Concrete Categorical Model of a Quantum Circuit Description Language with Measurement

Dongho Lee ✉ 🏠
Université Paris-Saclay, CentraleSupélec, LMF, France & CEA, List, France

Valentin Perrelle ✉
Université Paris-Saclay, CEA, List, France

Benoît Valiron ✉ 🏠
Université Paris-Saclay, CentraleSupélec, LMF, France

Zhaowei Xu ✉
Université Paris-Saclay, LMF, France

Abstract

In this paper, we introduce dynamic lifting to a quantum circuit-description language, following the Proto-Quipper language approach. Dynamic lifting allows programs to transfer the result of measuring quantum data – qubits – into classical data – booleans – . We propose a type system and an operational semantics for the language and we state safety properties. Next, we introduce a concrete categorical semantics for the proposed language, basing our approach on a recent model from Rios&Selinger for Proto-Quipper-M. Our approach is to construct on top of a concrete category of circuits with measurements a Kleisli category, capturing as a side effect the action of retrieving classical content out of a quantum memory. We then show a soundness result for this semantics.

2012 ACM Subject Classification Theory of computation → Categorical semantics

Keywords and phrases Categorical semantics, Operational semantics, Quantum circuit description language

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.51

Funding This work was supported in part by the French National Research Agency (ANR) under the research projects SoftQPRO ANR-17-CE25-0009-02 and PPS ANR-19-CE48-0014, by the DGE of the French Ministry of Industry under the research project PIA-GDN/QuantEx P163746-484124, and by STICAMSUD 21-SITC-20 Qapla'.

Acknowledgements The authors want to thank Christophe Chareton and Sébastien Bardin for enlightening discussions.

1 Introduction

In quantum computation, one considers a special kind of memory where data is encoded on the state of objects governed by the laws of quantum mechanics. The basic unit for quantum data is the quantum bit, or qubit, and in general, a quantum memory is understood as consisting in individually addressable qubits. As derived in the no-cloning theorem [23], qubits are *non-duplicable* objects. The state of a quantum memory can be represented by a unit vector in a complex Hilbert space. Elementary operations on qubits consist in unitary operations on the state space, called *quantum gates*, and *measurements*, which are probabilistic operations returning a classical boolean.

The usual model for quantum computation is the notion of quantum circuits. Quantum circuits consist of quantum gates and wires. A wire represents a qubit, and each gate, attached to one or several wires, is a unitary operation acting on the corresponding qubits. In this model, a computation consists in allocating a quantum register, applying a circuit (i.e. the list of gates, in order), followed with a measurement to get back classical data.

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 51; pp. 51:1–51:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The *QRAM model* [8] generalizes circuits: in this model, quantum computation is performed under the control of a classical host. It emits a stream of interleaved (pieces of) quantum circuits and measurements to the quantum co-processor. The quantum co-processor executes the instructions while returning the results of measurements on the fly to the classical host. In this model, the computation is not a fixed linear list of quantum gates: the quantum gates emitted to the quantum co-processor might depend on the results of intermediate measurements. Although quantum circuits and QRAM models are equivalent in terms of expressive power, practical quantum computation is more likely to be based on the QRAM model. For this reason, many programming languages and their semantics are based on the QRAM model [17, 11, 7, 22, 24, 20, 18].

An interesting implication of this model is that the quantum circuit construction in the classical host can be dependent on the result of a measurement: there is a transfer of information from the quantum co-processor to the classical host. This feature is implemented for example in Quipper [7, 1] and QWire [11, 12]. Following Quipper's convention, we call this transfer *dynamic lifting*: classical information is *lifted* from the quantum co-processor to the classical host. Some use-cases for dynamic lifting are as follows. First, quantum error correction typically interleaves unitaries and measurements. Other examples include subroutines with repeat-until-success, where the result of the measurement on one wire says whether the computation succeeded or not [4], and measurement-based quantum computation.

The classical control over the circuit construction imposed by dynamic lifting has not been explicitly formalized in the semantics of the circuit construction languages using it. To illustrate this problem, let us look at the program in Eq. (1). The syntax we use is presented in Section 2, so we explain what the program does here: The program measures the qubit v_c and obtains the updated state of the qubit together with the resulting boolean b. Based on b, it then either allocates a new qubit initialized by true, then free the qubit v_c, or simply returns ${v_c}^1$. Despite this simple structure, the program does not correspond to a circuit because of the classical control.

$$\text{exp} \quad ::= \quad \mathbf{let}\ \langle b, v_c\rangle = \text{meas}(v_c)\ \mathbf{in} \quad \mathbf{if}\ b\ \mathbf{then}\ \langle \text{init}(\text{tt}), \text{free}(v_c)\rangle\ \mathbf{else}\ \langle v_c, *\rangle \tag{1}$$

In QWire, the operational semantics performs normalization for composition and unbox operations but the classical control by dynamic lifting is hidden in the host term within the unbox. In Quipper, the operational semantics is encoded in Haskell's monadic type system and captures a notion of dynamic circuit including measurements. However, this semantics has never been fully formalized in the context of higher-order, functional quantum languages.

Besides operational semantics, programming languages for quantum circuits have been formalized using denotational semantics based on density matrices [11] and categorical semantics based on symmetric monoidal categories [16, 14, 9, 14, 5], or on the category of C^*-algebras [19, 13]. However, these examples of formalization do not solve the problem in that they either ignore the structure of circuit or keep the term with dynamic lifting abstract. In particular, in [14, 5], the authors construct expressive categorical models for the family of circuits – or parameterized circuits – and linear dependent type theory, respectively, while they do not provide semantics of dynamic lifting explicitly.

Our goal in this paper is to find a model and formalize a semantics for interleaved quantum circuits and dynamic lifting. The problem rests in how to analyze the structure of the computation without requiring the quantum co-processor to decide on the value of

[1] The program actually returns a pair consisting of a qubit and the unit term $*$ so that it is well-typed; we assume that the return type of free is the unit-type.

measurement. The interest of such a model and semantics is that it can serve as a test-bed to explore properties of the language. Mixing non-duplicable data, higher-order and circuits in a language yields a non-trivial system, and dependent-types were for instance only added recently for Proto-Quipper [5], with the use of such tools.

Contributions. In this paper, we propose both a small step operational semantics and a categorical semantics for a typed language – called Proto-Quipper-L – extending quantum lambda calculus [17] with circuit construction operators (box and unbox) and circuit constants. The formalization extends the one of Proto-Quipper [15]: circuits are generalized to *quantum channels* enabling the formalization of the semantics of the dynamic lifting. A quantum computation that only consists of unitary gates deterministically reduces to only one possible value. On the other hand, a quantum computation with dynamic lifting might reduce to distinct values depending on the results of the measurements. We support this by making circuits not only lists but trees branching over the results of measurements: we call such objects *quantum channels*. The language is then extended with a notion of branching terms, representing the possible choices along the computation. We prove the usual safety properties for the language: subject reduction, progress and termination (Lemmas 7, 8 and 9).

Next, we propose a sound categorical model for Proto-Quipper-L. The model is based on Proto-Quipper-M, the work of Rios&Selinger [14]. It consists of two categories: a symmetric monoidal category abstracting the notion of circuit, and an extension capturing classical computation and circuit manipulation. A morphism in the former category becomes a circuit-element in the extension. If this construction captures a sound notion of circuit, in its abstract formulation it is not *a priori* amenable to dynamic lifting. To answer the issue, we propose a concrete instantiation of the model in which dynamic lifting can be represented: we define a concrete, symmetric monoidal closed category for representing quantum channels, and, based on the construction proposed in [14], a linear category admitting a strong monad F representing the branching side-effect associated with the measurement. Following [10, 21], we use the Kleisli category $\overline{\overline{M}}_F$ to represent terms of Proto-Quipper-L.

In fact, branching monad in our categorical model corresponds to the `Circ` monad in Quipper which models non-deterministic branching in the level of type system. Although it is standard to use monad to model non-deterministic side effects, it was not clear whether such a monadic structure could be set up on the categorical model of parameterized circuits by Rios&Selinger [14]. The main result of the paper is to show how to do it, using a concrete category of quantum channels. We validate the model by showing a soundness property (Theorem 18).

2 Syntax, Types and Operational Semantics

In this section, we present the syntax of a minimal lambda-calculus for manipulating quantum channels and booleans. The language is an extension of Proto-Quipper [15].

In Proto-Quipper, the quantum lambda calculus presented in [17] is extended with circuit operators and constants. Circuit operators give an efficient way to construct circuits instead of having to sequentially apply all the gates one-by-one. Specifically, two operators on circuits are added to the language: the box operator allows us to use quantum circuits as classical data, while the unbox operator applies a boxed circuit to an argument (usually a structured set of qubits called *pattern*). Boxed circuits are first-class objects and can come with useful circuit operators like reverse and control. Technically, a circuit object in Proto-Quipper can be seen as a tuple (p, C, M) where p is structured set of the input wires of a circuit C, matching the input type of the circuit. M is a term corresponding to the output of the circuit. Along the reduction of M, the circuit C is possibly updated with new gates. The term M is open, and the output wires of C are used in M linearly, meaning that each output wire appears in M exactly once.

However, Proto-Quipper does not support dynamic lifting within circuits. To extend Proto-Quipper with dynamic lifting, we replace circuits with quantum channels and redefine the circuit operators box and unbox over quantum channels.

2.1 Quantum Channels

A quantum channel is the generalization of a quantum circuit: a tree structure where branching captures the action of measurement. In essence, quantum channels are instances of QCAlg, defined by the following grammar.

$$(\text{QCAlg}) \qquad Q, Q_1, Q_2 \ ::= \ \epsilon(W) \ \mid \ U(W)\ Q \ \mid \ \text{init}\ b\ w\ Q \ \mid \ \text{meas}\ w\ Q_1\ Q_2 \ \mid \ \text{free}\ w\ Q.$$

The symbols w, b, and W respectively refer to wires, booleans, and finite sets of wires. The channel $\epsilon(W)$ stands for the empty computation on the qubits W. $U(W)\ Q$ represent the unitary operator U acting on the qubits W, followed by the operations stored in the quantum channel Q. In general, U can range over a fixed set of unitary operations: we write arity(U) for the arity of U. The operator init $b\ w\ Q$ creates and initializes the wire w in state b, followed by the channel Q possibly using the newly allocated qubit. The operator meas represents the conditional branching on the result of a measurement. In our interpretation, a measurement is non-destructive: the wire being measured is still allocated and can be acted upon. The two channels Q_1 and Q_2 stand for the two possible branches to follow based on the measurement. Finally, free $w\ Q$ frees the qubit w before running Q. From now on, we call the instance of QCAlg as quantum channel object.

