Proto-Quipper with dynamic lifting

Peng Fu, Kohei Kishida, Neil J. Ross, Peter Selinger

April 28, 2022

Abstract

Quipper is a functional programming language for quantum computing. Proto-Quipper is a family of languages aiming to provide a formal foundation for Quipper. In this paper, we extend Proto-Quipper-M with a construct called dynamic lifting, which is present in Quipper. By virtue of being a circuit description language, Proto-Quipper has two separate runtimes: circuit generation time and circuit execution time. Values that are known at circuit generation time are called parameters, and values that are known at circuit execution time are called states. Dynamic lifting is an operation that enables a state, such as the result of a measurement, to be lifted to a parameter, where it can influence the generation of the next portion of the circuit. As a result, dynamic lifting enables Proto-Quipper programs to interleave classical and quantum computation. We describe the syntax of a language we call Proto-Quipper-Dyn. Its type system uses a system of modalities to keep track of the use of dynamic lifting. We also provide an operational semantics, as well as an abstract categorical semantics for dynamic lifting based on enriched category theory. We prove that both the type system and the operational semantics are sound with respect to our categorical semantics. Finally, we give some examples of Proto-Quipper-Dyn programs that make essential use of dynamic lifting. A concrete categorical model is given in a companion paper [FKRS22].

1 Introduction

1.1 Quipper and Proto-Quipper

Quipper is a functional programming language for quantum computing [GLR+13b, GLR+13a]. The overall aim of Quipper is to allow quantum algorithms to be specified at a level of abstraction that is similar to how the algorithm might be described in a research paper, and to compile this down to the level of individual quantum gates, producing a logical quantum circuit. Quipper has been used to program a set of nontrivial algorithms from the quantum computing literature, and it has been used to generate quantum circuits consisting of trillions of gates. As a circuit description language, Quipper shares some of the traits of hardware description languages. In particular, it has two notions of runtime: The first of these is circuit generation time. This is when a Quipper program is run to generate a quantum circuit. The second is circuit execution time. This is when a quantum circuit is executed by a quantum computer or a simulator.

Quipper is a practical language, implemented as an embedded language in Haskell. As such, it lacks formal foundations such as operational and denotational semantics. This motivates the development of Proto-Quipper, a family of experimental languages that aim to provide formal semantics for fragments of Quipper. In Proto-Quipper-S [Ros15], a linear type system with subtyping as well as a small-step operational semantics were introduced. Later, Proto-Quipper-M [RS18] gave a linear type system that features a big-step operational semantics and a sound categorical semantics. Recently, Proto-Quipper-D [FKS20, FKRS20] proposed a type system with linear dependent types as well as a fibrational categorical semantics.

1.2 Dynamic lifting and the interaction of the two runtimes

Proto-Quipper, like Quipper, distinguishes two runtimes. Moreover, Proto-Quipper gives a formal account of parameters and states. A parameter is a value that is known at circuit generation time, such as a boolean
value for an if-then-else expression. A `state` is a value that is only known at circuit execution time, such as the actual state of a qubit or classical bit in a circuit. The type system of Proto-Quipper reflects this distinction. Among the types, there is a subset of `parameter` types, such as `Nat` and `Bool`, whose elements can be duplicated and discarded. There is also a subset of `state` types, such as `Qubit` and `Bit`, which are linear so that their elements cannot in general be duplicated or discarded. One of the fundamental design decisions of Proto-Quipper is that parameter types and state types belong to the same universe of types, so that one can form compound types that are part parameter and part state. An example of this is the type `Bool ⊗ Qubit`, whose elements are pairs of a boolean (a parameter) and a qubit (a state). Another example is the type of lists of qubits. Here, the length of the list is a parameter (known at circuit generation time), but the actual qubits in the list are states (known at circuit execution time). In this way, Proto-Quipper differs, e.g., from Qwire, an embedded quantum circuit description language in which parameters and states belong to separate universes [PRZ17].

In Quipper, the two runtimes can interact with each other. A priori, it is clear that states can depend on parameters. For example, we can initialize a qubit based on a boolean parameter, simply by inserting a gate at circuit generation time to initialize the qubit in one state or another. The opposite direction is more complicated. Usually, circuit execution happens after circuit generation, and in this case, it is clear that a state cannot be converted to a parameter. However, there are some quantum algorithms that require circuit generation and circuit execution to be interleaved. Here, a state, such as the outcome of a measurement in a circuit, may be used to inform the generation of the next part of the circuit. To enable such interleaving, Quipper provides a construct called dynamic lifting, which enables a state to be lifted to a parameter in certain situations. For example, dynamic lifting permits the result of a measurement, which is a state of type `Bit`, to be lifted to a parameter of type `Bool`. It is important to note that dynamic lifting is an expensive operation, as it requires control to pass from circuit evaluation time back to circuit generation time. This requires the real-time quantum computer to put all of its active qubits into long-term storage while spending an indeterminate amount of time awaiting further instructions from the classical computer in charge of circuit generation.

