Logical Methods in Computer Science
Volume 16, Issue 1, 2020, pp. 30:1–30:24
https://lmcs.episciences.org/

Submitted Nov. 15, 2017
Published Mar. 10, 2020

CLASSICAL CONTROL, QUANTUM CIRCUITS AND LINEAR LOGIC IN ENRICHED CATEGORY THEORY

MATHYS RENNELA [a] AND SAM STATON [b]

[a] Radboud University, Nijmegen, The Netherlands
e-mail address: mathys.rennela@gmail.com

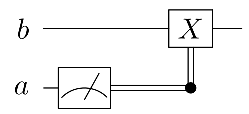
[b] Oxford University, Oxford, The United Kingdom
e-mail address: sam.staton@cs.ox.ac.uk

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. We show that this structure is also related to linear/non-linear models. 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.

Introduction

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

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 on a qubit b:



(0.1)

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

Key words and phrases: Enriched categories, categorical semantics, linear type theory, quantum circuits, relative monad, quantum domain theory.

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 [23, 28] and of prior work of the authors [33, 35]. This is a fascinating direction, and sheds light, for example, on the structure of the quantum teleportation algorithm (e.g. [28, Example 6]). However, the general connection between physics and higher-order quantum functions is yet unclear. Although some recent progress has been made [19], 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 [13] and LiQUi|⟩ [43]. This is the approach that we study in this paper.

Embedded languages and enriched categories. Our work revolves around a new calculus that we call 'EWire' (§1). It is a minor generalization of the QWire language [29,32]. QWire idealizes some aspects of the architecture of Quipper and LiQUi|⟩. 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 (0.1) corresponds to the instruction below (0.2) 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.

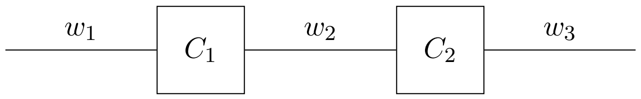
$$\begin{aligned} -; a, b : \mathrm{qubit} \ \vdash \ C \ \stackrel{\mathrm{def}}{=} \ & x \leftarrow \mathbf{gate\ meas}\ a; (x, y) \leftarrow \mathbf{gate}\ (\mathbf{bit\text{-}control}\ X)\,(x, b); \\ & () \leftarrow \mathbf{gate\ discard}\ x; \mathbf{output}\ y \qquad : \ \mathrm{qubit} \end{aligned} \tag{0.2}$$

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

$$t \ \stackrel{\mathrm{def}}{=} \ \mathbf{box}\ (a, b) \Rightarrow C(a, b) \qquad (\text{where } C \text{ is as in (0.2)}) \tag{0.3}$$

creates a closed term of type Circ(qubit⊗qubit,qubit) in the host language. We recover the instruction C in the circuit language (0.2) from the boxed expression t in the host language (0.3) 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

$$\text{comp} \ :\ \text{Circ}(W_1, W_2) \times \text{Circ}(W_2, W_3) \to \text{Circ}(W_1, W_3) \tag{0.4}$$

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 2, 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 (0.4) is precisely composition in the sense of enriched categories.

This enriched categorical framework is then incorporated in the definition of categorical models of EWire (Definition 2.1), which is associated to the following proposition, building on a partial normalization procedure proposed in [29] for QWire expressions, which is shown to be sound with respect to a semantics in completely positive maps [29, App. A–B].

Proposition 2.3 (paraphrased)**.** *The program equations that are used to justify the steps in the normalization procedure in [29] are sound in any EWire model.*

The idea of denoting a language for circuits within a symmetric monoidal category is an established idea (e.g. [11]). The novel point here is enrichment.

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.

Proposition 2.2 (paraphrased)**.** *Taking finite-dimensional C*-algebras and completely positive unital maps as a categorical model of the circuit language yields a categorical model of QWire in which the types qubit and bit are respectively interpreted by the C*-algebras M_2 and $\mathbb{C} \oplus \mathbb{C}$.*

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) 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.

Linear/non-linear models. Quantum resources cannot be duplicated or discarded, whereas classical resources can. This suggests a connection to linear logic [12], as many others have observed [6, 14, 22, 23, 28, 37, 38]. In fact, enriched category theory is closely related to linear logic through the following observation (see also [9, 24, 25]).

Lemma 4.2 (paraphrased)**.** *The following data are equivalent:*

- *A symmetric monoidal closed category enriched in a cartesian closed category, with copowers.*
- *A 'linear/non-linear' (LNL) model in the sense of Benton [4]: a symmetric monoidal adjunction between a cartesian closed category and a symmetric monoidal closed category.*

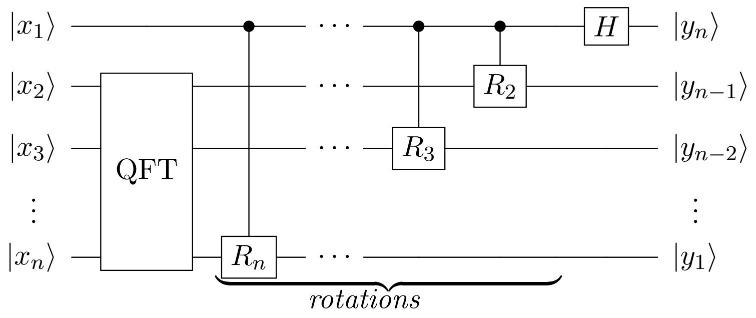
Thus, from the semantic perspective, enriched category theory is a way to study fragments and variations of linear logic. We show that every LNL model, with some minor extra structure, induces an EWire model (see Proposition 4.4). We call these structures LNL-EWire models.

So LNL-EWire extends EWire with some additional connectives. However, although these additional connectives have an established type theoretic syntax, their meaning from the perspective of quantum physics is less clear at present. For example, LNL assumes a first class linear function space, but categories of completely positive unital maps (or CPTP maps) are typically not monoidal closed. Nonetheless, there is a semantic way to understand their relevance, through the following theorem, which is proved using a variation on Day's construction [7], following other recent work [23, 27, 37].

Theorem 4.5 (paraphrased)**.** *Every EWire model (with a locally presentable base) embeds in an LNL-EWire model.*

Recursive types and recursive terms. Within our 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 by recursion, 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 $\left(\begin{smallmatrix}1 & 0\\ 0 & e^{2\pi i/2^n}\end{smallmatrix}\right)$.



(0.5)

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 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 [33], 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 (Section 3). 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.

Relation to previous work. This paper expands and develops the paper [36] presented at The Thirty-third Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXIII). This version includes numerous elaborations, most notably Section 4 on linear/non-linear models is entirely new.

1. 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 [29] ('quantum wire language'), and we make the connection precise in Section 1.2. We discuss extensions of EWire in Section 3.

We assume two classes of basic wire types.

- Classical wire types, ranged over by $\mathbf{a}, \mathbf{b}, \dots$. 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 $\alpha, \beta, \dots$. These wire types only exist in the circuit language. For example, the type of qubits.