We define a notion of validity for quantum channels: Q is *valid* whenever, for instance, an init-node introduces a non-existing wire, or whenever a free-node acts on an existing wire. One subtlety consists in deciding what is an output wire for a branching quantum channel. For instance, consider $Q = \text{meas}\ w_1\ (\text{init}\ b\ w_2\ (\epsilon\{w_1, w_2\}))\ (\epsilon\{w_1\})$. This quantum channel admits as output $\{w_1, w_2\}$ on the left branch and $\{w_1\}$ on the right branch. We formalize this notion and write $\mathbf{out}(Q)$ for a tree-structured set of outputs of Q: Here, $\mathbf{out}(Q)$ is $[\{w_1, w_2\}, \{w_1\}]$. We also define $\mathbf{all}(Q)$ to stand for the set of *all* of the wires appearing in Q, and $\mathbf{in}(Q)$ for the set of input wires. We give a formal definition of validity from the following definition of state of quantum channel.

▶ **Definition 1** (State of quantum channel). A *bunch* of elements of X is a binary tree where only the leaves are indexed, with elements of X. Formally, if x ranges over X, a bunch is built from the grammar $c_1, c_2 \ ::= \ x \mid [c_1, c_2]$. The ternary relation "st" formalizes what it means for a quantum channel to be valid. It is defined as the smallest relation satisfying the rules presented in Table 1. Informally, we say that a quantum channel Q is *valid* whenever there is some set of wires V and a bunch of sets of wires c such that $\mathbf{st}(Q, V, c)$ is derivable. Moreover, such V and c are called input and output wires of Q, respectively.

2.2 Syntax of the Terms

Having extended the notion of circuit to the notion of quantum channel, we turn to the question of the definition of the language. Compared to previous Proto-Quipper instances [15, 14, 5], there are two main changes. The first one concerns the circuit constant; the other one concerns the fact that one has to deal with non-deterministic branching computations.

Table 1 Valid quantum channel.

$$\frac{}{\mathbf{st}(\epsilon(W),\ W,\ W)} \qquad \frac{\begin{array}{l}W_1 \subseteq W\\ \mathrm{arity}(U) = |W_1|\end{array} \qquad \mathbf{st}(Q,\ W,\ c)}{\mathbf{st}(U(W_1)\ Q,\ W,\ c)} \qquad \frac{w \notin W \qquad \mathbf{st}(Q,\ W \cup \{w\},\ c)}{\mathbf{st}(\mathrm{init}\ b\ w\ Q,\ W,\ c)}$$

$$\frac{w \in W \qquad \mathbf{st}(Q_1,\ W,\ c_a) \qquad \mathbf{st}(Q_2,\ W,\ c_b)}{\mathbf{st}(\mathrm{meas}\ w\ Q_1\ Q_2,\ W,\ [c_a, c_b])} \qquad \frac{w \in W \qquad \mathbf{st}(Q,\ W \setminus \{w\},\ c)}{\mathbf{st}(\mathrm{free}\ w\ Q,\ W,\ c)}$$

Table 2 Proto-Quipper-L: terms, values, patterns, branching terms, branching values, types and pattern types.

$$\begin{array}{rl}
M, M_a, M_b & ::= x \ \mid\ * \ \mid\ \mathrm{tt} \ \mid\ \mathrm{ff} \ \mid\ (p, Q, m) \ \mid\ \lambda x.M \ \mid\ M_a M_b \ \mid\ \langle M_a, M_b\rangle \ \mid \\
& \qquad \mathbf{let}\ \langle x, y\rangle = M_a\ \mathbf{in}\ M_b \ \mid\ \mathbf{if}\ M\ \mathbf{then}\ M_a\ \mathbf{else}\ M_b \ \mid\ \mathrm{box}_P \ \mid\ \mathrm{unbox} \\
V, V_a, V_b & ::= x \mid * \mid \mathrm{tt} \mid \mathrm{ff} \mid \lambda x.M \mid \langle V_a, V_b\rangle \mid (p, Q, v) \mid \mathrm{box}_P \mid \mathrm{unbox} \mid \mathrm{unbox}(V) \\
p, p_a, p_b & ::= x \ \mid\ * \ \mid\ \langle p_a, p_b\rangle \\
m, m_a, m_b & ::= M \mid\ [m_a, m_b] \\
v, v_a, v_b & ::= V \mid\ [v_a, v_b] \\
A, A_a, A_b & ::= I \ \mid\ \mathrm{bool} \ \mid\ \mathbf{qubit} \ \mid\ \mathrm{QChan}(P, A) \ \mid\ A_a \multimap A_b \ \mid\ A_a \otimes A_b \ \mid\ !\ A \\
P, P_a, P_b & ::= I \ \mid\ \mathbf{qubit} \ \mid\ P_a \otimes P_b
\end{array}$$

We call the new language Proto-Quipper-L and define it as shown in Table 2. The constant $*$ stands for the unit term, while tt and ff stands for the booleans true and false. The term (p, Q, m) corresponds to a quantum channel object: p is a *pattern*: a structured set of input wires of a *valid* quantum channel Q, and m is a *branching term* that will match the branching structure of Q for valid quantum channel objects. For simplicity wire identifiers and term variables range over the same set of names. We then have the quantum channel operators box and unbox from Proto-Quipper: box makes a quantum channel out of a function, while unbox turns a quantum channel into a function. The rest of the constructors of the language are standard: abstraction, application, pair, let, and conditional statements. We define a notion of value in the standard way, apart from the fact that unbox(V) is also a value (as it is a function). Finally, branching terms and values are constructed using the branching constructor $[-, -]$. A term of the form $[M, N]$ represents a computation that has probabilistically branched and that is performing either M or N. This is novel compared to Proto-Quipper. We denote the set of free variables of a term m with $\mathrm{FV}(m)$.

One could argue that the language is missing constructors for unitary gates, qubit allocation and measurement. As in the case of Proto-Quipper, they can be defined with the unbox and quantum channel object. For instance, we can construct a measurement operation inputting a qubit and outputting a boolean and the measured wire as meas $::=$ unbox$\,(x, \mathrm{meas}\ x\ (\epsilon\{x\})\ (\epsilon\{x\}), [\langle \mathrm{tt}, x\rangle, \langle \mathrm{ff}, x\rangle])$. The tuple consists of a singleton wire name x, the quantum channel (meas x $\epsilon\{x\}$ $\epsilon\{x\}$), and the branching tree $[\langle \mathrm{tt}, x\rangle, \langle \mathrm{ff}, x\rangle]$. Note that the measurement operator we wrote here returns both a qubit and a boolean: we could discard the qubit with the use of a quantum channel constructor "free" if we only wanted to output a boolean. Similarly, we can also build the macros init_b and free which respectively allocates a new qubit in state b and frees a qubit, as init_b $::=$ unbox$(*, \mathrm{init}\ b\ x\,(\epsilon\{x\}), x)\,*$ and free $::=$ unbox$\,(x, \mathrm{free}\ x\ (\epsilon(\emptyset)), *)$. We can similarly define terms for unitary application by encapsulating the QCAlg constructors U inside a quantum channel object.

Table 3 Proto-Quipper-L: Typing Rules.

$$\frac{}{!\Delta,\ (x:A) \vdash x : A}\,(\text{var}) \qquad \frac{!\Delta,\ Q \vdash M : !A}{!\Delta,\ Q \vdash M : A}\,(\text{d}) \qquad \frac{!\Delta \vdash V : A \qquad V \text{ is value}}{!\Delta \vdash V : !A}\,(\text{p}) \qquad \frac{}{!\Delta \vdash * : I}\,(\text{I})$$

$$\frac{!\Delta,\ Q,\ (x:A_a) \vdash M : A_b}{!\Delta,\ Q \vdash \lambda x.M : A_a \multimap A_b}\,(\multimap_I) \qquad \frac{!\Delta,\ Q_a \vdash M_a : A_a \multimap A_b \qquad !\Delta,\ Q_b \vdash M_b : A_a}{!\Delta,\ Q_a,\ Q_b \vdash M_a M_b : A_b}\,(\multimap_E)$$

$$\frac{!\Delta,\ Q_a \vdash M : \text{bool} \qquad \begin{array}{l} !\Delta,\ Q_b \vdash M_1 : A \\ !\Delta,\ Q_b \vdash M_2 : A \end{array}}{!\Delta,\ Q_a,\ Q_b \vdash \textbf{if}\ M\ \textbf{then}\ M_1\ \textbf{else}\ M_2 : A}\,(\text{if}) \qquad \frac{\begin{array}{c} !\Delta,\ Q_a \vdash M_a : A_a \otimes A_b \\ !\Delta,\ Q_b,\ (x:A_a),\ (y:A_b) \vdash M_b : A \end{array}}{!\Delta,\ Q_a,\ Q_b \vdash \textbf{let}\ \langle x, y\rangle = M_a\ \textbf{in}\ M_b : A}\,(\otimes_E)$$

$$\frac{}{!\Delta \vdash \text{tt} : \text{bool}}\,(\text{tt}) \qquad \frac{}{!\Delta \vdash \text{ff} : \text{bool}}\,(\text{ff}) \qquad \frac{!\Delta,\ Q_a \vdash M_a : A_a \qquad !\Delta,\ Q_b \vdash M_b : A_b}{!\Delta,\ Q_a,\ Q_b \vdash \langle M_a, M_b\rangle : A_1 \otimes A_2}\,(\otimes_I)$$

$$\frac{}{!\Delta \vdash \text{box}_P : !(P \multimap A) \multimap !\text{QChan}(P,A)}\,(\text{box}) \qquad \frac{}{!\Delta \vdash \text{unbox} : \text{QChan}(P,A) \multimap (P \multimap A)}\,(\text{unbox})$$

$$\frac{\gamma_a \vdash m_a : A \qquad \gamma_b \vdash m_b : A}{\gamma_a \times \gamma_b \vdash [m_a, m_b] : A}\,(\text{b}) \qquad \frac{p \vDash P \qquad \mathbf{vBind}(!\Delta, \mathbf{out}(Q), m, A)}{!\Delta \vdash (p, Q, m) : !\text{QChan}(P,A)}\,(\text{QChan}_I)$$

2.3 Type System

Types of Proto-Quipper-L are defined as in Table 2. Following the standard strategy [17, 15, 3] to account for the non-duplicability brought by the quantum memory, we are using a type system based on linear logic [6]. Types consist in the constant types I, bool, **qubit**; the function type $A_a \multimap A_b$; the type for pairs $A_a \otimes A_b$; the type $!A$ of duplicable terms of type A; the type of quantum channels $\text{QChan}(P, A)$ with input of type P and output of type A, where P refers to patterns, that is, first-order types constructed from **qubit**s and tensors.

