists, suggesting a dependent type such as

$$\mathrm{fourier} : (n : \mathrm{Nat}) \to \mathrm{Circ}(\mathrm{QList}(n), \mathrm{QList}(n))$$

but we leave these kinds of elaboration of the type system to future work.

The categorical essence of recursive data types is algebraic compactness. In short, one says that a category $\mathbf{C}$ is algebraically compact (for a specific class of endofunctors) when every endofunctor $F : \mathbf{C} \to \mathbf{C}$ has a canonical fixpoint, which is the initial F-algebra [3]. In earlier work [24], the first author has shown that the category of W*-algebras is algebraically compact and enriched in dcpo's, and so this is a natural candidate for a semantics of the language. In brief: circuit types are interpreted as W*-algebras, and circuits are interpreted as completely positive sub-unital maps; host types are interpreted as dcpo's; in particular the collection of circuits $\mathrm{Circ}(W, W')$ is interpreted as the dcpo of completely positive sub-unital maps, with the Löwner order. In this way, we provide a basic model for a quantum type theory with recursive types. We conclude the present work with what is, to our knowledge, the first categorical semantics of a language that can accommodate QFT in this parameterized way.

2 Functional programming and quantum circuits

We introduce a new calculus called EWire as a basis for analysing the basic ideas of embedding a circuit language inside a host functional programming language. EWire (for 'embedded wire language') is based on QWire [22] ('quantum wire language'), and we make the connection precise in Section 2.2. One may add other features, for instance as discussed in Section 4.1.

We assume two classes of basic wire types.

- Classical wire types, ranged over by $\mathbf{a}, \mathbf{b}, \ldots$. The wire types exist in both the circuit language and the host language. For example, the type of classical bits, or Booleans.
- Circuit-only wire types, ranged over by $\alpha, \beta, \ldots$. These wire types only exist in the circuit language. For example, the type of qubits.

From these basic types we build all wire types:

$$W, W' ::= I \mid W \otimes W' \mid \mathbf{a} \mid \mathbf{b} \mid \alpha \mid \beta \ldots$$

We isolate the classical wire types, which are the types not using any circuit-only basic types:

$$V, V' \ ::= \ I \mid V \otimes V' \mid \mathbf{a} \mid \mathbf{b} \ldots$$

We also assume a collection $\mathcal{G}$ of basic gates, each assigned an input and an output wire type. We write $\mathcal{G}(W_{\mathrm{in}}, W_{\mathrm{out}})$ for the collection of gates of input type W_{in} and output type W_{out}.

In addition to the embedded circuit language, we consider a host language. This is like Moggi's monadic metalanguage [20] but with special types for the classical wire types $\mathbf{a}$, $\mathbf{b}$ and a type $\mathrm{Circ}(W, W')$ of circuits for any wire types W and W'. So the host types are

$$A, B \ ::= \ A \times B \mid 1 \mid A \to B \mid T(A) \mid \mathrm{Circ}(W, W') \mid \mathbf{a} \mid \mathbf{b}$$

The monad T is primarily to allow probabilistic computations, although one might also add other side effects to the host language. Notice that every classical wire type V can be understood as a first order host type $|V|$, according to the simple translation, called *lifting*:

$$\left|V \otimes V'\right| \stackrel{\text{def}}{=} |V| \times \left|V'\right| \quad |I| \stackrel{\text{def}}{=} 1 \quad |\mathbf{a}| \stackrel{\text{def}}{=} \mathbf{a}$$

2.1 Circuit typing and host typing

A well-formed circuit judgement $\Gamma; \Omega \vdash C : W$ describes a circuit with input context $\Omega = (w_1 : W_1 \cdots w_n : W_n)$ (for $n \in \mathbb{N}$) and output wire type W under the context of host language variables $\Gamma = (x_1 : A_1 \cdots x_m : A_m)$ (for $m \in \mathbb{N}$). Wires are organised in patterns given by the grammar $p ::= w \mid () \mid (p, p)$ associated to the following set of rules:

$$\frac{-}{\cdot \implies () : 1} \qquad \frac{-}{w : W \implies w : W} \qquad \frac{\Omega_1 \implies p_1 : W_1 \qquad \Omega_2 \implies p_2 : W_2}{\Omega_1, \Omega_2 \implies (p_1, p_2) : W_1 \otimes W_2}$$

Linear type theory for circuits

The first five term formation rules are fairly standard for a linear type theory. These are the constructions for sequencing circuits, one after another, and ending by outputting the wires, and for splitting a tensor-product type into its constituents. The fifth rule includes the basic gates in the circuit language.

$$\frac{\Gamma; \Omega_1 \vdash C_1 : W_1 \qquad \Omega \implies p : W_1 \qquad \Gamma; \Omega, \Omega_2 \vdash C_2 : W_2}{\Gamma; \Omega_1, \Omega_2 \vdash p \leftarrow C_1; C_2 : W_2} \qquad \frac{\Omega \implies p : W}{\Gamma; \Omega \vdash \mathbf{output}\ p : W}$$

$$\frac{\Omega \implies p : 1 \quad \Gamma; \Omega' \vdash C : W}{\Gamma; \Omega, \Omega' \vdash () \leftarrow p; C : W} \qquad \frac{\Omega \implies p : W_1 \otimes W_2 \quad \Gamma; w_1 : W_1, w_2 : W_2, \Omega' \vdash C : W}{\Gamma; \Omega, \Omega' \vdash (w_1, w_2) \leftarrow p; C : W}$$

$$\frac{\Omega_1 \implies p_1 : W_1 \qquad \Omega_2 \implies p_2 : W_2 \qquad \Gamma; \Omega_2, \Omega \vdash C : W}{\Gamma; \Omega_1, \Omega \vdash p_2 \leftarrow \mathbf{gate}\ g\, p_1; C : W}\ g \in \mathcal{G}(W_1, W_2)$$

For example, coin flipping is given by the following circuit:

$$\text{flip} \stackrel{\text{def}}{=} a \leftarrow \mathbf{gate}\ \text{init}_0\ (); a' \leftarrow \mathbf{gate}\ \text{H}\ a; b \leftarrow \mathbf{gate}\ \text{meas}\ a'; \mathbf{output}\ b$$

Interaction between the circuits and the host

A well-formed host judgement $\Gamma \vdash t : A$ describes a host-language program of type A in the context of host language variables Γ. The next set of typing rules describe the interaction between the host language and the circuit language. The host can run a circuit and get the result. Since this may have side effects, for example probabilistic behaviour, it returns a monadic type.

$$\frac{\Gamma; \cdot \vdash C : W}{\Gamma \vdash \mathbf{run}\ C : T(|W|)}\ W\ \textit{classical}$$

The next two rules concern boxing a circuit as data in the host language, and then unboxing the data to form a circuit fragment in the circuit language. Notice that unboxing requires a pure program of type $\text{Circ}(W_1, W_2)$, rather than effectful program of type $T(\mathrm{Circ}(W_1, W_2))$. For example, you cannot unbox a probabilistic combination of circuits. The monadic notation clarifies this point.

$$\frac{\Omega \implies p : W_1 \qquad \Gamma; \Omega \vdash C : W_2}{\Gamma \vdash \mathbf{box}\ (p : W_1) \Rightarrow C : \mathrm{Circ}(W_1, W_2)} \qquad \frac{\Gamma \vdash t : \mathrm{Circ}(W_1, W_2) \quad \Omega \implies p : W_1}{\Gamma; \Omega \vdash \mathbf{unbox}\ t\, p : W_2}$$

Finally we consider dynamic lifting, which, informally, allows us to send classical data to and from the host program while the quantum circuit is running.

$$\frac{\Gamma \vdash t : |W|}{\Gamma; - \vdash \mathbf{init}\ t : W}\ W\ \textit{classical} \qquad \frac{\Omega \implies p : |W| \qquad \Gamma, x : W; \Omega' \vdash C : W'}{\Gamma; \Omega, \Omega' \vdash x \Leftarrow \mathbf{lift}\ p; C : W'}\ W\ \textit{classical}$$

These rules are in addition to the standard typing rules for the host language, following Moggi [20] (see Appendix A).

