Proto-Quipper with Reversing and Control

Peng Fu[a], Kohei Kishida[b], Neil J. Ross[c], Peter Selinger[c]

[a] *University of South Carolina, Columbia, SC, USA*
[b] *University of Illinois at Urbana-Champaign, Urbana, IL, USA*
[c] *Dalhousie University, Halifax, NS, Canada*

Abstract

The quantum programming language Quipper supports circuit operations such as reversing and controlling certain quantum circuits. Additionally, Quipper provides a function called *with-computed*, which can be used to program circuits of the form $g; f; g^\dagger$. The latter is a common pattern in quantum circuit design. One benefit of using with-computed, as opposed to constructing the circuit $g; f; g^\dagger$ directly from g, f, and $g^\dagger$, is that it facilitates an important optimization. Namely, if the resulting circuit is later controlled, only f needs to be controlled; the circuits g and $g^\dagger$ need not even be controllable.

In this paper, we formalize a semantics for reversible and controllable circuits, using a dagger symmetric monoidal category $\mathbf{R}$ to interpret reversible circuits, and a new notion we call a *controllable category* $\mathbf{N}$, which encompasses the control and with-computed operations in Quipper. We extend the language Proto-Quipper with reversing, control and the with-computed operation. Since not all circuits are reversible and/or controllable, we use a type system with modalities to track reversibility and controllability. This generalizes the modality of Fu-Kishida-Ross-Selinger 2023. We give an abstract categorical semantics, and show that the type system and operational semantics are sound with respect to this semantics. Lastly, we give a construction of a categorical model using an idea similar to Fu-Kishida-Ross-Selinger 2022.

Keywords: Quantum programming languages, reverse, control, Proto-Quipper, type system, operational semantics, categorical semantics

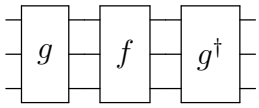
1. Introduction

The goal of this paper is to devise a strongly typed, functional programming language that can formally treat the quantum operations of reversing and control. Many quantum algorithms take essential advantage of reversing, control, and their associated operation "with-computed". At the same time, some operations, such as state preparations and measurements, can make circuits irreversible or uncontrollable. It is therefore desirable to equip quantum programming languages with a type system that can prevent the user from trying to reverse the irreversible or to control the uncontrollable. In this paper, we incorporate reversing, control, and with-computed into the language *Proto-Quipper*.

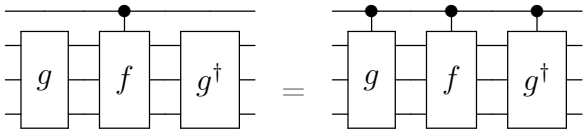
Proto-Quipper is a family of programming languages that aims to provide a formal syntax and semantics for various features of the embedded quantum programming language Quipper [16, 17]. Like Quipper, Proto-Quipper is a quantum circuit description language. When a Proto-Quipper program is run, it outputs a quantum circuit; this generated circuit can later be executed on a quantum computer. We refer to these two runtimes as "circuit generation time" and "circuit execution time", respectively. There are various errors that programmers can introduce by attempting the impossible—for example, to duplicate linear data, to apply circuit operations to a program that is not a circuit, etc.—but since Proto-Quipper is strongly typed, it can catch such errors at compile time, rather than at circuit generation time or circuit execution time. Previous research on Proto-Quipper includes providing a syntax and semantics for circuit boxing [26, 27], supporting recursion [23], combining linear types and dependent types [10, 14], and incorporating dynamic lifting [4, 11, 12, 22]. This paper defines a new variant Proto-Quipper-C, which extends the type system of Proto-Quipper with reversibility and controllability.

1.1. Reversing, control, and with-computed

In Quipper, the `controlled` function inputs a circuit and returns the controlled version of the circuit. The `reverse` function inputs a circuit and returns its adjoint. The program `(with_computed` g f`)` first performs a circuit g, then a circuit f, and finally the adjoint of g.



The *with-computed* circuit pattern is very common in quantum computing. For example, Bennett's method for constructing reversible classical circuits [1] uses this pattern to initialize ancillary qubits and then uncompute them; and the quantum Fourier transform (QFT) addition circuit [8] conjugates a series of controlled rotation gates by the QFT. This pattern is used so often that Quipper implements the following optimization: when controlling a quantum circuit $g; f; g^\dagger$ generated by the with-computed function, it suffices to control the circuit f.



In fact, the circuits g and $g^\dagger$ need not even be controllable. Thus the left-hand side of the above circuit identity is more general than the right-hand side. It is also more efficient, since it uses a smaller number of controlled gates. Note that in quantum computing, a controlled gate may require substantially more resources than its uncontrolled version. For example, a common way of measuring resources is the *T-count*, i.e., the number of T-gates required to implement a gate. While a controlled-not gate has T-count 0, a controlled-controlled-not (Toffoli) gate has T-count at least 4 [20].

1.2. Modalities for reversing and control

In Quipper, controlling an uncontrollable circuit or reversing a non-reversible circuit will give rise to runtime errors. We want to incorporate reversing and control in Proto-Quipper in such a way that erroneously controlling or reversing a circuit is detected as a typing error at compile time. We achieve this by introducing a notion of modality $\alpha \in \{0, 1, 2\}$. Here, the modality 2 is associated with circuits that are both controllable and reversible, for example a Hadamard gate. The modality 1 is associated with circuits that are reversible but not controllable, for example an initialization gate. (Recall that "reversing" means taking the adjoint of a circuit, not necessarily the inverse. For a more detailed discussion of this circuit model, see [17, Sec. 4.2].) The modality 0 is associated with circuits that are neither controllable nor reversible, for example a measurement gate. Note that because all controllable quantum gates are unitary and therefore reversible, we do not include a modality for circuits that are controllable but not reversible.

We can use modalities to annotate the type of a circuit. For example, the values of the type $\mathbf{Circ}_\alpha(S, U)$ are circuits with input type S, output type U and a modality annotation α. One important feature of modalities is that they are composable. Suppose we want to compose a circuit $\mathbf{Circ}_\alpha(S, U)$ with a circuit $\mathbf{Circ}_\beta(U, S')$. The resulting circuit will have type $\mathbf{Circ}_{\alpha\wedge\beta}(S, S')$, where $\alpha \wedge \beta = \min(\alpha, \beta)$. This means that as long as we know the modalities for the basic gate sets, we can devise a type system to track the modalities of the circuits constructed from basic gates. Moreover, the reversing, control, and with-computed operations can be given the following types:

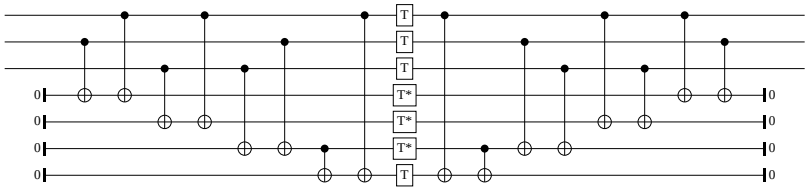
$$\begin{aligned}
\mathsf{control}_S &: \mathbf{Circ}_2(U, U) \to \mathbf{Circ}_2(S \otimes U, S \otimes U). \\
\mathsf{reverse} &: \mathbf{Circ}_\alpha(U, S) \to \mathbf{Circ}_\alpha(S, U), && \text{where } \alpha > 0. \\
\mathsf{withComputed} &: \mathbf{Circ}_\alpha(U, S) \times \mathbf{Circ}_2(S, S) \to \mathbf{Circ}_2(U, U), && \text{where } \alpha > 0.
\end{aligned}$$

Note that withComputed requires its first argument to be at least reversible, and the resulting circuit is again controllable.

1.3. Programming with reversing and control

Our modal type system makes it possible to detect errors, like controlling an uncontrollable circuit, at compile time. In a practical implementation (such as the one available from [9]), it is moreover possible for the type checker to automatically infer modalities, so that the programmer does not need to explicitly specify them in the source code.

Here, we give a basic example of using control and with-computed in the prototype implementation of Proto-Quipper from [9]. It is well-known that a CCZ gate can be implemented by 7 T-gates with T-depth one [30]. See the following circuit. Note that $0 \vdash$ is our notation for initializing a qubit in the $|0\rangle$ state, and $\dashv 0$ is our notation for terminating a qubit when it is known to be in the $|0\rangle$ state. See Section 4.2 of [17] for more details.



We can define the above circuit in Proto-Quipper, using its `withComputed` operator.

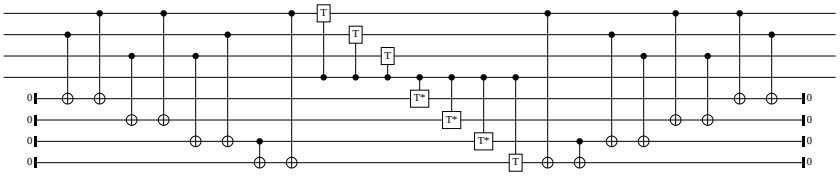
```
my_ccz : Circ(Qubit * Qubit * Qubit, Qubit * Qubit * Qubit)
my_ccz = withComputed box_cnot_circuit box_parallel_T
```

The function `box_parallel_T` generates the 7 parallel T gates in the middle of the circuit, and the function `box_cnot_circuit` generates the initialization gates and the cnot circuits before the parallel T gates. Their definitions are available in `examples/CCZ.dpq` in [9].

We can use Proto-Quipper's `control` operator to add an extra control to the CCZ circuit.

```
ctrl_ccz : Circ(Qubit * (Qubit * Qubit * Qubit),
                Qubit * (Qubit * Qubit * Qubit))
ctrl_ccz = control my_ccz
```

This generates the following circuit. We can see that only the T-gates in the middle are controlled.



If instead, we program the CCZ circuit without using the `withComputed` operation, we will get an error when trying to control it. In the program below, `my_ccz'` is defined by manual composition, where `cnot_circuit_rev` is the reverse version of `cnot_circuit`.

```
cnot_circuit_rev :
  !(Qubit * Qubit * Qubit * Qubit * Qubit * Qubit * Qubit
    -> Qubit * Qubit * Qubit)
cnot_circuit_rev = unbox (reverse (boxCirc cnot_circuit))

my_ccz' : Circ(Qubit * Qubit * Qubit, Qubit * Qubit * Qubit)
my_ccz' =
  boxCirc $ \input -> cnot_circuit_rev (parallel_T (cnot_circuit input))

-- The following gives rise to a typing error.
ctrl_my_ccz' : Circ(Qubit * (Qubit * Qubit * Qubit),
                    Qubit * (Qubit * Qubit * Qubit))
ctrl_my_ccz' = control my_ccz'
```

Controlling the circuit `my_ccz'` gives rise to a typing error, because according to the semantics of `control`, we would have to control all the gates in the circuit `my_ccz'`, which includes the non-controllable initialization gates. So in order to control the CCZ circuit containing initialization gates, we must construct the circuit using the `withComputed` operation.

1.4. Related work

The reverse, control and with-computed operations originally appeared in Quipper [17], but were lacking a formal semantics. In our work on Proto-Quipper [11, 12], we gave type systems with modalities for a feature called dynamic lifting, and provided a construction of a categorical model based on *biset*-enrichment. In this paper, we extend the type system to handle multiple modalities such as reversing and control, and we generalize the method of *biset*-enrichment to *triset*-enrichment to construct a categorical model for our type system.

Many practical quantum programming languages, such as Qiskit and Cirq, also include the concepts of reversibility and controllability. However, these languages are imperative, and errors of reversibility or controllability are treated as runtime exceptions [15, 19]. The functional quantum programming language QWire [25] supports circuit reversal, but not the control and with-computed operations. The language Silq [2] has type annotations **qfree** and **mfree**, which denote classical functions and functions that do not use measurement, respectively. Silq's core language supports reversing, and the control operation is supported by an if-then-else construct. The main difference between Silq and Proto-Quipper is that Silq is not a circuit description language and does not have a notion of boxed circuits.

We are only aware of one language besides Quipper that has a with-computed operation, namely ProjectQ [18, 32]. There are also libraries for automatic synthesis of uncomputation circuits [24], and optimizing compute-uncompute circuits [31]. However, none of these languages or libraries are strongly typed.

1.5. Contributions

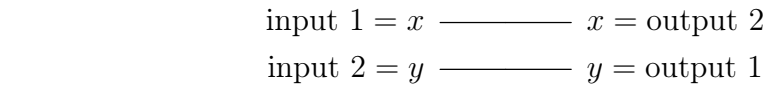
In this paper, we formalize a notion of controllable category. We extend the Proto-Quipper type system of [12] with modalities for reversing and control and define an operational semantics. We provide safety properties that guarantee that a well-typed program is free of runtime errors. We also give an axiomatization of an abstract categorical model of Proto-Quipper

with reversing, control, and the with-computed operation, and show that the operational semantics is sound for it. Lastly, we give a construction of a concrete categorical model that satisfies all the axioms of our abstract categorical model.

This paper is an extended version of the conference paper [13], with more details and proofs. Compared to this earlier work, we added the construction of the concrete categorical model.

2. Controlling a permutation circuit

In Proto-Quipper, there are two ways to represent the swap operation. One way is by permuting the logical order of the circuit outputs without using any gates, as in the following diagram.

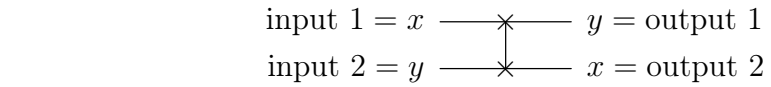


(1)

The above circuit is generated by the following Proto-Quipper program.

```
f : !(Qubit * Qubit -> Qubit * Qubit)
f input = let (x, y) = input in (y, x)
```

The other way is by using an explicit swap gate, as in the following diagram.



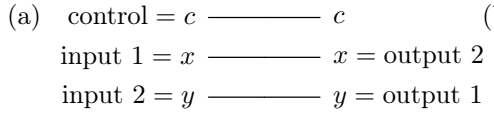
(2)

The corresponding Proto-Quipper program is the following.

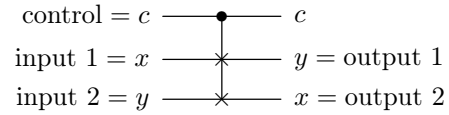
```
g : !(Qubit * Qubit -> Qubit * Qubit)
g input = let (x, y) = input in Swap x y
```

These circuits are semantically equivalent: each of them sends (x, y) to (y, x). However, care must be taken when controlling these circuits, since controlling them naively may produce the following non-equivalent circuits.

(a)



(b)



(3)

While the second circuit is correct, the first one is not. Indeed, the first circuit will send input 1 to output 2 independently of the state of the control

qubit. This is not the correct behavior, since the swapping should only take place when the control qubit is in the state $|1\rangle$. When the control qubit is in the state $|0\rangle$, the functions f and g should both behave like the identity function on `Qubit * Qubit`.

Therefore, the programming language must be aware not only of the circuit, but also of the implicit permutation of any qubits performed by the language. If such an operation is controlled, explicit swap gates must sometimes be inserted. Proto-Quipper-C does this correctly, whereas the original Quipper implementation did not. In Proto-Quipper-C, controlling circuit (1) and circuit (2) both result in (3)(b), not (3)(a).

3. A formal model of reversing, control and with-computed

Before we can define a programming language for building quantum circuits, we must specify what a quantum circuit is. However, there exist many different classes of circuits; for example, they differ by what data they can manipulate (only qubits, or also classical bits, and/or more exotic objects like qutrits), what the built-in gates are (for example, the Clifford+T gate set, Toffoli+Hadamard, rotations by arbitrary angles), whether or not qubit initialization and termination is supported, whether measurement is considered as a gate, and so on. Therefore, rather than tailoring our programming language to a specific class of circuits, we make both the language and its operational and denotational semantics *parametric* on a given class of circuits. This is analogous to making a classical programming language parametric on some signature of built-in operations. For us, quantum circuits are abstractly given as the morphisms of monoidal categories with certain properties, which we now specify.

We start from a given symmetric monoidal category $\mathbf{M}$. In practice, the objects of $\mathbf{M}$ are typically generated from wire types such as **Bit** and **Qubit**, and the morphisms are quantum circuits generated from a finite set of gates. In the following, we define what we mean by a reversible subcategory of $\mathbf{M}$. Recall that a dagger symmetric monoidal category is a symmetric monoidal category equipped with a contravariant, identity-on-objects, involutive functor such that for all morphisms $f : A \to B$, we have $f^\dagger : B \to A$. Moreover, the dagger functor is required to be compatible with the symmetric monoidal structure [29].

Definition 3.1. By a *reversible subcategory* of $\mathbf{M}$, we mean a dagger symmetric monoidal category $\mathbf{R}$, together with an identity-on-objects, faithful,

symmetric (strong) monoidal functor $I : \mathbf{R} \to \mathbf{M}$. We usually regard I as an inclusion functor, i.e., we regard $\mathbf{R}$ as a subcategory of $\mathbf{M}$.

A morphism f of $\mathbf{R}$ is called *unitary* if $f \circ f^\dagger = \mathrm{id}$ and $f^\dagger \circ f = \mathrm{id}$. Note that we do not require all morphisms of $\mathbf{R}$ to be unitary. That is, our notion of reversible morphism means a morphism that has an adjoint, not necessarily an inverse. However, if the morphism is unitary, its adjoint is of course its inverse. The dagger functor provides a semantics for the operation of reversing a quantum circuit.

In order to capture the notion of control and with-computed, we define a notion of controllable category $\mathbf{N}$. To motivate the following definition, we should first clarify some use cases. The most common example of control from quantum computing is the control usually represented by a black dot "$\bullet$". This kind of control "fires" the gate if the control qubit is in state $|1\rangle$, and acts as the identity otherwise. There are also other ways of controlling a qubit; for example, the white dot "$\circ$" fires if the control qubit is in state $|0\rangle$. A more general kind of control in quantum computing is as follows: Let V, W be Hilbert spaces, let K be a subspace of V, and let $G : W \to W$ be a gate. We define the K-controlled gate $\mathsf{ctrl}_K(G) : V \otimes W \to V \otimes W$ by $\mathsf{ctrl}_K(G)(v \otimes w) = v \otimes G(w)$ if $v \in K$ and $\mathsf{ctrl}_K(G)(v \otimes w) = v \otimes w$ if $v \in K^\perp$, extended to all other states by linearity. Note that this general notion of control not only encompasses the black and white dot, but also many other kinds; for example, it is possible to control a gate on the state $|+\rangle$, to control on multiple qubits and/or classical bits, in which case the subspace K may or may not be 1-dimensional. Even the trivial control is possible, namely when $K = V$; in this case the gate always "fires".