From these basic types we build all wire types:

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

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

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

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 [26] but with special types for the classical wire types **a**, **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*:

$$\left|V \otimes V'\right| \overset{\mathrm{def}}{=} |V| \times \left|V'\right| \qquad |I| \overset{\mathrm{def}}{=} 1 \qquad |\mathbf{a}| \overset{\mathrm{def}}{=} \mathbf{a}$$

1.1. **Circuit typing and host typing.** To describe the well-formed terms, we consider two kinds of context: circuit contexts

$$\Omega = (w_1 : W_1 \cdots w_n : W_n) \qquad (\text{for } n \in \mathbb{N}).$$

and host language contexts

$$\Gamma = (x_1 : A_1 \cdots x_m : A_m) \qquad (\text{for } m \in \mathbb{N}).$$

We will define a well-formed circuit judgement

$$\Gamma; \Omega \vdash C : W$$

and a well-formed host language judgement

$$\Gamma \vdash t : A.$$

Following QWire [29], the circuit language is based on patterns. Wires are organised in patterns given by the grammar

$$p ::= w \mid () \mid (p, p)$$

associated to the following set of rules, defining a ternary relation $\Omega \implies p : W$:

$$\frac{-}{\cdot \implies () : 1} \qquad \frac{-}{w : W \implies w : W}$$

$$\frac{\Omega_1 \implies p_1 : W_1 \qquad \Omega_2 \implies p_2 : W_2}{\Omega_1, \Omega_2 \implies (p_1, p_2) : W_1 \otimes W_2} \qquad \frac{\Omega_1, w_1 : W_1, w_2 : W_2, \Omega_2 \implies p : W}{\Omega_1, w_2 : W_1, w_1 : W_2, \Omega_2 \implies p : W}$$

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, for splitting a tensor-product type into its constituents, and the exchange rule. The sixth rule includes the basic gates in the circuit language.

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

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

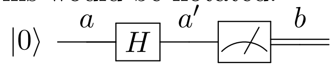
$$\frac{\Gamma; \Omega_1, w_1 : W_1, w_2 : W_2, \Omega_2 \vdash C : W}{\Gamma; \Omega_1, w_2 : W_2, w_1 : W_1, \Omega_2 \vdash C : W}$$

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

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

$$\mathrm{flip} \overset{\mathrm{def}}{=} a \leftarrow \mathbf{gate}\ \mathrm{init}_0\ (); a' \leftarrow \mathbf{gate}\ \mathrm{H}\ a; b \leftarrow \mathbf{gate}\ \mathrm{meas}\ a'; \mathbf{output}\ b \tag{1.1}$$

In quantum circuit notation, this would be notated:



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.

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

For example, flip (1.1) is an expression in the circuit language of type bit, and $(\mathbf{run}\ \mathrm{flip}) : T(\mathrm{bit})$ is an expression of the host language that performs the coin flip.

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 $\mathrm{Circ}(W_1, W_2)$, 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.

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

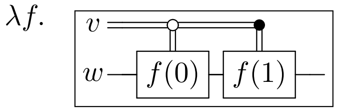
For example, we can define a host function

$$(|V| \to \mathrm{Circ}(W_1, W_2)) \to \mathrm{Circ}(V \otimes W_1, W_2)$$

converting a host language parameter to a circuit wire, by

$$\lambda f.\, \mathbf{box}\ \Big((v, w) \Rightarrow \Big(x \Leftarrow \mathbf{lift}\ v; \mathbf{unbox}(f x)(w)\Big)\Big)$$

If $V = \mathrm{bit}$, then this might be informally notated



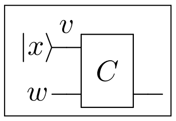
This function has an inverse,

$$\mathrm{Circ}(V \otimes W_1, W_2) \to (V \to \mathrm{Circ}(W_1, W_2))$$

transforming a classical circuit wire to a parameter in the host language:

$$\lambda C.\, \lambda x.\, \mathbf{box}\ \Big(w \Rightarrow \Big(v \Leftarrow \mathbf{init}\ x; \mathbf{unbox}\ (C)(v, w)\Big)\Big).$$

This might be informally notated

$\lambda C.\lambda x.$ 

As we will see in (2.2), these functions together correspond to a copower structure in enriched category theory.

For a further example of these interactions, if we had conditionals in the host language, then the illustration of classical control in the introduction (0.1) could be notated

$$\begin{aligned}\cdot; a:\text{qubit}, b:\text{qubit} \vdash\ & a' \leftarrow \text{meas } a; x \Leftarrow \textbf{lift } a';\\ & \textbf{unbox}\Big(\text{if } x \text{ then } \textbf{box}(b' \Rightarrow (b'' \leftarrow \textbf{gate } X\, b'; \textbf{output } b''))\\ & \qquad \text{else } \textbf{box}(b' \Rightarrow \textbf{output } b')\Big)\end{aligned} \tag{1.2}$$

Notice that the classical control is handled in the host language.
Additional typing rules for the host language. Recall that the types of the host language are

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

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

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

to which we need to add the typing rules for the monad [26], associated respectively to the unit and the strong Kleisli composition:

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

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

1.2. **QWire.** The language QWire of Paykin, Rand and Zdancewic [29] 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 $\in \mathcal{G}(\text{qubit}, \text{bit})$ and new $\in \mathcal{G}(\text{bit}, \text{qubit})$.

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:

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

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

$$\text{meas}_W : \text{Circ}(W, \overline{W}) \qquad \text{and} \qquad \text{new}_W : \text{Circ}(\overline{W}, W)$$

for all wires W. These are defined by induction on W. For example,

$$\text{meas}_I \overset{\text{def}}{=} \text{id} \quad \text{meas}_{\text{bit}} \overset{\text{def}}{=} \text{id}$$

$$\text{meas}_{\text{qubit}} \overset{\text{def}}{=} \textbf{box } p \Rightarrow p' \leftarrow \textbf{gate } \text{meas}\, p; \textbf{output } p'$$

$$\text{meas}_{W \otimes W'} \overset{\text{def}}{=} \textbf{box } (w, w') \Rightarrow \ x \leftarrow \textbf{unbox } \text{meas}_W\, w;\ x' \leftarrow \textbf{unbox } \text{meas}_{W'}\, w'; \\ \textbf{output } (x, x')$$

and new_W is defined using $\text{new} \in \mathcal{G}(\text{bit}, \text{qubit})$ 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:

$$\textbf{qwire-run}(C) \overset{\text{def}}{=} \textbf{run}(x \leftarrow C; \textbf{unbox}\text{ meas } x)$$

$$(x \Leftarrow \textbf{qwire-lift}\, p\, ;\, C) \overset{\text{def}}{=} y \Leftarrow \textbf{lift}\, p\, ;\, x \leftarrow \textbf{unbox}\text{ new } y\, ;\, C$$

2. Categorical models of EWire

We introduce the categorical semantics of EWire. Our semantic models are based around enriched category theory [18]. The relevance of **Set**-enriched copowers to quantum algorithms has previously been suggested by Jacobs [15]. On the other hand, copowers and enrichment play a key role in the non-quantum enriched effect calculus [9,25] and other areas [21,24,31,41]. Nonetheless, the connection that we establish with the EWire syntax appears to be novel.