Dynamic lifting is important because it can be used to express quantum algorithms that require interleaving circuit execution time and circuit generation time. While there are many quantum algorithms that do not require such interleaving, there are some that do. An example is magic state distillation [BK05]. Here, the goal is to prepare a qubit in some target state. We start with a large number of qubits, say \( n \) of them, each of which is a rough approximation of the target state. We then apply a probabilistic “distillation” procedure which yields on average, say, \( n/4 \) qubits that are better approximations of the target state; the remaining qubits are wasted. By repeated distillation steps, we eventually wind up with a small number of qubits that are excellent approximations of the target state. In such a situation, dynamic lifting is essential because after each distillation step, we must throw away the wasted qubits, but we do not know ahead of time which ones (or indeed, how many) there will be. Thus which future gates will be applied depends on the outcomes of previous measurements. With the help of dynamic lifting, these algorithms can be naturally expressed as functions in the programming language.

The concept of dynamic lifting is different from measurement, and the two should not be confused. Measurement is merely a gate in a circuit, turning a quantum bit (a state) into a classical bit (also a state). Dynamic lifting is an operation of the programming language, turning a classical bit (a state) into a boolean (a parameter).

### 1.3 A type system for dynamic lifting

Previous versions of Proto-Quipper lacked dynamic lifting. Modeling dynamic lifting is a challenging problem. To better understand the issues involved, it is useful to know that there are two things that can be done with circuits in Proto-Quipper. On the one hand, circuits can be run on a quantum device. On the other hand, circuits can be `boxed`, which means, converted to a data structure that can then be inspected and manipulated. The ability to box circuits is crucial to Quipper’s ability to express algorithms at a natural level of abstraction, because algorithms are often described in terms of meta-operations on circuits. For example, a circuit may be constructed by applying gate transformations to an existing circuit, reversing or
repeating it, etc.

Now it is clear that dynamic lifting only makes sense in the context of a circuit that is actually being executed, rather than one that is merely being boxed. We will keep track of this in the programming language by adding modalities to the type system and a corresponding monad to the semantics.

Before we can describe the operational or denotational semantics of Proto-Quipper-Dyn, we must be more precise about what we mean by a “circuit”. We must also specify what it means to “execute” a circuit. There are many different notions of circuits, differing, for example, in which collection of gates is provided. Rather than specializing to one of these, we take a more general point of view: a circuit is simply a morphism in a symmetric monoidal category $\mathbf{M}$, which we assume to be given ahead of time, but otherwise arbitrary (subject to some properties). Similarly, for the execution of circuits, we assume given another symmetric monoidal category $\mathbf{Q}$ of quantum operations. Conceptually, the morphisms of $\mathbf{M}$ are syntactic entities; thus, $\mathbf{M}$ is typically a category that is free generated (say by a collection of gates). On the other hand, we think of the morphisms of $\mathbf{Q}$ as physical operations, which can be performed on a quantum computer. The categories $\mathbf{M}$ and $\mathbf{Q}$ have the same objects, and there is an interpretation functor $J : \mathbf{M} \to \mathbf{Q}$.

Operationally, dynamic lifting is an operation that reads the state of a bit in $\mathbf{Q}$, and returns a boolean value. Since a bit state can be the result of a measurement, the read operation for dynamic lifting is nondeterministic, i.e., it can return different boolean values with probabilities governed by measurements. The nondeterministic nature of the dynamic lifting suggests that it should be modeled as a monadic operation $\mathbf{M} \to \mathbf{M}$.

We therefore conceptualize the types of Proto-Quipper-Dyn as the objects of a single category $\mathbf{A}$, with a monad $T : \mathbf{A} \to \mathbf{A}$, called the dynamic lifting monad. This will be done in such a way that $\mathbf{M}$ is fully embedded in $\mathbf{A}$, and $\mathbf{Q}$ is fully embedded in the Kleisli category $Kl(T(\mathbf{A}))$, in a way that makes the following diagram commute.

$$
\begin{array}{ccc}
\mathbf{M} & \xrightarrow{\psi} & \mathbf{A} \\
\downarrow J & & \downarrow E \\
\mathbf{Q} & \xrightarrow{\phi} & Kl(T(\mathbf{A}))
\end{array}
$$

Here, $J$ is the given interpretation functor, and $E$ is the canonical functor from $\mathbf{A}$ to $Kl(T(\mathbf{A}))$. We then model dynamic lifting as a map $\text{dynlift} : \text{Bit} \to T\text{Bool} \in Kl(T(\mathbf{A}))$ such that the following diagram commutes.

$$
\begin{array}{ccc}
\text{Bit} & \xrightarrow{\text{init}} & \mathbf{Bool} \\
& \xrightarrow{\text{dynlift}} & \downarrow \\
& & T\text{Bool}
\end{array}
$$

Note that dynamic lifting is a morphism of the Kleisli category; this makes sense because it is essentially a side-effecting read operation. More generally, any computation that potentially uses dynamic lifting will have type $A \to TB$.