Conventionally, a typing judgment consists in a typing context, which maps variables to types, and a term assigned with a type. However, in Proto-Quipper-L, the term can be a branching term. Although the terms of all branches in a branching term are assigned with the same type, they may have different typing contexts. This is formalized in two distinct definitions of typing judgments: regular typing judgments $\Gamma \vdash M : A$ where where Γ is a list of typed variables and M is a non-branching term, and *branching typing judgments* $\gamma \vdash m : A$, where m is a branching term and γ is an branching typing context: $\gamma ::= \Gamma \mid \gamma_1 \times \gamma_2$.

A judgment is valid if it can be derived from the typing rules presented in Table 3. The rules ensure that various constraints necessary for soundness are satisfied. One can note that all terms constituting a branching term share the same type ; that valid branching typing judgments have branching contexts and terms with the same tree structure ; that a quantum channel object is duplicable with type !QChan ; that box sends a duplicable function to a duplicable quantum channel object, and that unbox sends a quantum channel object to a function. One can also note that only values can be promoted to duplicable objects: this is due to the call-by-value reduction strategy we follow. The relation $\mathbf{vBind}(!\Delta, \mathbf{out}(Q), m, A)$ in the (QChan_I) rule ensures that one can derive typing derivations for each term leaf of m given that the output wires of the quantum channel Q is assigned with type **qubit** within the typing context. The relation $p \vDash P$ simply states that the shapes of p and P match and that the variables occurring in p are pairwise distinct.

The rules for **vBind** are found in Table 4. Note that the non-linear context $!\Delta$ is a list of pairs of variables and non-linear types. We denote by $\mathbf{FV}(!\Delta)$ the set of variables in $!\Delta$. In fact, the condition $(X \cap \mathbf{FV}(!\Delta) = \emptyset)$ is implicitly assumed by the definition of the typing judgment $(!\Delta,\ (x : \mathbf{qubit})_{x \in X} \vdash M : A)$.

Table 4 Validity of binding in quantum channel constant.

$$\frac{X \cap \mathbf{FV}(!\Delta) = \emptyset \qquad !\Delta,\ (x : \mathbf{qubit})_{x \in X} \vdash M : A}{\mathbf{vBind}(!\Delta, X, M, A)}(\mathbf{vBind}_{nb}) \qquad \frac{}{* \vDash I} \qquad \frac{}{x \vDash \mathbf{qubit}}$$

$$\frac{\mathbf{vBind}(!\Delta, c_a, m_a, A) \qquad \mathbf{vBind}(!\Delta, c_b, m_b, A)}{\mathbf{vBind}(!\Delta, [c_a, c_b], [m_a, m_b], A)}(\mathbf{vBind}_{b}) \qquad \frac{\forall i,\ p_i \vDash P_i \quad \mathbf{FV}(p_1) \cap \mathbf{FV}(p_2) = \emptyset}{\langle p_1, p_2 \rangle \vDash P_1 \otimes P_2}$$

▶ **Example 2.** In Section 2.2 we defined three macros: meas, free and init_b. We can type meas with $!(\mathbf{qubit} \multimap (\mathrm{bool} \otimes \mathbf{qubit}))$ and free with $!(\mathbf{qubit} \multimap I)$. For init_b, note that because there is a final argument "$*$", it is really an application and we can therefore only type it with **qubit** and not !**qubit**: this is expected, as we don't want to be able to construct duplicable qubits. With these types, we can now type the term exp in Eq (1) of Section 1: we can derive the judgment $v_c : \mathbf{qubit} \vdash \mathrm{exp} : \mathbf{qubit} \otimes I$.

▶ Remark 3. In general, there can be more than one typing derivation for a typing judgment but, for the types I, **qubit** or bool, there is a unique typing derivation when the term is a value. We call these types *basic types*.

2.4 Operational Semantics

The computational model we have in mind for the language is a reduction-based semantics specialized to circuit construction: the operational semantics is modeling an I/O side-effect, where gates are emitted and buffered in a quantum channel. Based on Proto-Quipper [15], the operational semantics we describe therefore updates a *configuration* consisting of a pair (Q, m): a buffered QCAlg object and a branching term. The term m is reduced up to a value representing the final state of the computation. Along the computation, quantum gates might be emitted to the co-processor: the quantum channel Q keeps track of these. One can notice that a configuration corresponds to a quantum channel constant without the input wires, where there is a minor relaxation on the linearity of the output wires of Q in m which will be recovered when we define well-typed configuration.

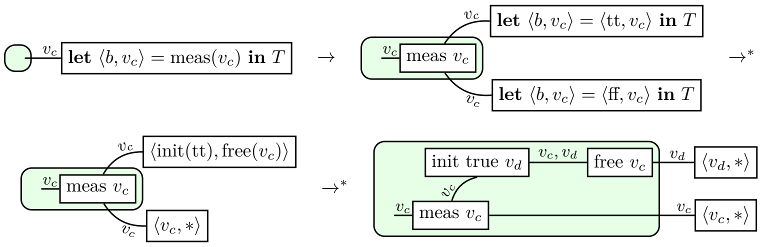
▶ **Definition 4.** A *circuit-buffering configuration* is a pair (Q, m) as described above. It is said to be *valid* whenever Q is valid, Q and m share the same tree-structure, and whenever output wires of Q corresponds to free variables of m (following the tree-structure). So for instance, $V \subseteq \mathbf{FV}(M)$ implies the validity of $(\epsilon(V),\ M)$, and whenever $(Q_1,\ m_a)$ and $(Q_2,\ m_b)$ are valid so is $(\mathrm{meas}\ w\ Q_1\ Q_2,\ [m_a, m_b])$.

▶ Remark 5. In order to define the reduction rules, we need to be able to extend a configuration with new wires. For instance, let us consider the term $(\epsilon\{x, y\}, \mathbf{if}\ (N\,x)\ \mathbf{then}\ y\ \mathbf{else}\ y)$ with N some term not containing y. Evaluating this configuration requires to first evaluate $N\,x$ and possibly append a few gates to $\epsilon\{x, y\}$. However, this can be factorized as first evaluating $(\epsilon\{x\}, N\,x)$ to (Q, V) and then adding back the wire y to the resulting quantum channel Q. We therefore define an operator **extend** taking a quantum channel and a set of wire names, adding them as unused wires to the quantum channel.

The reduction rules for Proto-Quipper-L are defined in Table 5. (See Section A.1 for more details.) Rules (a.x) always hold (b ranges over $\{\mathrm{tt}, \mathrm{ff}\}$). In Rules (b.1), p is a pattern of same shape as P made from dynamically allocated fresh variables. In Rule (b.2), p and V have the same shape, and σ is a substitution mapping p to V. Provided that $(Q, m) \to (Q', m')$, we have

Table 5 Reduction rules for operational semantics.

(a.1)	$(\epsilon(W), (\lambda x.M)V) \to (\epsilon(W), M[V/x])$	$(\epsilon(W_{C[M]}),\ C[M]) \to (\mathbf{extend}(Q, W_{C[-]}),\ C[m])$	(c)
(a.2)	$(\epsilon(W), \mathbf{let}\ \langle x, y\rangle = \langle V, U\rangle\ \mathbf{in}\ M) \to (\epsilon(W), M[V/x, U/y])$	$(Q,\ [m_a, m_b]) \to (Q',\ [m_c, m_d])$	(d.1)
(a.3)	$(\epsilon(W), \mathbf{if}\ \mathrm{b}\ \mathbf{then}\ M_{\mathrm{tt}}\ \mathbf{else}\ M_{\mathrm{ff}}) \to (\epsilon(W), M_b)$	$(Q,\ [m_a, v]) \to (Q',\ [m_c, v])$	(d.2)
(b.1)	$(\epsilon(\emptyset), \mathrm{box}_P\, V) \to (\epsilon(\emptyset), (p, \epsilon(\mathrm{FV}(p)), Vp))$	$(Q,\ [v, m_b]) \to (Q',\ [v, m_d])$	(d.3)
(b.2)	$(\epsilon(\mathbf{FV}(V)), (\mathrm{unbox}(p,\ Q,\ u))V) \to (\sigma(Q), \sigma(u))$	$(G\ Q_1,\ m_a) \to (G\ Q_3,\ m_c)$	(d.3)



v_c
let $\langle b, v_c\rangle = \mathrm{meas}(v_c)$ in T
$\to$
v_c
meas v_c
v_c
let $\langle b, v_c\rangle = \langle \mathrm{tt}, v_c\rangle$ in T
v_c
let $\langle b, v_c\rangle = \langle \mathrm{ff}, v_c\rangle$ in T
$\to^*$
v_c
meas v_c
v_c
$\langle \mathrm{init}(\mathrm{tt}), \mathrm{free}(v_c)\rangle$
v_c
$\langle v_c, *\rangle$
$\to^*$
init true v_d
v_c, v_d
free v_c
v_d
$\langle v_d, *\rangle$
v_c
v_c
meas v_c
v_c
$\langle v_c, *\rangle$

Figure 1 Reduction of the term of Eq (1).

$(\epsilon(\emptyset), (p, Q, m)) \to (\epsilon(\emptyset), (p, Q', m'))$. Provided that we have that $(\epsilon(W_M),\ M) \to (Q,\ m)$, that $\mathbf{all}(Q) \cap W_N = \emptyset$ and that $\mathbf{all}(Q) \cap W_V = \emptyset$, the class of rules (c) apply. There, $C[-]$ ranges over $[-]N$, $V[-]$, $\langle[-], N\rangle$, $\langle V, [0]\rangle$, if $[-]$then M_a else M_b and let $\langle x, y\rangle = [-]$ in N. We use syntactic sugar for combining terms and branching terms, as in $C[m]$. It corresponds to the term constructor applied to each leaf of m, for instance: for $m = [[N_1, N_2], N_3]$, $C[m] := [[C[N_1], C[N_2]], C[N_3]]$. In Rules (d.x), Q stands for meas $w\ Q_1\ Q_2$ and Q' for meas $w\ Q_3\ Q_4$. These rules apply whenever $(Q_1,\ m_a) \to (Q_3,\ m_c)$ and $(Q_2,\ m_b) \to (Q_4,\ m_d)$. In (d.3), G ranges over $U(W)$, init $b\ w$ and free w.