In Appendix B we recall a reduction relation on circuits, based on [22], which reduces a circuit with no free host variables to an expression in the following grammar

$$N ::= \mathbf{output}\ p \mid w \leftarrow \mathbf{gate}\ g\, p; N \mid x \Leftarrow \mathbf{lift}\ p; N \mid () \leftarrow w; N \mid (w_1, w_2) \leftarrow w; N$$

The reduction works by rearranging patterns and resolving unboxed boxes.

2.2 QWire

The language QWire of Paykin, Rand and Zdancewic [22] is an instance of EWire where:

- there is one classical wire type, bit, and one circuit-only wire type, qubit.
- there are basic gates meas $\in \mathcal{G}(\mathrm{qubit}, \mathrm{bit})$ and new $\in \mathcal{G}(\mathrm{bit}, \mathrm{qubit})$.

A subtle difference between EWire and QWire is that in QWire one can directly run a circuit of type qubit, and it will produce a bit, automatically measuring the qubit that results from the circuit. To run a circuit of type qubit in EWire, one must append an explicit measurement at the end of the circuit. These explicit measurements can be appended automatically, to give a translation from QWire proper to this instantiation of EWire. We now summarize how this is done. We first define a translation $(\overline{-})$ from *all* wire types to classical wire types:

$$\overline{W \otimes W'} \overset{\text{def}}{=} \overline{W} \otimes \overline{W'} \quad \overline{I} \overset{\text{def}}{=} I \quad \overline{\mathrm{bit}} \overset{\text{def}}{=} \mathrm{bit} \quad \overline{\mathrm{qubit}} \overset{\text{def}}{=} \mathrm{bit}$$

Then, from an arbitrary wire type W, we can extract a host type $|\overline{W}|$.

From the basic gates meas and new we can define circuits $\mathrm{meas}_W : \mathrm{Circ}(W, \overline{W})$ and $\mathrm{new}_W : \mathrm{Circ}(\overline{W}, W)$ for all wires W. These are defined by induction on W. For example,

$$\mathrm{meas}_I \overset{\text{def}}{=} \mathrm{id} \quad \mathrm{meas}_{\mathrm{bit}} \overset{\text{def}}{=} \mathrm{id}$$
$$\mathrm{meas}_{\mathrm{qubit}} \overset{\text{def}}{=} \mathbf{box}\ p \Rightarrow p' \leftarrow \mathbf{gate}\ \ \mathrm{meas}\, p; \mathbf{output}\ p'$$

$$\mathrm{meas}_{W\otimes W'} \stackrel{\mathrm{def}}{=} \mathbf{box}\ (w,w') \Rightarrow\ x \leftarrow \mathbf{unbox}\ \ \mathrm{meas}_W\, w;\ x' \leftarrow \mathbf{unbox}\ \ \mathrm{meas}_{W'}\, w'; \\ \mathbf{output}\ (x,x')$$

and new_W is defined using new $\mathcal{G}(\mathrm{bit},\mathrm{qubit})$ similarly. Then we define the following derived syntax, so that run and lift can be used at all wire types, not just the classical ones:

$$\mathbf{qwire\text{-}run}(C) \stackrel{\mathrm{def}}{=} \mathbf{run}(x \leftarrow C; \mathbf{unbox}\ \mathrm{meas}\ x)$$
$$(x \Leftarrow \mathbf{qwire\text{-}lift}\, p\,;\, C) \stackrel{\mathrm{def}}{=} y \Leftarrow \mathbf{lift}\, p\,;\, x \leftarrow \mathbf{unbox}\ \mathrm{new}\ y\,;\, C$$

3 Categorical models of EWire

Let us define a sufficient set of properties which ensure that a pair of categories corresponds to a categorical model in which one can interpret EWire, in order to reason about circuits and identify their denotational meaning. We assume that the circuit language is parametrized by a fixed collection of gates, noted $\mathcal{G}$.

Enriched categories. Our development is based on the theory of enriched categories, which are increasingly widely used in programming language theory. We recall some basics. If $\mathbf{H}$ is a category with finite products $\times$, then a category $\mathbf{C}$ enriched in $\mathbf{H}$ is given by a collection of objects together with

- for each pair of objects A and B in $\mathbf{C}$, an object $\mathbf{C}(A,B)$ of $\mathbf{H}$;
- for each object A of $\mathbf{C}$, a morphism $1 \to \mathbf{C}(A,A)$ in $\mathbf{H}$;
- for objects A, B, C of $\mathbf{C}$, a morphism $\mathbf{C}(A,B) \times \mathbf{C}(B,C) \to \mathbf{C}(A,C)$ in $\mathbf{H}$

such that composition satisfies the identity and unit laws. If $\mathbf{C}$ and $\mathbf{D}$ are enriched in $\mathbf{H}$, an enriched functor F is a mapping from the collection of objects of $\mathbf{C}$ to the collection of objects of $\mathbf{D}$ together with, for objects A and B of $\mathbf{C}$, a morphism $\mathbf{C}(A,B) \to \mathbf{D}(F(A),F(B))$, respecting composition and identities in a suitable way.

For a first example, a locally small category is a category for which the collection of morphisms $\mathbf{C}(A,B)$ is a set; this is a category enriched in the category **Set** of sets and functions.

As a first illustration of the importance of enriched categories in computer science, recall that a model of the typed lambda calculus is a cartesian closed category, which is a category $\mathbf{H}$ with finite products that is enriched in itself.

Symmetric monoidal enriched categories. Recall that a symmetric monoidal category is a category $\mathbf{C}$ together with a distinguished object I and a functor $\otimes : \mathbf{C} \times \mathbf{C} \to \mathbf{C}$ together with coherent associativity, identity and symmetry natural isomorphisms. Any category with products is an example of this, but more generally symmetric monoidal categories model linear type theories where weakening and contraction might not hold. An $\mathbf{H}$-*enriched symmetric monoidal category* is defined in a similar way except that the functor must be an enriched functor and the isomorphisms must also be enriched natural transformations.

(Recall that for $\mathbf{H}$-enriched functors $F, G : \mathbf{C} \to \mathbf{D}$ between $\mathbf{H}$-enriched categories $\mathbf{C}$ and $\mathbf{D}$, an enriched natural transformation $\eta : F \Rightarrow G$ is a family of morphisms $\{\eta_c : 1 \to \mathbf{D}(Fc, Gc)\}_{c \in \mathrm{Obj}(\mathbf{C})}$ in the category $\mathbf{H}$ which satisfy a straightforward naturality condition. See [13] for more details.)

Computational effects. Embedding the circuit language requires the use of some computational effects in the host language. When the circuit language involves quantum measurement, then the closed host term $\vdash \mathbf{run}(\mathrm{flip}) : T(\mathrm{bit})$ is a coin toss, and so the semantics of the host language must accommodate probabilistic features.

Following Moggi, we model this by considering a cartesian closed category $\mathbf{H}$ with an enriched monad on it. Recall that an enriched monad is given by an endofunctor T on $\mathbf{H}$ together with a unit morphism $\eta : X \to T(X)$ for each X in $\mathbf{H}$, and a bind morphism $\mathbf{H}(X, T(Y)) \to \mathbf{H}(T(X), T(Y))$ for objects X and Y, subject to the monad laws [20].

The idea is that deterministic, pure programs in the host language are interpreted as morphisms in $\mathbf{H}$. Probabilistic, effectful programs in the host language are interpreted as Kleisli morphisms, i.e. morphisms $X \to T(Y)$.

Relative monads. The monads of Moggi are a slightly higher order notion. In a truly first order language, this type arguably should not exist. In particular, there is no wire type $T(A)$ of all quantum computations. To resolve this mismatch, authors have proposed alternatives such as relative monads [1] and monads with arities [4].

A *relative adjunction* is given by three functors $J : \mathbf{B} \to \mathbf{D}$, $L : \mathbf{B} \to \mathbf{C}$ and $R : \mathbf{C} \to \mathbf{D}$ such that there is a natural isomorphism $\mathbf{C}(L(b), c) \cong \mathbf{D}(J(b), R(c))$. We write $L \;{}_J\!\dashv R$ and call *relative monad* the functor $RL : \mathbf{B} \to \mathbf{D}$.