2.1. Preliminaries.

Enriched categories. Let $\mathbf{H}$ be a cartesian closed category. Recall that a category $\mathbf{C}$ enriched in $\mathbf{H}$ is given by a collection of objects and, for each pair of objects c and d, an object $\mathbf{C}(c, d)$ in $\mathbf{H}$, together with morphisms for composition $\mathbf{C}(c, d) \times \mathbf{C}(d, e) \to \mathbf{C}(c, e)$ and identities $1 \to \mathbf{C}(c, c)$, subject to associativity and identity laws. Much of category theory can be reformulated in the enriched setting [18].

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 $\vdash \textbf{run}(\text{flip}) : T(\text{bit})$ 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 [26].

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 provide a first class type of computations $T(A)$. In a truly first order language, such as a circuit language, although computations are present, they are not first class. In particular, there is no *wire* type $T(\text{qubit})$ of all quantum computations. To resolve this kind of mismatch, authors have proposed alternatives such as relative monads [2] and monads with arities [5].

An ordinary *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 bijection

$$\mathbf{C}(L(b), c) \cong \mathbf{D}(J(b), R(c)) \tag{2.1}$$

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 replacing the natural bijection (2.1) with a antural isomorphism. 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 and weighted limits. In enriched category theory it is appropriate to consider weighted colimits and limits, which are a generalization of the ordinary notions. As a first step, we consider 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

$$\mathbf{C}(h \odot A, B) \cong \mathbf{H}(h, \mathbf{C}(A, B)) \tag{2.2}$$

natural in B.

Weighted colimits combine the notions of copowers and of ordinary conical colimits. If $\mathbf{J}$ is a $\mathbf{H}$-enriched category, then we may try to take a colimit of a diagram $D : \mathbf{J} \to \mathbf{C}$ weighted by a functor $W : \mathbf{J}^{\mathrm{op}} \to \mathbf{H}$. Here $\mathbf{C}$ is an $\mathbf{H}$-enriched category and D and W are $\mathbf{H}$-functors. A *cylinder* for (W, D) is given by an object X and a $\mathbf{H}$-natural family of morphisms $W(j) \to \mathbf{C}(D(j), X)$; the weighted limit, $\mathrm{colim}_j^W D_j$ if it exists, is the universal cylinder.

2.2. **EWire models.**

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 $\mathcal{G}$.

Definition 2.1. A *categorical model of EWire* $(\mathbf{C}, \mathbf{H}, \mathbf{H}_0, T)$ is given by the following data:

(1) A cartesian closed category $\mathbf{H}$ with a strong monad T on $\mathbf{H}$. This is needed to interpret the host language.
(2) 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.

(3) 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 $\mathrm{Circ}(W, W')$.
(4) 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.
(5) 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.
(6) 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

$$[\![\mathrm{Circ}(W, W')]\!] \overset{\mathrm{def}}{=} \mathbf{C}(W, W') \in \mathrm{Obj}(\mathbf{H})$$

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. 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

$$\begin{aligned}\mathrm{lift}_{h'} : \mathbf{H}(h \times h', \mathbf{C}(X, Y)) \cong \mathbf{H}(h, \mathbf{H}(h', \mathbf{C}(X, Y))) &\cong \mathbf{H}(h, \mathbf{C}(h' \odot X, Y)) \\ &\cong \mathbf{H}(h, \mathbf{C}(J(h') \otimes X, Y))\end{aligned}$$

so that

$$[\![\Gamma; \Omega, \Omega' \vdash x \Leftarrow \mathbf{lift}\ p; C : W]\!] = \mathrm{lift}_{[\![|V|]\!]}([\![\Gamma, x : |V| ; \Omega' \vdash C : W]\!])$$

As typing rules for the monad, the operations **return** and **let** are denoted respectively by the unit and the strong Kleisli composition [26].

Since we're enforcing explicit measurement here, the denotation of the operation **init** for a term t of first order host type $|V|$ (for a classical wire type V) and the denotation of the operation **run** for a circuit C whose output wire type is the type W are given by Def. 2.1(6).

$$[\![\Gamma; \cdot \vdash \mathbf{init}\ t : V]\!] = [\![\Gamma]\!] \xrightarrow{t} [\![V]\!] \xrightarrow{\mathbf{C}(I, J(-))} \mathbf{C}(I, J([\![V]\!]))$$
$$[\![\Gamma \vdash \mathbf{run}\ C : T(|V|)]\!] = [\![\Gamma]\!] \xrightarrow{C} \mathbf{C}(I, J([\![V]\!])) \xrightarrow{\mathrm{run}_{[\![V]\!]}} T([\![V]\!])$$

The denotations of the remaining instructions are given by the following composite morphisms in $\mathbf{H}$. They crucially use the enriched composition, which is a morphism in $\mathbf{H}$.

$$\begin{aligned}&[\![\Gamma; \Omega_1, \Omega_2 \vdash p \leftarrow C; C' : W']\!] = \\ &\quad [\![\Gamma]\!] \xrightarrow{C, C'} \mathbf{C}([\![\Omega_1]\!], [\![W]\!]) \times \mathbf{C}([\![\Omega, \Omega_2]\!], [\![W']\!]) \overset{p}{\cong} \mathbf{C}([\![\Omega_1]\!], [\![\Omega_2]\!]) \times \mathbf{C}([\![\Omega, \Omega_2]\!], [\![W']\!]) \\ &\qquad \xrightarrow{(\mathrm{inj}_\Omega \times -)} \mathbf{C}([\![\Omega, \Omega_1]\!], [\![\Omega, \Omega_2]\!]) \times \mathbf{C}([\![\Omega, \Omega_2]\!], [\![W']\!]) \xrightarrow{\circ} \mathbf{C}([\![\Omega, \Omega_1]\!], [\![W']\!])\end{aligned}$$

$$\begin{aligned}&[\![\Gamma; \Omega_1, \Omega \vdash p_2 \leftarrow \mathbf{gate}\ g\, p_1; C : W]\!] = \\ &\quad [\![\Gamma]\!] \cong 1 \times [\![\Gamma]\!] \xrightarrow{[\![g]\!] \times C} \mathbf{C}([\![W_1]\!], [\![W_2]\!]) \times \mathbf{C}([\![\Omega_2, \Omega]\!], [\![W]\!]) \\ &\qquad \overset{p_1, p_2}{\cong} \mathbf{C}([\![\Omega_1]\!], [\![\Omega_2]\!]) \times \mathbf{C}([\![\Omega, \Omega_2]\!], [\![W']\!]) \\ &\qquad \xrightarrow{(\mathrm{inj}_\Omega \times -)} \mathbf{C}([\![\Omega, \Omega_1]\!], [\![\Omega, \Omega_2]\!]) \times \mathbf{C}([\![\Omega, \Omega_2]\!], [\![W']\!]) \xrightarrow{\circ} \mathbf{C}([\![\Omega, \Omega_1]\!], [\![W']\!])\end{aligned}$$

where C, C' is the product map, and inj_Ω is the injection map which sends circuits in $\mathbf{C}([\![\Omega_1]\!], [\![\Omega_2]\!])$ to circuits in $\mathbf{C}([\![\Omega, \Omega_1]\!], [\![\Omega, \Omega_2]\!])$. The later corresponds to the operation of adding a wire to a circuit.