▶ **Example 6.** As an example, we show the reduction of the term shown in Eq. (1). For convenience, we define T as **if** b **then** $\langle \mathrm{init}(\mathrm{tt}), \mathrm{free}(v_c)\rangle$ **else** $\langle v_c, *\rangle$. Figure 1 shows the reduction of the term. (check Section A.2 for more details). We use a graphical representation for configuration. A green box represents a quantum channel whose leaves are linked to square-boxed terms. The edges represent bundles of wires, which can contain multiple wires and can be empty.

In the first line, the measurement in the term is reduced by the structural rule for *let* and the reduction rule for measurement creating a branching term. Then, each term at a leaf of the tree is reduced into the left-most configuration of the second line. Note how classical computation can happen inside the leaves. The second line of the figure shows the application of initialization and free operation. In particular, note how the tree expands as the computation progresses.

2.5 Type safety for Proto-Quipper-L

In order to state the type safety theorem, we need to extend typing derivations to configurations. We write $!\Delta \vdash (Q, m) : A$ whenever $\mathbf{vBind}(!\Delta, \mathbf{out}(Q), m, A)$ and Q is valid. Note that the definition implies that the output wires of the quantum channel correspond to the linear variables of type **qubit** in the context of the typing derivation. In any case, we can now state type safety for Proto-Quipper-L, as follows.

▶ **Lemma 7** (Subject reduction). *For any configurations* (Q_1, m_1) *and* (Q_2, m_2) *such that* $(Q_1, m_1) \to (Q_2, m_2)$*, if* $\vdash (Q_1, m_1) : A$*, then* $\vdash (Q_2, m_2) : A$*.*

▶ **Lemma 8** (Progress). *If* $(\vdash (Q_1, m_1) : A)$*, then either there exists* (Q_2, m_2) *such that* $(Q_1, m_1) \to (Q_2, m_2)$ *or* m_1 *is a branching value.*

▶ **Lemma 9** (Termination). *Given a well-typed configuration* $\vdash (Q, m) : A$*, any reduction sequence starting with* (Q, m) *is terminating.*

3 Categorical semantics

In this section, we turn to the question of developing a categorical semantics for Proto-Quipper-L. The categorical semantics of circuit-description languages and Proto-Quipper in particular originates from Rios&Selinger [14]. They developed a model parametrized by a symmetric monoidal category M. In their model one can therefore interpret higher-order circuit-description languages, and several extensions of the semantics [5, 9] have been discussed. However, none of them were shown to be able to capture dynamic lifting: the possibility to change behavior depending on the result of a measurement.

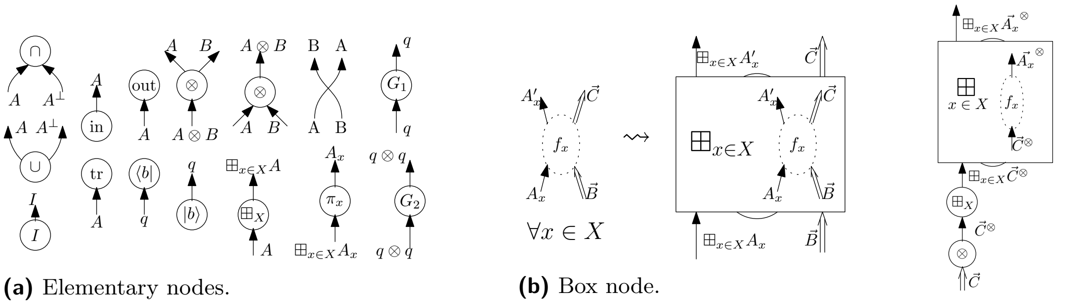
Our proposal. What we propose in this paper is a concrete, symmetric monoidal category M such that applying Rios&Selinger's construction gives us also access to an interpretation of dynamic lifting. The model we propose follows Moggi's categorical interpretation of side effect [10] and models the action of measurement using a (strong) monad. Our semantics is therefore based on: (1) A category of *diagrams*, serving as graph-like abstractions of quantum channels. This category is compact-closed and features products: it matches the requirements of the base category $\overline{M}$ in Rios&Selinger's work. This category is discussed in Section 3.1. (2) The category $\overline{\overline{M}}$, extending $\overline{M}$ with the same procedure as Rios&Selinger. This category is the category of *values*, following Moggi's computational interpretation. It is presented in Section 3.2. (3) A strong monad on $\overline{\overline{M}}$ that we denote with F. This monad encapsulates *computations* involving measurements: a general term of Proto-Quipper-L is therefore interpreted inside the Kleisli category $\overline{\overline{M}}_F$: This is the main novelty compared to other models of Proto-Quipper-like languages [14, 5, 9], and the critical reason for the possibility to interpret dynamic lifting. This is discussed in Section 3.4.

Finally, we discuss the soundness of the model and presents a few examples. For sake of space, the presentation of the definitions and results is only kept to a minimum: more information is available in the appendix.

3.1 Categories of Diagrams

In this section, we aim at building a category of quantum channels. We first define a graph-based language: we call the corresponding terms *diagrams* to distinguish them from the terms of QCAlg of Section 2.1: these are directed graphs with edges labeled with *marks*. We then build the category $\overline{M}$ out of these terms.

Marks. Formally, we define *marks* with the grammar $M ::= q \mid M \otimes M \mid \boxplus_{i \in X} M_i \mid M^{\perp}$, where X ranges over the class of sets, and is subject to the equivalence relation defined as $\boxplus_{i \in I} \boxplus_{j \in J} M_{(i,j)} = \boxplus_{j \in J} \boxplus_{i \in I} M_{(i,j)}$; $(M_1 \otimes M_2)^{\perp} = M_1^{\perp} \otimes M_2^{\perp}$; $(\boxplus_{i \in I} M_i)^{\perp} = \boxplus_{i \in I} (M_i^{\perp})$; $(M^{\perp})^{\perp} = M$; $\boxplus_{l \in L} \boxplus_{x \in l} M_{(l,x)} = \boxplus_{x \in l_1 ++ \cdots ++ l_n} M_{(l,x)}$, whenever $L = [l_1, \ldots, l_n]$. Note that $\boxplus_{i \in \emptyset} M_i$ acts as a unit for $\boxplus$: we denote it with I. If $A = [A_1 \ldots A_n]$ is a list of marks, we use the notation $A^{\otimes}$ for $A_1 \otimes \cdots \otimes A_n$. We also use a binary notation for $\boxplus$ when the indexed set contains 2 elements: $\boxplus_{x \in \{a,b\}} A_x = A_a \boxplus A_b$.



(a) Elementary nodes.

(b) Box node.

(c) Product.

Figure 2 Diagram Nodes and Product.

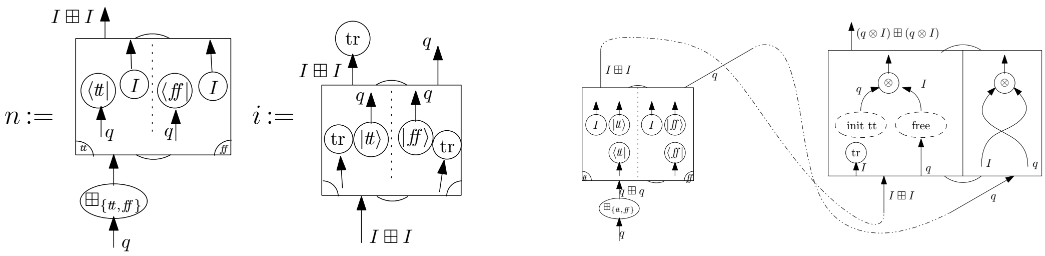
▶ Remark 10. Box node is a way of representing additive connectives of intuitionistic linear logic. It can be considered as a set of different proofs depending on the choice made for the additive connective. Note that we are following the convention of linear logic for $(-)^{\perp}$, where the $(-)^{\perp}$ operator is not changing the order of tensors.

Diagrams. A diagram is a (possibly infinite) directed graph with edges indexed with marks and built from *elementary nodes* and *boxes*. A diagram is not necessarily a connected graph. By graphical convention, all edges are flowing upward: a diagram is therefore acyclic.

Elementary nodes make the basic building blocks of diagrams: they are shown on Figure 2a. As we work with directed graphs, each edge connected to a node is either an input or an output for that node. There are several kinds of elementary nodes: the structural nodes for capturing the compact closed structure: $\cup$, $\cap$, $\otimes$, I and the swap-node (also written σ); the structural nodes for handling the product: $\boxplus$ for the diagonal map and π for the projection; the structural nodes for pointing inputs in and outputs out of diagrams; the nodes specifically for quantum: $|b\rangle$ and $\langle b|$, with b ranging over booleans, where the former stands for initialization and the latter for projection onto the corresponding basis, tr for representing tracing (also useful for products), G_1 for unary unitary gates and G_2 for binary gates. Note that the nodes allows more expressivity than what we need: for instance, tr and $\langle b|$ are indistinguishable. We nonetheless keep them in order to draw attention on the correspondence with quantum computation and an obvious mapping to completely positive maps. For the sake of legibility, we do not draw in and out nodes unless necessary.

Presented in Figure 2b, a box-node is built from a family of diagrams. They should all share the same input and output marks except for one pair of input/output (represented on the left of the box-node). As a node, box-node has the same input and output marks as its contained diagrams except that the left-most marks: these are the $\boxplus$ of all left-most marks of the family. We represent juxtaposition of edges as a double arrows. This node is the last piece needed for representing products.

Equivalence relation on diagrams. We define an equivalence relation on diagrams. The equivalence is given with local rules that can be extended to larger diagrams coherently: subgraphs can be rewritten inside a larger graph. These rules exactly capture what is needed for the categorical semantics to work. For instance, we include all of the rules for compact closed categories [16]. We also for instance need the fact that the π-node acts as a projection over box-nodes. The complete list can be found in the appendix.