Enriched relative adjunctions and enriched relative monads are defined in the obvious way, by requiring J, L and R to be enriched functors and the adjunction to be an enriched adjunction. In an enriched relative monad $T = RL$, the bind operation is a morphism of type $\mathbf{D}(J(X), T(Y)) \to \mathbf{D}(T(X), T(Y))$.

Copowers. A copower is a generalization of an n-fold coproduct. Let n be a natural number, and let A be an object of a category $\mathbf{C}$ with sums, the copower $n \odot A$ is the n fold coproduct $A + \cdots + A$. This has the universal property that to give a morphism $n \odot A \to B$ is to give a family of n morphisms $A \to B$. In general, if $\mathbf{C}$ is a category enriched in a category $\mathbf{H}$, and A is an object of $\mathbf{C}$ and h an object of $\mathbf{H}$, then the *copower* is an object $h \odot A$ together with a family of isomorphisms $\mathbf{C}(h \odot A, B) \cong \mathbf{H}(h, \mathbf{C}(A, B))$, natural in B.

The relevance of **Set**-enriched copowers to quantum algorithms has previously been suggested by Jacobs [11]. On the other hand, copowers and enrichment play a key role in the non-quantum enriched effect calculus [7,19] and other areas [16,18,23,29]. Nonetheless, our connection with the EWire syntax appears to be novel.

Definition 3.1 A *categorical model of EWire* $(\mathbf{H}, \mathbf{H}_0, \mathbf{C}, T)$ is given by the following data:

(i) A cartesian closed category $\mathbf{H}$ with a strong monad T on $\mathbf{H}$. This is needed to interpret the host language.

(ii) A small full subcategory $j : \mathbf{H}_0 \subseteq \mathbf{H}$. The idea is that the objects of $\mathbf{H}_0$ interpret the first order host types, equivalently, the classical wire types: the types that exist in both the host language and the circuit language.

(iii) An $\mathbf{H}$-enriched symmetric monoidal category $(\mathbf{C}, \otimes, I)$. This allows us to interpret the circuit language, and the $\mathbf{H}$-enrichment allows us to understand the host types $\mathrm{Circ}(W, W')$.

(iv) The category $\mathbf{C}$ has copowers by the objects of $\mathbf{H}_0$. The copower induces a functor $J : \mathbf{H}_0 \to \mathbf{C}$ defined by $J(h) = h \odot I$. Then, we have a natural isomorphism

$$\mathbf{C}(J(h), C) = \mathbf{C}(h \odot I, C) \cong \mathbf{H}(j(h), \mathbf{C}(I, C))$$

and therefore a j-relative adjunction $J \;{}_j\!\dashv \mathbf{C}(I, -)$ between circuits and (host) terms. This functor $J : \mathbf{H}_0 \to \mathbf{C}$ interprets the translation between first order host types and classical wire types.

(v) For each object A of $\mathbf{C}$, the functor $A \otimes - : \mathbf{C} \to \mathbf{C}$ preserves copowers. This makes the functor J symmetric monoidal, and makes the relative adjunction an enriched relative adjunction.

(vi) There is an enriched relative monad morphism

$$\mathrm{run}_h : \mathbf{C}(I, J(h)) \to T(j(h))$$

where the enriched relative monad $\mathbf{C}(I, J(-)) : \mathbf{H}_0 \to \mathbf{H}$ is induced by the enriched j-relative adjunction $J \;{}_j\!\dashv \mathbf{C}(I, -)$. This is the interpretation of running a quantum circuit, producing some classical probabilistic outcome.

If the category $\mathbf{C}$ has a given object $[\![\alpha]\!]$ for each basic quantum wire type α, and $\mathbf{H}_0$ has a given object $[\![\mathbf{a}]\!]$ for each basic classical wire type $\mathbf{a}$, then we can interpret all wire types W as objects of $\mathbf{C}$:

$$[\![1]\!] \overset{\mathrm{def}}{=} I \quad [\![\mathbf{a}]\!] \overset{\mathrm{def}}{=} J([\![\mathbf{a}]\!]) \quad [\![W \otimes W']\!] \overset{\mathrm{def}}{=} [\![W]\!] \otimes [\![W']\!].$$

If the category $\mathbf{C}$ also has a given morphism $[\![g]\!] : [\![W_1]\!] \to [\![W_2]\!]$ for every gate $g \in \mathcal{G}(W_1, W_2)$, then we can interpret the circuit langauge inside $\mathbf{C}$.

In light of those axioms, and to every categorical model of EWire, we associate the following denotational semantics. First, we define as promised the denotation of the host type $\mathrm{Circ}(W, W')$ by $[\![\mathrm{Circ}(W, W')]\!] \overset{\mathrm{def}}{=} \mathbf{C}(W, W')$, an object of the category $\mathbf{H}$. The semantics of the other host types is given as follows:

$$[\![1]\!] \overset{\mathrm{def}}{=} 1 \quad [\![A \times A']\!] \overset{\mathrm{def}}{=} [\![A]\!] \times [\![A']\!] \quad [\![A \to A']\!] \overset{\mathrm{def}}{=} ([\![A]\!] \to [\![A']\!]) \quad [\![T(A)]\!] \overset{\mathrm{def}}{=} T([\![A]\!]).$$

Ordered context of wires Ω have the following semantics:

$$[\![\langle\cdot\rangle]\!] = I \qquad [\![w : W]\!] = [\![W]\!] \qquad [\![\Omega, \Omega']\!] = [\![\Omega]\!] \otimes [\![\Omega']\!]$$

A circuit judgement $\Gamma; \Omega \vdash t : W$ is denoted by

$$[\![\Gamma; \Omega \vdash t : W]\!] \in \mathbf{H}([\![\Gamma]\!], \mathbf{C}([\![\Omega]\!], [\![W]\!]))$$

relying on the assumption that the category $\mathbf{H}$ is a model of the host language. A host type $\mathrm{Circ}(W, W')$ is interpreted as the hom-object $\mathbf{C}([\![W]\!], [\![W']\!])$, in the category $\mathbf{H}$. In this setting, denotations of boxing and unboxing instructions are trivial. Indeed, notice that whenever $\Omega \implies p : W_1$ holds, we have $[\![\Omega]\!] \cong [\![W_1]\!]$, and we put

$$[\![\Gamma \vdash \mathbf{box}\ (p : W_1) \Rightarrow C : \mathrm{Circ}(W_1, W_2)]\!] = [\![\Gamma; \Omega \vdash C : W_2]\!]$$

$$[\![\Gamma; \Omega \vdash \mathbf{unbox}\ t\, p : W_2]\!] = [\![\Gamma \vdash t : \mathrm{Circ}(W_1, W_2)]\!]$$

The denotation of **output** $p : W$ is the identity when the type of the pattern p is not a sum type, and is the i-th projection when p is of the form $\mathbf{in}_i\, p'$. Moreover, instructions $\Gamma; \Omega, \Omega' \vdash () \leftarrow p; C : W$ and $\Gamma; \Omega' \vdash C : W$ (resp. $\Gamma; \Omega, \Omega' \vdash (w_1, w_2) \leftarrow p; C : W$ and $\Gamma; w_1 : W_1, w_2 : W_2, \Omega' \vdash C : W$) have isomorphic denotations whenever $\Omega \implies p : 1$ holds (resp. $\Omega \implies p : W_1 \otimes W_2$ holds).

The lift construction is interpreted by the copower. In detail, for every object h of $\mathbf{H}$, and every object h' of $\mathbf{H}_0$, we consider the isomorphism

$$\mathrm{lift}_h : \mathbf{H}(h \times h', \mathbf{C}(X, Y)) \cong \mathbf{H}(h, \mathbf{H}(h', \mathbf{C}(X, Y))) \cong \mathbf{H}(h, \mathbf{C}(h' \odot X, Y))$$

Since we're enforcing explicit measurement here, the denotation of the operation **run** for a circuit C whose output wire type is the type W is given by Def. 3.1(vi).