Working in an abstract monoidal category, we will not talk about subspaces and their orthogonal complements. Instead, we consider a monoid $\mathbf{C}$ of *controls*, regarded as a discrete monoidal category, together with a monoidal functor $F : \mathbf{C} \to \mathbf{N}$. We write $\underline{K}$ for $F(K)$. The idea is that each $K \in \mathbf{C}$ specifies a kind of control on the object $\underline{K} \in \mathbf{N}$.

Definition 3.2. Let $\mathbf{R}$ be a dagger symmetric monoidal category. By a *controllable category* of $\mathbf{R}$, we mean a dagger symmetric monoidal category $\mathbf{N}$ with the same objects as $\mathbf{R}$, and equipped with the following structure:

(**a**) For all $A, B \in \mathbf{N}$, a function $- \bullet - : \mathbf{R}(B, A) \times \mathbf{N}(A, A) \to \mathbf{N}(B, B)$ (also denoted by $\mathsf{withComputed}$) such that:

$$\mathrm{id} \bullet h = h$$

$$(g_1; g_2) \bullet h = g_1 \bullet (g_2 \bullet h)$$
$$(g_1 \otimes g_2) \bullet (h_1 \otimes h_2) = (g_1 \bullet h_1) \otimes (g_2 \bullet h_2)$$
$$(g \bullet h)^\dagger = g \bullet h^\dagger.$$

Here we use the semicolon to mean forward composition of morphisms.

(**b**) A discrete monoidal category $\mathbf{C}$ of *controls*, together with a monoidal functor mapping $K \in \mathbf{C}$ to $\underline{K} \in \mathbf{N}$.

(**c**) For all $A \in \mathbf{N}$ and $K \in \mathbf{C}$, a function $\mathsf{ctrl}_K : \mathbf{N}(A, A) \to \mathbf{N}(\underline{K} \otimes A, \underline{K} \otimes A)$ such that the following hold:

$$\mathsf{ctrl}_K(\mathrm{id}_A) = \mathrm{id}_{\underline{K} \otimes A}$$

$$\begin{array}{ccc} \underline{I} \otimes A & \xrightarrow{\mathsf{ctrl}_I(h)} & \underline{I} \otimes A \\ \downarrow \cong & & \downarrow \cong \\ A & \xrightarrow{h} & A, \end{array}$$

$$\begin{array}{ccc} (\underline{A \otimes B}) \otimes C & \xrightarrow{\mathsf{ctrl}_{A \otimes B}(h)} & (\underline{A \otimes B}) \otimes C \\ \downarrow \cong & & \downarrow \cong \\ \underline{A} \otimes (\underline{B} \otimes C) & \xrightarrow{\mathsf{ctrl}_A(\mathsf{ctrl}_B(h))} & \underline{A} \otimes (\underline{B} \otimes C), \end{array}$$

$$\begin{array}{ccc} (\underline{A \otimes B}) \otimes C & \xrightarrow{\mathsf{ctrl}_{A \otimes B}(h)} & (\underline{A \otimes B}) \otimes C \\ \downarrow \cong & & \downarrow \cong \\ (\underline{B \otimes A}) \otimes C & \xrightarrow{\mathsf{ctrl}_{B \otimes A}(h)} & (\underline{B \otimes A}) \otimes C. \end{array}$$

(**d**) An identity-on-objects, strict dagger symmetric monoidal functor $G : \mathbf{N} \to \mathbf{R}$ such that

$$G(g \bullet h) = g; G(h); g^\dagger$$

Remarks.

- The functor G gives an intended interpretation for the with-computed function. It captures the common quantum computation pattern $g; h; g^\dagger$.

- The functor G preserves the equalities of condition (a). For example, $G(\mathrm{id} \bullet h) = \mathrm{id}; G(h); \mathrm{id}^\dagger = G(h)$. Note that we do not require $g \bullet \mathrm{id}_A = \mathrm{id}_B$, because this would imply $g; g^\dagger = g; \mathrm{id}_A; g^\dagger = G(g \bullet \mathrm{id}_A) = G(\mathrm{id}_B) = \mathrm{id}_B$, which is not an assumption we make for $\mathbf{R}$. For similar reasons, we do not require $g \bullet (h_1; h_2) = (g \bullet h_1); (g \bullet h_2)$.

To illustrate how the above notions are applicable to quantum computing, we can consider a "standard model" of these axioms, parameterized by a class of quantum circuits. Suppose the circuits are generated by some types of wires (for example, **Qubit** and **Bit**) and some set of gates. Suppose that some distinguished subset of the gates are reversible, and a further subset of these are controllable. Also suppose that the class of circuits is equipped with some primitive control gadgets, for example, a black-dot and a white-dot control on qubits, and that for each gate, all of its controlled versions are also gates. The category $\mathbf{M}$ consists of all circuits, arranged into a symmetric monoidal category (the symmetric monoidal structure permits the crossing of wires, but such crossings are not considered to be gates). Let $\mathbf{R}$ be the subcategory of circuits that are made from reversible gates. Finally, the structure of $\mathbf{N}$ is somewhat interesting. The morphisms of $\mathbf{N}$ are circuits made from gates of two colors, say red and green. Each red gate is reversible and each green gate is controllable (and in particular, each controllable gate comes in two different colors). All morphisms in $\mathbf{N}$ are reversible, and reversing does not change color. The objects of the monoidal category $\mathbf{C}$ are sequences of control gadgets, for example $\bullet \circ \circ$, and $\underline{\bullet \circ \circ} = \mathbf{Qubit} \otimes \mathbf{Qubit} \otimes \mathbf{Qubit}$. The control function adds controls to the green gates. The with-computed function takes any morphism g of $\mathbf{R}$ and any morphism f of $\mathbf{N}$, and produces $g; f; g^\dagger$, where g and $g^\dagger$ have been colored red. The functor G is the forgetful functor that forgets the colors.

4. A type system for reversing, control and with-computed

In this section, we define the syntax and type system of Proto-Quipper-C, a version of Proto-Quipper with support for reversing, control and with-computed. We assume that we are given three fixed small categories $\mathbf{M}$, $\mathbf{R}$, and $\mathbf{N}$ as specified in the previous section. The type system and (in later sections) the operational semantics and categorical semantics are all parameterized by this choice.

Modalities	α, β	::=	$2 \mid 1 \mid 0$
Types	A, B	::=	$\mathbf{Unit} \mid W \mid \mathbf{Bool} \mid !_\alpha A \mid A \multimap_\alpha B$
			$\mid \mathbf{Circ}_\alpha(S, U) \mid A \otimes B$
Parameter Types	P, R	::=	$\mathbf{Unit} \mid \mathbf{Bool} \mid !_\alpha A \mid \mathbf{Circ}_\alpha(S, U) \mid P \otimes R$
Simple Types	S, U	::=	$\mathbf{Unit} \mid W \mid S \otimes U$
Terms	M, N	::=	$c \mid x \mid \ell \mid \mathsf{unit} \mid \lambda x.M \mid MN \mid (M, N)$
			$\mid \mathsf{let}\ (x, y) = N\ \mathsf{in}\ M \mid \mathsf{lift}\ M \mid \mathsf{force}\, M$
			$\mid \mathsf{circ}(S, C, U) \mid \mathsf{apply}(M, N) \mid \mathsf{box}_U M$
			$\mid \mathsf{reverse}\, M \mid \mathsf{control}_K M$
			$\mid \mathsf{withComputed}\, M\, N$
Simple Terms	a, b	::=	$\ell \mid \mathsf{unit} \mid (a, b)$
Contexts	Γ	::=	$\cdot \mid x : A, \Gamma \mid \ell : W, \Gamma$
Parameter contexts	Φ	::=	$\cdot \mid x : P, \Phi.$
Label Contexts	Σ	::=	$\cdot \mid \ell : W, \Sigma$
Values	V	::=	$c \mid x \mid \ell \mid \mathsf{unit} \mid \lambda x.M \mid (V, V') \mid \mathsf{lift}\ M$
			$\mid \mathsf{circ}(S, C, U)$
Circuits	C, D	$\in$	$\mathbf{N}([\![S]\!], [\![U]\!]) \mid \mathbf{R}([\![S]\!], [\![U]\!]) \mid \mathbf{M}([\![S]\!], [\![U]\!])$

Figure 1: The syntax of Proto-Quipper-C

4.1. Syntax

The syntax of Proto-Quipper-C is shown in Figure 1. It is similar to the one specified in [12], except for the highlighted terms and types.

Simple types. The letter W ranges over a set of *wire types*, which in typical applications will be types that represent a single wire in a circuit, such as **Qubit** and **Bit**. Each wire type represents a particular object of the category $\mathbf{M}$, and tensors of wire types are called *simple types*. We write $[\![S]\!]$ to denote the object corresponding to S in the category $\mathbf{M}$, $\mathbf{N}$, or $\mathbf{R}$ (note that they all have the same objects).

Labels. We also assume that ℓ ranges over a set of *labels*, which are distinct from variables and are used as pointers to places in a circuit where a gate can be attached. Unlike variables, labels cannot be substituted, and they can only have wire types. A *simple term* is essentially a tuple of labels. We call a typing context that contains only labels a *label context* (denoted by Σ). If $\Sigma = \ell_1 : W_1, \ldots, \ell_n : W_n$, we write $[\![\Sigma]\!]$ for the object corresponding

to Σ in $\mathbf{M}$, $\mathbf{N}$, or $\mathbf{R}$, i.e., $[\![W_1]\!] \otimes \ldots \otimes [\![W_n]\!]$.

Parameter types. A certain subset of the types are called *parameter types*. These are the types whose values can be freely duplicated and discarded, whereas values of simple types are resources. Also, the values of parameter types are computed at circuit generation time, whereas the values of simple types are merely tuples of labels, which are placeholders for values that will be computed at circuit execution time. A typing context that contains only parameter types is a *parameter context* (denoted by Φ). We have included **Bool** as a typical example of a parameter type, but in a real language, one would of course have many more such types (such as **Nat**, **Int**, sum types, etc.). Such types can be handled by standard methods and we omit a comprehensive treatment of them.

Terms. As usual, x ranges over a set of *variables*. The symbol c is used for constant symbols of the language, such as True and False for the type **Bool**, as well as other appropriate constants. The terms of the lambda calculus are fairly standard. Lift and force are from linear lambda calculus, and are used to construct and deconstruct a term of type $!_\alpha A$. A value of type $!_\alpha A$ is a suspended computation that can be forced by force. We will discuss the remaining terms later.

Modalities. The symbols α and β range over modalities, which we take to be numbers in the lattice $2 > 1 > 0$. Here, $\alpha = 2$ indicates that a circuit is reversible and controllable; $\alpha = 1$ indicates that a circuit is reversible but not necessarily controllable, and $\alpha = 0$ indicates a general circuit that may be neither reversible nor controllable. The modality in the type $!_\alpha A$ means that when its value is forced, it may append a gate that has modality α to the then-current circuit. Similarly, the modality in the type $A \multimap_\alpha B$ indicates that when its value is applied to an argument, it may append a gate that has modality α.

Circuits. Since Proto-Quipper-C is a circuit description language, we must have some terms in the language that represent circuits. We write such terms as $\mathsf{circ}(S, C, U)$, where $C : [\![S]\!] \to [\![U]\!]$ is a morphism in the appropriate circuit category $\mathbf{M}$, $\mathbf{R}$, or $\mathbf{N}$. Note that in this context, category theory is used not as a denotational semantics (that will be done in Section 6), but as part of the *syntax* of the language. This is effectively the same as adding a constant symbol for every possible circuit. We write $\mathbf{Circ}_\alpha(S, U)$ for the type of quantum circuits with inputs S and outputs U (which are simple types), subject to the modality α as described above. The terms apply and box are used for constructing and deconstructing circuits; they will be explained in

$$\frac{}{\Phi, x : A \vdash_2 x : A}\ \mathit{var} \qquad \frac{}{\Phi, \ell : W \vdash_2 \ell : W}\ \mathit{label}$$

$$\frac{}{\Phi \vdash_2 \mathsf{unit} : \mathbf{Unit}}\ \mathit{unit} \qquad \frac{}{\Phi \vdash_2 c : A_c}\ \mathit{const}$$

$$\frac{\Gamma, x : A \vdash_\alpha M : B}{\Gamma \vdash_2 \lambda x.M : A \multimap_\alpha B}\ \mathit{lambda} \qquad \frac{\Phi, \Gamma_1 \vdash_{\alpha_1} M : A \multimap_\beta B \quad \Phi, \Gamma_2 \vdash_{\alpha_2} N : A}{\Phi, \Gamma_1, \Gamma_2 \vdash_{\alpha_1 \wedge \alpha_2 \wedge \beta} MN : B}\ \mathit{app}$$

$$\frac{\Phi \vdash_\alpha M : A}{\Phi \vdash_2 \mathsf{lift}\, M : !_\alpha A}\ \mathit{lift} \qquad \frac{\Phi, \Gamma_1 \vdash_{\alpha_1} M : A \quad \Phi, \Gamma_2 \vdash_{\alpha_2} N : B}{\Phi, \Gamma_1, \Gamma_2 \vdash_{\alpha_1 \wedge \alpha_2} (M, N) : A \otimes B}\ \mathit{pair}$$

$$\frac{\Gamma \vdash_\beta M : !_\alpha A}{\Gamma \vdash_{\alpha \wedge \beta} \mathsf{force}\ M : A}\ \mathit{force} \qquad \frac{\Phi, \Gamma_1, x : A, y : B \vdash_{\alpha_1} M : C \quad \Phi, \Gamma_2 \vdash_{\alpha_2} N : A \otimes B}{\Phi, \Gamma_1, \Gamma_2 \vdash_{\alpha_1 \wedge \alpha_2} \mathsf{let}\ (x, y) = N\ \mathsf{in}\ M : C}\ \mathit{let}$$

$$\frac{C \in \mathbf{D}^\alpha([\![S]\!], [\![U]\!])}{\Phi \vdash_2 \mathsf{circ}(S, C, U) : \mathbf{Circ}_\alpha(S, U)}\ \mathit{circ} \qquad \frac{\Phi, \Gamma_1 \vdash_\alpha M : \mathbf{Circ}_\gamma(S, U) \quad \Phi, \Gamma_2 \vdash_\beta N : S}{\Phi, \Gamma_1, \Gamma_2 \vdash_{\alpha \wedge \beta \wedge \gamma} \mathsf{apply}(M, N) : U}\ \mathit{apply}$$

$$\frac{\Gamma \vdash_\alpha M : !_\beta(S \multimap_\gamma U)}{\Gamma \vdash_\alpha \mathsf{box}_S\ M : \mathbf{Circ}_{\beta \wedge \gamma}(S, U)}\ \mathit{box} \qquad \frac{\Gamma \vdash_\alpha M : \mathbf{Circ}_2(S, S)}{\Gamma \vdash_\alpha \mathsf{control}_K M : \mathbf{Circ}_2(\underline{K} \otimes S, \underline{K} \otimes S)}\ \mathit{ctrl}$$

$$\frac{\Gamma \vdash_\alpha M : \mathbf{Circ}_\gamma(S, U) \quad \gamma > 0}{\Gamma \vdash_\alpha \mathsf{reverse}\, M : \mathbf{Circ}_\gamma(U, S)}\ \mathit{rev} \qquad \frac{\begin{array}{c}\Phi, \Gamma_1 \vdash_\alpha M : \mathbf{Circ}_\gamma(U, S) \quad \gamma > 0 \\ \Phi, \Gamma_2 \vdash_\beta N : \mathbf{Circ}_2(S, S)\end{array}}{\Phi, \Gamma_1, \Gamma_2 \vdash_{\alpha \wedge \beta} \mathsf{withComputed}\, M\, N : \mathbf{Circ}_2(U, U)}\ \mathit{wc}$$

Figure 2: The typing rules

more detail in the section on the operational semantics. We include the terms $\mathsf{reverse}\, M$, $\mathsf{control}_K M$, and $\mathsf{withComputed}\, M\, N$ for reversing, control and the with-computed operation. The subscript K ranges over the objects of the category $\mathbf{C}$, which is part of the given structure of the control category $\mathbf{N}$. We moreover assume that $\underline{K}$ is a simple type (rather than just an arbitrary object).

In a practical implementation, programmers will not usually write values of the form $\mathsf{circ}(S, C, U)$ directly. Instead, basic quantum gates such as the Hadamard gate will usually be predefined as a library function $H : !(\mathbf{Qubit} \multimap \mathbf{Qubit})$, and users will use operations such as box and unbox to combine these into larger circuits.

4.2. Typing rules

The typing rules are shown in Figure 2. Typing judgments are of the form $\Gamma \vdash_\alpha M : A$. Here, the modality α asserts that during the evaluation of the term M, gates of modality α may be appended to the current circuit. Recall that we write $\alpha \wedge \beta$ to denote the greatest lower bound, or $\min(\alpha, \beta)$. To facilitate the statement of the typing rules, we treat the contexts as unordered lists.

Except for the modalities, the typing rules are similar to the ones in [12]. We make the following observations about the modalities. Since a value cannot append any gates, it does not change the current circuit state. Therefore values always have modality 2, i.e., $\Gamma \vdash_2 V : A$. In the *const* rule, A_c is the type of the constant symbol c. The *lambda* and *lift* rules store the modality α of M in the respective types $A \multimap_\alpha B$ and $!_\alpha A$. Similarly, in the typing rule *box*, the modalities β, γ jointly determine the modality of the circuit type. Conversely, in the rules *app*, *force*, and *apply*, the modality in the types $A \multimap_\alpha B$, $!_\alpha A$, and $\mathbf{Circ}_\gamma(S, U)$ affects the modality of the current term. In the rule *circ*, we write $\mathbf{D}^\alpha$ to mean $\mathbf{M}$ if $\alpha = 0$, and $\mathbf{R}$ if $\alpha = 1$, and $\mathbf{N}$ if $\alpha = 2$. The *rev* rule requires the circuit to be reversible, i.e., γ must be at least 1. Similarly, the rule *ctrl* requires a controllable circuit, i.e., modality 2. It also requires the circuit to have matching input and output types, i.e., $\mathbf{Circ}_2(S, S)$ rather than $\mathbf{Circ}_2(S, U)$. Finally, the typing rule for with-computed requires the term N to be a controllable circuit, whereas the term M only needs to be reversible. The resulting term $\mathsf{withComputed}\, M\, N$ is again a controllable circuit.