As mentioned above, our type system must also distinguish quantum circuits that are being executed from quantum circuits that are being boxed. Naturally, since the latter may not use dynamic lifting, they are maps in the category $\mathbf{A}$ while the former are maps in the Kleisli category $Kl(T(\mathbf{A}))$. As a practical matter for programmer convenience, it would be awkward to have $T$ as an explicit type constructor that must be mentioned everywhere in the program. Instead, we use a system of modalities to keep track of the dynamic lifting monad $T$. More specifically, we annotate a typing judgment with a modality, i.e., $\Gamma \vdash \alpha M : A$, where $\alpha \in \{0,1\}$. When $\alpha = 0$, it means that the term $M$ represents a morphism $[\Gamma] \to T[A]$ in the Kleisli category $Kl(T(\mathbf{A}))$. When $\alpha = 1$, it means that the term $M$ represents a morphism $[\Gamma] \to [A]$ in $\mathbf{A}$. An example of the typing rule for dynamic lifting is the following (where $\text{Meas} : \text{Qubit} \to \text{Bit}$ represents the measurement gate).

$$
\ell : \text{Qubit} \vdash_1 \text{Meas}(\ell) : \text{Bit}
$$

$$
\ell : \text{Qubit} \vdash_0 \text{dynlift}(\text{Meas}(\ell)) : \text{Bool}
$$

If we have a quantum circuit $\text{Qubit} \to \text{Bit}$, it can be run by a quantum computer and the measurement result of type $\text{Bit}$ will be lifted to a parameter of type $\text{Bool}$. Note that the $\text{dynlift}$ operation sets the modality
of the typing judgment to 0, and as a result, we have a map $Qubit \rightarrow TBool$ in the Kleisli category. The use of modalities in our type system ensures that the term $\text{Meas}(\ell)$ can be turned into a boxed circuit, whereas it will be a compile time typing error to try to box the term $\text{dynlift}(\text{Meas}(\ell))$.

1.4 Operational semantics

Next, let us take a look at the operational semantics of Proto-Quipper-Dyn. In previous versions of Proto-Quipper, the operational semantics used configurations of the form $(C, M)$, where $C$ is the circuit being currently constructed, and $M$ is a term. On the other hand, in the quantum lambda calculus [SV09], which is not a circuit construction language but intended to run directly on a quantum computer, the operational semantics used configurations of the form $(Q, M)$, where $Q$ is the current quantum state and $M$ is a term.

In a sense, Proto-Quipper-Dyn is a combination of these prior languages: it is a language for circuit construction (via the boxing operation), but it is also a language for running quantum operations (as otherwise dynamic lifting would not be possible). Consequently, our operational semantics uses both kinds of configurations: those of the form $(Q, M)$ are only used for top-level computations that actually run on a quantum device, and those of the form $(C, M)$ are used during boxing. These two kinds of configurations correspond closely to the two runtimes, since configuration of the form $(C, M)$ are used for circuit construction and those of the form $(Q, M)$ are used for circuit execution. They also correspond to the two categories $M$ and $Q$.

Consequently, the evaluation rules take two different forms. Evaluation at circuit generation time takes the form $(C, M) \downarrow (C', V)$, where $C$ is a circuit. The type system ensures that such an evaluation does not involve dynamic lifting, so it can be done entirely with a classical computer and the evaluation is deterministic. On the other hand, evaluation at circuit execution time takes the form $(Q, M) \downarrow \sum_i p_i(Q_i, V_i)$, where $Q$ represents a quantum state. Since $M$ can use dynamic lifting, the result of such an evaluation rule is probabilistic, with outcome $(Q_i, V_i)$ happening with probability $p_i$.

1.5 Related work

In recent work, Lee et al. [LPVX21] used a different approach to extending Proto-Quipper with dynamic lifting. Instead of distinguishing circuit generation time and circuit execution time, they work with a single runtime modeled by a category of quantum channels, which are a generalization of quantum circuits with a notion of branching for measurement results. As a result, their categorical semantics and type system do not distinguish quantum circuits from quantum computations that use dynamic lifting.

Another version of Proto-Quipper with a form of dynamic lifting is proposed by Colledan and Dal Lago [CL22]. Their language uses a very general version of dynamic lifting, even more general than the one present in the Quipper language, which allows for measurements to be conditional on the outcomes of prior measurements. As a consequence, the output type of their circuit can depend on the outcomes of the measurements specified in the computation. While this alternative notion of dynamic lifting is interesting in its own right, their language does not come equipped with a denotational semantics.

Qwire [PRZ17] is a quantum programming language that also supports dynamic lifting. Qwire has a host language and a circuit language. The host language describes the computation of the classical computer, while the circuit language describes the computation of the quantum computer. Qwire has a denotational semantics for the circuit language, but not for the host language. Dynamic lifting is part of the syntax in their circuit language. Therefore Qwire’s notion of quantum circuit differs from Proto-Quipper’s notion of quantum circuit. For example, printing a circuit in Qwire could fail because of dynamic lifting, whereas circuit printing in Proto-Quipper-Dyn can not fail, because our type system ensures that a value of a circuit type does not use dynamic lifting.