The denotations of the remaining instructions are given as follows.

$$[\![\Gamma; \Omega_1, \Omega \vdash p_2 \leftarrow \mathbf{gate}\ g\, p_1; C : W]\!] = [\![\Gamma; \Omega_2, \Omega \vdash C : W]\!] \circ ([\![g]\!] \otimes \mathrm{id})$$

$$[\![\Gamma; \Omega_1, \Omega_2 \vdash p \leftarrow C; C' : W']\!] = [\![\Gamma; \Omega, \Omega_2 \vdash C' : W']\!] \circ ([\![\Omega_1 \vdash C : W]\!] \otimes \mathrm{id})$$

Consider the operational semantics given in Appendix B. Assuming that $\mathbf{H}$ is a sound categorical model of the host language, one obtains the following theorem by straightforward induction on typing judgements. (This is similar to the proof in [22, App. B].)

Theorem 3.2 (Soundness) *For every denotational semantics induced by a categorical model of EWire, if the circuit judgement $\cdot; \Omega \vdash C : W$ holds and the circuit C reduces to a circuit C', then $[\![\cdot; \Omega \vdash C : W]\!] = [\![\cdot; \Omega \vdash C' : W]\!]$.*

It is now time to elaborate an example. Our view on the semantics of quantum computing relies on the theory of C*-algebras. The positive elements of C*-algebras correspond to observables in quantum theory, and we understand quantum computations as linear maps that preserve positive elements, in other words, 'observable transformers'. Circuits $(\cdot; (x : W) \vdash C : W')$ will be interpreted as completely positive unital maps $[\![W']\!] \to [\![W]\!]$. The reverse direction is in common with predicate transformer semantics for conventional programming.

In short, a *(unital) C*-algebra* (e.g. [27]) is a vector space over the field of complex numbers that also has multiplication, a unit and an involution, satisfying associativity and unit laws for multiplication, involution laws (e.g. $x^{**} = x$, $(xy)^* = y^*x^*$, $(\alpha x)^* = \bar{\alpha}(x^*)$) and such that the spectral radius provides a norm making it a Banach space.

There are two crucial constructions of C*-algebras: matrix algebras and direct sums. *Matrix algebras* provide a crucial example of C*-algebras. For example, the algebra M_2 of 2×2 complex matrices represents the type of qubits. The *direct sum* of two C*-algebras, $A \oplus B$, is the set of pairs with componentwise algebra structure. For instance, $\mathbb{C} \oplus \mathbb{C}$ represents the type of classical bits. Every finite-dimensional C*-algebra is a direct sum of matrix algebras.

The tensor product $\otimes$ of finite dimensional C*-algebras is uniquely determined by two properties: (i) that $M_k \otimes M_l \cong M_{k \times l}$, and (ii) that $A \otimes (-)$ and $(-) \otimes B$ preserve direct sums. In particular $M_k \otimes A$ is isomorphic to the algebra of $(k \times k)$-matrices valued in A.

We do not focus here on linear maps that preserve all of the C*-algebra structure, but rather on completely positive maps. An element $x \in A$ is *positive* if it can be written in the form $x = y^*y$ for $y \in A$. These elements correspond to quantum observables. A map $f : A \to B$, linear between the underlying vector spaces, is *positive* if it preserves positive elements. A linear map is *unital* if it preserves the multiplicative unit. A linear map f is *completely positive* if the map $(M_k \otimes f) : M_k \otimes A \to M_k \otimes B$ is positive for every k. This enables us to define a functor $C \otimes (-)$ for every finite dimensional C*-algebra C. Thus finite dimensional C*-algebras and completely positive unital linear maps form a symmetric monoidal category. There are completely positive unital maps corresponding to initializing quantum data, performing unitary rotations, and measurement, and in fact all completely positive unital maps arise in this way (e.g. [28,30]).

Proposition 3.3 *The triplet* $(\mathbf{FdC^*\text{-}Alg}_{\mathrm{CPU}}^{\mathrm{op}}, \mathbf{Set}, \mathcal{D})$ *is a model of EWire, formed by the category* $\mathbf{FdC^*\text{-}Alg}_{\mathrm{CPU}}$ *of finite-dimensional C*-algebras and completely positive unital maps, the cartesian closed category* **Set** *of sets and functions, and the probability distribution monad* $\mathcal{D}$ *over* **Set**. *In fact it is a model of QWire, with* $[\![qubit]\!] \stackrel{def}{=} M_2$ *and* $[\![bit]\!] \stackrel{def}{=} \mathbb{C} \oplus \mathbb{C}$.

Proof. See Appendix C. □

Steps towards subsets and variations on quantum computation

Completely positive maps between C*-algebras allow for all quantum operations, but sometimes one would focus on a variation of quantum computation, or a restricted set of gates, such as the stabiliser gates.

Definition 3.4 A *category of quantum computation* is a subcategory **Q** of the category $\mathbf{C^*\text{-}Alg}_{\mathrm{CPU}}$ of C*-algebras together with completely positive unital maps. For the sake of coherence, we also require:

(i) Initiality: the C*-algebra $\mathbb{C}$ of complex numbers is in $\mathbf{Q}$.

(ii) Closure under matrix algebras: if A is in $\mathbf{Q}$ then so is $M_n \otimes A$ ($n \in \mathbb{N}$).

(iii) Closure under matrices of morphisms: for every pair (A, B) of C*-algebras in $\mathbf{Q}$, if the map f in $\mathbf{Q}(A, B)$ then the map $(M_n \otimes f)$ in $\mathbf{Q}(M_n \otimes A, M_n \otimes B)$.

Different choices for the category $\mathbf{Q}$ give different classes of states and unitaries for our language, making $\mathbf{Q}$ the 'categorical signature' of the subset of quantum mechanics associated with the collection $\mathcal{G}$ of gates which parametrize QWire.

For example, we define a category of quantum computation which only contains matrix algebras and the completely positive unital maps generated by the stabilizer states of the Clifford group [26, Sec. 1.3]. We call this category **Clifford**; it corresponds to stabilizer quantum mechanics [2], which can be efficiently simulated by classical computers.

Then, considering that the single-qubit Clifford group together with the gate $T = \begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{pmatrix}$ can approximate any single-qubit unitary up to an arbitrary accuracy [5], adding the completely positive unital map $T^* - T$ generated by the gate T to the maps of the category **Clifford** forms a category **CliffordT** that is arguably the smallest category which corresponds to a reasonably complete subset of quantum theory.

4 A step towards quantum domain theory

A W*-algebra is an unital C*-algebra A whose unit interval is a dcpo, with sufficiently many normal states, i.e. normal completely positive unital maps $A \to \mathbb{C}$. We write $\mathbf{W^*\text{-}Alg}_{\mathrm{CPSU}}$ for the category of W*-algebras and completely positive subunital maps, which is known for being $\mathbf{Dcpo}_\perp$-enriched (see e.g. [24]), where $\mathbf{Dcpo}_\perp$ is the category of pointed dcpos and Scott-continuous maps.

In fact, its opposite category is part of a categorical model of QWire (as shown in Appendix C), when one considers the restricted version of the monad of subvaluations $V = \mathbf{dcGEMod}([0,1]^{(-)}, [0,1])$ on $\mathbf{Dcpo}_\perp$ (see e.g. [12, Section 5.6]), where the category **dcGEMod** is the category of directed-complete generalized effect modules and Scott-continuous effect module homomorphisms, also introduced as a category of quantum predicates in [25,26].

Proposition 4.1 *$(\mathbf{W^*\text{-}Alg}^{\mathrm{op}}_{\mathrm{CPSU}}, \mathbf{Dcpo}_\perp, V)$ is a categorical model of QWire.*

The remaining part of this section is devoted to an investigation of the extra syntax supported by the category $\mathbf{W^*\text{-}Alg}^{\mathrm{op}}_{\mathrm{CPSU}}$. We argue that the present work constitutes a milestone for the development of quantum domain theory. The interested reader will find the proofs in Appendix C.