5. Operational semantics

In this section, we give an operational semantics of our language. We follow the usual paradigm of keeping type checking separate from evaluation, i.e., after type checking succeeds, the evaluation is done on untyped terms. Accordingly, in Section 5.1, we provide evaluation rules for untyped terms (which could potentially lead to runtime errors). In Section 5.2, we prove that well-typed terms do not cause runtime errors.

5.1. Evaluation rules

In this section, we define a big-step, call-by-value operational semantics for Proto-Quipper-C. Like the syntax and the type system, the operational semantics is parameterized by the triple of categories $\mathbf{M}$, $\mathbf{R}$, and $\mathbf{N}$. The

$$\frac{(C_1, \Sigma_1, M) \Downarrow (C_2, \Sigma_2, \lambda x.M') \quad (C_2, \Sigma_2, N) \Downarrow (C_3, \Sigma_3, V) \quad (C_3, \Sigma_3, [V/x]M') \Downarrow (C_4, \Sigma_4, V')}{(C_1, \Sigma_1, MN) \Downarrow (C_4, \Sigma_4, V')}\ app$$

$$\frac{(C, \Sigma, M) \Downarrow (C', \Sigma', \mathsf{lift}\ M') \quad (C', \Sigma', M') \Downarrow (C'', \Sigma'', V)}{(C, \Sigma, \mathsf{force}\ M) \Downarrow (C'', \Sigma'', V)}\ force$$

$$\frac{(C, \Sigma, N) \Downarrow (C', \Sigma', (V_1, V_2)) \quad (C', \Sigma', [V_1/x, V_2/y]M) \Downarrow (C'', \Sigma'', V)}{(C, \Sigma, \mathsf{let}\ (x, y) = N\ \mathsf{in}\ M) \Downarrow (C'', \Sigma'', V)}\ let$$

$$\frac{(C, \Sigma, M) \Downarrow (C', \Sigma', V_1) \quad (C', \Sigma', N) \Downarrow (C'', \Sigma'', V_2)}{(C, \Sigma, (M, N)) \Downarrow (C'', \Sigma'', (V_1, V_2))}\ pair$$

$$\frac{\begin{array}{c}(C, \Sigma, M) \Downarrow (C', \Sigma', \mathsf{lift}\ M') \\ \mathrm{gen}(S) = (a, \Sigma'') \\ ([\![a]\!]^\dagger, \Sigma'', M'\ a) \Downarrow (D, \Sigma''', b) \\ \mathrm{ungen}(b, \Sigma''') = U\end{array}}{(C, \Sigma, \mathsf{box}_S\ M) \Downarrow (C', \Sigma', \mathsf{circ}(S, [\![b]\!] \circ D, U))}\ box$$

$$\frac{\begin{array}{c}(C_1, \Sigma_1, M) \Downarrow (C_2, \Sigma_2, \mathsf{circ}(S, D, U)) \\ (C_2, \Sigma_2, N) \Downarrow (C_3, \Sigma_3, a) \\ \mathrm{gen}(U) = (b, \Sigma) \\ (C', \Sigma') = \mathrm{append}(C_3, \Sigma_3, a, D, b, \Sigma)\end{array}}{(C_1, \Sigma_1, \mathsf{apply}(M, N)) \Downarrow (C', \Sigma', b)}\ apply$$

$$\frac{(C, \Sigma, M) \Downarrow (C', \Sigma', \mathsf{circ}(S, D, U))}{(C, \Sigma, \mathsf{reverse}\, M) \Downarrow (C', \Sigma', \mathsf{circ}(U, D^\dagger, S))}\ rev$$

$$\frac{(C, \Sigma, M) \Downarrow (C', \Sigma', \mathsf{circ}(S, D, U))}{(C, \Sigma, \mathsf{control}_K M) \Downarrow (C', \Sigma', \mathsf{circ}(\underline{K} \otimes S, \mathsf{ctrl}_K D, \underline{K} \otimes S))}\ ctrl$$

$$\frac{(C, \Sigma, M) \Downarrow (C', \Sigma', \mathsf{circ}(S, D_1, U)) \quad (C', \Sigma', N) \Downarrow (C'', \Sigma'', \mathsf{circ}(U, D_2, U))}{(C, \Sigma, \mathsf{withComputed}\, M\, N) \Downarrow (C'', \Sigma'', \mathsf{circ}(S, D_1 \bullet D_2, S))}\ wc$$

Figure 3: The operational semantics

operational semantics is defined in Figure 3. It is defined on configurations that are triples (C, Σ, M), where M is a term, Σ is a label context, and $C : X \to [\![\Sigma]\!]$ is a morphism in $\mathbf{M}$, $\mathbf{R}$ or $\mathbf{N}$. We call the pair (C, Σ) the *circuit state* of the configuration. The evaluation of a closed program M begins with the empty circuit state, i.e., $(\mathrm{id}_I, \emptyset)$.

The *app*, *force*, *let* and *pair* rules are standard. They do not directly modify the underlying circuit state.

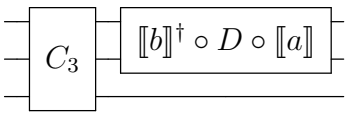
The *box* and *apply* rules use some notations that warrant explaining. If Σ is a label context, a a simple term, and S a simple type, we write $\Sigma \vdash a : S$ instead of $\Sigma \vdash_2 a : S$. In this case, Σ and S determine each other uniquely given a. We write $\mathrm{ungen}(a, \Sigma)$ for the unique S such that $\Sigma \vdash a : S$. Conversely, we write $\mathrm{gen}(S) = (a, \Sigma)$ to indicate the generation of a fresh simple term a and a corresponding label context Σ such that $\Sigma \vdash a : S$. There is an obvious interpretation $[\![a]\!] : [\![\Sigma]\!] \to [\![S]\!]$ as an isomorphism in $\mathbf{M}$, $\mathbf{R}$, or $\mathbf{N}$.

In the *box* rule, the codomain of D is $[\![\Sigma''']\!]$ and by the definition of ungen, we have $\Sigma''' \vdash b : U$. Then $[\![S]\!] \xrightarrow{D} [\![\Sigma''']\!] \xrightarrow{[\![b]\!]} [\![U]\!]$ is a morphism in $\mathbf{M}$, $\mathbf{R}$ or $\mathbf{N}$.

In the *rev* and *ctrl* rules, the morphism D is reversed/controlled using the reverse/control function from the category it belongs to. If D is not a morphism in $\mathbf{R}$ or $\mathbf{N}$, respectively, then this will cause a runtime error. The type preservation property (Theorem 5.2) will ensure that there are no such runtime errors. In the *wc* rule, we again abuse the notation; if $D_1 \in \mathbf{N}$, we apply the functor $G : \mathbf{N} \to \mathbf{R}$ before the "$\bullet$" operation. It is a runtime error if $D_1 \in \mathbf{M}$ or $D_2 \notin \mathbf{N}$.

In the *apply* rule, by the definition of configurations, we have $C_3 : X \to [\![\Sigma_3]\!]$, for some object X. We also have $D : [\![S]\!] \to [\![U]\!]$. Moreover, $\mathrm{gen}(U) = (b, \Sigma)$ implies that $\Sigma \vdash b : U$ and $[\![b]\!] : [\![\Sigma]\!] \to [\![U]\!]$. Let Σ'_3 be the unique label context such that $\Sigma'_3 \vdash a : S$. We assert that Σ_3 can be written in the form $\Sigma_3 = \Sigma'_3, \Sigma''_3$ (it is a runtime error if the assertion fails). We have $[\![a]\!] : [\![\Sigma'_3]\!] \to [\![S]\!]$. Note that $([\![b]\!]^\dagger \circ D \circ [\![a]\!]) : [\![\Sigma'_3]\!] \to [\![\Sigma]\!]$. Let $\Sigma' = \Sigma, \Sigma''_3$, and let $C' : X \to [\![\Sigma]\!]$ be the following composition:

$$C' = X \xrightarrow{C_3} [\![\Sigma_3]\!] \xrightarrow{\cong} [\![\Sigma'_3]\!] \otimes [\![\Sigma''_3]\!] \xrightarrow{([\![b]\!]^\dagger \circ D \circ [\![a]\!]) \otimes \mathrm{id}} [\![\Sigma]\!] \otimes [\![\Sigma''_3]\!] \xrightarrow{\cong} [\![\Sigma']\!] \quad =$$



.

We write $(C', \Sigma') = \text{append}(C_3, \Sigma_3, a, D, b, \Sigma)$ for this operation. With a slight abuse of notation, this composition is always possible, even when the morphisms belong to different categories. For example, if $D \in \mathbf{R}$ and $C_3 \in \mathbf{M}$, we can apply the functor $I : \mathbf{R} \to \mathbf{M}$ to D before the composition.

Example. Since the evaluation rules of Figure 3 are a bit complex, we give an example illustrating their use. Consider the configuration

$$(\mathrm{id}_I,\ \emptyset,\ \mathsf{box}_S\,\mathsf{lift}(\lambda x.\mathsf{let}(a_1, a_2) = x\ \mathsf{in}\ \mathsf{apply}(\mathsf{circ}(S, \mathsf{CNOT}, S), (a_2, a_1)))),$$

where $S = \mathbf{Qubit} \otimes \mathbf{Qubit}$ and $\mathsf{CNOT} : \mathbf{Qubit} \otimes \mathbf{Qubit} \to \mathbf{Qubit} \otimes \mathbf{Qubit}$ is a morphism in $\mathbf{N}$. Using the *box* rule, we first call $\text{gen}(S)$, which returns a pair of fresh labels (ℓ_1, ℓ_2) and the label context $\Sigma = \ell_1 : \mathbf{Qubit}, \ell_2 : \mathbf{Qubit}$. Next, using the rules *app* and *let*, we evaluate

$$([\![(\ell_1, \ell_2)]\!]^\dagger,\ \Sigma,\ (\lambda x.\mathsf{let}(a_1, a_2) = x\ \mathsf{in}\ \mathsf{apply}(\mathsf{circ}(S, \mathsf{CNOT}, S), (a_2, a_1))))(\ell_1, \ell_2))$$

to

$$(([\![(\ell_1, \ell_2)]\!]^\dagger,\ \Sigma,\ \mathsf{apply}(\mathsf{circ}(S, \mathsf{CNOT}, S), (\ell_2, \ell_1))).$$

Then by the *apply* rule, it is evaluated to

$$([\![(\ell_3, \ell_4)]\!]^\dagger \circ \mathsf{CNOT} \circ [\![(\ell_2, \ell_1)]\!] \circ [\![(\ell_1, \ell_2)]\!]^\dagger,\ \Sigma',\ (\ell_3, \ell_4)),$$

where $\text{gen}(S)$ returns fresh labels (ℓ_3, ℓ_4) and the label context $\Sigma' = \ell_3 : \mathbf{Qubit}, \ell_4 : \mathbf{Qubit}$. Therefore, continuing with the *box* rule, we obtain

$$(\mathrm{id}_I,\ \emptyset,\ \mathsf{circ}(S, [\![(\ell_3, \ell_4)]\!] \circ [\![(\ell_3, \ell_4)]\!]^\dagger \circ \mathsf{CNOT} \circ [\![(\ell_2, \ell_1)]\!] \circ [\![(\ell_1, \ell_2)]\!]^\dagger, S).$$

Note that $[\![(\ell_3, \ell_4)]\!] \circ [\![(\ell_3, \ell_4)]\!]^\dagger = \mathrm{id}_S$ and $[\![(\ell_2, \ell_1)]\!] \circ [\![(\ell_1, \ell_2)]\!]^\dagger = \gamma : \mathbf{Qubit} \otimes \mathbf{Qubit} \to \mathbf{Qubit} \otimes \mathbf{Qubit}$, where the morphism γ comes from the symmetric monoidal structure on $\mathbf{N}$. Thus, the final configuration is

$$(\mathrm{id}_I,\ \emptyset,\ \mathsf{circ}(S, \mathsf{CNOT} \circ \gamma, S)).$$

Note that the effect of the morphism γ is to switch two wires without inserting an explicit swap gate (it is, for example, the same as the interpretation of the function `f` in Section 2). If we later control the circuit $\mathsf{circ}(S, \mathsf{CNOT} \circ \gamma, S)$, this morphism γ will be replaced by a controlled swap gate, just like circuit (b) in Section 2. Note that this replacement is not handled at the level of the operational semantics, but is handled by the ctrl_K function, which is built into the circuit model.

5.2. Type preservation

In order to formulate type preservation, we first define a notion of *well-typed configuration*.

Definition 5.1 (Well-typed configuration)**.** We define a well-typed configuration $S \vdash_\alpha (C, \Sigma, M) : A; \Sigma'$ to mean: There exist β, γ, Σ'' such that $C \in \mathbf{D}^\beta([\![S]\!], [\![\Sigma]\!])$, $\Sigma = (\Sigma'', \Sigma')$, $\Sigma'' \vdash_\gamma M : A$ and $\alpha = \beta \wedge \gamma$.

Theorem 5.2 (Type preservation)**.** *Suppose* $S \vdash_\alpha (C_1, \Sigma_1, M) : A; \Sigma'$ *and* $(C_1, \Sigma_1, M) \Downarrow (C_2, \Sigma_2, V)$*. Then* $S \vdash_\alpha (C_2, \Sigma_2, V) : A; \Sigma'$*.*

Proof. The proof is by induction on the derivation of $(C_1, \Sigma_1, M) \Downarrow (C_2, \Sigma_2, V)$. □

6. A categorical semantics for reversing and control

6.1. The abstract model

As before, we assume that the categories **M**, **R**, and **N** are given. These three categories lack the necessary structures to interpret a programming language. Instead, we will interpret the typing rules in a category **A**, whose structure we now describe.

Definition 6.1. A category **A** is a *model for Proto-Quipper-C* if it has the following structure.

(**a**) **A** is symmetric monoidal closed. We write $\epsilon : (A \multimap B) \otimes A \to B$ for the application map.

(**b**) **A** has coproducts. Note that the tensor distributes over coproducts, because $- \otimes A$ is a left adjoint.

(**c**) There is an adjunction $p \dashv \flat : \mathbf{Set} \to \mathbf{A}$ where p is a strong monoidal functor.

(**d**) **A** is equipped with idempotent, commutative, and strong monads T_0 and T_1. (A monad T is called idempotent to mean that its multiplication $\mu : T^2 \to T$ is a natural isomorphism.) Moreover, $T_0T_1 \cong T_0 \cong T_1T_0$ via natural isomorphisms $\eta_{T_0} : T_0 \to T_1T_0$ and $T_0\eta : T_0 \to T_0T_1$ for the unit η of T_1. We write $s : TA \otimes B \to T(A \otimes B)$ for the strength and $t : A \otimes TB \to T(A \otimes B)$ for the costrength.

(**e**) There are full, faithful, and strong monoidal embeddings $\phi_3 : \mathbf{N} \hookrightarrow \mathbf{A}$, $\phi_2 : \mathbf{R} \hookrightarrow Kl_{T_1}(\mathbf{A})$, and $\phi_1 : \mathbf{M} \hookrightarrow Kl_{T_0}(\mathbf{A})$ that make the following diagram commute, where E_1 and E_0 are the canonical identity-on-objects functors.

$$\begin{array}{ccc} \mathbf{N} & \xrightarrow{\phi_3} & \mathbf{A} \\ {\scriptstyle G}\downarrow & & \downarrow{\scriptstyle E_1} \\ \mathbf{R} & \xrightarrow{\phi_2} & Kl_{T_1}(\mathbf{A}) \\ {\scriptstyle I}\downarrow & & \downarrow{\scriptstyle E_0} \\ \mathbf{M} & \xrightarrow[\phi_1]{} & Kl_{T_0}(\mathbf{A}) \end{array} \tag{4}$$

Explicitly, for any $f \in \mathbf{A}(A, B)$ we have $E_1(f) \in Kl_{T_1}(\mathbf{A})(A, B)$ given by $E_1(f) = \eta^1_B \circ f$ for the unit η^1 of T_1, and for any $g \in Kl_{T_1}(\mathbf{A})(A, B)$ we have $E_0(g) \in Kl_{T_0}(\mathbf{A})(A, B)$ given by $E_0(g) = A \xrightarrow{g} T_1 B \xrightarrow{\eta^0_{T_1 B}} T_0 T_1 B \xrightarrow{(T_0 \eta^1_B)^{-1}} T_0 B$ for the unit η^0 of T_0.

Conditions (a)–(c) are not new; they appear in models of many versions of Proto-Quipper starting with [26]. Because $p : \mathbf{Set} \to \mathbf{A}$ is a left adjoint and is strong monoidal, we can deduce that $p(X) \cong p(\sum_{x \in X} 1) \cong \sum_{x \in X} p(1) \cong \sum_{x \in X} I$. Due to the adjunction $p \dashv \flat$, we also have $\mathbf{Set}(X, \flat B) \cong \mathbf{A}(p(X), B) \cong \mathbf{A}(\sum_{x \in X} I, B) \cong \prod_{x \in X} \mathbf{A}(I, B) \cong \mathbf{Set}(X, \mathbf{A}(I, B))$. Therefore by Yoneda's principle, we have $\flat(B) \cong \mathbf{A}(I, B)$. We write $\mathsf{force} : p\flat \to \mathrm{id}$ for the counit of the comonad $p\flat$.

For convenience, we define $T_2 = \mathrm{id}$ to be the identity monad on $\mathbf{A}$. Then in condition (d), the monads T_0, T_1, and T_2 represent (via their Kleisli categories) general circuits, reversible circuits, and controllable circuits, respectively. Recall that we write $\mathbf{D}^\alpha$ for the categories $\mathbf{M}$, $\mathbf{R}$, and $\mathbf{N}$ when $\alpha = 0$, 1, and 2, respectively. Condition (e) ensures that morphisms of simple types in the Kleisli category $Kl_{T_\alpha}(\mathbf{A})$ correspond to morphisms in $\mathbf{D}^\alpha$.