(a) Morphisms in $\overline{M}$.

(b) Figure for Example 17.

Figure 3 Examples of Morphisms.

Category of Diagrams. Based on the definition of diagrams, we define the category of diagrams $\overline{M}$: object are lists of marks $[A_1, \dots, A_n]$, and a morphism $[A_1, \dots, A_n] \to [B_1, \dots, B_m]$ is a diagram with $\textcircled{\scriptsize in}$-nodes of marks A_i and $\textcircled{\scriptsize out}$-nodes of marks B_i, modulo the equivalence relation defined on diagrams. We use the notation $\vec{A}$ for the list of the A_i's. An identity morphism is a diagram consisting of a bunch of simple edges connecting $\textcircled{\scriptsize in}$ and $\textcircled{\scriptsize out}$ nodes. Composition consists in identifying $\textcircled{\scriptsize out}$ and $\textcircled{\scriptsize in}$ nodes of diagrams. The category $\overline{M}$ is symmetric monoidal: The unit is $I = []$, the empty list, and the monoidal structure is given with $\otimes : \overline{M} \times \overline{M} \to \overline{M}$ defined as $[A_1, \dots, A_n] \otimes [B_1, \dots, B_m] = [A_1, \dots, A_n, B_1, \dots, B_m]$, and $f \otimes g$ the juxtaposition of diagrams. As for standard graphical representation of symmetric monoidal structure [16], the associativity, unit laws and symmetry of the tensor product follow their graphical conventions. Finally, the operation on marks $(-)^\perp$ lifts to a contravariant functor, giving a compact-closed structure to $\overline{M}$. It then admits an internal hom: $\vec{A} \multimap \vec{B}$ can be defined as $[A_1, \dots, A_n] \multimap [B_1, \dots, B_m] = [A_1^\perp, \dots, A_n^\perp, B_1, \dots, B_m]$. Thanks to the $\textcircled{\scriptsize π}$-nodes and the corresponding diagram equivalence rules, the category $\overline{M}$ also has products: for any family of objects $\{\vec{A_x} \mid x \in X\}$ indexed by a set X, let $\times_{x \in X} \vec{A_x} = [\boxplus_{x \in X} \vec{A_x}^{\otimes}]$ be the product of the family of objects. Then, the family of projections $\pi_x : \times_{x \in X} A_x \to A_x$ is given by the π-node. Finally, for any family of maps $\{f_x : C \to A_x\}_{x \in X}$, the morphisms $\langle f_x \rangle : C \to \times_{x \in X} A_x$ is the diagram presented in Figure 2c. As an abuse of notation we use one $\textcircled{\scriptsize $\otimes$}$ for tensoring several wires at once.

▶ Remark 11. The category of diagrams is strongly inspired from proof nets: tensor nodes correspond to multiplicative connectives while boxes correspond to additive connectives.

Examples of morphisms in $\overline{M}$. Lastly, we show in Figure 3a two interesting morphisms in the category $\overline{M}$. The morphism $n : [q] \to [I \boxplus I]$ corresponds to the measure: in each branch we perform a projection, and we keep in the output the information of where we were. Note that the semantics does not state what is doing $\langle tt|$: what is important is to (1) "remove" the q-wire, and (2) keep as information if we are on the "true" or the "false" part. The morphism i corresponds to qubit creation: it takes a boolean $I \boxplus I$, initializes a qubit depending on its state and "forgets" the boolean. As a last example we can build the injections $I \to I \boxplus I$ in a similar way to n: first a $\textcircled{\scriptsize $\boxplus$}$-node, followed with a box-node where we trace out the component we do not need.

▶ Remark 12. The object $[I \boxplus I]$ corresponds to the bit-type in Quipper or in Proto-Quipper, corresponding to boolean values within the quantum co-processor, and manipulated with circuits in Quipper. For simplicity we did not include such a bit-type in the language, but it does exist in the model.

▶ **Definition 13** (Intepreting QCAlg terms)**.** Let us use the notation $q^{\otimes n}$ to represent a list $[q, ...q]$ of size n. A QCAlg-term Q can be interpreted as a $\overline{M}$-morphism $[\![Q]\!] : A \to B$, where $A = q^{\otimes \mathbf{in}(Q)}$ and B of tree-shape for instance $(q^{\otimes n_1} \boxplus q^{\otimes n_2}) \boxplus q^{\otimes n_3}$, following the tree-shape of out(Q). The $\overline{M}$-morphism $[\![Q]\!]$ is then defined by induction, using the idea presented above: initialization and unitary gates are simply composed, and the branches of a meas operation are encapsulated inside box-nodes.

3.2 Coproduct completion

Coproduct completion allows us to define families of circuits [14, 5]: the categorical structure clearly separate what is *purely quantum* and what is *parameter* to the computation: we have *parametric* families of quantum channels. Formally, this is done using the coproduct completion $\overline{\overline{M}}$ of $\overline{M}$. In this completion, an object corresponds to a pair $(X, (A_x)_{x\in X})$ where X is a set and A_x is an object of $\overline{M}$ for each $x \in X$: This should be understood as a *parametric* families of objects of $\overline{M}$. A morphism from $(X, (A_x))$ to $(Y, (B_y))$ corresponds to a pair $(f_0, (f_x)_{x\in X})$ where $f_0 : X \to Y$ is a set function and $f_x : A_x \to B_{f_0(x)}$ is a morphism in $\overline{M}$ for each $x \in X$. Intuitively, to each choice of parameter x we have a $\overline{M}$-morphisms $A_x \to A_{f_0(x)}$. Composition is defined with $(g_0, (g_y)) \circ (f_0, (f_x)) = (g_0 \circ f_0, (g_{f_0(x)} \circ f_x))$ where $(g_0, (g_y)) : (Y, (B_y)) \to (Z, (C_x))$ and $(f_0, (f_x)) : (X, (A_x)) \to (Y, (B_y))$ are morphisms in $\overline{\overline{M}}$, while the identity is $\mathbf{id}_A = (\mathbf{id}_X, (\mathbf{id}_{A_x}))$ for an object $A = (X, (A_x))$.

According to Rios&Selinger [14], the category $\overline{\overline{M}}$ is symmetric monoidal closed, and features products and co-products. In particular, the monoidal unit is $(\{\emptyset\}, (I))$ (where $\emptyset$ stands for the only representative of the singleton-set), and when $A = (X, (A_x))$ and $B = (Y, (B_y))$, the tensor on objects is $A \otimes B = (X \times Y, (A_x \otimes B_y)_{(x,y)})$ and the internal hom is $A \multimap B = (X \to Y, (C_f)_{f\in X\to Y})$ ($X \to Y$ is the set of all set-functions from X to Y and C_f refers to the product $\boxplus_{x\in X}(A_x \multimap B_{f(x)})$ of internal homs in $\overline{M}$). Note that the product is defined by $\boxplus$ in the case of the category of diagrams. Also note that compared to [14], we can capitalize on the concrete structure of the category for the proofs involving the coproduct completion. For instance, the associativity is trivial in our category $\overline{M}$.

Finally, in order to model the type operator "!", Rios&Selinger rely on Benton's linear/non-linear model [2], based on an adjunction between a symmetric monoidal closed category and a cartesian closed category. In our case, as in [14] the adjunction is built between the SMCC $\overline{\overline{M}}$ and the cartesian closed category $\mathbf{Set}$. The two functors of the adjunction are $p : \mathbf{Set} \to \overline{\overline{M}}$, defined on objects as $p(X) = (X, (I)_X)$, and $b : \overline{\overline{M}} \to \mathbf{Set}$, defined on objects as $b(X, (A_x)) = \sum_{x\in X} \overline{M}(I, A_x)$ where $\overline{M}(I, A_x)$ is the set of morphisms between the objects I and A_x of the category $\overline{M}$ and $\sum_{x\in X} \overline{M}(I, A_x)$ is the disjoint union of all such sets over X . From the adjunction, one can then construct a comonad "!" defined as $! = p \circ b$.

▶ Remark 14. In $\overline{\overline{M}}$ there are two classes of interesting objects. The *parameters* are objects of the form $(X, (I)_{x\in X})$: the family consists in trivial objects of $\overline{M}$, and the only information is given by…the parameter. The *state* object is the dual: the parameter is trivial and the family is of size 1 with only one object of $\overline{M}$. It is then of the form $(\{\emptyset\}, (A))$. One therefore has two booleans: a parameter boolean $b_p = (\{tt, ff\}, (I)_{\{tt,ff\}})$ and the state boolean $b_s = (\{\emptyset\}, (I \boxplus I))$ living in $\overline{M}$.

3.3 Monad for Branching Computation

According to Rios&Selinger, the category $\overline{\overline{M}}$ together with the structure sketched in Section 3.2 forms a model of Proto-Quipper-M. We shall now see how our concrete construction can also support dynamic lifting, therefore forming a model of Proto-Quipper-L.

The main problem consists in *lifting* a branching sitting inside a quantum channel – i.e. inside the category $\overline{M}$ – to turn it into a coproduct on which one can act upon in the classical world, represented by the category $\overline{\overline{M}}$: as in Remark 14, we need to lift a state-boolean into a parameter-boolean. Our strategy consists in defining a strong monad (F, μ, η, t) to capture the action of retrieving such a branching: a term featuring measurement (and dynamic lifting) is therefore represented within the Kleisli category $\overline{\overline{M}}_F$, following Moggi's [10] view on side-effects.

The functor $F : \overline{\overline{M}} \to \overline{\overline{M}}$ is defined as follows. For an object $A = (X, (A_x))$, we define $F(A) = (\mathrm{mset}(X), ([\boxplus_{x \in l} A_x^{\otimes}])_{l \in \mathrm{mset}(X)})$, where $\mathrm{mset}(X)$ is the set of multisets of X, while for a morphism $f = (f_0, (f_x)) : A \to B$ we set $F(f) = (g_0 : \mathrm{mset}(X) \to \mathrm{mset}(Y), g_l : [\boxplus_{x \in l} A_x^{\otimes}] \to [\boxplus_{y \in g_0(l)} B_y^{\otimes}])$, where $g_0 = \{[x_0, \ldots, x_n] \mapsto [f_0(x_0), \ldots, f_0(x_n)]\}$ and where g_l is defined as shown on the right.