1.6 Contributions

In this paper, we describe the syntax and type system of an extension of Proto-Quipper with dynamic lifting, called Proto-Quipper-Dyn. The type system uses a system of modalities to keep track of the use of dynamic lifting. We also provide an operational semantics, using two different kinds of configurations to model circuit
generation time and circuit execution time. We further provide an abstract categorical semantics for this language, in which dynamic lifting is modeled by a map \( \text{Bit} \to T\text{Bool} \), where \( T \) is a monad encapsulating circuit execution. By an “abstract” categorical semantics, we mean that we only state the properties that a categorical model must satisfy to give a sound interpretation of the language, without constructing an actual concrete example of such a model. We give such a concrete model in a companion paper [FKRS22].

The rest of the paper is organized as follows: In Section 2 we briefly recall the basics of enriched category theory, and then we give an axiomatization of a general categorical semantics for dynamic lifting. In Section 3 we define a type system for dynamic lifting that uses a system of modalities. We then show how a typing judgment with modalities is interpreted as a morphism in our categorical model. In Section 4, we define a call-by-value big-step operational semantics for our language. We show that the operational semantics is sound with respect to the enriched categorical semantics. In Section 5, we prove that the operational semantics satisfies type preservation and that the type system guarantees error freeness. We also show that the operational semantics is sound with respect to the enriched categorical semantics. In Section 6 we give some applications of dynamic lifting in Proto-Quipper-Dyn. We finish the paper with some concluding remarks in Section 6.

2 An enriched categorical semantics for dynamic lifting

In this section we will give a general categorical semantics for dynamic lifting. Our categorical semantics is based on enriched categories, which are generalizations of ordinary categories. In enriched categories, instead of hom-sets, one works with hom-objects, which are objects in a monoidal category.

Definition 2.1. Let \( \mathcal{V} \) be a monoidal category. A \( \mathcal{V} \)-enriched category \( \mathbb{A} \) (or \( \mathcal{V} \)-category for short) is given by the following:

- A class of objects, also denoted \( \mathbb{A} \).

- For any \( A, B \in \mathbb{A} \), an object \( \mathbb{A}(A, B) \) in \( \mathcal{V} \).

- For any \( A \in \mathbb{A} \), a morphism in \( u_A : I \to \mathbb{A}(A, A) \) in \( \mathcal{V} \), called the identity on \( A \).

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

- The composition and identity morphisms must satisfy suitable diagrams in \( \mathcal{V} \) (see [Kel82, Bor94]).

Remarks.  
- Many concepts from non-enriched category theory 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 to [Kel82, Bor94] for comprehensive introductions.

- In the rest of this paper, when we speak of a map \( f : A \to B \) in a \( \mathcal{V} \)-enriched category \( \mathbb{A} \), we mean a morphism of the form \( f : I \to \mathbb{A}(A, B) \) in \( \mathcal{V} \). Furthermore, when \( g : B \to C \) is another map in \( \mathbb{A} \), we write \( g \circ f : A \to C \) as a shorthand for

\[
I \xrightarrow{f \otimes g} \mathbb{A}(A, B) \otimes \mathbb{A}(B, C) \xrightarrow{\cdot} \mathbb{A}(A, C).
\]

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

Ordinary symmetric monoidal categories can be generalized to enriched categories as well.

\(^1\text{V stands for “underlying” because the letter } U \text{ serves another purpose in this paper.}\)
Definition 2.2. Let $\mathcal{V}$ be a symmetric monoidal category. A $\mathcal{V}$-category $\mathbf{A}$ is symmetric monoidal if it is equipped with the following:

- There is an object $I \in \mathbf{A}$ called the tensor unit. For any $A, B \in \mathbf{A}$, there is an object $A \otimes B \in \mathbf{A}$. Moreover, for any $A_1, A_2, B_1, B_2 \in \mathbf{A}$, there is a morphism
  \[ \text{Tensor} : \mathbf{A}(A_1, B_1) \otimes \mathbf{A}(A_2, B_2) \to \mathbf{A}(A_1 \otimes A_2, B_1 \otimes B_2) \]
  in $\mathcal{V}$. The tensor product is a bifunctor in the sense that $\text{Tensor} \circ (u_A \otimes u_B) = u_{A \otimes B}$ for the identity maps $u_A, u_B, u_{A \otimes B}$, and the following diagram commutes for any $A, A_1, B, B_1, C, C_1, C_2 \in \mathbf{A}$.