For any $S, U \in \mathrm{obj}(\mathbf{M})$, the maps unbox and box are defined by

$$\mathsf{unbox} : \mathbf{D}^\beta(S, U) \xrightarrow{\cong} \mathbf{A}(S, T_\beta U) \xrightarrow{\text{curry}} \mathbf{A}(I, S \multimap T_\beta U) \xrightarrow{\cong} \flat(S \multimap T_\beta U).$$

$$\begin{aligned} \mathsf{box} : \flat T_\alpha(S \multimap T_\beta U) &\xrightarrow{\cong} \mathbf{Set}(1, \flat T_\alpha(S \multimap T_\beta U)) \xrightarrow{\cong} \mathbf{A}(I, T_\alpha(S \multimap T_\beta U)) \\ &\xrightarrow{k} \mathbf{A}(S, T_{\alpha \wedge \beta} U) \xrightarrow{\cong} \mathbf{D}^{\alpha \wedge \beta}(S, U). \end{aligned}$$

Here, $k : \mathbf{A}(I, T_\alpha(S \multimap T_\beta U)) \to \mathbf{A}(S, T_{\alpha\wedge\beta} U)$ is given by

$$\mathbf{A}(I, T_\alpha(S \multimap T_\beta U)) \to \mathbf{A}(I, S \multimap T_\alpha T_\beta U) \to$$

$$\mathbf{A}(I, S \multimap T_{\alpha\wedge\beta} U) \to \mathbf{A}(S, T_{\alpha\wedge\beta} U),$$

where the first map comes from the strength. Note that box is the inverse of unbox when $\alpha = 2$.

Given $\mathbf{M}, \mathbf{R}$ and $\mathbf{N}$, one can construct a concrete category $\mathbf{A}$ satisfying the required properties, using a generalization of our earlier work on biset enrichment [12]. We give this construction in Section 7.

6.2. Interpretation

We will interpret the typing rules in the category $\mathbf{A}$. The following is the interpretation for types.

$$\begin{array}{lcl} [\![\mathbf{Qubit}]\!] & = & \mathbf{Qubit} \\ [\![A \otimes B]\!] & = & [\![A]\!] \otimes [\![B]\!] \\ [\![\mathbf{Circ}_\alpha(S,U)]\!] & = & p\mathbf{D}^\alpha([\![S]\!], [\![U]\!]) \\ [\![!_\alpha A]\!] & = & p\flat T_\alpha [\![A]\!] \\ [\![A \multimap_\alpha B]\!] & = & [\![A]\!] \multimap T_\alpha [\![B]\!] \end{array}$$

We interpret a typing context Γ as the tensor of all of its objects (denoted by $[\![\Gamma]\!]$). Each valid typing judgment $\Gamma \vdash_\alpha M : A$ will be interpreted as a morphism $[\![M]\!] : [\![\Gamma]\!] \to T_\alpha [\![A]\!]$ in $\mathbf{A}$. The interpretation is defined by induction on the typing rules. Here we show how to define it for a few cases. The remaining cases are defined similarly.

- Case
$$\frac{\Gamma \vdash_\alpha M : !_{\alpha_1}(S \multimap_{\alpha_2} U)}{\Gamma \vdash_\alpha \mathsf{box}_S M : \mathbf{Circ}_{\alpha_1\wedge\alpha_2}(S,U)}$$
We define $[\![\mathsf{box}_S M]\!]$ to be
$$[\![\Gamma]\!] \xrightarrow{[\![M]\!]} T_\alpha p\flat T_{\alpha_1}([\![S]\!] \multimap T_{\alpha_2}[\![U]\!]) \xrightarrow{T_\alpha p\mathsf{box}} T_\alpha p\mathbf{D}^{\alpha_1\wedge\alpha_2}([\![S]\!], [\![U]\!]).$$

- Case
$$\frac{C \in \mathbf{D}^\alpha([\![S]\!], [\![U]\!])}{\Phi \vdash_2 \mathsf{circ}(S, C, U) : \mathbf{Circ}_\alpha(S,U)}$$

Since $C : [\![S]\!] \to [\![U]\!]$ is a morphism in $\mathbf{D}^\alpha$, we define $[\![\Phi \vdash_2 \mathsf{circ}(S, C, U) : \mathbf{Circ}_\alpha(S, U)]\!]$ as the following composition, where $1_C : 1 \to \mathbf{D}^\alpha([\![S]\!], [\![U]\!])$ is such that $1_C(*) = C$ and $\mathsf{discard} = p!_X, !_X \in \mathbf{Set}(X, 1)$.

$$[\![\Phi]\!] \xrightarrow{\mathsf{discard}} p1 \xrightarrow{p1_C} p\mathbf{D}^\alpha([\![S]\!], [\![U]\!]).$$

- Case
$$\frac{\Gamma \vdash_\alpha M : \mathbf{Circ}_2(U, U)}{\Gamma \vdash_\alpha \mathsf{control}_K M : \mathbf{Circ}_2(\underline{K} \otimes U, \underline{K} \otimes U)}$$

By the induction hypothesis, we have $[\![M]\!] : [\![\Gamma]\!] \to T_\alpha p\mathbf{N}([\![U]\!], [\![U]\!])$. Using the function

$$\mathsf{ctrl}_K : \mathbf{N}([\![U]\!], [\![U]\!]) \to \mathbf{N}([\![\underline{K}]\!] \otimes [\![U]\!], [\![\underline{K}]\!] \otimes [\![U]\!]),$$

we define $[\![\mathsf{control}_K M]\!]$ to be the following.

$$[\![\Gamma]\!] \xrightarrow{[\![M]\!]} T_\alpha p\mathbf{N}([\![U]\!], [\![U]\!]) \xrightarrow{T_\alpha p(\mathsf{ctrl}_K)} T_\alpha p\mathbf{N}([\![\underline{K}]\!] \otimes [\![U]\!], [\![\underline{K}]\!] \otimes [\![U]\!])$$

- Case
$$\frac{\begin{array}{c}\Gamma_1 \vdash_\alpha M : \mathbf{Circ}_\gamma(U, S) \quad \gamma > 0 \\ \Gamma_2 \vdash_\beta N : \mathbf{Circ}_2(S, S)\end{array}}{\Gamma_1, \Gamma_2 \vdash_{\alpha \wedge \beta} \mathsf{withComputed}\, MN : \mathbf{Circ}_2(U, U)}$$

Suppose $\alpha = \beta = \gamma = 2$ and $\Gamma_1 \vdash_2 M : \mathbf{Circ}_2(U, S)$. By the induction hypotheses, we have

$$pG_{U,S} \circ [\![M]\!] : [\![\Gamma_1]\!] \to p\mathbf{N}([\![U]\!], [\![S]\!]) \to p\mathbf{R}([\![U]\!], [\![S]\!]),$$

$$[\![N]\!] : [\![\Gamma_2]\!] \to p\mathbf{N}([\![S]\!], [\![S]\!]).$$

Using the function

$$\mathsf{withComputed} : \mathbf{R}([\![U]\!], [\![S]\!]) \times \mathbf{N}([\![S]\!], [\![S]\!]) \to \mathbf{N}([\![U]\!], [\![U]\!]),$$

we define

$[\![\mathsf{withComputed}\, MN]\!]$ to be the following (where m is the mediating map for the strong monoidal functor p):

$$[\![\Gamma_1]\!] \otimes [\![\Gamma_2]\!] \xrightarrow{(pG_{U,S} \circ [\![M]\!]) \otimes [\![N]\!]} p\mathbf{R}([\![U]\!], [\![S]\!]) \otimes p\mathbf{N}([\![S]\!], [\![S]\!])$$

$$\xrightarrow{m} p(\mathbf{R}(\llbracket U \rrbracket, \llbracket S \rrbracket) \times \mathbf{N}(\llbracket S \rrbracket, \llbracket S \rrbracket)) \xrightarrow{p(\mathsf{withComputed})} p\mathbf{N}(\llbracket U \rrbracket, \llbracket U \rrbracket).$$

Suppose $\alpha = \beta = 2$, $\gamma = 1$ and $\Gamma_1 \vdash_2 M : \mathbf{Circ}_1(U, S)$. By the induction hypotheses, we have

$$\llbracket M \rrbracket : \llbracket \Gamma_1 \rrbracket \to p\mathbf{R}(\llbracket U \rrbracket, \llbracket S \rrbracket),$$

$$\llbracket N \rrbracket : \llbracket \Gamma_2 \rrbracket \to p\mathbf{N}(\llbracket S \rrbracket, \llbracket S \rrbracket).$$

We define $\llbracket \mathsf{withComputed}\, MN \rrbracket$ to be the following.

$$\llbracket \Gamma_1 \rrbracket \otimes \llbracket \Gamma_2 \rrbracket \xrightarrow{\llbracket M \rrbracket \otimes \llbracket N \rrbracket} p\mathbf{R}(\llbracket U \rrbracket, \llbracket S \rrbracket) \otimes p\mathbf{N}(\llbracket S \rrbracket, \llbracket S \rrbracket)$$

$$\xrightarrow{m} p(\mathbf{R}(\llbracket U \rrbracket, \llbracket S \rrbracket) \times \mathbf{N}(\llbracket S \rrbracket, \llbracket S \rrbracket)) \xrightarrow{p(\mathsf{withComputed})} p\mathbf{N}(\llbracket U \rrbracket, \llbracket U \rrbracket)$$

The remaining cases (e.g. when $\alpha, \beta \neq 2$) are similar.

- Case

$$\frac{\Phi, \Gamma_1 \vdash_\alpha M : \mathbf{Circ}_\gamma(S, U) \quad \Phi, \Gamma_2 \vdash_\beta N : S}{\Phi, \Gamma_1, \Gamma_2 \vdash_{\alpha \wedge \beta \wedge \gamma} \mathsf{apply}(M, N) : U}$$

By the induction hypotheses, we have

$$\llbracket M \rrbracket : \llbracket \Phi \rrbracket \otimes \llbracket \Gamma_1 \rrbracket \to T_\alpha p\mathbf{D}^\gamma(\llbracket S \rrbracket, \llbracket U \rrbracket)$$

$$\llbracket N \rrbracket : \llbracket \Phi \rrbracket \otimes \llbracket \Gamma_2 \rrbracket \to T_\beta \llbracket S \rrbracket.$$

We therefore define $\llbracket \mathsf{apply}(M, N) \rrbracket$ as the following.

$$\begin{array}{lll}
\llbracket \Phi \rrbracket \otimes \llbracket \Gamma_1 \rrbracket \otimes \llbracket \Gamma_2 \rrbracket & \xrightarrow{\Delta \otimes \llbracket \Gamma_1 \rrbracket \otimes \llbracket \Gamma_2 \rrbracket} & \llbracket \Phi \rrbracket \otimes \llbracket \Phi \rrbracket \otimes \llbracket \Gamma_1 \rrbracket \otimes \llbracket \Gamma_2 \rrbracket \\
& \xrightarrow{\cong} & \llbracket \Phi \rrbracket \otimes \llbracket \Gamma_1 \rrbracket \otimes \llbracket \Phi \rrbracket \otimes \llbracket \Gamma_2 \rrbracket \\
& \xrightarrow{\llbracket M \rrbracket \otimes \llbracket N \rrbracket} & T_\alpha p\mathbf{D}^\gamma(\llbracket S \rrbracket, \llbracket U \rrbracket) \otimes T_\beta \llbracket S \rrbracket \\
& \xrightarrow{t} & T_\alpha(p\mathbf{D}^\gamma(\llbracket S \rrbracket, \llbracket U \rrbracket) \otimes T_\beta \llbracket S \rrbracket) \\
& \xrightarrow{T_\alpha s} & T_\alpha T_\beta(p\mathbf{D}^\gamma(\llbracket S \rrbracket, \llbracket U \rrbracket) \otimes \llbracket S \rrbracket) \\
& \xrightarrow{T_\alpha T_\beta(p\,\mathsf{unbox} \otimes \llbracket S \rrbracket)} & T_\alpha T_\beta(p\flat(\llbracket S \rrbracket \multimap T_\gamma \llbracket U \rrbracket) \otimes \llbracket S \rrbracket) \\
& \xrightarrow{T_\alpha T_\beta(\mathsf{force} \otimes \llbracket S \rrbracket)} & T_\alpha T_\beta((\llbracket S \rrbracket \multimap T_\gamma \llbracket U \rrbracket) \otimes \llbracket S \rrbracket) \\
& \xrightarrow{T_\alpha T_\beta \epsilon} & T_\alpha T_\beta T_\gamma \llbracket U \rrbracket \\
& \xrightarrow{\cong} & T_{\alpha \wedge \beta \wedge \gamma} \llbracket U \rrbracket
\end{array}$$

6.3. Soundness of operational semantics

We now prove that the operational semantics is sound with respect to the categorical model **A**. To do so, we first interpret a well-typed configuration as a morphism in **A**.

Definition 6.2. Suppose $S \vdash_\alpha (C, \Sigma, M) : A; \Sigma'$ is a well-typed configuration, with $C \in \mathbf{D}^\beta([\![S]\!], [\![\Sigma]\!])$, $\Sigma = (\Sigma'', \Sigma')$, $\Sigma'' \vdash_\gamma M : A$ and $\alpha = \beta \wedge \gamma$. We define $[\![(C, \Sigma, M)]\!] : [\![S]\!] \to T_\alpha([\![A]\!] \otimes [\![\Sigma']\!])$ as follows:

$$[\![S]\!] \xrightarrow{C} T_\beta([\![\Sigma]\!]) \xrightarrow{\cong} T_\beta([\![\Sigma'']\!] \otimes [\![\Sigma']\!])$$

$$\xrightarrow{T_\beta([\![M]\!] \otimes [\![\Sigma']\!])} T_\beta(T_\gamma[\![A]\!] \otimes [\![\Sigma']\!]) \to T_\alpha([\![A]\!] \otimes [\![\Sigma']\!]),$$

where the last step uses the strength and the natural isomorphism $T_\beta T_\gamma \cong T_\alpha$.

Theorem 6.3 (Soundness of the evaluation)**.** *If* $S \vdash_\alpha (C, \Sigma, M) : A; \Sigma'$ *and* $(C, \Sigma, M) \Downarrow (C', \Sigma', V)$*, then* $[\![(C, \Sigma, M)]\!] = [\![(C', \Sigma', V)]\!]$.

Proof. See Appendix A. □

We remark that our semantics also satisfies computational adequacy, but in our setting, it is a trivial consequence of soundness. This is because if V is a value of type $\mathbf{Circ}(S, U)$, then V is by definition a circuit, i.e., a morphism of the appropriate category **M**, **R**, or **N**. Its semantics is essentially itself. It immediately follows that $[\![V]\!] = [\![V']\!]$ implies $V = V'$. Since our language is also terminating, adequacy for any closed term of type $\mathbf{Circ}(S, U)$ then follows from soundness.

7. A concrete model for reversing and control

In this section, we will show how to use the categories **N**, **R**, and **M** to construct a category **A** that satisfies the conditions of Definition 6.1. Our construction is based on enriched categories (see Appendix B for some background on enrichment). We first define a category of *trisets*.

Definition 7.1. Write **3** for the diagram $0 \to 1 \to 2$. By *trisets*, we mean objects of the presheaf category $\mathbf{Set}^{\mathbf{3^{op}}}$. Explicitly,

- A triset X is a tuple $(X_0, X_1, X_2, f_1, f_2)$ of three sets X_0, X_1, X_2 and two functions $f_1 : X_1 \to X_0$ and $f_2 : X_2 \to X_1$.

- A morphism from $(X_0, X_1, X_2, f_1, f_2)$ to $(Y_0, Y_1, Y_2, g_1, g_2)$ is a triple (h_0, h_1, h_2) of functions making the following commutative diagram.

$$\begin{array}{ccc} X_2 & \xrightarrow{h_2} & Y_2 \\ \downarrow{\scriptstyle f_2} & & \downarrow{\scriptstyle g_2} \\ X_1 & \xrightarrow{h_1} & Y_1 \\ \downarrow{\scriptstyle f_1} & & \downarrow{\scriptstyle g_1} \\ X_0 & \xrightarrow{h_0} & Y_0 \end{array}$$

Analogously, we write **2** for the diagram $0 \to 1$ and refer to objects in $\mathbf{Set}^{\mathbf{2^{op}}}$ as *bisets*: they are tuples $(X_0, X_1, f : X_1 \to X_0)$, and their morphisms are commutative squares.

Notation and remarks.

- Given a triset X, we write X_0, X_1, X_2 for its 0-, 1-, 2-components.
- We write $\mathrm{Hom}_{\mathbf{Set}^{\mathbf{3^{op}}}}(X, Y)$ for the hom-*set* of morphisms from a triset X to another Y.
- Being a presheaf category, $\mathbf{Set}^{\mathbf{3^{op}}}$ is cartesian closed and hence self-enriched: it has an exponential $X \Rightarrow Y$, which constitutes $\mathbf{Set}^{\mathbf{3^{op}}}(X, Y)$, the hom-*object* as opposed to hom-set of $\mathbf{Set}^{\mathbf{3^{op}}}$. Explicitly, $X \Rightarrow Y$ is a triset given by

$$\begin{array}{lcl} (X \Rightarrow Y)_0 & = & \mathrm{Hom}_{\mathbf{Set}}(X_0, Y_0), \\ (X \Rightarrow Y)_1 & = & \mathrm{Hom}_{\mathbf{Set}^{\mathbf{2^{op}}}}((X_0, X_1, f_1), (Y_0, Y_1, g_1)), \\ (X \Rightarrow Y)_2 & = & \mathrm{Hom}_{\mathbf{Set}^{\mathbf{3^{op}}}}(X, Y), \end{array}$$

  where f_1 and g_1 are the $(0 \to 1)$-components of X and Y.
- Similarly, $\mathbf{Set}^{\mathbf{2^{op}}}$ is self-enriched and we distinguish its hom-objects $\mathbf{Set}^{\mathbf{2^{op}}}(X, Y)$ and hom-sets $\mathrm{Hom}_{\mathbf{Set}^{\mathbf{2^{op}}}}(X, Y)$.
- We often write $\mathcal{V}_3$ for $\mathbf{Set}^{\mathbf{3^{op}}}$, $\mathcal{V}_2$ for $\mathbf{Set}^{\mathbf{2^{op}}}$, and $\mathcal{V}_1$ for $\mathbf{Set}$.
- $\mathcal{V}_2$ and $\mathbf{Set}$, and indeed any categories enriched in them, can be regarded as enriched in $\mathcal{V}_3$, via Δ_1 and $\Delta_1\Delta_0$ in the following definition.