▶ **Example 15.** The lifting of the state boolean b_s of Remark 14 to the parameter boolean b_p is then a $\overline{\overline{M}}$-map $\mathrm{lb} : b_s \to F(b_p)$, where $F(b_p)$ is $(\mathrm{mset}\{\mathit{tt}, \mathit{ff}\}, (\boxplus_{x \in l} I)_l)$. The map lb is defined as $(\mathrm{lb}_0, (f_x)_x)$ where $\mathrm{lb}_0 : \{\emptyset\} \to \mathrm{mset}\{\mathit{tt}, \mathit{ff}\}$ sends $\emptyset$ to $[\mathit{tt}, \mathit{ff}]$, and where $\mathrm{lb}_\emptyset : I \boxplus I \to I \boxplus I$ is simply defined as the identity. In the other direction, the $\overline{\overline{M}}$-map $b_p \to b_s$ consists of the constant set-function on $\emptyset$ together with the injections $I \to I \boxplus I$ discussed in Section 3.1.

▶ Remark 16. In Quipper dynamic lifting is implemented in the `Circ` monad which corresponds to the strong monad of F in our model. The branching side-effect corresponds to the `RW_Read` constructor of the `Circ` monad.

3.4 Interpreting Typed Terms and Configurations

In this section, we introduce an interpretation of Proto-Quipper-L within the Kleisli category $\overline{\overline{M}}_F$. As it is customary, types are mapped to objects while typing derivations are mapped to morphisms. When typed terms admit a unique typing derivation this entails a unique denotation for typed terms. In our situation, due to the promotion and dereliction rules typing derivations are not necessarily unique: we therefore adjust the statements of the lemmas and theorems accordingly. However, in the case of values of basic types, thanks to Remark 3 and the type safety properties, the denotation of closed terms of basic types is independent from the choice of typing derivation: this gives Corollary 19.

The interpretation $[\![A]\!]$ of a type A is directly built against the categorical structure: $[\![I]\!] = (\{\emptyset\}, (I))$, $[\![\mathrm{bool}]\!] = (\{\mathrm{tt}, \mathrm{ff}\}, (I, I))$, $[\![\mathbf{qubit}]\!] = (\{\emptyset\}, ([q]))$, $[\![A_a \multimap A_b]\!] = [\![A_a]\!] \multimap_{\overline{\overline{M}}_F} [\![A_b]\!]$ the internal hom in the category $\overline{\overline{M}}_F$, $[\![A_a \otimes B_b]\!] = [\![A_a]\!] \otimes [\![A_b]\!]$, $[\![!A]\!] = ![\![A]\!] = (p \circ b)[\![A]\!]$. Finally, for quantum channels we follow Rios&Selinger's strategy by defining $[\![\mathrm{QChan}(P, B)]\!] = p(\overline{\overline{M}}_F([\![P]\!], [\![A]\!]))$. In our situation, the set $\overline{\overline{M}}_F(A, B)$ is isomorphic to $\overline{M}(A, B)$ when A and B are state objects: in this situation, QChan-types indeed correspond to morphisms of the category $\overline{M}$, i.e. quantum channels: this is used to interpret the box and unbox operators. The quantum channel constant is just an encapsulation over Definition 13. Finally, a typed configuration $!\Delta \vdash (Q, m) : A$ is interpreted as the composition of Q (i.e. we first "compute" Q) followed with the interpretation of M.

▶ **Example 17.** The term exp of Eq.(1) in Section 1 has for interpretation a morphism $(\{\emptyset\}, (q)) \to (\mathrm{mset}\{(\emptyset, \emptyset)\}, (q)_l)$ defined as $(f_0, (f_\emptyset))$ where $f_0(\emptyset) = [(\emptyset, \emptyset), (\emptyset, \emptyset)]$ and $f_\emptyset$ is defined as shown in Figure 3b (the dashed lines are meant to be vertical). The bottom box-node represents the measurement ($I \boxplus I$ being the result) and the upper one the test. The top result is a $\boxplus$-superposition of 2 copies of $q \otimes I$, as expected: these stand for the two "classical" possibilities.

In general, soundness of categorical semantics states that the categorical interpretation of the typing derivation is preserved over the reduction. However, there can be multiple type derivations for each type judgement, in our type system, because of the reason explained above. Therefore, in this paper, we show that for a type judgement and a typing derivation, there exists a particular typing judgement of the reduced type judgement which has the same interpretation of the original typing derivation.

▶ **Theorem 18** (Soundness). *For any configurations (Q_1, m_1) and (Q_2, m_2) such that $(Q_1, m_1) \to (Q_2, m_2)$, if $\vdash (Q_1, m_1) : A$, then for any typing derivation π_1 of $\vdash (Q_1, m_1) : A$, there exists a typing derivation π_2 of $\vdash (Q_2, m_2) : A$ such that $[\![\pi_1]\!] = [\![\pi_2]\!]$.*

Finally, from the type safety properties, we can derive the following, making it possible to define the interpretation of a closed term of basic type.

▶ **Corollary 19.** *All the typing derivations of a closed term of basic type share the same interpretation.*

4 Conclusion

In this paper, we introduce the language Proto-Quipper-L which formalizes several features of Quipper (dynamic lifting, higher-order function, circuit composition, and branching) while treating the qubits linearly using the type system. On one hand we propose a type system and an operational semantics which explains the meaning of programs as a set of reduction rules. On the other hand, we propose a concrete categorical model of the language which is proven to be sound, meaning that the semantics is preserved over the operational semantics.

On one side, the model is closely related to models of intuitionistic linear logic. Diagrams are akin to proof nets: tensor nodes correspond to multiplicative connectives while boxes correspond to additive connectives. On the other side, they can be considered as an extension of diagrammatic languages for quantum processes [19].

Our concrete semantics makes it possible to describe a monad, following closely Quipper's operational semantics encoded in Haskell's type system. With this semantics we are able to answer an open question in the community: finding a categorical representation of dynamic lifting for a circuit-description language.

References

1 Linda Anticoli, Carla Piazza, Leonardo Taglialegne, and Paolo Zuliani. Towards quantum programs verification: from Quipper circuits to qpmc. In *International Conference on Reversible Computation*, pages 213–219. Springer, 2016.

2 P 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.

3 Benjamin Bichsel, Maximilian Baader, Timon Gehr, and Martin T. Vechev. Silq: a high-level quantum language with safe uncomputation and intuitive semantics. In Alastair F. Donaldson and Emina Torlak, editors, *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*, pages 286–300. ACM, 2020. doi:10.1145/3385412.3386007.

4 Qingxiuxiong Dong, Marco Túlio Quintino, Akihito Soeda, and Mio Murao. Success-or-draw: A strategy allowing repeat-until-success in quantum computation. *Phys. Rev. Lett.*, 126:150504, April 2021. doi:10.1103/PhysRevLett.126.150504.

5 Peng Fu, Kohei Kishida, and Peter Selinger. Linear dependent type theory for quantum programming languages. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 440–453, 2020.

6 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987.

7 Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. Quipper: A scalable quantum programming language. In Hans-Juergen Boehm and Cormac Flanagan, editors, *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI'13*, pages 333–342. ACM, 2013. `doi:10.1145/2491956.2462177`.

8 Emmanuel Knill. Conventions for quantum pseudocode. Technical report, Los Alamos National Lab., NM (United States), 1996.

9 Bert Lindenhovius, Michael Mislove, and Vladimir Zamdzhiev. Enriching a linear/non-linear lambda calculus: A programming language for string diagrams. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 659–668. ACM, 2018.

10 Eugenio Moggi. Notions of computation and monads. *Information and computation*, 93(1):55–92, 1991.

11 Jennifer Paykin, Robert Rand, and Steve Zdancewic. QWIRE: a core language for quantum circuits. In Giuseppe Castagna and Andrew D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL'17*, pages 846–858. ACM, 2017. `doi:10.1145/3009837.3009894`.

12 Robert Rand, Jennifer Paykin, and Steve Zdancewic. QWIRE practice: Formal verification of quantum circuits in coq. In Bob Coecke and Aleks Kissinger, editors, *Proceedings 14th International Conference on Quantum Physics and Logic, QPL 2017*, volume 266 of *Electronic Proceedings in Theoretical Computer Science*, pages 119–132, 2017. `doi:10.4204/EPTCS.266.8`.

13 Mathys Rennela and Sam Staton. Classical control, quantum circuits and linear logic in enriched category theory. *Log. Methods Comput. Sci.*, 16(1), 2020. `doi:10.23638/LMCS-16(1:30)2020`.

14 Francisco Rios and Peter Selinger. A categorical model for a quantum circuit description language. In Bob Coecke and Aleks Kissinger, editors, *Proceedings 14th International Conference on Quantum Physics and Logic, QPL 2017*, volume 266 of *Electronic Proceedings in Theoretical Computer Science*, pages 164–178, 2018. `doi:10.4204/EPTCS.266.11`.

15 Neil J. Ross. *Algebraic and logical methods in quantum computation*. PhD thesis, Dalhousie University, 2015.

16 Peter Selinger. A survey of graphical languages for monoidal categories. In *New structures for physics*, pages 289–355. Springer, 2010.

17 Peter Selinger and Benoît Valiron. A lambda calculus for quantum computation with classical control. *Mathematical Structures in Computer Science*, 16(3):527–552, 2006.

18 Robert S. Smith, Michael J. Curtis, and William J. Zeng. A practical quantum instruction set architecture. *arXiv preprint*, 2016. `arXiv:1608.03355`.

19 Sam Staton. Algebraic effects, linearity, and quantum programming languages. *SIGPLAN Not.*, 50(1):395–406, January 2015. `doi:10.1145/2775051.2676999`.

20 Krysta M. Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. Q#: Enabling scalable quantum computing and development with a high-level domain-specific language. *arXiv preprint*, 2018. `arXiv:1803.00652`.

21 Benoît Valiron. *Semantics for a higher order functional programming language for quantum computation*. PhD thesis, University of Ottawa, 2008.

22 Dave Wecker and Krysta M. Svore. LIQUi$|\rangle$: A software design architecture and domain-specific language for quantum computing. *arXiv preprint*, 2014. `arXiv:1402.4467`.

23 William K. Wootters and Wojciech H. Zurek. A single quantum cannot be cloned. *Nature*, 299:802–803, October 1982. `doi:10.1038/299802a0`.

24 Bernhard Ömer. *Structured quantum programming*. PhD thesis, Technical University of Vienna, 2003.

A Operational semantics

A.1 Reduction

The reduction rules for Proto-Quipper-L are defined as follows.