- There are the following $\mathcal{V}$-natural isomorphisms in $\mathbf{A}$ and they satisfy the same coherence diagrams for symmetric monoidal categories.
  \[ l_A : I \otimes A \to A \]
  \[ r_A : A \otimes I \to A \]
  \[ \gamma_{A,B} : A \otimes B \to B \otimes A \]
  \[ \alpha_{A,B,C} : (A \otimes B) \otimes C \to A \otimes (B \otimes C) \]

If the $\mathcal{V}$-category $\mathbf{A}$ is symmetric monoidal, then its underlying category $\mathcal{V}(\mathbf{A})$ is symmetric monoidal. For any maps $f : A_1 \to B_1, g : A_2 \to B_2$ in $\mathbf{A}$, we write the map $f \otimes g : A_1 \otimes A_2 \to B_1 \otimes B_2$ as a shorthand for the following composition.

\[ I \xrightarrow{f \otimes g} \mathbf{A}(A_1, B_1) \otimes \mathbf{A}(A_2, B_2) \xrightarrow{\text{Tensor}} \mathbf{A}(A_1 \otimes A_2, B_1 \otimes B_2) \]

2.1 An axiomatization of enriched categorical models of dynamic lifting

In the following, we assume $\mathcal{V}$ to be a cartesian closed category with coproducts. For any $A, B \in \mathcal{V}$, we write $A \times B$ for the cartesian product, $A \Rightarrow B$ for the exponential object, and $1 \in \mathcal{V}$ for the terminal object. Since $\mathcal{V}$ is cartesian closed, it is self-enriched, i.e., $\mathcal{V}$ is a $\mathcal{V}$-category where the hom-objects are defined by $\mathcal{V}(A,B) := A \Rightarrow B$.

We will now focus on defining a $\mathcal{V}$-enriched category $\mathbf{A}$ that models dynamic lifting. We give a sequence of definitions that specify a sequence of properties [3, 11], which will culminate in Definition 2.8 of a model for Proto-Quipper with dynamic lifting.

Definition 2.3. A $\mathcal{V}$-category $\mathbf{A}$ is a linear-non-linear programming language model if

(a) $\mathbf{A}$ has coproducts and is symmetric monoidal closed, i.e., it is symmetric monoidal and there is a $\mathcal{V}$-adjunction $- \otimes A \dashv A \rightarrow -$ for each $A \in \mathbf{A}$.

(b) $\mathbf{A}$ is equipped with a $\mathcal{V}$-adjunction

\[ p : \mathcal{V} \to \mathbf{A} \quad \dashv \quad \mathbf{A} \to \mathcal{V} \]

such that $p$ is a strong monoidal $\mathcal{V}$-functor.

Remarks. The requirement that $\mathbf{A}$ has coproducts and is symmetric monoidal closed implies that it can model function types and sum types in a functional programming language. Moreover, since $- \otimes A$ is a left adjoint $\mathcal{V}$-functor for any $A \in \mathbf{A}$, it preserves the coproducts, so the tensor products distribute over coproducts in $\mathbf{A}$.
• The adjunction in $[\mathcal{B}]$ is often called a linear-non-linear adjunction [Ben95]. Here, the assumption that $p$ is a strong monoidal $\mathcal{V}$-functor means that there exist isomorphisms $e : I \rightarrow p1$ and $m : pX \otimes pY \rightarrow p(X \times Y)$ making some diagrams commute (see Appendix $A$).

• Since $p$ is strong monoidal and $\mathcal{V}$ is cartesian, for any $X \in \mathcal{V}$, there are maps $\text{discard}_X : pX \rightarrow I$ and $\text{dup}_X : pX \rightarrow pX \otimes pX$ in $A$. Moreover, for any map $f : X \rightarrow Y$ in $\mathcal{V}$, we have the following in $A$.

$$\text{dup}_Y \circ pf = (pf \otimes pf) \circ \text{dup}_X$$

We call objects of the form $pX \in A$ parameter objects, since they can be duplicated and discarded. For example, $\text{Bool} := I + I \cong p1 + p1 \cong p(1 + 1)$ is a parameter object.

• For any $X \in \mathcal{V}, B \in A$, we write $\delta$ for the isomorphism $\delta : A(pX, B) \cong \mathcal{V}(X, \text{flat} B)$, and force$_{B}$ for the counit $\text{force}_{B} : p\text{flat} B \rightarrow B$.

**Definition 2.4.** A convex space is a set $X$ equipped with a convex sum operation, which assigns to any $x, y \in X$ and $p, q \in [0, 1]$ such that $p + q = 1$ an element $px + qy \in X$, subject to certain standard conditions, which are detailed in Appendix $B$. A category is enriched in convex spaces if each hom-set is equipped with the structure of a convex space, and moreover, composition is bilinear with respect to convex sum, i.e., $(pf + qg) \circ h = p(f \circ h) + q(g \circ h)$ and $h \circ (pf + qg) = p(h \circ f) + q(h \circ g)$.

As mentioned in the introduction, Proto-Quipper-Dyn is parameterized by two (ordinary) categories $M$ and $Q$ of circuits and quantum operations, respectively. We now specify the properties that these categories must satisfy.