Definition 7.2. We define the following functors.

- $U_0 : \mathbf{Set}^{\mathbf{2^{op}}} \to \mathbf{Set} :: (X_0, X_1, f_1) \mapsto X_0$.
- $\Delta_0 : \mathbf{Set} \to \mathbf{Set}^{\mathbf{2^{op}}} :: X_0 \mapsto (X_0, X_0, \mathrm{id})$.
- $U_1 : \mathbf{Set}^{\mathbf{3^{op}}} \to \mathbf{Set}^{\mathbf{2^{op}}} :: (X_0, X_1, X_2, f_1, f_2) \mapsto (X_0, X_1, f_1)$.
- $\Delta_1 : \mathbf{Set}^{\mathbf{2^{op}}} \to \mathbf{Set}^{\mathbf{3^{op}}} :: (X_0, X_1, f_1) \mapsto (X_0, X_1, X_1, f_1, \mathrm{id})$.

We have adjunctions $U_0 \dashv \Delta_0$ and $U_1 \dashv \Delta_1$, with $U_0\Delta_0 = \mathrm{id}$ and $U_1\Delta_1 = \mathrm{id}$. These give rise to idempotent monads $T_0 = \Delta_1\Delta_0U_0U_1$ and $T_1 = \Delta_1U_1$ in $\mathbf{Set}^{\mathbf{3^{op}}}$, which act on objects as follows.

$$T_1(X_0, X_1, X_2, f_1, f_2) = (X_0, X_1, X_1, f_1, \mathrm{id}),$$
$$T_0(X_0, X_1, X_2, f_1, f_2) = (X_0, X_0, X_0, \mathrm{id}, \mathrm{id}).$$

It is easy to verify that $T_1T_0 \cong T_0 \cong T_0T_1$ via natural isomorphisms $\eta_{T_0} : T_0 \to T_1T_0$ and $T_0\eta : T_0 \to T_0T_1$ (for the unit η of T_1).

These functors can be regarded as $\mathcal{V}_3$-functors by regarding $\mathcal{V}_2$ and $\mathbf{Set}$ as $\mathcal{V}_3$-enriched via Δ_1 and $\Delta_1\Delta_0$. Then U_1 and Δ_1, as $\mathcal{V}_3$-functors, have the following triset morphisms as their (X, X')- and (Y, Y')-components, $U_{1,XX'} : \mathbf{Set}^{\mathbf{3^{op}}}(X, X') \to \Delta_1\mathbf{Set}^{\mathbf{2^{op}}}(U_1X, U_1X')$ and $\Delta_{1,YY'} : \Delta_1\mathbf{Set}^{\mathbf{2^{op}}}(Y, Y') \to \mathbf{Set}^{\mathbf{3^{op}}}(\Delta_1Y, \Delta_1Y')$, for $X, X' \in \mathbf{Set}^{\mathbf{3^{op}}}$ and $Y, Y' \in \mathbf{Set}^{\mathbf{2^{op}}}$.

$$\begin{array}{ccc}
\mathrm{Hom}_{\mathbf{Set}^{\mathbf{3^{op}}}}(X, X') & \xrightarrow{U_{1,XX'}} & \mathrm{Hom}_{\mathbf{Set}^{\mathbf{2^{op}}}}(U_1X, U_1X') \\
\downarrow{\scriptstyle U_{1,XX'}} & & \downarrow{\scriptstyle \mathrm{id}} \\
\mathrm{Hom}_{\mathbf{Set}^{\mathbf{2^{op}}}}(U_1X, U_1X') & \xrightarrow{\mathrm{id}} & \mathrm{Hom}_{\mathbf{Set}^{\mathbf{2^{op}}}}(U_1X, U_1X') \\
\downarrow{\scriptstyle U_{0,XX'}} & & \downarrow{\scriptstyle U_{0,XX'}} \\
\mathrm{Hom}_{\mathbf{Set}}(U_0U_1X, U_0U_1X') & \xrightarrow{\mathrm{id}} & \mathrm{Hom}_{\mathbf{Set}}(U_0U_1X, U_0U_1X')
\end{array}$$

$$\begin{array}{ccc}
\mathrm{Hom}_{\mathbf{Set}^{\mathbf{2^{op}}}}(Y, Y') & \xrightarrow{\Delta_{1,XY}} & \mathrm{Hom}_{\mathbf{Set}^{\mathbf{3^{op}}}}(\Delta_1Y, \Delta_1Y') \\
\downarrow{\scriptstyle \mathrm{id}} & & \downarrow{\scriptstyle U_{1,YY'}} \\
\mathrm{Hom}_{\mathbf{Set}^{\mathbf{2^{op}}}}(Y, Y') & \xrightarrow{\mathrm{id}} & \mathrm{Hom}_{\mathbf{Set}^{\mathbf{2^{op}}}}(Y, Y') \\
\downarrow{\scriptstyle U_{0,YY'}} & & \downarrow{\scriptstyle U_{0,YY'}} \\
\mathrm{Hom}_{\mathbf{Set}}(U_0Y, U_0Y') & \xrightarrow{\mathrm{id}} & \mathrm{Hom}_{\mathbf{Set}}(U_0Y, U_0Y')
\end{array}$$

U_0 and Δ_0 as $\mathcal{V}_3$-functors behave in the obviously analogous way.

Next, we construct a $\mathcal{V}_3$-enriched category, $\mathbf{D}$, that encapsulates all of $\mathbf{N}$, $\mathbf{R}$, $\mathbf{M}$ whereas its underlying category is $\mathbf{N}$. Similarly, we construct a $\mathcal{V}_2$-enriched category, $\mathbf{C}$, that encapsulates $\mathbf{R}$ and $\mathbf{M}$ whereas its underlying category is $\mathbf{R}$. ($\mathbf{M}$ is trivially the third in a series after $\mathbf{D}$ and $\mathbf{C}$: it is $\mathcal{V}_1$-enriched, is its own underlying category, and encapsulates itself.)

Definition 7.3. We define a triset-enriched category $\mathbf{D}$ and a biset-enriched category $\mathbf{C}$ by:

- $\mathrm{obj}(\mathbf{D}) = \mathrm{obj}(\mathbf{C}) = \mathrm{obj}(\mathbf{M})$ (and hence equal to $\mathrm{obj}(\mathbf{R})$ and $\mathrm{obj}(\mathbf{N})$).
- For any $A, B \in \mathbf{M}$, the hom-objects $\mathbf{D}(A,B)$ and $\mathbf{C}(A,B)$ are the following triset and biset, respectively, where $G : \mathbf{N} \to \mathbf{R}$ and $I : \mathbf{R} \to \mathbf{M}$ are the inclusion functors.

$$\begin{array}{ccc}
\mathbf{N}(A,B) & & \\
\downarrow{\scriptstyle G_{AB}} & & \\
\mathbf{R}(A,B) & \qquad & \mathbf{R}(A,B) \\
\downarrow{\scriptstyle I_{AB}} & & \downarrow{\scriptstyle I_{AB}} \\
\mathbf{M}(A,B) & & \mathbf{M}(A,B)
\end{array}$$

We can then apply U_1 to hom-trisets and obtain a $\mathcal{V}_3$-functor $u_1 : \mathbf{D} \to \mathbf{C}$, where we regard $\mathbf{C}$ as enriched in $\mathcal{V}_3$ via Δ_1, that is the identity on objects and that has the following component for each pair $A, B \in \mathbf{D}$:

$$\begin{array}{ccc}
\mathbf{N}(A,B) & \xrightarrow{G_{AB}} & \mathbf{R}(A,B) \\
\downarrow{\scriptstyle G_{AB}} & & \downarrow{\scriptstyle \mathrm{id}} \\
\mathbf{R}(A,B) & \xrightarrow{\mathrm{id}} & \mathbf{R}(A,B) \\
\downarrow{\scriptstyle I_{AB}} & & \downarrow{\scriptstyle I_{AB}} \\
\mathbf{M}(A,B) & \xrightarrow{\mathrm{id}} & \mathbf{M}(A,B)
\end{array}$$

We also apply U_0 and obtain the analogous $\mathcal{V}_3$-functor $u_0 : \mathbf{C} \to \mathbf{M}$, where again we enrich $\mathbf{M}$ in $\mathcal{V}_3$ via $\Delta_1\Delta_0$.

The enriched categories $\mathbf{D}$ and $\mathbf{C}$ inherit the symmetric monoidal structure of $\mathbf{N}$, $\mathbf{R}$, $\mathbf{M}$. These monoidal categories are, however, not closed. We

therefore use enriched Yoneda embeddings of $\mathbf{D}$, $\mathbf{C}$, $\mathbf{M}$ into enriched categories of enriched presheaves.

A key to obtaining the enriched categories of enriched presheaves over $\mathbf{D}$, $\mathbf{C}$, $\mathbf{M}$ is the following lemma. (With a slight abuse of notation, we henceforth write u_1 instead of u_1^{op}.)

Lemma 7.4. *Let $\mathcal{V}_3^{\mathbf{D}^{\mathrm{op}}}$ and $\mathcal{V}_2^{\mathbf{C}^{\mathrm{op}}}$ be the* ordinary *categories of $\mathcal{V}_3$-enriched presheaves $H : \mathbf{D}^{\mathrm{op}} \to \mathcal{V}_3$ and $\mathcal{V}_2$-enriched presheaves $F : \mathbf{C}^{\mathrm{op}} \to \mathcal{V}_2$, respectively. Then there is an* ordinary *functor $\upsilon_1 : \mathcal{V}_3^{\mathbf{D}^{\mathrm{op}}} \to \mathcal{V}_2^{\mathbf{C}^{\mathrm{op}}}$ with the following properties.*

*(**a**) Given any $\mathcal{V}_3$-functor $H : \mathbf{D}^{\mathrm{op}} \to \mathcal{V}_3$, $\upsilon_1 H$ is the unique $\mathcal{V}_3$-functor making the square below commute.*

$$\begin{array}{ccc} \mathbf{D}^{\mathrm{op}} & \xrightarrow{H} & \mathcal{V}_3 \\ \downarrow{\scriptstyle u_1} & & \downarrow{\scriptstyle U_1} \\ \mathbf{C}^{\mathrm{op}} & \xrightarrow{\upsilon_1 H} & \mathcal{V}_2 \end{array}$$

*(**b**) In particular, $\upsilon_1(y_3 A) = y_2 A$ for any $A \in \mathbf{D}$.*

There is also an ordinary *functor $\upsilon_0 : \mathcal{V}_2^{\mathbf{C}^{\mathrm{op}}} \to \mathcal{V}_1^{\mathbf{M}^{\mathrm{op}}}$ satisfying the analogous properties. It follows that $\upsilon_0 \circ \upsilon_1 : \mathcal{V}_3^{\mathbf{D}^{\mathrm{op}}} \to \mathcal{V}_1^{\mathbf{M}^{\mathrm{op}}}$ satisfies the analogous properties.*

Proof. We show the case of υ_1; the case of υ_0 is similar. Given $H : \mathbf{D}^{\mathrm{op}} \to \mathcal{V}_3$, its (A, B)-component H_{AB} is a triset morphism $(H_{AB,0}, H_{AB,1}, H_{AB,2}) : \mathbf{D}(B, A) \to \mathcal{V}_3(HA, HB)$. So define $\upsilon_1 H : \mathbf{C}^{\mathrm{op}} \to \mathcal{V}_2$ with

$$\upsilon_1 H(A) = U_1(HA)$$

and $(\upsilon_1 H)_{AB}$ the biset morphism

$$(H_{AB,0}, H_{AB,1}) : \mathbf{C}(B, A) \to \mathcal{V}_2(U_1(HA), U_1(HB)).$$

(a): Let us first show the commutativity. $(\upsilon_1 H) \circ u_1 = U_1 \circ H$ since $(\upsilon_1 H) \circ u_1(A) = \upsilon_1 H(A) = U_1(HA)$ and since $u_{1,BA}$ and $(\upsilon_1 H)_{AB}$ constitute the left and right halves of the first of the following two diagrams while H_{AB}

and $U_{1,AB}$ do the latter.

$$\begin{array}{ccccc}
\mathbf{N}(B,A) & \xrightarrow{G_{BA}} & \mathbf{R}(B,A) & \xrightarrow{H_{AB,1}} & \mathrm{Hom}_{\mathbf{Set}^{\mathbf{2^{op}}}}(U_1HA, U_1HB) \\
\downarrow{\scriptstyle G_{BA}} & & \downarrow{\scriptstyle \mathrm{id}} & & \downarrow{\scriptstyle \mathrm{id}} \\
\mathbf{R}(B,A) & \xrightarrow{\mathrm{id}} & \mathbf{R}(B,A) & \xrightarrow{H_{AB,1}} & \mathrm{Hom}_{\mathbf{Set}^{\mathbf{2^{op}}}}(U_1HA, U_1HB) \\
\downarrow{\scriptstyle I_{BA}} & & \downarrow{\scriptstyle I_{BA}} & & \downarrow{\scriptstyle U_{0,AB}} \\
\mathbf{M}(B,A) & \xrightarrow{\mathrm{id}} & \mathbf{M}(B,A) & \xrightarrow{H_{AB,0}} & \mathrm{Hom}_{\mathbf{Set}}(U_0U_1HA, U_0U_1HB)
\end{array}$$

$$\begin{array}{ccccc}
\mathbf{N}(B,A) & \xrightarrow{H_{AB,2}} & \mathrm{Hom}_{\mathbf{Set}^{\mathbf{3^{op}}}}(HA, HB) & \xrightarrow{U_{1,AB}} & \mathrm{Hom}_{\mathbf{Set}^{\mathbf{2^{op}}}}(U_1HA, U_1HB) \\
\downarrow{\scriptstyle G_{BA}} & & \downarrow{\scriptstyle U_{1,AB}} & & \downarrow{\scriptstyle \mathrm{id}} \\
\mathbf{R}(B,A) & \xrightarrow{H_{AB,1}} & \mathrm{Hom}_{\mathbf{Set}^{\mathbf{2^{op}}}}(U_1HA, U_1HB) & \xrightarrow{\mathrm{id}} & \mathrm{Hom}_{\mathbf{Set}^{\mathbf{2^{op}}}}(U_1HA, U_1HB) \\
\downarrow{\scriptstyle I_{BA}} & & \downarrow{\scriptstyle U_{0,AB}} & & \downarrow{\scriptstyle U_{0,AB}} \\
\mathbf{M}(B,A) & \xrightarrow{H_{AB,0}} & \mathrm{Hom}_{\mathbf{Set}}(U_0U_1HA, U_0U_1HB) & \xrightarrow{\mathrm{id}} & \mathrm{Hom}_{\mathbf{Set}}(U_0U_1HA, U_0U_1HB)
\end{array}$$

These diagrams give the same triset morphism from the left column to the right, since $H_{AB,1} \circ G_{AB} = U_{1,AB} \circ H_{AB,2}$ (as in the upper left square in the second diagram).

Now the uniqueness is clear: Let K be another functor $K : \mathbf{C}^{\mathrm{op}} \to \mathcal{V}_2$ such that $K \circ u_1 = U_1 \circ H$. Then $KA = K \circ u_1(A) = U_1(HA) = \upsilon_1 H(A)$. Furthermore, $f = K_{AB} \circ u_{1,BA}$ equals the same triset morphism from the left column to the right above, so $K_{AB,0} = f_0 = H_{AB,0}$ and $K_{AB,1} = f_1 = H_{AB,1}$, and since K_{AB} has the form $\Delta_1 g$, we have $K_{AB,2} = K_{AB,1} = H_{AB,1}$, so $K_{AB} = (\upsilon_1 H)_{AB}$. Thus $K = \upsilon_1 H$.

(b): By the uniqueness part of (a), it is enough to show that y_3A and y_2A make the square there commute in place of H and $\upsilon_1 H$, i.e., that $U_1 \circ y_3(A) = y_2 \circ u_1(A) = y_2A$. This holds since, for any object $B \in \mathbf{D}$, we have $(U_1 \circ y_3(A))(B) = U_1\mathbf{D}(B,A) = \mathbf{C}(B,A) = (y_2A)(B)$, and for any $C, D \in \mathbf{D}$, the following diagram commutes.

$$\begin{array}{ccc}
\mathbf{D}(D,C) & \xrightarrow{(y_3A)_{CD}} & \mathcal{V}_3(\mathbf{D}(C,A), \mathbf{D}(D,A)) \\
\downarrow{\scriptstyle u_{1,DC}} & & \downarrow{\scriptstyle U_{1,\mathbf{D}(C,A)\mathbf{D}(D,A)}} \\
\mathbf{C}(D,C) & \xrightarrow{(y_2A)_{CD}} & \mathcal{V}_2(\mathbf{C}(C,A), \mathbf{C}(D,A))
\end{array}$$

□

Lemma 7.4 helps us to regard $\mathcal{V}_3^{\mathbf{D}^{\mathrm{op}}}$ as a $\mathcal{V}_3$-category $\overline{\mathbf{D}}$, by setting a hom-triset $\overline{\mathbf{D}}(H,K)$ to be

$$\mathcal{V}_3^{\mathbf{D}^{\mathrm{op}}}(H,K) \xrightarrow{\upsilon_1} \mathcal{V}_2^{\mathbf{C}^{\mathrm{op}}}(\upsilon_1 H, \upsilon_1 K) \xrightarrow{\upsilon_0} \mathcal{V}_1^{\mathbf{M}^{\mathrm{op}}}(\upsilon_0\upsilon_1 H, \upsilon_0\upsilon_1 K).$$

Similarly for $\mathcal{V}_2^{\mathbf{D}^{\mathrm{op}}}$. Lemma 7.4 then extends to the enriched version.