Reduction rules for classical computation. The following rules always hold (b is tt or ff)

$$\begin{aligned}(\epsilon(W),(\lambda x.M)V) &\to (\epsilon(W),M[V/x])\\ (\epsilon(W),\textbf{let }\langle x,y\rangle=\langle V,U\rangle\textbf{ in }M) &\to (\epsilon(W),M[V/x,U/y])\\ (\epsilon(W),\textbf{if }\mathrm{b}\textbf{ then }M_{\mathrm{tt}}\textbf{ else }M_{\mathrm{ff}}) &\to (\epsilon(W),M_b)\end{aligned}$$

Reduction rules for circuit operations. Provided that **new** is an operator that creates free variables during the computation, meaning that these free variables do not appear in both classical and quantum contexts and that the term **new**(P) is a pattern of same shape as P made out of these new variables, we have

$$\frac{p=\mathbf{new}(P)\qquad W_p=\mathbf{supp}(p)}{(\epsilon(\emptyset),\mathrm{box}_P\,V)\to(\epsilon(\emptyset),(p,\epsilon(W_p),Vp))}\qquad\frac{\mathbf{shape}(p)=\mathbf{shape}(V)\qquad\sigma=\mathbf{bind}(p,V)}{(\epsilon(\mathbf{FV}(V)),(\mathrm{unbox}(p,\ Q,\ u))V)\to(\sigma(Q),\sigma(u))}$$

Structural reduction rule for quantum channel constant. Provided that $(Q,m)\to(Q',m')$, we have $(\epsilon(\emptyset),(p,Q,m))\to(\epsilon(\emptyset),(p,Q',m'))$.

Structural reduction rules for empty quantum channel. Provided that $(\epsilon(W_M),\ M)\to(Q,\ m)$, that $\mathbf{all}(Q)\cap W_N=\emptyset$ and that $\mathbf{all}(Q)\cap W_V=\emptyset$, we have

$$\begin{aligned}(\epsilon(W_M\cup W_N),\ MN) &\to (\mathbf{extend}(Q,W_N),\ mN)\\ (\epsilon(W\cup W_V),\ VM) &\to (\mathbf{extend}(Q,W_V),\ Vm)\\ (\epsilon(W_M\cup W_N),\ \langle M,N\rangle) &\to (\mathbf{extend}(Q,W_N),\ \langle m,N\rangle)\\ (\epsilon(W_M\cup W_V),\ \langle V,M\rangle) &\to (\mathbf{extend}(Q,W_V),\ \langle V,m\rangle)\\ (\epsilon(W_M\cup W_N),\ \textbf{if }M\textbf{ then }M_a\textbf{ else }M_b)) &\to (\mathbf{extend}(Q,W_N),\ \textbf{if }m\textbf{ then }M_a\textbf{ else }M_b)\\ (\epsilon(W_M),\ \textbf{let }\langle x,y\rangle=M\textbf{ in }N) &\to (\mathbf{extend}(Q,W_N),\ \textbf{let }\langle x,y\rangle=m\textbf{ in }N)\end{aligned}$$

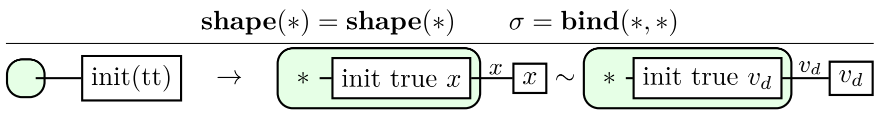
We use syntactic sugar combining terms and branching terms, as in mM. It corresponds to the term constructor applied to every leafs of m, for instance: for $m=[[N_1,N_2],N_3]$, $[[N_1,N_2],N_3]M:=[[N_1M,N_2M],N_3M]$.

Structural reduction rules for non-empty quantum channel. Assume that $(Q_1,\ m_a)\to(Q_3,\ m_c)$ and $(Q_2,\ m_b)\to(Q_4,\ m_d)$. Then

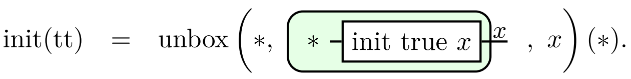
$$\begin{aligned}((\mathrm{meas}\ w\ Q_1\ Q_2),\ [m_a,m_b]) &\to ((\mathrm{meas}\ w\ Q_3\ Q_4),\ [m_c,m_d])\\ ((\mathrm{meas}\ w\ Q_1\ Q_2),\ [m_a,v]) &\to ((\mathrm{meas}\ w\ Q_3\ Q_2),\ [m_c,v])\\ ((\mathrm{meas}\ w\ Q_1\ Q_2),\ [v,m_b]) &\to ((\mathrm{meas}\ w\ Q_1\ Q_4),\ [v,m_d])\\ (U(W)\ Q_1,\ m_a) &\to (U(W)Q_3,\ m_c)\\ (\mathrm{init}\ b\ w\ Q_1,\ m_a) &\to (\mathrm{init}\ b\ w\ Q_3,\ m_c)\\ (\mathrm{free}\ w\ Q_1,\ m_a) &\to (\mathrm{free}\ w\ Q_3,\ m_c)\end{aligned}$$

A.2 Derivation of the example of Example 6

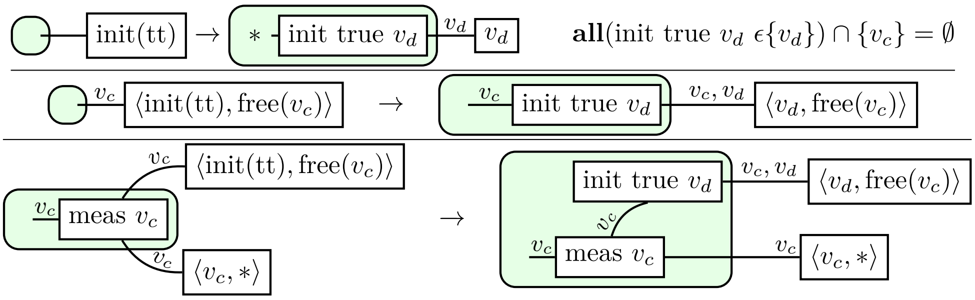
Let us explain how the tree expands as the computation progresses for example 6. First, we show that $((\epsilon\{\}, \mathrm{init}(\mathrm{tt})) \to ((\mathrm{init\ true}\ x\ \epsilon\{x\}, x)$ as follows.



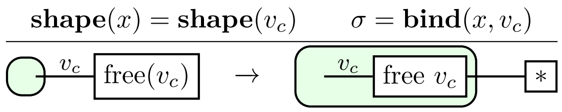
where we let



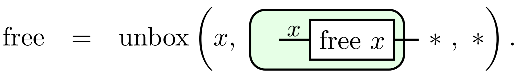
Then we can show the following reduction:



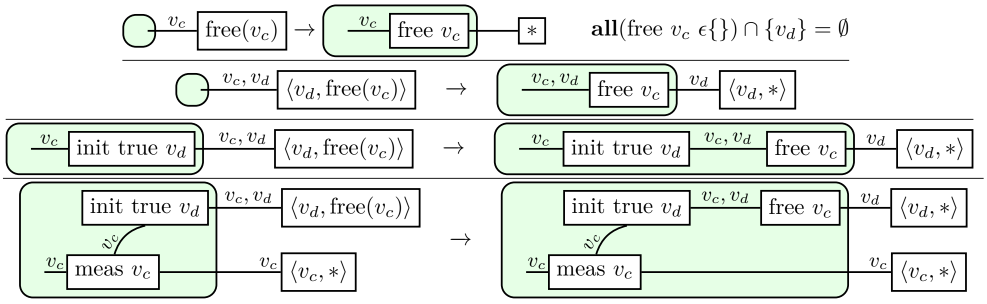
Next, we show the last reduction step of the example.



Recall that



Then we can show the following reduction:



B Categorical semantics

B.1 Equivalence of diagrams

Complete list of the equivalence rules that are used to construct the categorical model is shown in Figure 4.

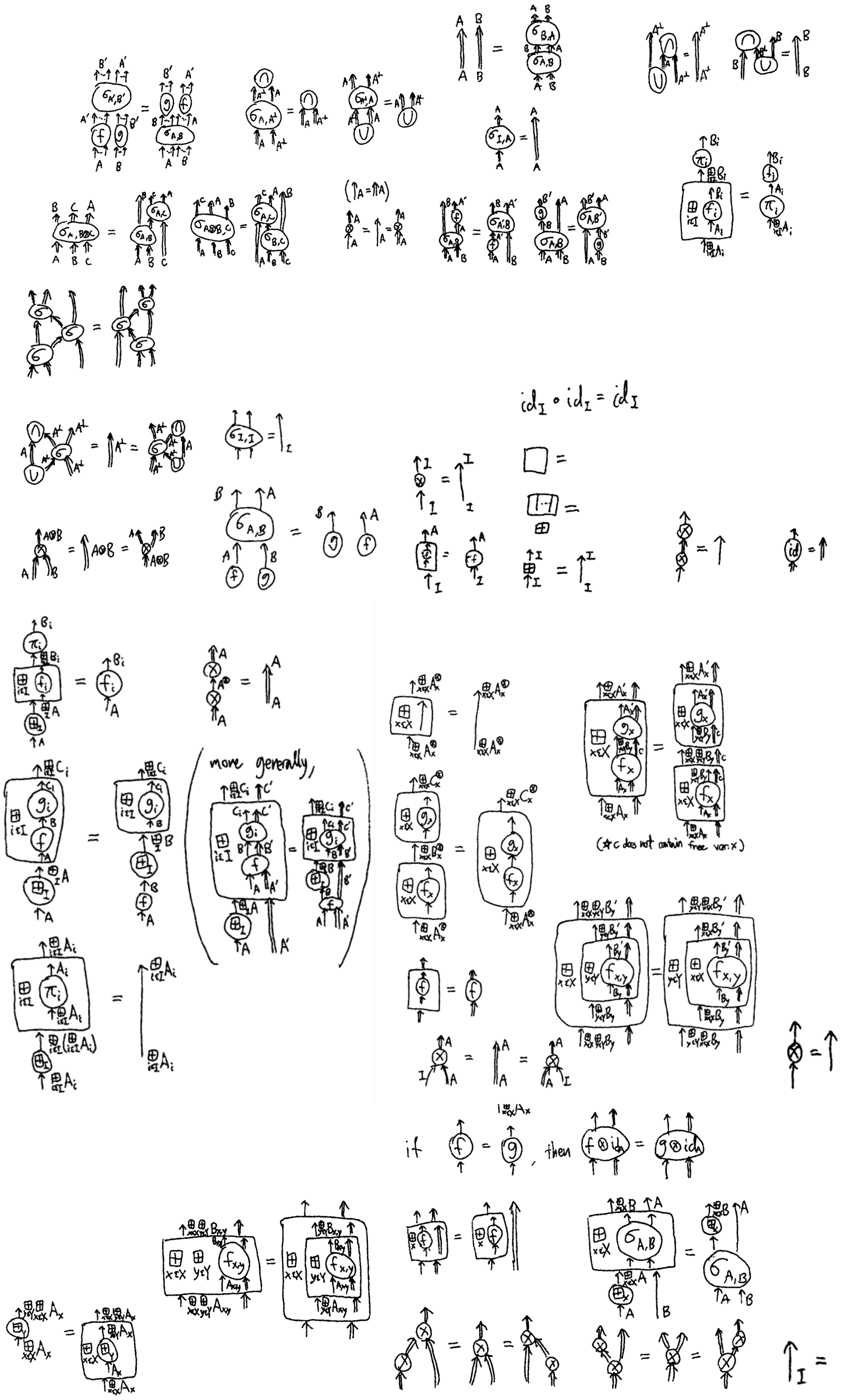
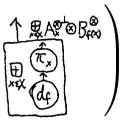
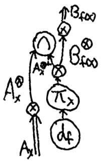
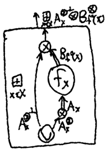