**Assumption 2.5.** We assume that we are given two symmetric monoidal categories $M$ and $Q$, satisfying the following properties:

1. $M$ and $Q$ have the same objects, including a distinguished object called $\text{Bit}$. The category $M$ has distinguished morphisms zero, one : $I \rightarrow \text{Bit}$.

2. $Q$ has a coproduct $\text{Bit} = I + I$, and the tensor product in $Q$ distributes over this coproduct.

3. There exists a given strict monoidal functor $J : M \rightarrow Q$ that is the identity on objects and $J(\text{zero}) = \text{inj}_1 : I \rightarrow I + I, J(\text{one}) = \text{inj}_2 : I \rightarrow I + I$. We call $J$ the interpretation functor.

4. The category $Q$ is enriched in convex spaces.

5. For any $A \in Q$, and $f : I \rightarrow \text{Bit} \otimes A \in Q$, we have $f = p_1(\text{inj}_1 \otimes f_1) + p_2(\text{inj}_2 \otimes f_2)$, where $\text{inj}_1, \text{inj}_2 : I \rightarrow I + I$ and $p_1, p_2 \in [0, 1]$ are uniquely determined real numbers such that $p_1 + p_2 = 1$. When $p_i \neq 0$, the map $f_i$ is also unique.

The categories $M$ and $Q$ are not only used in the categorical semantics, but also in the operational semantics of Proto-Quipper-Dyn (i.e., to run the program, we must know what a circuit is and what a quantum operation is). Therefore, these categories should be regarded as given as part of the language specification, rather than as a degree of freedom in the semantics. On the other hand, nothing in the operational or denotational semantics depends on particular properties of $M$ and $Q$ other than properties (1)-(5) above. Therefore, Proto-Quipper-Dyn can handle a wide variety of possible circuit models and physical execution models.

In practice, the category $M$ will be a category of quantum circuits and the category $Q$ will be a category of quantum operations. These categories will typically have additional objects, such as $\text{Qubit}$ and perhaps $\text{Qutrit}$, and additional morphisms, such as $H : \text{Qubit} \rightarrow \text{Qubit}$ and $\text{Meas} : \text{Qubit} \rightarrow \text{Bit}$. Assumption (5) means that any morphism with domain $I$ and a bit state in its codomain is a convex sum of two morphisms. This property is used in the rule for dynamic lifting in the operational semantics.

**Definition 2.6.** Suppose the $\mathcal{V}$-enriched category $A$ is a linear-non-linear programming model. We say it supports box-unbox operations if the following hold.
(c) There is a fully faithful embedding $\psi : M \hookrightarrow V(A)$ and $\psi$ is strong monoidal.

(d) Let $S$ denote the set of objects in the image of $\psi$. For any $S, U \in S$, there is an isomorphism

$$b(S \to U) \cong A(S, U).$$

Condition (c) implies that there is a circuit subcategory in $A$. Using condition (d), we define $\boxe = p(e)$ and $\unboxe = p(e^{-1})$, and there is an isomorphism $p(b(S \to U)) \cong pA(S, U)$. Elements of $pA(S, U)$ correspond to boxed circuits with input $S$ and output $U$.

If a $V$-enriched category $A$ satisfies (c)–(f), then it is a model for Proto-Quipper without dynamic lifting. For example, the Set-enriched category $M$ in [RS18] is such a model. To support dynamic lifting, we define the following monad to account for the category $Q$.

**Definition 2.7.** Let $A$ be a symmetric monoidal $V$-category and let $T : A \to A$ be a $V$-monad on $A$. We say $T$ is a **dynamic lifting monad** if the following hold.

(e) $T$ is a commutative strong $V$-monad. For any $A, B \in A$, we write $t_{A,B} : A \otimes TB \to T(A \otimes B)$ for the strength and $s_{A,B} : TA \otimes B \to T(A \otimes B)$ for the costrength.

(f) Let $V(A)$ be the underlying category of $A$, let $VT$ be the underlying monad of $T$, and let $Kl_{VT}(V(A))$ be the Kleisli category of $VT$. The Kleisli category $Kl_{VT}(V(A))$ is enriched in convex spaces.

(g) There are the following fully faithful embeddings:

$$M \xhookrightarrow{\psi} V(A),$$

$$Q \xhookrightarrow{\phi} Kl_{VT}(V(A)).$$

These embedding functors are strong monoidal, and $\phi$ preserves the convex sum. Moreover, the following diagram commutes for any $S, U \in M$.

$$
\begin{array}{ccc}
M(S, U) & \xrightarrow{\psi_{S,U}} & V(A)(S, U) \\
\downarrow{J_{S,U}} & & \downarrow{E_{S,U}} \\
Q(S, U) & \xrightarrow{\phi_{S,U}} & Kl_{VT}(V(A))(S, U)
\end{array}
$$

Here $E : V(A) \to Kl_{VT}(V(A))$ is the the functor such that $E(A) = A$ and $E(f) = \eta \circ f$.