Lemma 7.5. *There is a $\mathcal{V}_3$-functor $\upsilon_1 : \mathcal{V}_3^{\mathbf{D}^{\mathrm{op}}} \to \mathcal{V}_2^{\mathbf{C}^{\mathrm{op}}}$ and a $\mathcal{V}_2$-functor $\upsilon_0 : \mathcal{V}_2^{\mathbf{C}^{\mathrm{op}}} \to \mathcal{V}_1^{\mathbf{M}^{\mathrm{op}}}$ satisfying the properties* (a) *and* (b) *as in Lemma 7.4.*

Note that $\mathcal{V}_1$- and $\mathcal{V}_2$-categories and functors are enriched in $\mathcal{V}_3$ as well. The $\mathcal{V}_3$-enriched functors U_0, Δ_0, U_1, and Δ_1 among $\mathcal{V}_3$, $\mathcal{V}_2$, $\mathcal{V}_1$ can be lifted by postcomposition to the presheaf categories.

Definition 7.6. We define the following $\mathcal{V}_3$-enriched functors.

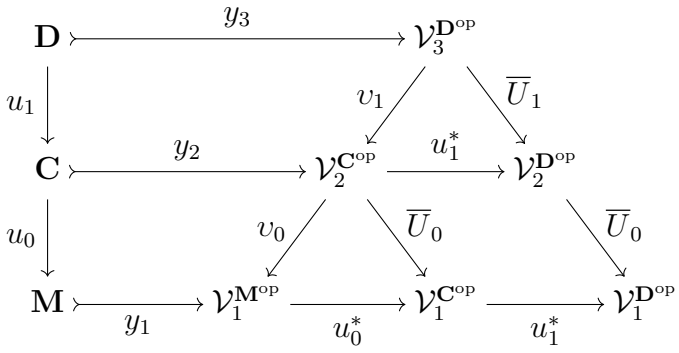
$$\begin{aligned}
\overline{U}_0 &: \mathcal{V}_2^{\mathbf{D}^{\mathrm{op}}} \to \mathcal{V}_1^{\mathbf{D}^{\mathrm{op}}} :: F \mapsto U_0 \circ F,\\
\overline{\Delta}_0 &: \mathcal{V}_1^{\mathbf{D}^{\mathrm{op}}} \to \mathcal{V}_2^{\mathbf{D}^{\mathrm{op}}} :: F \mapsto \Delta_0 \circ F,\\
\overline{U}_1 &: \mathcal{V}_3^{\mathbf{D}^{\mathrm{op}}} \to \mathcal{V}_2^{\mathbf{D}^{\mathrm{op}}} :: F \mapsto U_1 \circ F,\\
\overline{\Delta}_1 &: \mathcal{V}_2^{\mathbf{D}^{\mathrm{op}}} \to \mathcal{V}_3^{\mathbf{D}^{\mathrm{op}}} :: F \mapsto \Delta_1 \circ F.
\end{aligned}$$

Many properties of U_0, etc., are also lifted. In particular, we will use the property that $\overline{U}_0\overline{\Delta}_0 = \mathrm{id}$ and $\overline{U}_1\overline{\Delta}_1 = \mathrm{id}$.

Definition 7.7. We define the following $\mathcal{V}_3$-functors by precomposition. (We abuse notation and write u_1^* for two $\mathcal{V}_3$-functors.)

$$\begin{aligned}
u_1^* &: \mathcal{V}_2^{\mathbf{C}^{\mathrm{op}}} \to \mathcal{V}_2^{\mathbf{D}^{\mathrm{op}}} :: F \mapsto F \circ u_1,\\
u_0^* &: \mathcal{V}_1^{\mathbf{M}^{\mathrm{op}}} \to \mathcal{V}_1^{\mathbf{C}^{\mathrm{op}}} :: G \mapsto G \circ u_0,\\
u_1^* &: \mathcal{V}_1^{\mathbf{C}^{\mathrm{op}}} \to \mathcal{V}_1^{\mathbf{D}^{\mathrm{op}}} :: H \mapsto H \circ u_1.
\end{aligned}$$

Lemma 7.8. *The following diagram of $\mathcal{V}_3$-functors commutes, where the y_i's are enriched Yoneda embeddings.*

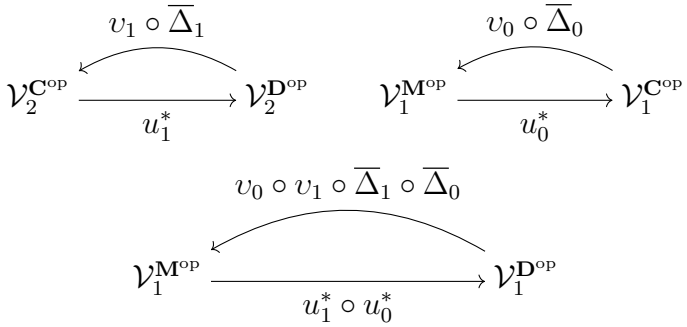


(5)

Proof. The trapezoids on the left side commute by Lemma 7.5(b). The triangles commute by the commutativity part of Lemma 7.5(a). The parallelogram commutes since precomposition $- \circ u_1$ and postcomposition $U_0 \circ -$ commute by associativity $U_0 \circ (- \circ u_1) = (U_0 \circ -) \circ u_1$. □

Moreover, the bases of the (three) triangles in (5) are isomorphisms.

Lemma 7.9. *The following opposite pairs of $\mathcal{V}_3$-functors are mutually inverse, making $\mathcal{V}_2^{\mathbf{C}^{\mathrm{op}}} \cong \mathcal{V}_2^{\mathbf{D}^{\mathrm{op}}}$ and $\mathcal{V}_1^{\mathbf{M}^{\mathrm{op}}} \cong \mathcal{V}_1^{\mathbf{C}^{\mathrm{op}}} \cong \mathcal{V}_1^{\mathbf{D}^{\mathrm{op}}}$.*



The latter two isomorphisms entail that $u_1^* : \mathcal{V}_1^{\mathbf{C}^{\mathrm{op}}} \to \mathcal{V}_1^{\mathbf{D}^{\mathrm{op}}}$ is also an isomorphism.

Proof. We show the case of $\mathcal{V}_2^{\mathbf{C}^{\mathrm{op}}} \cong \mathcal{V}_2^{\mathbf{D}^{\mathrm{op}}}$; the other cases are similar. On the one hand, the upper triangle in (5) implies $u_1^* \circ \upsilon_1 \circ \overline{\Delta}_1 = \overline{U}_1 \circ \overline{\Delta}_1 = \mathrm{id}$. On the other, we have $\upsilon_1 \circ \overline{\Delta}_1 \circ u_1^* = \mathrm{id}$ because, for any $F : \mathbf{C}^{\mathrm{op}} \to \mathcal{V}_2$, $U_1 \circ \Delta_1 \circ F \circ u_1 = F \circ u_1$ implies $\upsilon_1 \circ \overline{\Delta}_1 \circ u_1^*(F) = \upsilon_1(\Delta_1 \circ F \circ u_1) = F$ by the uniqueness part of Lemma 7.5(a). □

To obtain an instance of Definition 6.1, enrichment is not needed. We therefore take the underlying categories and functors of the ones we have discussed, using the same notation as the enriched ones for the underlying

ones henceforth. The diagram (5) then boils down to

$$\begin{array}{ccc} \mathbf{N} & \xrightarrow{\quad y_3 \quad} & \mathcal{V}_3^{\mathbf{D}^{\mathrm{op}}} \\ {\scriptstyle G}\downarrow & & \downarrow{\scriptstyle \overline{U}_1} \\ \mathbf{R} & \xrightarrow{\quad y_2' \quad} & \mathcal{V}_2^{\mathbf{D}^{\mathrm{op}}} \\ {\scriptstyle I}\downarrow & & \downarrow{\scriptstyle \overline{U}_0} \\ \mathbf{M} & \xrightarrow[\quad y_1' \quad]{} & \mathcal{V}_1^{\mathbf{D}^{\mathrm{op}}} \end{array} \tag{6}$$

where y_2' and y_1' are enriched Yoneda embeddings followed by isomorphisms. We can then use Day's convolution [7] to define tensor products so that $\mathcal{V}_3^{\mathbf{D}^{\mathrm{op}}}$, $\mathcal{V}_2^{\mathbf{D}^{\mathrm{op}}}$, $\mathcal{V}_1^{\mathbf{D}^{\mathrm{op}}}$ are symmetric monoidal closed and all the functors in (6) are strong monoidal. $\mathcal{V}_3^{\mathbf{D}^{\mathrm{op}}}$, $\mathcal{V}_2^{\mathbf{D}^{\mathrm{op}}}$, $\mathcal{V}_1^{\mathbf{D}^{\mathrm{op}}}$ are cocomplete as well.

The adjunctions $U_0 \dashv \Delta_0$ and $U_1 \dashv \Delta_1$ are lifted to $\overline{U}_0 \dashv \overline{\Delta}_0$ and $\overline{U}_1 \dashv \overline{\Delta}_1$, giving rise to idempotent monads

$$\begin{aligned} \overline{T}_0 &= \overline{\Delta}_1\overline{\Delta}_0\overline{U}_0\overline{U}_1 : \overline{\mathbf{D}} \to \overline{\mathbf{D}} :: F \mapsto T_0 \circ F, \\ \overline{T}_1 &= \overline{\Delta}_1\overline{U}_1 : \overline{\mathbf{D}} \to \overline{\mathbf{D}} :: F \mapsto T_1 \circ F \end{aligned}$$

in $\overline{\mathbf{D}} = \mathcal{V}_3^{\mathbf{D}^{\mathrm{op}}}$. The idempotence indeed means that $\mathcal{V}_2^{\mathbf{D}^{\mathrm{op}}}$ and $\mathcal{V}_1^{\mathbf{D}^{\mathrm{op}}}$ are the Kleisli categories of $\overline{T}_1$ and $\overline{T}_0$, respectively. Therefore, as we will show, (6) essentially serves as (4) of Definition 6.1.

There is one tiny (and evil) detail, however: $\mathcal{V}_2^{\mathbf{D}^{\mathrm{op}}}$ and $\mathcal{V}_1^{\mathbf{D}^{\mathrm{op}}}$ as Kleisli categories are not represented using the same objects as $\mathcal{V}_3^{\mathbf{D}^{\mathrm{op}}}$; so $\overline{U}_1$ and $\overline{U}_0$ are not identities on objects. Nevertheless, we have

Fact 7.10. *In the following diagram expanding* (6)*, let* $Kl_{\overline{T}_1}(\overline{\mathbf{D}})$ *and* $Kl_{\overline{T}_0}(\overline{\mathbf{D}})$ *be the Kleisli categories represented by the same objects as* $\overline{\mathbf{D}}$ *and Kleisli morphisms. Let* E_1 *and* E_0 *be the identity-on-objects functors as in Definition 6.1*(e) *(with* $\overline{\mathbf{D}}$ *in place of* $\mathbf{A}$*). And let* e_2 *be the functor that maps* X *to* $\overline{U}_1X$ *and whose* (X, Y)*-components are the isomorphisms*

$$e_{2,XY} : \mathrm{Hom}_{\overline{\mathbf{D}}}(X, \overline{\Delta}_1\overline{U}_1Y) \to \mathrm{Hom}_{\mathcal{V}_2^{\mathbf{D}^{\mathrm{op}}}}(\overline{U}_1X, \overline{U}_1Y)$$

of the adjunction $\overline{U}_1 \dashv \overline{\Delta}_1$*; and similarly for* e_1*. Then* e_2 *and* e_1 *are equiva-*

lences of categories, and the diagram commutes.

$$\begin{array}{ccccc}
\mathbf{N} & \xrightarrow{y_3} & \mathcal{V}_3^{\mathbf{D}^{\mathrm{op}}} & = & \overline{\mathbf{D}} \\
{\scriptstyle G}\downarrow & & \downarrow{\scriptstyle \overline{U}_1} & & \downarrow{\scriptstyle E_1} \\
\mathbf{R} & \xrightarrow{y_2'} & \mathcal{V}_2^{\mathbf{D}^{\mathrm{op}}} & \xleftarrow{e_2} & Kl_{\overline{T}_1}(\overline{\mathbf{D}}) \\
{\scriptstyle I}\downarrow & & \downarrow{\scriptstyle \overline{U}_0} & & \downarrow{\scriptstyle E_0} \\
\mathbf{M} & \xrightarrow[y_1']{} & \mathcal{V}_1^{\mathbf{D}^{\mathrm{op}}} & \xleftarrow[e_1]{} & Kl_{\overline{T}_0}(\overline{\mathbf{D}})
\end{array}$$

We can therefore use the inverses of $e_{2,XY}$ and $e_{1,XY}$ to "divert" the Yoneda embeddings y_2' and y_1' to $Kl_{\overline{T}_1}(\overline{\mathbf{D}})$ and $Kl_{\overline{T}_0}(\overline{\mathbf{D}})$. Explicitly, define $\phi_2 : \mathbf{R} \to Kl_{\overline{T}_1}(\overline{\mathbf{D}})$ by $\phi_2(A) = y_3(A)$ and $\phi_{2,AB} = e^{-1}_{2,y_3(A)y_3(B)} \circ y'_{2,AB}$. Then it is essentially the Yoneda embedding, as $e_2 \circ \phi_2 = y_2'$. Similarly for $\phi_1 : \mathbf{M} \to Kl_{\overline{T}_0}(\overline{\mathbf{D}})$. And they make the following diagram commute, with $\phi_3 = y_3$.

$$\begin{array}{ccc}
\mathbf{N} & \xrightarrow{\phi_3} & \overline{\mathbf{D}} \\
{\scriptstyle G}\downarrow & & \downarrow{\scriptstyle E_1} \\
\mathbf{R} & \xrightarrow{\phi_2} & Kl_{\overline{T}_1}(\overline{\mathbf{D}}) \\
{\scriptstyle I}\downarrow & & \downarrow{\scriptstyle E_0} \\
\mathbf{M} & \xrightarrow[\phi_1]{} & Kl_{\overline{T}_0}(\overline{\mathbf{D}})
\end{array} \tag{7}$$

We are now ready to state the following main theorem.

Theorem 7.11. *The category $\mathbf{A} = \overline{\mathbf{D}}$, with the diagram (7), is a model for Proto-Quipper with reversing and control in the sense of Definition 6.1.*

Proof. (a), (b), (e), and the functors being strong monoidal have already been discussed for the equivalent case of (6). Here is (d) for (6): The idempotence and natural isomorphisms $\eta_{\overline{T}_0} : \overline{T}_0 \to \overline{T}_1\overline{T}_0$ and $\overline{T}_0\eta : \overline{T}_0 \to \overline{T}_0\overline{T}_1$ (for the unit η of $\overline{T}_1$) carry over from T_0 and T_1 straightforwardly. The proof of commutative strength is similar to the one in [11].

(c): We define $p(X) = \sum_{x \in X} I : \mathbf{Set} \to \overline{\mathbf{D}}$ and $\flat(A) = \overline{\mathbf{D}}(I, A) : \overline{\mathbf{D}} \to \mathbf{Set}$. (We write $\overline{\mathbf{D}}(I, A)$, etc., for hom-sets.) We then have $p \dashv \flat$ because

$$\begin{aligned}\overline{\mathbf{D}}(pX, B) &= \mathcal{V}_3(1, \overline{\mathbf{D}}(pX, B)) \cong \mathcal{V}_3(1, \overline{\mathbf{D}}(\sum_{x \in X} I, B)) \\ &\cong \mathcal{V}_3(1, \prod_{x \in X} \overline{\mathbf{D}}(I, B)) \cong \prod_{x \in X} \mathcal{V}_3(1, \overline{\mathbf{D}}(I, B)) \\ &\cong \mathbf{Set}(X, \overline{\mathbf{D}}(I, B)) = \mathbf{Set}(X, \flat(B)).\end{aligned}$$

□

8. Conclusion

In this paper, we showed how to extend Proto-Quipper with reversing, control, and the with-computed operation. Our language is parameterized by three categories $\mathbf{M}$, $\mathbf{R}$, and $\mathbf{N}$, which correspond to general quantum circuits, reversible circuits, and controllable circuits, respectively. We defined a type system that uses modalities to distinguish the different types of circuits. We provided an operational and a denotational semantics for the language; the latter takes the form of an abstract categorical model in which our modalities are represented by monads. We proved that the operational semantics is sound with respect to the categorical model. Lastly, we gave a construction of a categorical model for our abstract categorical semantics.

In Figure 4, we give an overview of the Proto-Quipper language family tree. Proto-Quipper is intended to formalize various language features that are available in Quipper [17]. The very first version of Proto-Quipper appeared in [27], where a type system with subtyping and a small step operational semantics were given. In [26], a version of Proto-Quipper (Proto-Quipper-M) is proposed, which came with a big-step operational semantics and categorical semantics. Later Proto-Quipper-M was extended with dependent types (Proto-Quipper-D [14]), dynamic lifting (Proto-Quipper-Dyn [12], Proto-Quipper-L [22] and Proto-Quipper-K [5]) and with index annotations to track resource usage (Proto-Quipper-RA [6] and Proto-Quipper-C [28]). We note that after the present work, the name Proto-Quipper-C (C for closure) was also used for the type system in [28]. It differs from the Proto-Quipper-C of this paper in that the notion of index used there is used for resource estimation, while ours is for tracking reversibility and controllability.

There are many possible directions for future work. For example, in this paper, we only considered the modalities of reversibility and controllability. Our construction of the type system and its categorical semantics can be

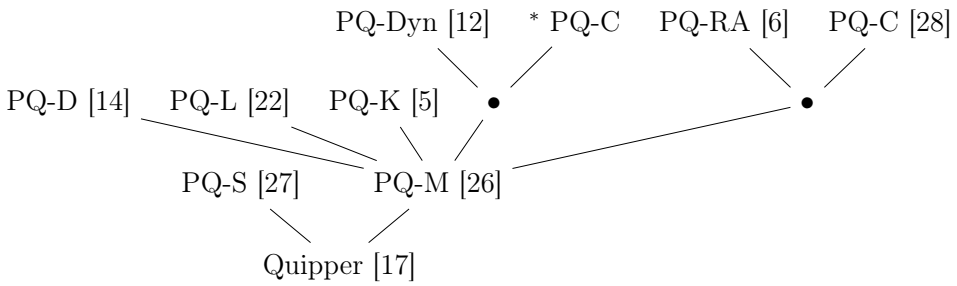


Figure 4: A genealogical tree for the Proto-Quipper language family. This paper's contribution is marked "*".

further generalized from a three-element set to a poset of modalities. Therefore it may be worthwhile to investigate combining our type system with the notion of modality in Proto-Quipper-Dyn [12]. Introducing additional modalities might also be useful for characterizing additional properties of gates. Although we have made a prototype implementation that includes a type inference algorithm, we did not formally study its properties. Another challenge for future work is to combine the modalities of this paper with the dependent types of Proto-Quipper-D [14]. Although our software implementation does support both modalities and dependent types, we have not considered a formal semantics for combining them. Similarly, we do not know the exact relationship between Proto-Quipper-RA [6, 28] and our type system.