Figure 4 Equivalence relation of diagrams.

Table 6 Definition of isomorphism between $b(A \multimap_{\overline{\overline{M}}} B)$ and $\overline{\overline{M}}(A, B)$.

$\mathrm{iso}_{\rightarrow} : b(A \multimap_{\overline{\overline{M}}} B) \rightarrow \overline{\overline{M}}(A, B)$:	$\mathrm{iso}_{\leftarrow} : \overline{\overline{M}}(A, B) \rightarrow b(A \multimap_{\overline{\overline{M}}} B)$:
Given (f, d_f), which is $\Big(f,$  $\Big)$, let $f_0 = f$ and $f_x =$ .	Given $(f_0, (f_x)_X)$, let $f = f_0$ and $d_f =$ 

B.2 Interpretation of type system

For a typing context $\Gamma = x_1 : A_1, \ldots, x_k : A_k$, assuming the variables are ordered by some linear order, $[\![\Gamma]\!] = [\![A_1]\!] \otimes \ldots \otimes [\![A_k]\!]$. Next, the branching typing context is interpreted as the coproduct of the objects assigned to the smaller branching typing contexts, namely: $[\![\gamma_1, \gamma_2]\!] = [\![\gamma_1]\!] + [\![\gamma_2]\!]$. Lastly, we interpret the typing derivation as a morphism in $\overline{\overline{M}}_F$.

B.2.1 Quantum channel types, Box and Unbox

As in [14], we interpret the quantum channel types $\mathbf{QChan}(A, B)$ as an object $p(\overline{\overline{M}}_F(A, B)) = (\overline{\overline{M}}_F(A, B), (I))$ in $\overline{\overline{M}}_F$ and $\overline{\overline{M}}$. Note that the object is a parameter object as in [14], which means that the object has the form of $(X, (I)_X)$ for some X. When we define the quantum channel types $\mathbf{QChan}(A, B)$ as a parameter object, box and unbox can be interpreted based on an isomorphism between the set $b(A \multimap_{\overline{\overline{M}}} B)$ and $\overline{\overline{M}}(A, B)$. In specific, we can define an isomorphism as in Table 6.

Given the isomorphism, we can define the morphisms for box and unbox as morphisms in $\overline{\overline{M}}$ as follows:

$$\begin{aligned}
\mathrm{unbox} &= p(\overline{\overline{M}}(A, F(B)) \xrightarrow{p(\mathrm{iso}_{\rightarrow})} (p \circ b)(A \multimap_{\overline{\overline{M}}} F(B)) \xrightarrow{\epsilon(A \multimap_{\overline{\overline{M}}} F(B))} (A \multimap_{\overline{\overline{M}}} F(B)) \\
\mathrm{box} &= (p \circ b)(A \multimap_{\overline{\overline{M}}} F(B)) \xrightarrow{p(\mathrm{iso}_{\leftarrow})} p(\overline{\overline{M}}(A, F(B))) \\
&\qquad \xrightarrow{p(\eta(\overline{\overline{M}}(A, F(B))))} (p \circ b \circ p)(\overline{\overline{M}}(A, F(B))).
\end{aligned}$$

where ϵ refers to the counit from the comonad !.

B.2.2 Quantum channel constants

We define a natural transformations called bif and merge for the measurement as in Table 7.

A quantum channel Q is interpreted as a morphism $[\![\mathbf{in}(Q)]\!] \rightarrow_{\overline{\overline{M}}_F} [\![\mathbf{out}(Q)]\!]$, where

$$[\![\mathbf{in}(Q)]\!] = (\{\emptyset\}, ([q]^{\otimes |\mathbf{in}(Q)|}))$$

$$[\![\mathbf{out}(Q)]\!] = \begin{cases} (\{\emptyset\}, ([q]^{\otimes |V|})) & \text{if } \mathbf{out}(Q) \text{ is a set } V \\ [\![o_1]\!] + [\![o_2]\!] & \text{if } \mathbf{out}(Q) = [o_1, o_2] \end{cases}$$

Table 7 Definition of bif and merge.

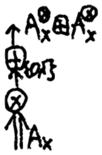
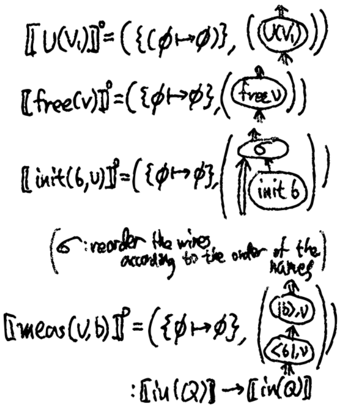
$\mathrm{bif}(A) : A \to_{\overline{\overline{M}}_F} A + A$	$\mathrm{merge}(A,B) : F(A) + F(B) \to_{\overline{\overline{M}}_F} A + B$
For an object $A = (X, (A_x))$, we let $\mathrm{bif}(X,(A_x)) = \begin{pmatrix} \{x \mapsto [(0,x),(1,x)]\}, \\ (f_x : A_x \to A_x^{\otimes} \boxplus A_x^{\otimes}) \end{pmatrix}$ where $f_x =$ .	For objects A, B, we let $\mathrm{merge}(A,B) = (\{$ $(0,[x_1,\ldots,x_k]) \mapsto [(0,x_1),\ldots,(0,x_k)],$ $(1,[y_1,\ldots,y_n]) \mapsto [(1,y_1),\ldots,(1,y_n)]\},$ $(\mathbf{id}_{[\boxplus_{x\in l} A_x^{\otimes}]})_{l:\mathbf{mset}(X)}$ $++(\mathbf{id}_{[\boxplus_{y\in l} B_y^{\otimes}]})_{l:\mathbf{mset}(Y)})$
It satisfies the following commute diagram for naturality: $$\begin{tikzcd} A \arrow[r, "\mathrm{bif}(A)"] \arrow[d, "f"] & F(A+A) \arrow[d, "F(f+f)"] \\ B \arrow[r, "\mathrm{bif}(B)"] & F(B+B) \end{tikzcd}$$	It satisfies the following commute diagram for naturality: $$\begin{tikzcd} F(A)+F(B) \arrow[r, "\mathrm{merge}(A,B)"] \arrow[d, "F(f)+F(g)"] & F(A+B) \arrow[d, "F(f+g)"] \\ F(A')+F(B') \arrow[r, "\mathrm{merge}(A',B')"] & F(A'+B') \end{tikzcd}$$

Table 8 Interpretation of quantum channel constants.

$[\![\epsilon(V)]\!] = \eta(\{\emptyset\},([q]^{\otimes\lvert V\rvert}))$ $[\![U(V_1)\ Q]\!] = [\![Q]\!] \circ [\![U(V_1)]\!]^0$ $[\![\mathrm{free}\ v\ Q]\!] = [\![Q]\!] \circ [\![\mathrm{free}(V)]\!]^0$ $[\![\mathrm{init}\ b\ v\ Q]\!] = [\![Q]\!] \circ [\![\mathrm{init}(b,v)]\!]^0$ $[\![\mathrm{meas}\ v\ Q_1\ Q_2]\!] =$ $$\begin{tikzcd} {[\![\mathbf{in}(\mathrm{meas}\ v\ Q_1\ Q_2)]\!]} \arrow[d, "\mathrm{bif};F\begin{pmatrix} [\![Q_1]\!]\circ[\![\mathrm{meas}(v,0)]\!]^0 \\ +[\![Q_2]\!]\circ[\![\mathrm{meas}(v,1)]\!]^0 \end{pmatrix}"] \\ F(F[\![\mathbf{out}(Q_1)]\!] + F[\![\mathbf{out}(Q_2)]\!]) \arrow[d, "F(\mathrm{merge});\mu"] \\ F([\![\mathbf{out}(Q_1)]\!] + [\![\mathbf{out}(Q_2)]\!]) \end{tikzcd}$$	where  ⟦U(V₁)⟧⁰=({(ϕ↦ϕ)}, (U(V₁))) ⟦free(v)⟧⁰=({ϕ↦ϕ}, (free v)) ⟦init(b,v)⟧⁰=({ϕ↦ϕ}, (σ init b)) (σ: reorder the wires according to the order of the names) ⟦meas(v,b)⟧⁰=({ϕ↦ϕ}, (|b⟩,v ⟨b|,v)) : ⟦in(Q)⟧ → ⟦in(Q)⟧

The interpretation of quantum channel is defined inductively as in Table 8 where μ represents the multiplication of the monad F. In addition, the elementary nodes in Table 8–($U(V_1)$), (free v), (| $b\rangle$, v) and ($\langle b$ |, v)–refers to the unitary gate node Ⓤ (which is either 1 or 2-qubits gate) applied to wires V_1, (tr) node applied to wire v, and (⟨b|) and (|b⟩) nodes applied to wire v, respectively.