(h) There are maps $\text{dynlift} : \text{Bit} \to T\text{Bool}$ and $\text{init} : \text{Bool} \to \text{Bit}$ in $A$ such that the following diagram commutes.

$$
\begin{array}{ccc}
\text{Bit} & \xrightarrow{\text{init}} & \text{Bool} \\
\downarrow{\text{dynlift}} & & \downarrow{\eta} \\
& & T\text{Bool}
\end{array}
$$

**Remarks.**

- The objects of the Kleisli category $Kl_{VT}(V(A))$ are the same as the objects of $A$, and the hom-set is given by $Kl_{VT}(V(A))(A, B) := V(A)(A, VT(B)) = V(1, A(TB))$ for any $A, B \in A$. Moreover, $V(1, A(TB)) = V(1, Kl_T(A)(A, B)) = V(Kl_T(A))(A, B)$.

- Since $T$ is a commutative strong $V$-monad, $VT$ is a commutative strong (ordinary) monad. Therefore the Kleisli category $Kl_{VT}(V(A))$ is monoidal. For any $f : A_1 \to VT B_1$ and $g : A_2 \to VT B_2$ in $Kl_{VT}(V(A))$, we define $f \otimes g \in Kl_{VT}(V(A))(A_1 \otimes A_2, B_1 \otimes B_2)$ to be the following

$$A_1 \otimes A_2 \xrightarrow{f \otimes g} VT B_1 \otimes VT B_2 \xrightarrow{\delta} VT(B_1 \otimes VT B_2) \xrightarrow{T_1} VT(VT B_1 \otimes B_2) \xrightarrow{\mu} VT(B_1 \otimes B_2).$$
Definition 2.8. We say a \( \alpha \)-lifting. When \( \alpha = 0 \), being reduced to a value. Such computations can therefore be carried out at circuit generation time. When [FKRS22], we give a construction of a concrete model based on\( \) satisfies (a)–(h).

Definition 3.1. In the rest of this paper, we will be focusing on showing this abstract categorical model that comes with any particular instance of Proto-Quipper-Dyn.

\[ \text{Circ} \]

The modality \( \alpha \) appears in the linear function type \( A \rightarrow_{\alpha} B \) and the linear exponential type \( !_{\alpha}A \). This is because the values of \( A \rightarrow_{\alpha} B \) and \( !_{\alpha}A \) are thunks and we use the modality \( \alpha \) in the types to keep track of the dynamic lifting within the thunks. \( \text{Circ}(S, U) \) denotes a type of circuits with input \( S \) and output \( U \). The values of this type are boxed quantum circuits. They can be further manipulated by meta-operations such as circuit reversal, circuit iteration, or printing; these operations are treated as constants in the language, i.e., we do not fix a particular set of such operations, but assume that they would be defined in a standard library that comes with any particular instance of Proto-Quipper-Dyn.

\[
\begin{align*}
\text{Modality} & \quad \alpha, \beta \quad ::= \quad 0 \mid 1 \\
\text{Types} & \quad A, B \quad ::= \quad \text{Unit} \mid \text{Qubit} \mid \text{Bit} \mid \text{Bool} \mid !_{\alpha}A \mid A \rightarrow_{\alpha} B \mid \text{Circ}(S, U) \mid A \otimes B \\
\text{Parameter Types} & \quad P, R \quad ::= \quad \text{Unit} \mid \text{Nat} \mid !_{\alpha}A \mid \text{Circ}(S, U) \\
\text{Simple Types} & \quad S, U \quad ::= \quad \text{Unit} \mid \text{Qubit} \mid \text{Bit} \mid S \otimes U \\
\text{Terms} & \quad M, N \quad ::= \quad c \mid x \mid \lambda x.M \mid M \cdot N \mid \text{Unit} \mid (a, c, b) \mid \text{apply}(M, N) \mid \text{force}M \\
& \quad \mid \text{lift} M \mid \text{box} U M \mid (M, N) \mid \text{let} (x, y) = N \in M \mid \text{dynlift} M \\
\text{Simple Terms} & \quad a, b \quad ::= \quad \ell \mid \text{Unit} \mid (a, b) \\
\text{Contexts} & \quad \Gamma \quad ::= \quad \cdot \mid x : A, \Gamma \mid \ell : \text{Qubit}, \Gamma \mid \ell : \text{Bit}, \Gamma \\
\text{Parameter contexts} & \quad \Phi \quad ::= \quad \cdot \mid x : P, \Phi. \\
\text{Label Contexts} & \quad \Sigma \quad ::= \quad \cdot \mid \ell : \text{Qubit}, \Sigma \mid \ell : \text{Bit}, \Sigma \\
\text{Values} & \quad V \quad ::= \quad x \mid \ell \mid \lambda x.M \mid \text{lift} M \mid (a, c, b) \mid (V, V') \mid \text{Unit} \\
\text{Circuits} & \quad \mathcal{C}, \mathcal{D} : \Sigma \rightarrow \Sigma' \\
\end{align*}
\]

Figure 1: The syntax for Proto-Quipper-Dyn

- Condition (2) expresses the requirement that the enriched category \( A \) must combine both categories \( M \) and \( Q \), i.e., they are subcategories of \( V(A) \) and its Kleisli category, respectively. Thus \( A \) has both quantum circuits and quantum operations. The commutative diagram implies that a circuit in \( A \) can be used as a quantum operation.