2.3. Example model: finite dimensional C*-algebras enriched in sets.

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'. For example, the observables of a qubit are positive elements of the non-commutative algebra M_2 of 2×2 complex matrices, and the observables of a bit are positive elements of the commutative algebra $\mathbb{C}^2$ of pairs of complex numbers. Circuit expressions $(\cdot; (x : W) \vdash C : W')$ 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. [39]) 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, as already mentioned, 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 C*-algebra $M_k(A)$ 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. [40, 42]).

Proposition 2.2. *The quadruplet* $(\mathbf{FdC^*\text{-}Alg}^{\mathrm{op}}_{\mathrm{CPU}}, \mathbf{Set}, \mathbb{N}, \mathcal{D})$ *is a model of EWire, formed by the opposite category of the category* $\mathbf{FdC^*\text{-}Alg}_{\mathrm{CPU}}$ *of finite-dimensional C*-algebras and completely positive unital maps, the cartesian closed category* $\mathbf{Set}$ *of sets and functions, the skeleton* $\mathbb{N}$ *of the category of finite sets and functions (which considers natural numbers as its objects), and the probability distribution monad* $\mathcal{D}$ *over* $\mathbf{Set}$*. In fact it is a model of QWire, with* $[\![qubit]\!] \overset{def}{=} M_2$ *and* $[\![bit]\!] \overset{def}{=} \mathbb{C} \oplus \mathbb{C}$.

Proof. The category $\mathbf{Set}$ 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.

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 [15]. We observe that the copower distributes over the coproduct, i.e.

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

and that composition is multiplication, i.e.

$$n \odot (m \odot A) = (nm) \odot B$$

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

$$\mathbf{FdC^*\text{-}Alg}^{\mathrm{op}}_{\mathrm{CPU}}(\mathbb{C}, \mathbb{C}^n) = \mathbf{FdC^*\text{-}Alg}_{\mathrm{CPU}}(\mathbb{C}^n, \mathbb{C}) \cong \mathcal{D}(n) \qquad \text{[10, Lemma 4.1]}$$

between states on $\mathbb{C}^n$ and the n-simplex

$$\mathcal{D}(n) := \{x \in [0,1]^n \mid \textstyle\sum_{i=1}^n x_i = 1\}$$

The semantics of types and gates is rather standard. Probabilities are in particular 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,

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

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

and so on. □

2.4. **Soundness of normalization steps.** In [29], a partial normalization procedure is proposed for QWire expressions, which is shown to be sound with respect to a semantics in completely positive maps [29, App. A–B]. We now remark that the program equations that are used to justify the steps in the normalization procedure are sound in any EWire model.

We say that an equation between circuit expressions

$$\Gamma; \Omega \vdash C_1 \equiv C_2 : W$$

is *sound* if the interpretations are equal, $[\![C_1]\!] = [\![C_2]\!]$, in every EWire model.

Proposition 2.3. *The following equations between circuits are sound:*

- $\boldsymbol{unbox}\ (\boldsymbol{box}\ w \Rightarrow C)\, p \equiv C[w \mapsto p]$
- $p \leftarrow \boldsymbol{output}\ p'; C \equiv C[p \mapsto p']$
- $w \leftarrow (p_2 \leftarrow \boldsymbol{gate}\ g\, p_1; N); C \equiv p_2 \leftarrow \boldsymbol{gate}\ g\, p_1; w \leftarrow N; C$
- $w \leftarrow (x \Leftarrow \boldsymbol{lift}\ p; C'); C \equiv x \Leftarrow \boldsymbol{lift}\ p; w \leftarrow C'; C$
- $() \leftarrow (); C \equiv C$
- $(w_1, w_2) \leftarrow (p_1, p_2); C \equiv C[\begin{smallmatrix} w_1 & \mapsto p_1 \\ w_2 & \mapsto p_2 \end{smallmatrix}]$
- $w \leftarrow (() \leftarrow w'; N); C \equiv () \leftarrow w'; w \leftarrow N; C$
- $w \leftarrow ((w_1, w_2) \leftarrow w'; N); C \equiv (w_1, w_2) \leftarrow w'; w \leftarrow N; C$

Proof notes. These equations are all straightforward consequences of the interpretation in monoidal categories, except for the first one, which is different in character, but trivial. □

We remark that the categorical model suggests other equations that could be used for a more advanced normalization algorithm, such as the following equations that come from the definition of copower (2.2)

$$x \Leftarrow \textbf{lift}\ p; \textbf{init}\ x \equiv p \qquad p \leftarrow \textbf{init}\ t; x \Leftarrow \textbf{lift}\ p; C \equiv C[x \mapsto t].$$

3. Towards a model with recursive quantum datatypes: enrichment in dcpos

Recall that a dcpo is a partial order which has all directed joins. A monotone function between dcpo's is continuous if it preserves directed joins. Let **Dcpo** be the category of dcpo's and continuous functions (e.g. [1]). The category is cartesian closed, hence a suitable situation for modelling a host language, but also supports recursion. If D is a dcpo with a bottom element $\perp$, then there is a fixed point combinator

$$Y : (D \to D) \to D$$

given by $Y(f) = \bigvee_{i=1}^{\infty} f^n(\perp)$, with the property that $f(Y(f)) = Y(f)$.

Any ordinary category is trivially enriched in dcpo's, by considering the hom-sets as flat partial orders. This leads us to an EWire model

$$(\mathbf{FdC^*\text{-}Alg}^{\mathrm{op}}_{\mathrm{CPU}}, \mathbf{Dcpo}, \mathbb{N}, \mathcal{V}) \tag{3.1}$$

where $\mathbb{N}$ is the subcategory of **Dcpo** comprising the flat, finite orders, and $\mathcal{V}$ is a probabilistic powerdomain monad on dcpos (e.g. [17]). We could use this model to interpret an extension of EWire with recursion in the host language.

We can also model iteration in the circuit language. One way to model iteration in the circuit language is to work with subunital maps. Recall that a map f between C*-algebras is subunital if $f(1) \leq 1$, i.e. $1 - f(1)$ is positive (e.g. [40, §6]). The subunital maps can be ordered by the Löwner partial order: $f \leq g$ if and only if $(g - f)$ is a positive map. This turns out to be a directed complete partial order between finite dimensional C*-algebras, and so the category $\mathbf{FdC^*\text{-}Alg}_{\mathrm{CPSU}}$ is enriched in **Dcpo** in a more interesting way [33, 34, 40].

Thus we have an EWire model:

$$(\mathbf{FdC^*\text{-}Alg}^{\mathrm{op}}_{\mathrm{CPSU}}, \mathbf{Dcpo}, \mathbb{N}, \mathcal{V}) \tag{3.2}$$

In this model, the runnable circuits correspond to subprobability distributions:

$$\mathbf{FdC^*\text{-}Alg}_{\mathrm{CPSU}}(\mathbb{C}^n, \mathbb{C}) \cong \mathcal{V}(n) \cong \{x \in [0,1]^n \mid \textstyle\sum_{i=1}^n x_i \leq 1\}$$

with, intuitively, $1 - \sum_{i=1}^n x_i$ the probability of diverging. In this model, the space of circuits $[\![\mathrm{Circ}(W_1, W_2)]\!] = \mathbf{FdC^*\text{-}Alg}_{\mathrm{CPSU}}([\![W_2]\!], [\![W_1]\!])$ is a dcpo with a bottom element (the zero map).