4.1 Extensions of EWire

Let us part away from the traditional notion of circuits in order to be able to deal with inputs and outputs of sum types and recursive types: in what follows, circuits are first-order linear instructions. Therefore, the structures that our circuit language manipulate are not *per se* circuits but generalised circuits.

Conditional branching

To extend EWire with conditional branching, one needs to introduce sum types (that is, $W ::= \cdots \mid W \oplus W'$ and $A ::= \cdots \mid A + A'$), and gates $\mathrm{in}_1 \in \mathcal{G}(W_1, W_1 \oplus W_2)$ and $\mathrm{in}_2 \in \mathcal{G}(W_2, W_1 \oplus W_2)$ for every pair of types W_1 and W_2. Additionally, we introduce case expressions

$$\mathbf{case}\ p\ \mathbf{of}\ (\mathbf{in}_1\, w_1 \to C_1 \mid \mathbf{in}_2\, w_2 \to C_2) : W$$

(where $p : W_1 \oplus W_2$, $C_1 : \mathrm{Circ}(W_1, W)$ and $C_2 : \mathrm{Circ}(W_2, W)$) and extend the grammar of patterns:

$$p ::= \cdots \mid \mathbf{in}_1\, p \mid \mathbf{in}_2\, p$$

Then, bit is $1 \oplus 1$. A wire type $W \oplus W'$ is classical if W and W' are classical, and then $|W \oplus W'| \stackrel{\text{def}}{=} |W| + |W'|$.

Patterns of sums are eliminated using $\dfrac{\Omega \implies p : W_i}{\Omega \implies \mathbf{in}_i\, p : W_1 \oplus W_2}\ i \in \{1, 2\}$ and the typing rule for branching is the following:

$$\frac{\Omega \implies p : W_1 \oplus W_2 \qquad \Gamma; w_1 : W_1, \Omega' \vdash C_1 : W \qquad \Gamma; w_2 : W_2, \Omega' \vdash C_2 : W}{\Gamma; \Omega, \Omega' \vdash \mathbf{case}\ p\ \mathbf{of}\ (\mathbf{in}_1\, w_1 \to C_1 \mid \mathbf{in}_2\, w_2 \to C_2) : W}$$

Recursive types

Let us complete the grammars of wire types and host types:

$$W ::= \cdots \mid X \mid \mu X.W \qquad A ::= \cdots \mid X \mid \mu X.A$$

A wire type $\mu X.W$ is classical if W is, and $|\mu X.W| \stackrel{\text{def}}{=} \mu X.\,|W|$.

We assume that $\mathcal{G}$ contains gates

$$\mathrm{fold}_{\mu X.W} \in \mathcal{G}(W[X \mapsto \mu X.W], \mu X.W) \quad \mathrm{unfold}_{\mu X.W} \in \mathcal{G}(\mu X.W, W[X \mapsto \mu X.W])$$

which corresponds to the folding/unfolding of a recursive type $\mu X.W$. For example, for the type QList of quantum lists defined by $\mathrm{QList} = \mu X.\mathrm{qubit} \otimes X \oplus 1$, we have $\mathrm{fold} \in \mathcal{G}(\mathrm{qubit} \otimes \mathrm{QList} \oplus 1, \mathrm{QList})$ and $\mathrm{unfold} \in \mathcal{G}(\mathrm{QList}, \mathrm{qubit} \otimes \mathrm{QList} \oplus 1)$. As another example, $\mathrm{QNat} \stackrel{\text{def}}{=} \mu X.X \oplus 1$ is a wire type of natural numbers and $\mathrm{Nat} = \mu X.X + 1$ is the host type of natural numbers. The type QNat is classical and $|\mathrm{QNat}| \stackrel{\text{def}}{=} \mathrm{Nat}$.

In line with [8], we introduce the recursive typing rules as type judgements $\Theta \vdash \tau$, which entail that the type τ is well-formed with respect to the context of distinct type variables Θ. We introduce the following set of rules for typing judgements:

$$\frac{}{\Theta \vdash 1} \qquad \frac{}{\Theta, X, \Theta' \vdash X} \qquad \frac{\Theta \vdash W_1 \quad \Theta \vdash W_2}{\Theta \vdash W_1 \circledast W_2} \circledast \in \{\otimes, \oplus\} \qquad \frac{\Theta, X \vdash W}{\Theta \vdash \mu X.W}$$

The typing judgement $\Theta \vdash \Gamma$ holds whenever Γ is a context of language variables such that $\Theta \vdash \tau$ holds for every variable $(x : \tau) \in \Gamma$.

At this point, one might question the interest for recursive types in a circuit-based language. In the traditional conceptualization of circuits, lists and other infinite data types must be instantiated at a specific length to be used as the input type of a circuit and therefore (iso)recursive types cannot appear in the wire types of a circuit.

Let us illustrate our interest for patterns of recursive types by focusing on the pattern p : QList. We want to implement the Quantum Fourier Transform. Taking inspiration from [22, Sec. 6.2] and [10], we assume a host language constant **CR** : Nat $\to$ Circ(qubit $\otimes$ qubit, qubit $\otimes$ qubit), so that (**CR** n) corresponds to the controlled rotation by $\frac{2\pi}{2^m}$ around the z-axis. Then the program rotations performs the rotations of a QFT circuit and the instruction (fourier) corresponds to the QFT, as illustrated in the circuit in the introduction.

```
length :
 Circ(QList,QNat⊗QList) =
box qs =>
  case qs of [] => output (0,[])
    | (q:qs') =>
      (n,qs') <- unbox length qs';
      output (S n,q:qs')
```

```
rotations :
 Nat -> Circ(qubit⊗QList,qubit⊗QList) =
lambda m. box (c,qs) =>
case qs of [] => output (c,[])
   | (q:qs') =>
    (n,qs') <- unbox length qs' ;
    n <= lift n ;
    (c,qs') <- unbox (rotations m) (c,qs') ;
    (c,q) <- unbox (CR(1+m-n)) (c,q) ;
    output (c,(q:qs'))
```

```
fourier : Circ(QList, QList) =
box qs =>
  case qs of [] => []
       | (q:qs') =>
            qs' <- unbox fourier qs' ;
            (n,qs') <- unbox length qs' ;
            n <= lift n ;
            (q,qs') <- unbox (rotations n) (q,qs')
            q <- gate H q ; output (q,qs')
```

Here we are using some standard syntactic sugar for recursive types and lists. For example, `length` is more verbosely written

```
   Y (lambda l. box qs =>
        qs <- unfold qs ;
        case qs of
            in2() => qs <- gate in1 () ; qs <- gate fold qs ;
                     z <- gate in1 () ; z <- gate fold z ;
                     output (z,qs)
          | in1(q,qs) =>
              (n,qs) <- unbox l qs ;
              qqs <- gate in1 (q,qs) ; qqs <- gate fold qqs ;
              n' <- gate in2 n ; n' <- gate fold n ;
              output (n',qqs) )
```

where Y is a fixed point combinator, defined using recursive types. (This standard QFT algorithm leaves the list in reverse order, and so for many purposes this program must be composed with a standard list reversal program, omitted here.)

Here, we consider recursive types because they are a quick way of introducing operations over lists, and these quickly fit into our categorical formalism. The drawback is that in quantum programming, it is useful to make the lengths of lists more explicit. For example, the type system does not tell us that (**unbox** `fourier qs`) has the same length as `qs`. This is a familiar problem in a functional language such as Haskell, but it is particularly inconvenient for a quantum circuit layout engine. A good way to deal with it would be through some kind of dependent types [22, 6.2], for instance allowing a type $\mathrm{QArray}(n)$ of arrays of qubits of size n and $\mathrm{fourier} : (n : \mathrm{Nat}) \to \mathrm{Circ}(\mathrm{QArray}(n), \mathrm{QArray}(n))$. However, what follows discusses categorical models of QWire in which one can denote recursive types, exploiting a presheaf-theoretic semantics which we intend to use as a foundation for a theory of quantum domains. Therefore, integrating dependent types to EWire/QWire and associating such types to an appropriate categorical semantics is left for future work.