Acknowledgements

We thank the referees for their thoughtful comments. This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and by the Air Force Office of Scientific Research under Award No. FA9550-21-1-0041.

References

[1] C. H. Bennett, Logical reversibility of computation, IBM Journal of Research and Development 17 (6) (1973) 525–532. doi:10.1147/rd.176.0525.

[2] B. Bichsel, M. Baader, T. Gehr, M. Vechev, Silq: A high-level quantum language with safe uncomputation and intuitive semantics, in: Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2020, Association for Computing Machinery, 2020, pp. 286–300. doi:10.1145/3385412.3386007.

[3] F. Borceux, Handbook of Categorical Algebra, Volume 2: Categories and Structures, Cambridge University Press, 1994.

[4] A. Colledan, U. Dal Lago, On dynamic lifting and effect typing in circuit description languages (extended version), available from arXiv:2202.07636 (2022).

[5] A. Colledan, U. Dal Lago, On Dynamic Lifting and Effect Typing in Circuit Description Languages, in: D. Kesner, P.-M. Pédrot (Eds.), 28th International Conference on Types for Proofs and Programs (TYPES 2022), Vol. 269 of Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2023, pp. 3:1–3:21. doi:10.4230/LIPIcs.TYPES.2022.3.

[6] A. Colledan, U. Dal Lago, Flexible type-based resource estimation in quantum circuit description languages, Proc. ACM Program. Lang. 9 (POPL). doi:10.1145/3704883.

[7] B. Day, On closed categories of functors, in: Reports of the Midwest Category Seminar IV, Vol. 137 of Springer Lecture Notes in Mathematics, 1970, pp. 1–38.

[8] T. G. Draper, Addition on a quantum computer, available from arXiv:quant-ph/0008033 (2000).

[9] P. Fu, Implementation of Proto-Quipper, available from https://gitlab.com/frank-peng-fu/dpq-remake (2025).

[10] P. Fu, K. Kishida, N. J. Ross, P. Selinger, A tutorial introduction to quantum circuit programming in dependently typed Proto-Quipper, in: Proceedings of the 12th International Conference on Reversible Computation, RC 2020, Oslo, Vol. 12227 of Lecture Notes in Computer Science, Springer, 2020, pp. 153–168, also available from arXiv:2005.08396. doi:10.1007/978-3-030-52482-1_9.

[11] P. Fu, K. Kishida, N. J. Ross, P. Selinger, A biset-enriched categorical model for Proto-Quipper with dynamic lifting, in: Proceedings of the 19th International Conference on Quantum Physics and Logic, QPL 2022, Oxford, Vol. 394 of Electronic Proceedings in Theoretical Computer Science, 2023, pp. 302–342. doi:10.4204/EPTCS.394.16.

[12] P. Fu, K. Kishida, N. J. Ross, P. Selinger, Proto-Quipper with dynamic lifting, in: Proceedings of the 50th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2023, Boston, Vol. 7 of Proceedings of the ACM on Programming Languages, 2023, pp. 309–334, also available from arXiv:2204.13041. doi:10.1145/3571204.

[13] P. Fu, K. Kishida, N. J. Ross, P. Selinger, Proto-Quipper with reversing and control, Electronic Proceedings in Theoretical Computer Science 426 (2025) 1–22. doi:10.4204/eptcs.426.1.

[14] P. Fu, K. Kishida, P. Selinger, Linear dependent type theory for quantum programming languages, in: Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2020, Saarbrücken, Germany, 2020, pp. 440–453, also available from arXiv:2004.13472. doi:10.1145/3373718.3394765.

[15] Google, Cirq Software Reference, https://quantumai.google/reference/python/cirq/ControlledOperation (Accessed July 29, 2026).

[16] A. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, B. Valiron, An introduction to quantum programming in Quipper, in: Proceedings of the 5th International Conference on Reversible Computation, RC 2013, Victoria, British Columbia, Vol. 7948 of Lecture Notes in Computer Science, Springer, 2013, pp. 110–124, also available from arXiv:1304.5485. doi:10.1007/978-3-642-38986-3_10.

[17] A. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, B. Valiron, Quipper: a scalable quantum programming language, in: Proceedings of the 34th Annual ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2013, Seattle, Vol. 48(6) of ACM SIGPLAN Notices, 2013, pp. 333–342, also available from arXiv:1304.3390. doi:10.1145/2499370.2462177.

[18] T. Häner, D. S. Steiger, K. Svore, M. Troyer, A software methodology for compiling quantum programs, Quantum Science and Technology 3 (2) (2018) 020501, also available from arXiv:1604.01401. doi:10.1088/2058-9565/aaa5cc.

[19] IBM, IBM quantum documentation, QuantumCircuit class, https://docs.quantum.ibm.com/api/qiskit/qiskit.circuit.QuantumCircuit#control (Accessed July 29, 2026).

[20] C. Jones, Novel constructions for the fault-tolerant Toffoli gate, Phys. Rev. A 87 (2013) 022328.

[21] G. M. Kelly, Basic concepts of enriched category theory, Vol. 64 of Lecture Notes of the London Mathematical Society, Cambridge University Press, 1982.

[22] D. Lee, V. Perrelle, B. Valiron, Z. Xu, Concrete categorical model of a quantum circuit description language with measurement, in: M. Bojanczyk, C. Chekuri (Eds.), 41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021, Vol. 213 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pp. 51:1–20, also available from arXiv:2110.02691. doi:10.4230/LIPIcs.FSTTCS.2021.51.

[23] B. Lindenhovius, M. Mislove, V. Zamdzhiev, Enriching a linear/nonlinear lambda calculus: A programming language for string diagrams, in: Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '18, Association for Computing Machinery, 2018, pp. 659–668, also available from arXiv:1804.09822. doi:10.1145/3209108.3209196.

[24] A. Paradis, B. Bichsel, S. Steffen, M. Vechev, Unqomp: synthesizing uncomputation in quantum circuits, in: Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2021, Association for Computing Machinery, 2021, p. 222–236. doi:10.1145/3453483.3454040.

[25] J. Paykin, R. Rand, S. Zdancewic, QWIRE: a core language for quantum circuits, in: Proceedings of the 44th ACM SIGPLAN Symposium

on Principles of Programming Languages, Vol. 52 of ACM SIGPLAN Notices, ACM, 2017, pp. 846–858. doi:10.1145/3009837.3009894.

[26] F. Rios, P. Selinger, A categorical model for a quantum circuit description language. Extended abstract, in: Proceedings of the 14th International Conference on Quantum Physics and Logic, QPL 2017, Nijmegen, Vol. 266 of Electronic Proceedings in Theoretical Computer Science, 2018, pp. 164–178. doi:10.4204/EPTCS.266.11.

[27] N. J. Ross, Algebraic and logical methods in quantum computation, Ph.D. thesis, Dalhousie University, Department of Mathematics and Statistics, available from arXiv:1510.02198 (2015).

[28] K. Sakayori, A. Colledan, U. Dal Lago, On circuit description languages, indexed monads, and resource analysis, Proc. ACM Program. Lang. 10 (POPL). doi:10.1145/3776666.

[29] P. Selinger, Dagger compact closed categories and completely positive maps, in: Proceedings of the 3rd International Workshop on Quantum Programming Languages, QPL 2005, Chicago, Vol. 170 of Electronic Notes in Theoretical Computer Science, Elsevier, 2007, pp. 139–163. doi:10.1016/j.entcs.2006.12.018.

[30] P. Selinger, Quantum circuits of T-depth one, Physical Review A 87 (2013) 042302 (4 pages), also available from arXiv:1210.0974. doi:10.1103/PhysRevA.87.042302.

[31] R. Sharma, S. Achour, Optimizing ancilla-based quantum circuits with spare, Proc. ACM Program. Lang. 9 (PLDI). doi:10.1145/3729253.

[32] D. S. Steiger, T. Häner, M. Troyer, ProjectQ: an open source software framework for quantum computing, Quantum 2 (2018) 49, also available from arXiv:1612.08091. doi:10.22331/q-2018-01-31-49.

Appendix A. Proof of soundness

We write $\delta : \mathbf{A}(pX, Y) \to \mathbf{Set}(X, \flat Y)$ for the bijection that arises from the adjunction $p \dashv \flat$.

Theorem A.1 (Soundness of the evaluation)**.** *If* $S \vdash_\alpha (C, \Sigma, M) : A; \Sigma'$ *and* $(C, \Sigma, M) \Downarrow (C', \Sigma', V)$*, then* $[\![(C, \Sigma, M)]\!] = [\![(C', \Sigma', V)]\!]$.

The proof is by induction on $(C, \Sigma, M) \Downarrow (C', \Sigma', V)$. We only give the proof for the rules *box*, *apply*, *ctrl*, and *wc*, as the remaining rules are routine.

- Case
$$\frac{\begin{array}{c}(C, \Sigma, M) \Downarrow (C', \Sigma', \mathsf{lift}\ M') \\ \mathrm{gen}(S) = (a, \Sigma'') \\ ([\![a]\!]^\dagger, \Sigma'', M'\ a) \Downarrow (D, \Sigma''', b) \\ \mathrm{ungen}(b, \Sigma''') = U\end{array}}{(C, \Sigma, \mathsf{box}_S M) \Downarrow (C', \Sigma', \mathsf{circ}(S, [\![b]\!] \circ D, U))}\ box$$

  Suppose $S' \vdash_2 (C, \Sigma, \mathsf{box}_S M) : \mathbf{Circ}_0(S, U); \Sigma'_2$. This implies that $C \in \mathbf{N}([\![S]\!], [\![\Sigma'_1]\!] \otimes [\![\Sigma'_2]\!])$ and $\Sigma'_1 \vdash_2 \mathsf{box}_S M : \mathbf{Circ}_0(S, U)$. Thus $\Sigma'_1 \vdash_2 M : !_0(S \multimap_2 U)$, $C' \in \mathbf{N}([\![S]\!], [\![\Sigma'_2]\!])$, and $\vdash_0 M' : S \multimap_2 U$. By the induction hypotheses, $[\![(C, \Sigma, M)]\!] = [\![(C', \Sigma', \mathsf{lift}\, M')]\!]$ and $[\![([\![a]\!]^\dagger, \Sigma'', M'a)]\!] = [\![(D, \Sigma''', b)]\!]$. Thus we have

$$[\![S']\!] \xrightarrow{C} [\![\Sigma'_1]\!] \otimes [\![\Sigma'_2]\!] \xrightarrow{[\![M]\!] \otimes [\![\Sigma'_2]\!]} p\flat T_0([\![S]\!] \multimap [\![U]\!]) \otimes [\![\Sigma'_2]\!]$$

$$\overset{IH1}{=} [\![S']\!] \xrightarrow{C'} I \otimes [\![\Sigma'_2]\!] \xrightarrow{p\delta[\![M']\!] \otimes [\![\Sigma'_2]\!]} p\flat T_0([\![S]\!] \multimap [\![U]\!]) \otimes [\![\Sigma'_2]\!]$$

  and

$$\begin{aligned} I \otimes [\![S]\!] &\xrightarrow{I \otimes [\![a]\!]^\dagger} I \otimes [\![\Sigma'']\!] \xrightarrow{[\![M']\!] \otimes [\![a]\!]} T_0([\![S]\!] \multimap [\![U]\!]) \otimes [\![S]\!] \\ &\xrightarrow{s} T_0([\![S]\!] \multimap [\![U]\!] \otimes [\![S]\!]) \xrightarrow{T_0 \epsilon} T_0[\![U]\!] \end{aligned}$$

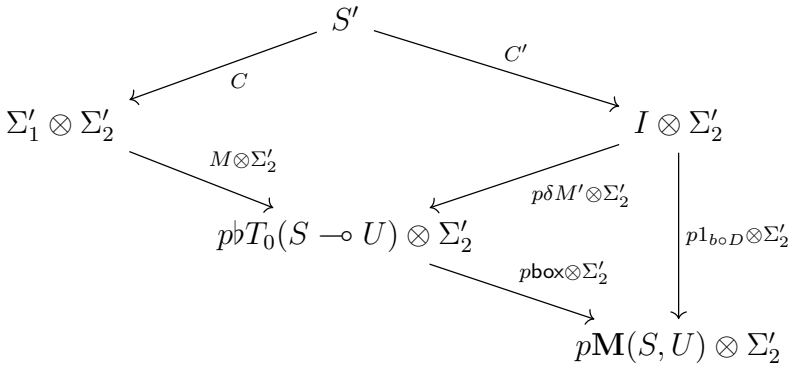
  =

$$\begin{aligned} I \otimes [\![S]\!] &\xrightarrow{[\![M']\!] \otimes [\![S]\!]} T_0([\![S]\!] \multimap [\![U]\!]) \otimes [\![S]\!] \\ &\xrightarrow{s} T_0([\![S]\!] \multimap [\![U]\!] \otimes [\![S]\!]) \xrightarrow{T_0 \epsilon} T_0[\![U]\!] \end{aligned}$$

$\overset{IH2}{=}$

$$[\![S]\!] \xrightarrow{D} T_0[\![\Sigma''']\!] \xrightarrow{T_0[\![b]\!]} T_0[\![U]\!].$$

To prove $[\![(C, \Sigma, \mathsf{box}_S M)]\!] = [\![(C', \Sigma', \mathsf{circ}(S, [\![b]\!] \circ D, U))]\!]$, we use the following commutative diagram. We remove all the semantic brackets for space.



The outer left path corresponds to $[\![(C, \Sigma, \mathsf{box}_S M)]\!]$ and the outer right path corresponds to $[\![(C', \Sigma', \mathsf{circ}(S, [\![b]\!] \circ D, U))]\!]$. The square commutes by *IH1*. The triangle commutes by the definition of box and *IH2*.

- Case

$$\frac{\begin{array}{c}(C_1, \Sigma_1, M) \Downarrow (C_2, \Sigma_2, \mathsf{circ}(S, D, U)) \\ (C_2, \Sigma_2, N) \Downarrow (C_3, \Sigma_3, a) \\ \mathrm{gen}(U) = (b, \Sigma_4) \\ (C', \Sigma_5) = \mathrm{append}(C_3, \Sigma_3, a, D, b, \Sigma_4)\end{array}}{(C_1, \Sigma_1, \mathsf{apply}(M, N)) \Downarrow (C', \Sigma_5, b)}\ \textit{apply}$$

Suppose $S' \vdash_0 (C_1, \Sigma_1, \mathsf{apply}(M, N)) : U; \Sigma'$, where $\Sigma_1 = (\Sigma'', \Sigma')$, $C_1 \in \mathbf{R}([\![S]\!], [\![\Sigma'']\!] \otimes [\![\Sigma']\!])$, $\Sigma'' \vdash_0 \mathsf{apply}(M, N) : U$, $\Sigma_1'' \vdash_0 M : \mathbf{Circ}_1(S, U)$ and $\Sigma_2'' \vdash_1 N : S$ and $\Sigma'' = (\Sigma_1'', \Sigma_2'')$, $\Sigma_2 = (\Sigma_2'', \Sigma')$, $\Sigma_3 = (\Sigma_2''', \Sigma')$. By the induction hypotheses, we know that $[\![(C_1, \Sigma_1, M)]\!] = [\![(C_2, \Sigma_2, \mathsf{circ}(S, D, U))]\!]$

and $[\![(C_2, \Sigma_2, N)]\!] = [\![(C_3, \Sigma_3, a)]\!]$. Thus

$$\begin{aligned}
[\![S']\!] &\xrightarrow{C_1} T_1([\![\Sigma_1'']\!] \otimes [\![\Sigma_2'']\!] \otimes [\![\Sigma']\!]) \\
&\xrightarrow{T_1([\![M]\!] \otimes [\![\Sigma_2'']\!] \otimes [\![\Sigma']\!])} T_1(T_0 p\mathbf{N}([\![S]\!], [\![U]\!]) \otimes [\![\Sigma_2'']\!] \otimes [\![\Sigma']\!]) \\
&\xrightarrow{T_1 s} T_1 T_0(p\mathbf{N}([\![S]\!], [\![U]\!]) \otimes [\![\Sigma_2'']\!] \otimes [\![\Sigma']\!]) \\
&\xrightarrow{\cong} T_0(p\mathbf{N}([\![S]\!], [\![U]\!]) \otimes [\![\Sigma_2'']\!] \otimes [\![\Sigma']\!])
\end{aligned}$$

$\overset{IH1}{=}$

$$[\![S']\!] \xrightarrow{C_2} T_0([\![\Sigma_2'']\!] \otimes [\![\Sigma']\!]) \xrightarrow{T_0(p1_D \otimes [\![\Sigma_2'']\!] \otimes [\![\Sigma']\!])} T_0(p\mathbf{N}([\![S]\!], [\![U]\!]) \otimes [\![\Sigma_2'']\!] \otimes [\![\Sigma']\!])$$

and

$$\begin{aligned}
[\![S']\!] &\xrightarrow{C_2} T_0([\![\Sigma_2'']\!] \otimes [\![\Sigma']\!]) \xrightarrow{T_0([\![N]\!] \otimes [\![\Sigma']\!])} T_0(T_1[\![S]\!] \otimes [\![\Sigma']\!]) \\
&\xrightarrow{T_0 s} T_0 T_1([\![S]\!] \otimes [\![\Sigma']\!]) \xrightarrow{\cong} T_0([\![S]\!] \otimes [\![\Sigma']\!])
\end{aligned}$$

$\overset{IH2}{=}$

$$[\![S']\!] \xrightarrow{C_3} T_0([\![\Sigma_2'''']\!] \otimes [\![\Sigma']\!]) \xrightarrow{T_0([\![a]\!] \otimes [\![\Sigma']\!])} T_0([\![S]\!] \otimes [\![\Sigma']\!]).$$