For this reason we can add to the host language a family of fixed point combinators:

$$Y_{A,W_1,W_2} : \Big((A \to \mathrm{Circ}(W_1, W_2)) \to (A \to \mathrm{Circ}(W_1, W_2)\Big) \to A \to \mathrm{Circ}(W_1, W_2)$$

We can use this to recursively define circuits. For a simple example, suppose we also add basic arithmetic functions to the host langauge, with standard interpretation in dcpos. Then the following function defines a family of circuits where `Hs n` is the composition of n Hadamard gates.

```
Hs : int -> Circ(qubit,qubit)
Hs = Y(lambda Hs. lambda n.
        if n=0 then box q => output q
        else box q => q' <- gate H q ; unbox (Hs (n-1)) q')
```

Note that `Hs (-1)` diverges, in the sense that it is interpreted by the zero map.

A model with recursive quantum data. A still more elaborate model begins from the observation that the category of W*-algebras and normal subunital maps is also enriched in **Dcpo** via the Löwner order.

This category $\mathbf{W}^*\text{-}\mathbf{Alg}_{\mathrm{CPSU}}^{\mathrm{op}}$ is symmetric monoidal when equipped with the spatial tensor product [20]. In earlier work [33], the first author has shown that $\mathbf{W}^*\text{-}\mathbf{Alg}_{\mathrm{CPSU}}$ supports the construction of recursive types: it is algebraically compact. Thus this is an EWire model that allows wire types encoding streams of qubits, lists of qubits, and so on.

Much like in [16, Section 5.6], we use the restricted version of the monad of subvaluations $V' = \mathbf{dcGEMod}([0,1]^{(-)}, [0,1])$ on **Dcpo**, where the category **dcGEMod** is the category of directed-complete generalized effect modules and Scott-continuous effect module homomorphisms, also considered as a category of quantum predicates in [34, 35]. Thus

$$(\mathbf{W}^*\text{-}\mathbf{Alg}_{\mathrm{CPSU}}^{\mathrm{op}}, \mathbf{Dcpo}, \mathbb{N}, \mathcal{V}')$$

is an EWire model.

In this model, we can interpret a wire type `qlist` with gates

- `isempty` $\in \mathcal{G}($`qlist`$,$ `bit` $\otimes$ `qlist`$)$
- `headtail` $\in \mathcal{G}($`qlist`$,$ `qbit` $\otimes$ `qlist`$)$
- `nil` $\in \mathcal{G}(I,$ `qlist`$)$
- `cons` $\in \mathcal{G}($`qbit` $\otimes$ `qlist`$,$ `qlist`$)$.

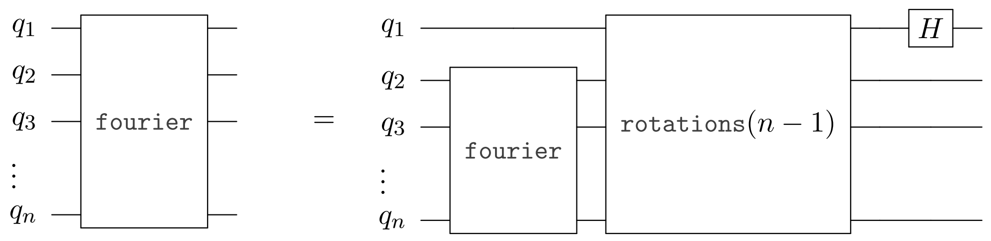
and a classical wire type `int`.

We want to implement the Quantum Fourier Transform (see Figure 1). Taking inspiration from [29, Sec. 6.2] and [13], we assume a host language constant

$$\mathbf{CR} : \mathrm{Nat} \to \mathrm{Circ}(\mathrm{qubit} \otimes \mathrm{qubit}, \mathrm{qubit} \otimes \mathrm{qubit})$$

so that $(\mathbf{CR}\, n)$ corresponds to the controlled rotation by $\frac{2\pi}{2^n}$ 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.

Here we are using some standard syntactic sugar for the host language, such as recursive definitions instead of a fixed point combinator. The recursive definitions encode the following informal recursive circuit definitions:

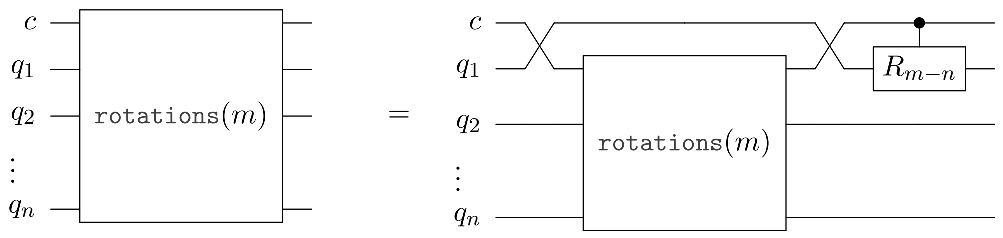


```
length : Circ(qlist,int⊗qlist) =
box qs =>
 (b,qs) <- isempty qs ; b <- lift b ;
 unbox (if b then box qs => n <- init 0 ; output (n,qs)
        else box qs => (h,t) <- gate headtail qs ;
                       (n,t) <- unbox length t ;
                       n <- lift n ; n' <- init (n+1) ;
                       qs <- gate cons (h,t) ;
                       output (n',qs)            ) qs

rotations : int -> Circ(qubit⊗qlist,qubit⊗qlist) =
lambda m. box (c,qs) =>
 (b,qs) <- isempty qs ; b <= lift b ;
 unbox (if b then box (c,qs) => output(c,qs)
        else box (c,qs) =>
             (n,qs) <- unbox length qs ; n <= lift n ;
             (q,qs') <- gate headtail qs
             (c,qs') <- unbox (rotations m) (c,qs')
             (c,q) <- unbox (CR (m-n)) (c,q) ;
             qs <- gate cons (q,qs')
             output (c,qs) )
       (c,qs)

fourier : Circ(qlist, qlist) =
box qs =>
  (b,qs) <- isempty qs ; b <= lift b ;
  unbox (if b then box qs => output qs
         else box qs =>
              (q,qs') <- gate headtail 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')     ) qs
```

Figure 1. Implementation of QFT



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 have specified the QFT algorithm using recursion, in a way that is familiar from basic functional programming. There are at least two drawbacks to this approach: (1) it is a priori unclear that the `unbox fourier qs` terminates in general; (2) the type system does not tell us that (`unbox fourier qs`) has the same length as `qs`. This is a familiar problem for list programming in a functional language such as Haskell, but it is particularly inconvenient for a quantum circuit layout engine, especially if one want to ensure that the terms that correspond to circuits are terminating and have a known size. A good way to deal with it would be through some kind of dependent types [29, Sec. 6.2], for instance allowing a type $\mathrm{QList}(n)$ of arrays of qubits of size n and

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

Integrating dependent types to EWire and associating such types to an appropriate categorical semantics is left for future work. Although this section discusses the integration of recursive types and how it leads to more refined presentations (in EWire) of quantum programs such as QFT, we do not provide a detailed explanation of the syntactic and semantic integration of recursive types. However, in a sequel of this paper co-authored by one of the authors of the present work [30], there is a detailed treatment of inductive types in quantum programming, both syntactically and semantically, with a computationally adequate denotational model based on W*-algebras.