4.2 Towards quantum domains

The notion of algebraic compactness provides a way to interpret recursive types. A $\mathbf{Dcpo}_{\perp!}$-enriched category $\mathbf{C}$ is *algebraically compact* [3] for locally continuous endofunctors if every locally continuous endofunctor F on $\mathbf{C}$ (i.e. F is such that all $F_{X,Y} : \mathbf{C}(X,Y) \to \mathbf{C}(FX,FY)$ are Scott-continuous) has a canonical fixpoint written μF, which is the initial F-algebra (where $\mathbf{Dcpo}_{\perp!}$ is the category of pointed dcpos and strict Scott-continuous maps).

Every algebraically compact category $\mathbf{C}$, as part of a categorical model $(\mathbf{C}, \mathbf{H}, T)$, is a domain-theoretic model of FPC [8, Def. 6.7] and as such provides a computationally adequate model for the language FPC, a functional programming language with recursive types [8, Th. 7.14].

Consider that every type judgement $\Theta \vdash W$ is denoted by a locally continuous functor $\mathbf{C}^n \to \mathbf{C}$, defined by inductions as follows:

$$[\![\Theta \vdash W]\!](\chi) = [\![W]\!] \text{ when } W \in \{1, \mathrm{bit}, \mathrm{qubit}\} \quad [\![\Theta \vdash X]\!] = \mathrm{Id}$$

$$[\![\Theta \vdash W_1 \star W_2]\!](\chi) = [\![\Theta \vdash W_1]\!](\chi) \star [\![\Theta \vdash W_2]\!](\chi) \text{ with } \star \in \{\otimes, \oplus\}$$

$$[\![\Theta \vdash \mu X.W]\!] = \mu[\![\Theta, X \vdash W]\!]$$

The algebraic compactness of the category of W*-algebras together with (completely) positive sub-unital maps has already been established [24]. This result establishes W*-algebras and completely positive unital maps as a categorical model of higher-order quantum computing. For example, the type QList is denoted by the fixpoint $\oplus_{k \geq 0} M_{2^k}$ of the endofunctor $F : X \mapsto X \otimes M_2 \oplus 1$.

Ultimately, we are interested in quantum domains, that is in finding models and convenient programming languages that mix quantum features and classical features more generally. Enriched category theory offers a compelling way to elevate ourselves from the theory of W*-algebras.

Indeed, whenever a category $\mathbf{C}$ is enriched over a category $\mathbf{H}$, one can form a category of $\mathbf{H}$-enriched presheaves $[\mathbf{C}^{\mathrm{op}}, \mathbf{H}]$, which inherits some of the structure of $\mathbf{H}$ and $\mathbf{C}$. In particular, when a category $\mathbf{C}$ is enriched over a category $\mathbf{Dcpo}$, order-enriched presheaves $\mathbf{C}^{\mathrm{op}} \to \mathbf{Dcpo}$ are the objects of the free colimit completion of the category $\mathbf{C}$ as a $\mathbf{Dcpo}$-enriched category. We recall that Malherbe et al. [17] also used presheaves in their steps towards a model of a higher-order quantum programming language.

Definition 4.2 Consider a categorical model of QWire $(\mathbf{C}, \mathbf{Dcpo}, T)$. A *quantum predomain* is an order-enriched contravariant presheaf $F : \mathbf{C}^{\mathrm{op}} \to \mathbf{Dcpo}$. A *quantum domain* is a quantum predomain whose *root* $F(1)$ is a pointed dcpo, i.e. a dcpo which has a least element.

The literature is rich of examples of dcpo structures over quantum systems, upon which quantum (pre)domains can be built. Normal states $\mathcal{NS}(A)$ and predicates $[0,1]_A$ of a C*-algebra A form dcpos whenever A is a W*-algebra. Projections $\mathrm{Proj}(A)$ on a finite-dimensional C*-algebra A, which are elements p of A which are self-adjoint (i.e., $p = p^*$) and idempotent (i.e., $p = p^2$), are continuous lattices, used in von Neumann's quantum logic. Furthermore, one can consider the dcpo $\mathcal{C}(A)$ of commutative C*-subalgebras of a C*-algebra A (seen as *classical views* of a quantum system) ordered by inclusion.

The following proposition will not come as a surprise to the reader familiar with enriched categories: mostly, it follows from the fact that the Yoneda embedding is full and faithful.

Proposition 4.3 *Consider a categorical model of QWire* $(\mathbf{C}, \mathbf{Dcpo}, T)$ *where* $\mathbf{C}$ *is a small category of quantum computation. Then, the triplet* $([\mathbf{C}^{\mathrm{op}}, \mathbf{Dcpo}], \mathbf{Dcpo}, T)$ *is a categorical model of QWire, where* $[\mathbf{C}^{\mathrm{op}}, \mathbf{Dcpo}]$ *is algebraically compact for locally continuous endofunctors.*

Consequently, the triplet $([\mathbf{FdC^*\text{-}Alg}^{\mathrm{op}}_{\mathrm{CPSU}}, \mathbf{Dcpo}], \mathbf{Dcpo}, V)$ is a categorical model of QWire, which will be investigated in future work.

Summary. We have introduced a new calculus, EWire (Sec. 2), for embedded circuits within an expressive host language. The language includes QWire as an instance (Sec. 2.2). We have proposed a notion of categorical model for EWire (Sec. 3) in which the relationship between the circuit and host language is explained in terms of enriched categories. Our first example of a model is based on the Set-enrichment of C*-algebras. Finally by considering a model of QWire based on W*-algebras and dcpos, we have introduced some recursive types to give a denotational semantics to the Quantum Fourier Transform.

Acknowledgement

The authors would like to thank Bart Jacobs, Michele Pagani and Shane Mansfield for helpful discussions, but also Bert Lindenhovius, Michael Mislove and Vladimir Zamdzhiev for hosting the first author at Tulane University during the elaboration of this work, and Jennifer Paykin for introducing us to the subtle aspects of QWire. The research leading to these results has received funding from the ERC grant agreement n. 320571, a Royal Society fellowship, and EPSRC grant EP/N007387/1.

References

[1] Thorsten Altenkirch, James Chapman, and Tarmo Uustalu. Monads need not be endofunctors. In *Proc. FOSSACS'10*, pages 297–311. Springer, 2010.

[2] Miriam Backens. The zx-calculus is complete for stabilizer quantum mechanics. *New Journal of Physics*, 16(9):093021, 2014.

[3] Michael Barr. Algebraically compact functors. *Journal of Pure and Applied Algebra*, 82(3):211–231, 1992.

[4] Clemens Berger, Paul-André Melliès, and Mark Weber. Monads with arities and their associated theories. *J. Pure Appl. Algebra*, 216(8-9):2029–2048, 2012.

[5] P. O. Boykin, T. Mor, M. Pulver, V. Roychowdhury, and F. Vatan. On universal and fault-tolerant quantum computing: a novel basis and a new constructive proof of universality for shor's basis. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 486–494, 1999.

[6] Brian Day. On closed categories of functors. In *Reports of the Midwest Category Seminar IV*, pages 1–38. Springer, 1970.

[7] Jeff Egger, Rasmus Ejlers Møgelberg, and Alex Simpson. The enriched effect calculus: syntax and semantics. *Journal of Logic and Computation*, 2012.

[8] Marcelo P Fiore and Gordon D Plotkin. An axiomatisation of computationally adequate domain theoretic models of FPC. In *Proc. LICS'94*, 1994.

[9] Robert Furber and Bart Jacobs. From Kleisli categories to commutative C*-algebras: probabilistic Gelfand duality. In *Proc. CALCO'13*, pages 141–157. Springer, 2013.

[10] Alexander S Green, Peter LeFanu Lumsdaine, Neil J Ross, Peter Selinger, and Benoît Valiron. Quipper: a scalable quantum programming language. In *Proc. PLDI'13*, pages 333–342. ACM, 2013.

[11] Bart Jacobs. On block structures in quantum computation. In *Proc. MFPS'13*, volume 298 of *Electron. Notes Theor. Comput. Sci.*, pages 233–255. Elsevier, 2013.