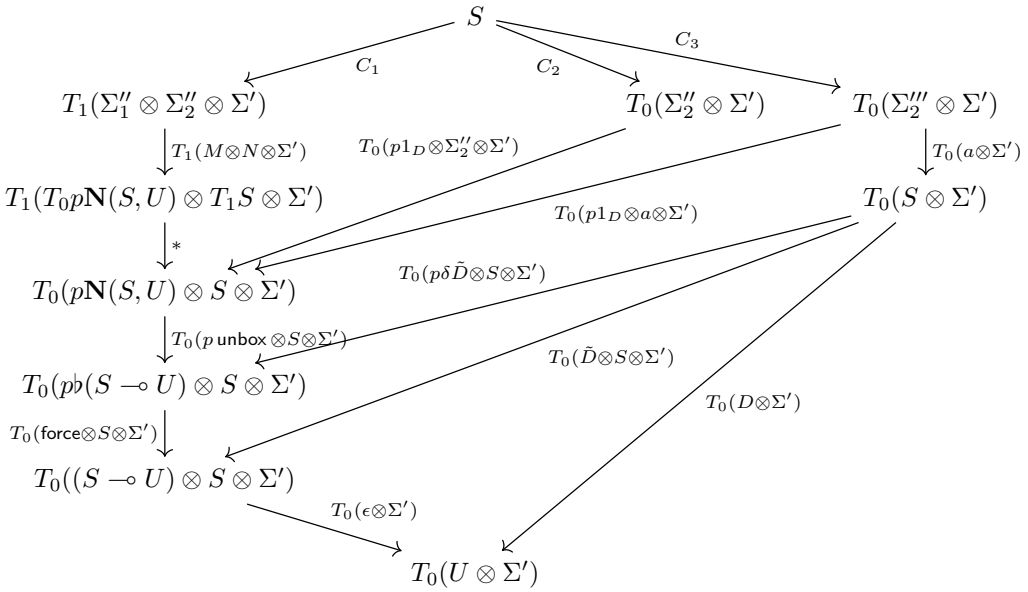
We want to show that $[\![(C_1, \Sigma_1, \mathsf{apply}(M, N))]\!] = [\![(C', \Sigma_5, b)]\!]$, where $(C', \Sigma_5) = \text{append}(C_3, \Sigma_3, a, D, b, \Sigma_4)$ is the following morphism in $\mathbf{M}$. So $\Sigma_5 = (\Sigma_4, \Sigma')$.

$$\begin{aligned}
[\![S']\!] &\xrightarrow{C_3} [\![\Sigma_2''']\!] \otimes [\![\Sigma']\!] \xrightarrow{[\![a]\!] \otimes [\![\Sigma']\!]} [\![S]\!] \otimes [\![\Sigma']\!] \\
&\xrightarrow{G(D) \otimes [\![\Sigma']\!]} [\![U]\!] \otimes [\![\Sigma']\!] \xrightarrow{[\![b]\!]^\dagger \otimes [\![\Sigma']\!]} [\![\Sigma_4]\!] \otimes [\![\Sigma']\!].
\end{aligned}$$

Since $\mathbf{M} \hookrightarrow Kl_{T_0}(\mathbf{A})$ and $\mathbf{N} \hookrightarrow \mathbf{A}$, the corresponding morphism in $\mathbf{A}$ is

$$\begin{aligned}
[\![S']\!] &\xrightarrow{C_3} T_0([\![\Sigma_2''']\!] \otimes [\![\Sigma']\!]) \xrightarrow{T_0([\![a]\!] \otimes [\![\Sigma']\!])} T_0([\![S]\!] \otimes [\![\Sigma']\!]) \\
&\xrightarrow{T_0(D \otimes [\![\Sigma']\!])} T_0([\![U]\!] \otimes [\![\Sigma']\!]) \xrightarrow{T_0([\![b]\!]^\dagger \otimes [\![\Sigma']\!])} T_0([\![\Sigma_4]\!] \otimes [\![\Sigma']\!]).
\end{aligned}$$

We therefore have the following commutative diagram.



The outer left path corresponds to $[\![(C_1, \Sigma_1, \mathsf{apply}(M,N))]\!]$ and the outer right path corresponds to $[\![(C', \Sigma_5, b)]\!]$. The $(*)$ denotes a sequence of strength/costrength maps and the isomorphism $T_\alpha T_\beta \to T_{\alpha \wedge \beta}$. The diagram commutes by *IH1*, *IH2*, $\epsilon \circ (\tilde{D} \otimes S) = D$, $\mathsf{force} \circ p\delta =$ id and the definition of unbox.

- Case

$$\frac{(C, \Sigma, M) \Downarrow (C', \Sigma''', \mathsf{circ}(U, D, U))}{(C, \Sigma, \mathsf{control}_K M) \Downarrow (C', \Sigma''', \mathsf{circ}(\underline{K} \otimes U, \mathsf{ctrl}_K D, \underline{K} \otimes U))}\ ctrl$$

  - Suppose $C : [\![S]\!] \to [\![\Sigma'']\!] \otimes [\![\Sigma']\!] \in \mathbf{N}$, $\Sigma = (\Sigma'', \Sigma')$ and $\Sigma'' \vdash_2 M : \mathbf{Circ}_2(U,U)$. By the induction hypothesis and type preservation (Theorem 5.2), we have

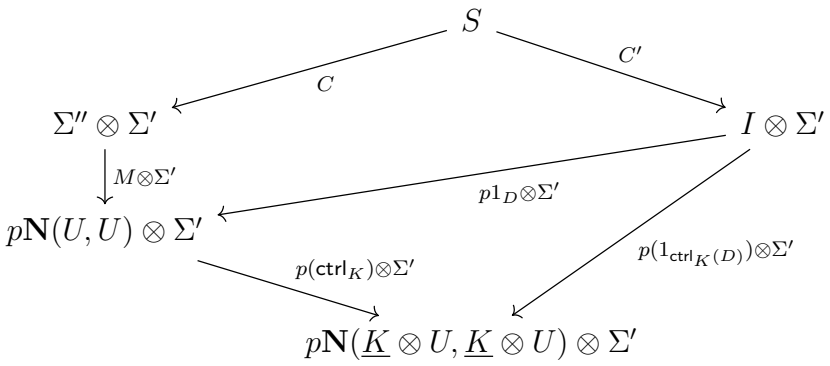
$$[\![(C, \Sigma, M)]\!] = [\![(C', \Sigma', \mathsf{circ}(U, D, U))]\!]$$

    and $S \vdash_2 (C', \Sigma', \mathsf{circ}(U,D,U)) : \mathbf{Circ}_2(U,U) : \Sigma'$, where $\vdash_2 \mathsf{circ}(U,D,U) : \mathbf{Circ}_2(U,U)$ and $\Sigma''' = \Sigma'$. So

$$[\![S]\!] \xrightarrow{C} [\![\Sigma'']\!] \otimes [\![\Sigma']\!] \xrightarrow{[\![M]\!] \otimes [\![\Sigma']\!]} p\mathbf{N}([\![U]\!], [\![U]\!]) \otimes [\![\Sigma']\!]$$

$$= [\![S]\!] \xrightarrow{C'} I \otimes [\![\Sigma']\!] \xrightarrow{p1_D \otimes [\![\Sigma']\!]} p\mathbf{N}([\![U]\!], [\![U]\!]) \otimes [\![\Sigma']\!].$$

Therefore we have the following commutative diagram.



The square commutes by the induction hypothesis and the triangle commutes by $\mathsf{ctrl}_K \circ 1_D = 1_{\mathsf{ctrl}K(D)}$.

– Now suppose $\Sigma = (\Sigma'', \Sigma')$, $C \in \mathbf{M}([\![S]\!], [\![\Sigma'']\!] \otimes [\![\Sigma']\!])$ and $\Sigma'' \vdash_1 M : \mathbf{Circ}_2(U, U)$. By the induction hypothesis, we have $[\![(C, \Sigma, M)]\!] = [\![(C', \Sigma''', \mathsf{circ}(U, D, U))]\!]$, i.e.,

$$[\![S]\!] \xrightarrow{C} T_0([\![\Sigma'']\!] \otimes [\![\Sigma']\!]) \xrightarrow{T_0([\![M]\!] \otimes [\![\Sigma']\!])} T_0(T_1 p\mathbf{N}([\![U]\!], [\![U]\!]) \otimes [\![\Sigma']\!])$$

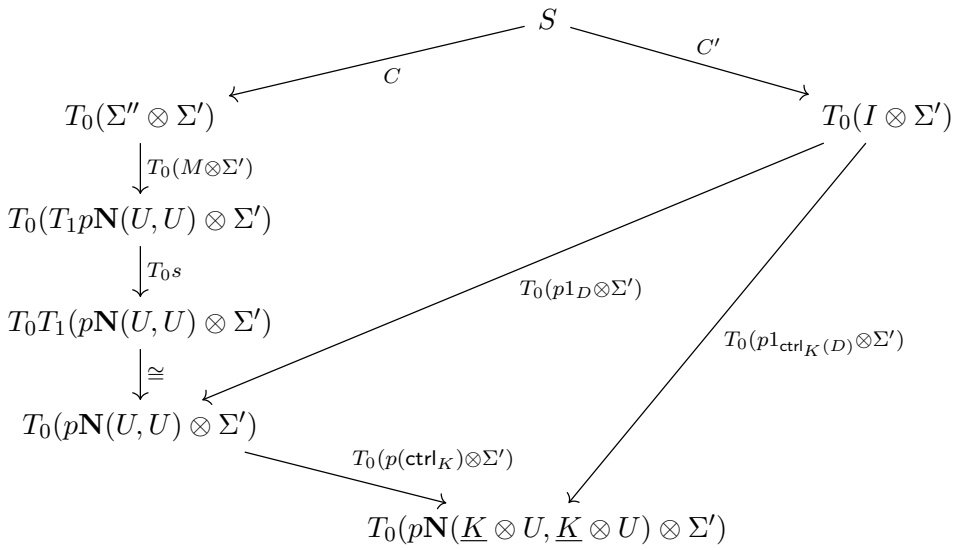
$$\xrightarrow{T_0 s} T_0(p\mathbf{N}([\![U]\!], [\![U]\!]) \otimes [\![\Sigma']\!])$$

$$= [\![S]\!] \xrightarrow{C'} T_0(I \otimes [\![\Sigma']\!]) \xrightarrow{T_0(p1_D \otimes [\![\Sigma']\!])} T_0(p\mathbf{N}([\![U]\!], [\![U]\!]) \otimes [\![\Sigma']\!]).$$

Note that $\Sigma''' = \Sigma'$. We have

$$[\![(C, \Sigma, \mathsf{control}_K M)]\!] = [\![(C', \Sigma''', \mathsf{circ}(\underline{K} \otimes U, \mathsf{ctrl}_K D, \underline{K} \otimes U))]\!]$$

by the following commutative diagram.



It commutes by the induction hypothesis and by $\mathsf{ctrl}_K \circ 1_D = 1_{\mathsf{ctrl}K(D)}$.

- Case

$$\frac{\begin{array}{c}(C, \Sigma, M) \Downarrow (C', \Sigma_1, \mathsf{circ}(U, D_1, S')) \\ (C', \Sigma_1, N) \Downarrow (C'', \Sigma_2, \mathsf{circ}(S', D_2, S'))\end{array}}{(C, \Sigma, \mathsf{withComputed}\ M\ N) \Downarrow (C'', \Sigma_2, \mathsf{circ}(U, D_1 \bullet D_2, U))}\ wc$$

  – Suppose $\Sigma = (\Sigma_1'', \Sigma_2'', \Sigma')$, $C \in \mathbf{N}([\![S]\!], [\![\Sigma_1'']\!] \otimes [\![\Sigma_2'']\!] \otimes [\![\Sigma']\!])$, $\Sigma_1'' \vdash_2 N : \mathbf{Circ}_2(S', S')$ and $\Sigma_2'' \vdash_2 M : \mathbf{Circ}_1(U, S')$. By the induction hypotheses, we have

$$[\![(C, \Sigma, M)]\!] = [\![(C', \Sigma_1, \mathsf{circ}(U, D_1, S'))]\!]$$

and $[\![(C', \Sigma_1, N)]\!] = [\![(C'', \Sigma_2, \mathsf{circ}(S', D_2, S'))]\!]$. Thus we have

$$[\![S]\!] \xrightarrow{C} [\![\Sigma_1'']\!] \otimes [\![\Sigma_2'']\!] \otimes [\![\Sigma']\!] \xrightarrow{[\![M]\!] \otimes [\![\Sigma_2'']\!] \otimes [\![\Sigma']\!]} p\mathbf{R}([\![U]\!], [\![S']\!]) \otimes [\![\Sigma_2'']\!] \otimes [\![\Sigma']\!]$$

$$\overset{IH1}{=} [\![S]\!] \xrightarrow{C'} I \otimes [\![\Sigma_2'']\!] \otimes [\![\Sigma']\!] \xrightarrow{p1_{D_1} \otimes [\![\Sigma_2'']\!] \otimes [\![\Sigma']\!]} p\mathbf{R}([\![U]\!], [\![S']\!]) \otimes [\![\Sigma_2'']\!] \otimes [\![\Sigma']\!]$$

and

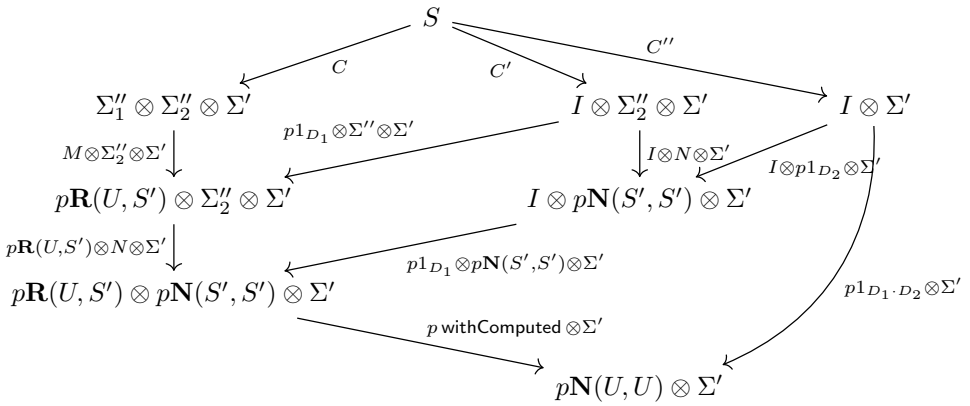
$$[\![S]\!] \xrightarrow{C'} [\![\Sigma_2'']\!] \otimes [\![\Sigma']\!] \xrightarrow{[\![N]\!] \otimes [\![\Sigma']\!]} p\mathbf{N}([\![S']\!], [\![S']\!]) \otimes [\![\Sigma']\!]$$

$$\overset{IH2}{=} [\![S]\!] \xrightarrow{C''} I \otimes [\![\Sigma']\!] \xrightarrow{p1_{D_2} \otimes [\![\Sigma']\!]} p\mathbf{N}([\![S']\!], [\![S']\!]) \otimes [\![\Sigma']\!].$$

We want to show

$$[\![(C, \Sigma, \mathsf{withComputed}\ M\ N)]\!] = [\![(C'', \Sigma_2, \mathsf{circ}(U, D_1 \bullet D_2, U))]\!].$$

This follows from the following commutative diagram. The outer leftmost path corresponds to $[\![(C, \Sigma, \mathsf{withComputed}\ M\ N)]\!]$ and the outer rightmost path corresponds to $[\![(C'', \Sigma_2, \mathsf{circ}(U, D_1 \bullet D_2, U))]\!]$.



The diagram commutes by *IH1*, *IH2*, naturality, and $\mathsf{withComputed} \circ (1_{D_1} \otimes 1_{D_2}) = 1_{D_1 \bullet D_2}$.

Appendix B. Background on enriched categories

Definition B.1. Let $\mathcal{V}$ be a monoidal category. A $\mathcal{V}$-enriched category $\mathbf{B}$ is given by the following:

- A class of objects, also denoted $\mathbf{B}$.
- For any $A, B \in \mathbf{B}$, an object $\mathbf{B}(A, B)$ in $\mathcal{V}$.
- For any $A \in \mathbf{B}$, a morphism $u_A : I \to \mathbf{B}(A, A)$ in $\mathcal{V}$, called the *identity* on A.

- For any $A, B, C \in \mathbf{B}$, a morphism $c_{A,B,C} : \mathbf{B}(A,B) \otimes \mathbf{B}(B,C) \to \mathbf{B}(A,C)$ in $\mathcal{V}$, called *composition*.

- The composition and identity morphisms must satisfy suitable diagrams in $\mathcal{V}$ (see e.g., [21] and [3]).

Remarks.

- Many concepts from the theory of non-enriched categories can be generalized to the enriched setting. For example, $\mathcal{V}$-functors, $\mathcal{V}$-natural transformations, $\mathcal{V}$-adjunctions, and the $\mathcal{V}$-Yoneda embedding are all straightforward generalizations of their non-enriched counterparts. We refer the reader to [21] and [3] for comprehensive introductions. Symmetric monoidal categories can also be generalized to the enriched setting.

- When we speak of a map $f : A \to B$ in a $\mathcal{V}$-enriched category $\mathbf{B}$, we mean a morphism of the form $f : I \to \mathbf{B}(A,B)$ in $\mathcal{V}$. Furthermore, when $g : B \to C$ is also a map in $\mathbf{B}$, we write $g \circ f : A \to C$ as a shorthand for

$$I \overset{f \otimes g}{\to} \mathbf{B}(A,B) \otimes \mathbf{B}(B,C) \overset{c}{\to} \mathbf{B}(A,C).$$

- A $\mathcal{V}$-enriched category $\mathbf{B}$ gives rise to an ordinary (non-enriched) category $V(\mathbf{B})$, called the *underlying category* of $\mathbf{B}$. The objects of $V(\mathbf{B})$ are the objects of $\mathbf{B}$ and the hom-sets of $V(\mathbf{B})$ are defined as $V(\mathbf{B})(A,B) = \mathcal{V}(I, \mathbf{B}(A,B))$, for any $A, B \in V(\mathbf{B})$. Similarly, a $\mathcal{V}$-functor $F : \mathbf{B} \to \mathbf{B}$ gives rise to a functor $VF : V(\mathbf{B}) \to V(\mathbf{B})$ and a $\mathcal{V}$-natural transformation $\alpha : F \to G$ gives rise to a natural transformation $V\alpha : VF \to VG$.