4. Relation to linear-non-linear models

Recent work by Selinger and collaborators [23, 37, 38] has suggested that a good setting for quantum programming might be the linear/non-linear (LNL) models of linear logic in the style of Benton [4]. This is, at first glance, a different approach to the QWire/EWire approach, for the following reason. In QWire/EWire the type of circuits $\mathrm{Circ}(W, W')$ is a type in the host language, and is not itself a wire type. By contrast, with LNL models, the linear function space $(W \multimap W')$ is itself an object of the category containing the linear types.

Rios and Selinger [37] gave a particular construction of an LNL model for quantum programming. Mislove et al. [22] have also been working on some related constructions. In this section we propose a variation on that construction that works for a broad class of EWire models. In this way we bring the two lines of research closer together.

Definition 4.1 [4]**.** A *linear/non-linear (LNL) model* is given by the following data:

(1) a symmetric monoidal closed category $\mathbf{L}$
(2) a cartesian closed category $\mathbf{H}$
(3) a symmetric monoidal adjunction $F \dashv G$ formed from symmetric monoidal functors $F : \mathbf{H} \to \mathbf{L}$ and $G : \mathbf{L} \to \mathbf{H}$.

There is an equivalent formulation of LNL models using enriched category theory. This plays a central role in work on the enriched effect calculus [9].

Lemma 4.2. *Let $\mathbf{H}$ be a cartesian closed category and $\mathbf{L}$ be a symmetric monoidal closed category. The following data are equivalent:*

(1) *structure making $\mathbf{L}$ an $\mathbf{H}$-enriched symmetric monoidal closed category with copowers;*
(2) *a symmetric monoidal adjunction between $\mathbf{H}$ and $\mathbf{L}$ (i.e. a LNL model).*

Proof. From (1) to (2), let the adjunction be the copower/hom adjunction

$$(-) \odot I \dashv \mathbf{L}(I, -) : (\mathbf{L}, \otimes) \to (\mathbf{H}, \times).$$

which is always a symmetric monoidal adjunction.

Since $\mathbf{L}$ is symmetric monoidal closed, each $- \otimes Z : \mathbf{L} \to \mathbf{L}$ has a right adjoint and so preserves weighted colimits, in particular copowers:

$$((X \odot W) \otimes Z) \cong X \odot (W \otimes Z)$$

In particular,

$$\begin{aligned} F(X \times Y) = (X \times Y) \odot I = X \odot (Y \odot I) = X \odot (I \otimes (Y \odot I)) \\ = (X \odot I) \otimes (Y \odot I) = F(X) \otimes F(Y). \end{aligned}$$

as required.

From (2) to (1), let the enrichment be given by

$$\mathbf{L}(X, Y) \overset{\text{def}}{=} G(X \multimap Y).$$

Then $\mathbf{L}$ has copowers given by $h \odot X \overset{\text{def}}{=} F(h) \otimes X$. For

$$\begin{aligned} \mathbf{H}(h' \times h, G(C \multimap D)) \cong \mathbf{C}(F(h' \times h), C \multimap D) \cong \mathbf{C}(F(h') \otimes F(h), C \multimap D) \\ \cong \mathbf{C}(F(h'), (F(h) \otimes C) \multimap D)) \cong \mathbf{H}(h', G(F(h) \otimes C) \multimap D) \end{aligned}$$

One must show that the symmetric monoidal closed structure is enriched. For example, we must give a map

$$G(C \multimap C') \times G(D \multimap D') \to G(C \otimes C' \multimap D \otimes D')$$

To give such a map is to give a map

$$FG(C \multimap C') \otimes FG(D \multimap D') \to (C \otimes C') \multimap (D \otimes D')$$

and we use the one that arises from the counit of the adjunction.

Finally we show that passing from (1) to (2) and back to (1) gives the same enriched structure up to isomorphism. But this is trivial, because

$$\mathbf{L}(I, X \multimap Y) \cong \mathbf{L}(X, Y)$$

And we must show that passing from (2) to (1) and back to (2) gives the same adjunction, but this is also fairly trivial because F, being a monoidal left adjoint, must preserve copowers and the monoidal structure:

$$F(X) \cong F(X \times 1) \cong X \odot F(1) \cong X \odot I. \qquad \square$$

Definition 4.3. An *LNL-EWire model* $(\mathbf{L}, \mathbf{H}, \mathbf{L}_0, \mathbf{H}_0, F, G)$ is given by a linear/non-linear model $(\mathbf{L}, \mathbf{H}, F, G)$ together with a small symmetric monoidal full subcategory $\mathbf{L}_0 \subseteq \mathbf{L}$ and a small full subcategory $\mathbf{H}_0 \subseteq \mathbf{H}$ such that F restricts to a functor $\mathbf{H}_0 \to \mathbf{L}_0$.

Proposition 4.4. *Every LNL-EWire model $(\mathbf{L}, \mathbf{H}, \mathbf{L}_0, \mathbf{H}_0, F, G)$ induces an EWire model, when we put*
- $\mathbf{C} = \mathbf{L}_0$*;*
- *The enrichment of* $\mathbf{C}$ *is given by* $\mathbf{C}(X, Y) = G(\mathbf{L}(X, Y))$*;*
- $T = GF$*.*

Proof. Let us verify that the axioms of Definition 2.1 are verified for the quadruplet $(\mathbf{L}_0, \mathbf{H}_0, \mathbf{H}, GF)$.

Axiom (3) follows from Lemma 4.2. For Axiom (4) we need to show that $\mathbf{L}_0$ has copowers by objects of $\mathbf{H}_0$. Lemma 4.2 already gives us that $\mathbf{L}$ has copowers, and so we must show that a copower $h \odot X$ is in $\mathbf{L}_0$ when $h \in \mathbf{H}_0$ and $X \in \mathbf{L}_0$. First, we observe that in $\mathbf{L}$, the following equation holds, because $(-) \otimes X$ preserves copowers since it a right adjoint.

$$(h \odot X) \cong (h \odot (I \otimes X)) \cong (h \odot I) \otimes X \tag{4.1}$$

LNL-EWire models are defined to be such that F restricts to a functor $\mathbf{H}_0 \to \mathbf{L}_0$. We know that $F \cong ((-) \odot I)$, so $(h \odot I)$ is in $\mathbf{L}_0$. It follows that $h \odot X$ is in $\mathbf{L}_0$ because, in an LNL-EWire model, $\mathbf{L}_0$ is a symmetric monoidal subcategory.

For Axiom (5), we must show that $A \otimes -$ preserves copowers. This again follows from $A \otimes -$ having a right adjoint, the closed structure, and hence preserving colimits.

For Axiom (6) we must give an enriched relative monad morphism, but by Lemma 4.2 this can be the identity morphism. $\square$

Theorem 4.5. *For every EWire model* $(\mathbf{C}, \mathbf{H}, \mathbf{H}_0, T)$ *there is an LNL-EWire model*

$$(\mathbf{L}, \mathbf{H}, \mathbf{L}_0, \mathbf{H}_0, F, G)$$

with a $\mathbf{L}_0 = \mathbf{C}$ *and a relative monad morphism* $GFj \to T$.