[12] Bart Jacobs. A recipe for state-and-effect triangles. In *Proc. CALCO'15*, volume 35, 2015.

[13] Max Kelly. *Basic concepts of enriched category theory*, volume 64. CUP Archive, 1982.

[14] Aleks Kissinger and Sander Uijlen. A categorical semantics for causal structure. In *Proc. LICS'17*, 2017.

[15] Andre Kornell. Quantum collections. *arXiv preprint arXiv:1202.2994*, 2012.

[16] Paul Blain Levy. *Call-by-push-value: A functional/imperative Synthesis*, volume 2. Springer Science & Business Media, 2012.

[17] Octavio Malherbe, Philip Scott, and Peter Selinger. Presheaf models of quantum computation: an outline. In *Computation, Logic, Games, and Quantum Foundations. The Many Facets of Samson Abramsky*, pages 178–194. Springer, 2013.

[18] Paul-André Melliès. Parametric monads and enriched adjunctions. Available from the author's webpage, 2012.

[19] R. E. Møgelberg and Sam Staton. Linear usage of state. *Logical Methods in Computer Science*, 10(1), 2014.

[20] Eugenio Moggi. Computational lambda-calculus and monads. In *Proc. LICS'89*, 1989.

[21] Michele Pagani, Peter Selinger, and Benoît Valiron. Applying quantitative semantics to higher-order quantum computing. In *Proc. POPL'14*, pages 647–658. ACM, 2014.

[22] Jennifer Paykin, Robert Rand, and Steve Zdancewic. QWIRE: a core language for quantum circuits. In *Proc. POPL'17*, pages 846–858. ACM, 2017.

[23] A J Power. Enriched Lawvere theories. *Theory Appl. Categories*, 6(7):83–93, 1999.

[24] Mathys Rennela. Towards a quantum domain theory: order-enrichment and fixpoints in W*-algebras. In *Proc. MFPS XXX*, volume 308, pages 289–307. Electron. Notes Theoret. Comput. Sci., 2014.

[25] Mathys Rennela. Operator algebras in quantum computation. *arXiv preprint arXiv:1510.06649*, 2015.

[26] Mathys Rennela and Sam Staton. Complete positivity and natural representation of quantum computations. In *Proc. MFPS XXXI*, volume 319, pages 369–385. Electron. Notes Theoret. Comput. Sci., 2015.

[27] Shôichirô Sakai. *C*-algebras and W*-algebras*. Springer Science & Business Media, 2012.

[28] Peter Selinger. Towards a quantum programming language. *Math. Struct. Comput. Sci.*, 14(04):527–586, 2004.

[29] Sam Staton. Freyd categories are enriched Lawvere theories. In *Proc. WACT*, Electron. Notes Theor. Comput. Sci., pages 197–206, 2013.

[30] Sam Staton. Algebraic effects, linearity, and quantum programming languages. In *Proc. POPL'15*, pages 395–406. ACM, 2015.

[31] Dave Wecker and Krysta M Svore. LIQUi|>: A software design architecture and domain-specific language for quantum computing. *arXiv preprint arXiv:1402.4467*, 2014.

A Additional typing rules for the host language

Recall that the types of the host language are

$$A, B \ ::= \ A \times B \mid 1 \mid A \to B \mid T(A) \mid \mathrm{Circ}(W, W') \mid \mathbf{a} \mid \mathbf{b}$$

The standard typing rules of the monadic metalanguage are the rules of the simply-typed λ-calculus

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash (\lambda x^A.t) : A \to B} \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t(u) : B}$$

Terms of product types are formed following four typing rules

$$\frac{-}{\Gamma \vdash \mathrm{unit} : 1} \qquad \frac{\Gamma \vdash t : A \quad \Gamma \vdash u : B}{\Gamma \vdash (t, u) : A \times B} \qquad \frac{\Gamma \vdash t : A \times B}{\Gamma \vdash \pi_1(t) : A} \qquad \frac{\Gamma \vdash t : A \times B}{\Gamma \vdash \pi_2(t) : B}$$

to which we need to add the typing rules for the monad [20], associated respectively to the unit and the strong Kleisli composition:

$$\frac{\Gamma \vdash t : B}{\Gamma \vdash \mathbf{return}(t) : T(B)} \qquad \frac{\Gamma \vdash t : T(B) \qquad \Gamma, x : B \vdash u : T(C)}{\Gamma \vdash \mathbf{let}\, x = t\, \mathbf{in}\, u : T(C)}$$

This is in addition to the typing rules for the interaction between the host language and the circuit language given in Section 2.

B Operational semantics of EWire

Let us start with the (partial) destruction of patterns.

$$\frac{}{p \leftarrow \mathbf{output}\ p'; C \implies C[p \mapsto p']} \qquad \frac{}{\mathbf{unbox}\ (\mathbf{box}\ w \Rightarrow C)\, p \implies C[w \mapsto p]}$$

$$\frac{}{() \leftarrow (); C \implies C} \qquad \frac{}{(w_1, w_2) \leftarrow (p_1, p_2); C \implies C[{}^{w_1\ \mapsto p_1}_{w_2\ \mapsto p_2}]}$$

The reduction system $\rightarrow$ on terms is defined in two parts: the reduction $\rightarrow_H$ is the operational semantics of the host language, while the reduction $\rightarrow_b$ is the operational semantics of boxed circuits (we refer to [22, Sec. 4] for its description), in such a way that $\rightarrow = \rightarrow_H \cup \rightarrow_b$. This allows us to define the following reduction rules for boxing and unboxing instructions.

$$\frac{C \implies C'}{\mathbf{box}\ (w : W) \Rightarrow C \rightarrow_b \mathbf{box}\ (w : W) \Rightarrow C'} \qquad \frac{t \rightarrow t'}{\mathbf{unbox}\ t\, p \Rightarrow \mathbf{unbox}\ t'\, p}$$

Then, we add the structural reduction rule $\dfrac{C \implies C'}{E[C] \implies E[C']}$ with

$$E ::= w \leftarrow \square; C \mid w \leftarrow \mathbf{gate}\ g\, p\ \mathbf{in}\ \square \mid () \leftarrow p; \square \mid (w_1, w_2) \leftarrow p; \square$$

Therefore, normal circuits are given by the following grammar:

$$N ::= \mathbf{output}\ p \mid w \leftarrow \mathbf{gate}\ g\, p; N \mid x \Leftarrow \mathbf{lift}\ p; N \mid () \leftarrow w; N \mid (w_1, w_2) \leftarrow w; N$$

Finally, commuting conversion rules allows to reduce even more instructions to normal forms, ensuring preservation, progress and normalization.

$$\frac{}{w \leftarrow (p_2 \leftarrow \mathbf{gate}\ g\, p_1; N); C \implies p_2 \leftarrow \mathbf{gate}\ g\, p_1; w \leftarrow N; C}$$

$$\frac{}{w \leftarrow (x \Leftarrow \mathbf{lift}\ p; C'); C \implies x \Leftarrow \mathbf{lift}\ p; w \leftarrow C'; C}$$

$$\frac{}{w \leftarrow (() \leftarrow w'; N); C \implies () \leftarrow w'; w \leftarrow N; C}$$

$$\frac{}{w \leftarrow ((w_1, w_2) \leftarrow w'; N); C \implies (w_1, w_2) \leftarrow w'; w \leftarrow N; C}$$

The following propositions are proven by straightforward induction. In particular, assuming that the reduction $\rightarrow_H$ satisfies preservation, progress and strong normalization one can deduce that the reduction $\rightarrow$ does too. The interested reader will find the proofs of the following propositions in [22, App. A].

Proposition B.1 (Preservation) *If the judgement $\Gamma; \Omega \vdash C : W$ holds and the circuit C reduces to the circuit C', then the judgement $\Gamma; \Omega \vdash C' : W$ holds.*

Proposition B.2 (Progress) *If the judgement $\Gamma;\Omega \vdash C : W$ holds, then either the circuit C is normal or there is a circuit C' such that C reduces to C', i.e. the relation $C \implies C'$ holds.*

Proposition B.3 (Normalization) *If the judgement $\Gamma;\Omega \vdash C : W$ holds, then there exists a normal circuit N such that the circuit C reduces to N in a finite number of steps.*

C Omitted proofs

C.1 Proof of Proposition 3.3

The category **Set** of sets and functions is the canonical example of cartesian closed category, and the distribution monad $\mathcal{D} : \mathbf{Set} \to \mathbf{Set}$ is a strong monad.