The proof of this theorem is postponed until after the following lemma, which is hinted in [18] and is likely well known in some circles.

Lemma 4.6. *Let* $\mathbf{C}$ *be a symmetric monoidal category enriched in a cartesian closed category* $\mathbf{H}$*, and suppose that* $\mathbf{H}$ *is also locally presentable as an enriched category. Let* $\mathcal{F}$ *be a class of weighted colimits in* $\mathbf{C}$*. Let* $\bar{\mathbf{C}} \overset{def}{=} [\mathbf{C}^{\mathrm{op}}, \mathbf{H}]_{\mathcal{F}}$ *be the* $\mathbf{H}$*-category of* $\mathcal{F}$*-limit preserving* $\mathbf{H}$*-functors.*

(1) *The restriction of the Yoneda embedding* $\mathbf{C} \to \bar{\mathbf{C}}$ *preserves* $\mathcal{F}$*-colimits, and exhibits the free* $\mathbf{H}$*-colimit cocompletion of* $\mathbf{C}$ *as a category with* $\mathcal{F}$*-colimits.*
(2) *If* $\mathcal{F}$*-colimits distribute over the monoidal structure, in that the canonical maps*

$$\mathrm{colim}^W_{j \in J}(C_j \otimes D) \;\longrightarrow\; (\mathrm{colim}^W_{j \in J} C_j) \otimes D$$

are isomorphisms, then $\bar{\mathbf{C}}$ *admits a symmetric monoidal closed structure, and the Yoneda embedding* $\mathbf{C} \to \bar{\mathbf{C}}$ *strongly preserves the symmetric monoidal structure.*

Proof. For (1), we first recall that it is straightforward to show that the Yoneda embedding factors through $\mathbf{C} \to \bar{\mathbf{C}}$, this follows from the definition of limit. Moreover, it is routine to check that this restricted Yoneda embedding preserves $\mathcal{F}$-colimits: this follows from the Yoneda lemma.

Less trivial is the fact that the category $\bar{\mathbf{C}}$ of $\mathcal{F}$-limit preserving functors is a reflective subcategory of the category of all functors $\hat{\mathbf{C}}$, i.e. that the embedding $\bar{\mathbf{C}} \to \hat{\mathbf{C}}$ has a left adjoint. This can be proved using a general method of orthogonality subcategories (e.g. [18, Ch. 6]), since a functor is in $\bar{\mathbf{C}}$ if it is orthogonal to the canonical morphism

$$\mathrm{colim}^W_i \mathbf{C}(-, d_i) \to \mathbf{C}(-, \mathrm{colim}^W_i d_i)$$

for each diagram in the class $\mathcal{F}$. From this reflection property we conclude that $\bar{\mathbf{C}}$ has all colimits.

Now we consider an $\mathcal{F}$-colimit preserving functor $F : \mathbf{C} \to \mathbf{D}$ where $\mathbf{D}$ is cocomplete. We show that F extends essentially uniquely to a functor $F_! : \bar{\mathbf{C}} \to \mathbf{D}$ that preserves all colimits.

$$\begin{tikzcd}
\mathbf{C} \arrow[r, "\mathbf{C}(=,-)"] \arrow[dr, "F"'] & \bar{\mathbf{C}} \arrow[d, dotted, "F_!"] \\
& \mathbf{D}
\end{tikzcd}$$

First recall that F already extends essentially uniquely to a functor $F_! : \hat{\mathbf{C}} \to \mathbf{D}$, which is right adjoint to the functor $\mathbf{D}(F(=), -) : \mathbf{D} \to \hat{\mathbf{C}}$. Moreover $\mathbf{D}(F(=), -)$ factors through $\bar{\mathbf{C}} \subseteq \hat{\mathbf{C}}$: this follows from the assumption that F preserves $\mathcal{F}$-colimits.

So the adjunction

$$F_! \dashv \mathbf{D}(F(=), -) : \mathbf{D} \to \hat{\mathbf{C}}$$

restricts to an adjunction

$$F_! \dashv \mathbf{D}(F(=), -) : \mathbf{D} \to \bar{\mathbf{C}}$$

and so $F_! : \bar{\mathbf{C}} \to \mathbf{D}$ preserves colimits, and is the required extension of F.

This functor $F_! : \bar{\mathbf{C}} \to \mathbf{D}$ is essentially unique as a colimit preserving functor such that $F_! \circ \mathbf{C}(=, -) \cong F$. Indeed suppose that $L : \bar{\mathbf{C}} \to \mathbf{D}$ preserves colimits and $L \circ \mathbf{C}(=, -) \cong F$. Since $\bar{\mathbf{C}}$ is a reflective subcategory of a presheaf category, it is 'total' hence L has a right adjoint $R : \mathbf{D} \to \bar{\mathbf{C}}$. By the Yoneda lemma,

$$R(d)(c) \cong \bar{\mathbf{C}}(\mathbf{C}(-, c), R(d)) \cong \mathbf{D}(L(\mathbf{C}(-, c)), d) \cong \mathbf{D}(F(c), d)$$

and so, by uniqueness of adjoints, $L \cong F_!$.

As for (2), recall that the presheaf category $\hat{\mathbf{C}}$ has a symmetric monoidal closed such that the Yoneda embedding $\mathbf{C} \to \hat{\mathbf{C}}$ is strongly monoidal. This structure is due to Day [7], and, for $P, Q \in \hat{\mathbf{C}}$, we have

$$(P \otimes_{\mathrm{Day}} Q)(Z) = \int^{X,Y} \mathbf{C}(X \otimes Y, Z) \times P(X) \times Q(Y)$$
$$(P \multimap Q)(X) = \hat{\mathbf{C}}(P, Q(X \otimes -))$$

If Q preserves $\mathcal{F}$-limits, then, because each $(X \otimes -) : \mathbf{C} \to \mathbf{C}$ preserves $\mathcal{F}$-limits too, $(P \multimap Q)$ preserves $\mathcal{F}$-limits. In other words, if $Q \in \bar{\mathbf{C}}$ then $(P \multimap Q) \in \bar{\mathbf{C}}$. This condition is sufficient for extending $(\multimap)$ to a symmetric monoidal closed structure on $\bar{\mathbf{C}} \subseteq \hat{\mathbf{C}}$, such that the reflection $\hat{\mathbf{C}} \to \bar{\mathbf{C}}$ is strongly monoidal [8]. (Note that this does not mean that the embedding $\bar{\mathbf{C}} \to \hat{\mathbf{C}}$ is strongly monoidal.) □

Proof. (Proof of Theorem 4.5.) Let $\mathcal{F}$ be the class of copowers by objects of $\mathbf{H}_0$. Let $\mathbf{L} \overset{\text{def}}{=} [\mathbf{C}^{\mathrm{op}}, \mathbf{H}]_{\mathcal{F}}$ be the category of $\mathbf{H}_0$-power preserving $\mathbf{H}$-functors. By Lemma 4.6, the category $\mathbf{L}$ has a symmetric monoidal closed structure, and the Yoneda embedding $\mathbf{C} \to \mathbf{L}$ preserves it. Let $\mathbf{L}_0 \overset{\text{def}}{=} \mathbf{C}$. By Lemma 4.2, this structure gives rise to an LNL model.