The category $\mathbf{FdC^*\text{-}Alg}^{\mathrm{op}}_{\mathrm{CPU}}$ of the opposite category of finite-dimensional C*-algebras and completely positive unital maps has a monoidal structure given by the tensor product of C*-algebras, finite sums given by direct sums and the C*-algebra $\mathbb{C}$ of complex numbers is the unit I.

In this setting, $\mathbf{H}_0$ is the category $\mathbb{N}$, skeleton of the category of finite sets and functions which considers natural numbers as its objects. The copower $n \odot A$ of a natural number $n \in \mathbb{N}$ and a C*-algebra A is the C*-algebra $n \odot A$, defined as the n-fold direct sum $A \oplus \cdots \oplus A$ like in [11]. We observe that the copower distributes over the coproduct ($n \odot (A \oplus B) = n \odot A \oplus n \odot B$) and that composition is multiplication ($n \odot (m \odot A) = nm \odot B$).

The copower $n \odot \mathbb{C}$ is the C*-algebra $\mathbb{C}^n$. Copowers are preserved by endofunctors $A \otimes -$.

$$A \otimes (n \odot B) = A \otimes B \oplus \cdots \oplus A \otimes B = n \odot (A \otimes B)$$

We still need to verify that we have a relative adjunction. Observing that

$$\mathbf{FdC^*\text{-}Alg}_{\mathrm{CPU}}(A, \mathbb{C}^n) \cong \mathbf{FdC^*\text{-}Alg}_{\mathrm{CPU}}(A, \mathbb{C}) \times \cdots \times \mathbf{FdC^*\text{-}Alg}_{\mathrm{CPU}}(A, \mathbb{C})$$

one deduces that

$$\mathbf{FdC^*\text{-}Alg}_{\mathrm{CPU}}(A, \mathbb{C}^n) \cong \mathbf{Set}(n, \mathbf{FdC^*\text{-}Alg}_{\mathrm{CPU}}(A, \mathbb{C}))$$

and therefore

$$\mathbf{FdC^*\text{-}Alg}^{\mathrm{op}}_{\mathrm{CPU}}(\mathbb{C}^n, A) \cong \mathbf{Set}(n, \mathbf{FdC^*\text{-}Alg}^{\mathrm{op}}_{\mathrm{CPU}}(\mathbb{C}, A))$$

Then, the symmetric monoidal functor $J : \mathbb{N} \to \mathbf{FdC^*\text{-}Alg}^{\mathrm{op}}_{\mathrm{CPU}}$ associates every natural number $n \in \mathbb{N}$ to the C*-algebra $\mathbb{C}^n$. The morphisms run_n are given by the isomorphism

$$\mathbf{FdC^*\text{-}Alg}^{\mathrm{op}}_{\mathrm{CPU}}(\mathbb{C}, \mathbb{C}^n) := \mathbf{FdC^*\text{-}Alg}_{\mathrm{CPU}}(\mathbb{C}^n, \mathbb{C}) \cong \mathcal{D}(n)$$

between states on $\mathbb{C}^n$ and the n-simplex $\mathcal{D}(n) := \{x \in [0,1]^n \mid \sum_i x_i = 1\}$ [9, Lemma 4.1].

The semantics of types and gates is rather standard. Probabilities are complex numbers $\mathbb{C}$ and a (classical) bit is therefore an element of the C*-algebra $\mathbb{C} \oplus \mathbb{C}$. Moreover, n-qubit systems are modelled in the C*-algebra M_{2^n}. In other words,

$$\mathbb{1} = \mathbb{C} \qquad \mathbb{bit} = \mathbb{C} \oplus \mathbb{C} \qquad \mathbb{qubit} = M_2 \qquad \mathbb{u} = u^\dagger - u \text{ (for every unitary } u \in \mathcal{U})$$

$$\text{meas} : \mathbb{C} \oplus \mathbb{C} \to M_2 : (a, b) \mapsto \left(\begin{smallmatrix} a & 0 \\ 0 & b \end{smallmatrix}\right) \qquad \text{new} : M_2 \to \mathbb{C} \oplus \mathbb{C} : \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right) \mapsto (a, b)$$

and so on.

C.2 Proof of Proposition 4.1

First, it has been established that the category $\mathbf{W^*\text{-}Alg}_{\text{CPSU}}^{\text{op}}$ is symmetric monoidal when equipped with the spatial tensor product [15] and it is a well-known fact that the category $\mathbf{Dcpo}_\perp$ of pointed dcpos and strict Scott-continuous maps is cartesian closed.

Much like in [12, Section 5.6], we introduce the restricted version of the monad of subvaluations $V = \mathbf{dcGEMod}([0,1]^{(-)}, [0,1])$ on $\mathbf{Dcpo}_\perp$, where the category **dcGEMod** is the category of directed-complete generalized effect modules and Scott-continuous effect module homomorphisms, also introduced as a category of quantum predicates in [25,26].

Recall that a strong monad over a monoidal closed category $\mathbf{K}$ is the same thing as a $\mathbf{K}$-enriched monad. The monad V is enriched over $\mathbf{Dcpo}_\perp$ and therefore a strong monad since the category $\mathbf{Dcpo}_\perp$ is (cartesian) closed.

Since there is a full and faithful functor which takes every W*-algebra A to its directed-complete generalized effect module $[0,1]_A$ of predicates [26], there is an equivalence

$$V(n) = \mathbf{dcGEMod}([0,1]^n, [0,1]) \cong \mathbf{W^*\text{-}Alg}_{\text{CPSU}}(\mathbb{C}^n, \mathbb{C}) \text{ for every } n \in \mathbb{N}$$

and therefore an equivalence $V(n) \cong \mathbf{W^*\text{-}Alg}_{\text{CPSU}}^{\text{op}}(\mathbb{C}, \mathbb{C}^n)$ for every $n \in \mathbb{N}$.

Building up on the constructions of Prop 3.3, we deduce that $(\mathbf{W^*\text{-}Alg}_{\text{CPSU}}, \mathbf{Dcpo}_\perp, V)$ is a categorical model of QWire.

C.3 Proof of Proposition 4.3

Variations of most of the categorical constructions required have been discussed in [26]. The coproduct of quantum predomains $F, G : \mathbf{C}^{\text{op}} \to \mathbf{Dcpo}$ is defined pointwise on the disjoint sum of dcpos, and so does the terminal object (resp. zero object) of the category of quantum (pre)domains and Scott-continuous maps (resp. strict Scott-continuous maps).

Since $\mathbf{C}$ is a small category, the Day convolution provides us with a symmetric monoidal closed category of quantum predomains [6] and algebraic compactness for locally continuous endofunctors is given by [8, Example 6.9].

The copower takes inspiration from the fact that the equivalence $M_n(A) \cong M_n \otimes A$ holds for every C*-algebra A and every natural number $n \in \mathbb{N}$. In short: for every

$n \in \mathbb{N}$, $n \odot F$ is the quantum predomain defined as the mapping $A \mapsto F(M_n(A))$ on objects and $f \mapsto F(M_n(f))$ on maps.

Finally, exploiting the fact that the enriched Yoneda embedding $\mathbf{y} : \mathbf{C} \to [\mathbf{C}^{\mathrm{op}}, \mathbf{Dcpo}]$ is full and faithful, we observe that there is a one-to-one correspondance between completely positive unital maps $I \to J(n)$ and Scott-continuous natural transformations $\mathbf{C}(-, I) \Rightarrow \mathbf{C}(-, J(n))$, which leads us to the following equivalence

$$T(n) \cong \mathbf{C}(I, J(n)) \cong [\mathbf{C}^{\mathrm{op}}, \mathbf{Dcpo}](I, J(n)) \text{ for every } n \in \mathbb{N}$$