Finally, we must give a relative monad morphism $GFj \to T$. Recall that we assume that in an LNL-EWire model, F restricts to a functor $\mathbf{H}_0 \to \mathbf{L}_0 = \mathbf{C}$. By Lemma 4.2, $F = ((-) \odot I)$, and this restriction property means that $\mathbf{L}_0$ is closed under copowers by objects of $\mathbf{H}_0$. So $GFj = \mathbf{L}(I, j(-) \odot I) \cong \mathbf{C}(I, j(-) \odot I)$, and the relative monad morphism is the (run) structure of the EWire model. □

From Theorem 4.5, we deduce that every EWire model $(\mathbf{C}, \mathbf{H}, \mathbf{H}_0, T)$ induces an exponential construction $! \overset{\text{def}}{=} J \circ \mathbf{L}(I, -) : \mathbf{L} \to \mathbf{L}$ where $\mathbf{L} \overset{\text{def}}{=} [\mathbf{C}^{\mathrm{op}}, \mathbf{H}]_{\mathcal{F}}$. For example, starting from the EWire model of Proposition 2.2

$$(\mathbf{FdC^*\text{-}Alg}^{\mathrm{op}}_{\mathrm{CPU}}, \mathbf{Set}, \mathbb{N}, \mathcal{D})$$

we arrive at an LNL-EWire model

$$\Big([\mathbf{FdC^*\text{-}Alg}_{\mathrm{CPU}}, \mathbf{Set}]_{\mathcal{F}}, \mathbf{Set}, \mathbf{FdC^*\text{-}Alg}^{\mathrm{op}}_{\mathrm{CPU}}, \mathbb{N}, F, G\Big)$$

where $\mathcal{F}$ is the class of finite products, and where $G : [\mathbf{FdC^*\text{-}Alg}_{\mathrm{CPU}}, \mathbf{Set}]_{\mathcal{F}}$ is given by $G(P) = P(\mathbb{C})$. This is close to the LNL model for Proto-Quipper proposed in [37].

Starting from the EWire model of (3.1),

$$(\mathbf{FdC^*\text{-}Alg}^{\mathrm{op}}_{\mathrm{CPU}}, \mathbf{Dcpo}, \mathbb{N}, \mathcal{V})$$

we arrive at an LNL-EWire model

$$\Big([\mathbf{FdC^*\text{-}Alg}_{\mathrm{CPU}}, \mathbf{Dcpo}]_{\mathcal{F}}, \mathbf{Dcpo}, \mathbf{FdC^*\text{-}Alg}^{\mathrm{op}}_{\mathrm{CPU}}, \mathbb{N}, F, G\Big)$$

where $\mathcal{F}$ is again the class of finite products. Starting from the EWire model of subunital maps (3.2),

$$(\mathbf{FdC^*\text{-}Alg}^{\mathrm{op}}_{\mathrm{CPSU}}, \mathbf{Dcpo}, \mathbb{N}, \mathcal{V})$$

we arrive at an LNL-EWire model

$$\Big([\mathbf{FdC^*\text{-}Alg}_{\mathrm{CPSU}}, \mathbf{Dcpo}]_{\mathcal{F}}, \mathbf{Dcpo}, \mathbf{FdC^*\text{-}Alg}^{\mathrm{op}}_{\mathrm{CPSU}}, \mathbb{N}, F, G\Big)$$

where now $[\mathbf{FdC^*\text{-}Alg}_{\mathrm{CPSU}}, \mathbf{Dcpo}]_{\mathcal{F}}$ comprises locally continuous product preserving functors. These models are related to the LNL model for recursion proposed by [22]. Indeed, presheaf models have been proposed for quantum programming by various authors, perhaps beginning from [23].

Summary. Having described a family of embedded languages for (quantum) circuits in Section 1, we described their denotational models in enriched category theory in Section 2. In Section 3, after an exploration of EWire models enriched over dcpos, we explored some of the possible extensions of the languages considered in this work. Finally in Section 4, we established a connection with the Benton's notion of linear-non-linear models.

Acknowledgements. The authors would like to thank Tom Hirschowitz, Bart Jacobs, Shane Mansfield, Paul-André Melliès, Michele Pagani, and Peter Selinger for helpful discussions, but also Bert Lindenhovius, Michael Mislove, and Vladimir Zamdzhiev for hosting the first author at Tulane University during the early stage of the elaboration of this work, and Jennifer Paykin and Robert Rand 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 the EPSRC grant EP/N007387/1.

References

[1] Samson Abramsky and Achim Jung. Domain theory. In *Handbook of logic in computer science*. Oxford University Press, 1994.

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

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

[4] Nick Benton. A mixed linear and non-linear logic: Proofs, terms and models. In *International Workshop on Computer Science Logic*, pages 121–135. Springer, 1994.

[5] 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.

[6] Ugo Dal Lago, Claudia Faggian, Benoît Valiron, and Akira Yoshimizu. The geometry of parallelism: classical, probabilistic, and quantum effects. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, pages 833–845. ACM, 2017.

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

[8] Brian Day. A reflection theorem for closed categories. *Journal of Pure and Applied Algebra*, 2:1–11, 1972.

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

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

[11] Dan R Ghica and Achim Jung. Categorical semantics of digital circuits. In *Proceedings of the 16th Conference on Formal Methods in Computer-Aided Design*, pages 41–48. FMCAD Inc, 2016.

[12] Jean-Yves Girard. Linear logic. *Theoretical computer science*, 50(1):1–101, 1987.

[13] 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.

[14] Ichiro Hasuo and Naohiko Hoshino. Semantics of higher-order quantum computation via geometry of interaction. *Annals of Pure and Applied Logic*, 168(2):404–469, 2017.

[15] 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.

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

[17] Claire Jones and Gordon D Plotkin. A probabilistic powerdomain of evaluations. In *Logic in Computer Science, 1989. LICS'89, Proceedings., Fourth Annual Symposium on*, pages 186–195. IEEE, 1989.

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

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

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

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

[22] Bert Lindenhovius, Michael Mislove, and Vladimir Zamdzhiev. A DCPO-enriched linear/non-linear model. *manuscript*, 2017.

[23] 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.

[24] Paul-André Melliès. Parametric monads and enriched adjunctions. 2012.

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

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

[27] Peter O'Hearn. On bunched typing. *Journal of functional Programming*, 13(4):747–796, 2003.

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

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

[30] Romain Péchoux, Simon Perdrix, Mathys Rennela, and Vladimir Zamdzhiev. Quantum programming with inductive datatypes: Causality and affine type theory. 2020.

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

[32] Robert Rand, Jennifer Paykin, and Steve Zdancewic. Qwire practice: Formal verification of quantum circuits in coq. *EPTCS*, 2017.

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

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

[35] 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.

[36] Mathys Rennela and Sam Staton. Classical control and quantum circuits in enriched category theory. In *Proc. MFPS 2017*, volume 336 of *ENTCS*, pages 257–279. Elsevier, 2018.

[37] Francisco Rios and Peter Selinger. A categorical model for a quantum circuit description language. *arXiv preprint arXiv:1706.02630*, 2017.

[38] Neil J. Ross. *Algebraic and Logical Methods in Quantum Computation.* PhD thesis, 2015.

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

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

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

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

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