- Since \( \psi(S) = \phi(S) \) for any \( S \in M, Q \), we define \( \text{Bit} = \psi(\text{Bit}) = \phi(\text{Bit}) \in A \).

- Condition (3) gives a categorical characterization of dynamic lifting. The map \( \text{dynlift} \) is not in the image of \( \phi \) or \( \psi \), and therefore it is neither a quantum circuit nor a quantum operation.

Definition 2.8. We say a \( \nu \)-enriched category \( A \) is a model for Proto-Quipper with dynamic lifting if it satisfies (a)–(h).

We have now axiomatized a general categorical model for Proto-Quipper with dynamic lifting. In [FKRS22], we give a construction of a concrete model based on biset-enrichment that satisfies (a)–(h). In the rest of this paper, we will be focusing on showing this abstract categorical model \( A \) is sound with respect to the type system and the operational semantics.

3 A type system for dynamic lifting

In this section, we present the syntax of Proto-Quipper-Dyn and a type system for dynamic lifting. Our typing judgments have the form \( \Gamma \vdash_{\alpha} M : A \), where \( \alpha ::= 0 \mid 1 \) is a modality used to keep track of dynamic lifting. When \( \alpha = 1 \), the term \( M \) is guaranteed not to perform any dynamic lifting operations while it is being reduced to a value. Such computations can therefore be carried out at circuit generation time. When \( \alpha = 0 \), \( M \) may invoke dynamic lifting so the evaluation of \( M \) needs to be performed at circuit execution time.

Definition 3.1 (Syntax). The syntax of Proto-Quipper-Dyn is in Figure 1.
The terms of our language are similar to the ones from [RS18], with the addition of a term construct for dynamic lifting $\text{dynlift} M$, which will be evaluated to a boolean value. A term of parameter type can be duplicated or discarded. A value of simple type corresponds to a state.

We make a distinction between variables and labels. A label $\ell$ corresponds to a wire in a circuit, or to an address of a bit or qubit state. Consequently, a label is a value that can only have type Bit or Qubit. Labels can only be renamed, not substituted. Every label context $\Sigma$ has an obvious interpretation $\llbracket \Sigma \rrbracket$ in the category $M$ as a tensor of the appropriate sequence of the objects Qubit and Bit. We write $D : \Sigma \rightarrow \llbracket \Sigma' \rrbracket$.

**Definition 3.2 (Typing).** The typing rules are in Figure 2.

We write $\alpha \& \beta$ for the boolean conjunction of $\alpha$ and $\beta$ so that, e.g., $0 \& 1 = 0$. If $\Gamma_1 = \Phi$, $\Gamma'_2$ and $\Gamma_2 = \Phi$, $\Gamma'_1$, we write $\Gamma_1 + \Gamma_2$ for $\Phi$, $\Gamma'_1$, $\Gamma'_2$.

In the $\text{var}$ rule, we require a parameter context $\Phi$. In the $\text{lift}$ and $\text{lambda}$ rules, the modality $\alpha$ is moved to the type and the current modality (i.e., modality in the conclusion) is set to 1. This is because the lift and lambda terms are values, and values do not perform dynamic lifting. In fact, all values have modality 1.

In elimination rules such as $\text{app}$ and $\text{force}$, the modality in the type affects the current modality of the typing judgment through boolean conjunction. This is related to how the evaluations are performed for these terms. For example, when evaluating the term $MN$, we will first evaluate $M$, then evaluate $N$ and finally perform a beta-reduction. Thus, the evaluation of $MN$ could perform dynamic lifting of $\alpha_1 = 0$, $\alpha_2 = 0$, or $\beta = 0$. Consequently, the modality for the typing judgment of $MN$ is the boolean conjunction of all these related modalities.

By the $\text{dynlift}$ rule, an application of dynamic lifting sets the current modality to 0, signifying that a dynamic lifting is performed. In the $\text{box}$ rule, a term $M$ can only be boxed into a circuit if it has type $!_1(S \rightarrow_1 U)$. This ensures that the term $M$ does not use dynamic lifting. Thus, when evaluating the term (box $S M$), a dynamic lifting cannot occur. This prevents a class of runtime errors in Quipper that are caused by boxing functions that use dynamic lifting.

In the $\text{apply}$ rule, depending on the modality $\alpha_1$ and $\alpha_2$, the term $\text{apply}(M, N)$ either appends the quantum circuit $M$ to $N$, which is done at circuit generation time, or applies the quantum operation $M$ to $N$, which is done at circuit execution time. The $\text{circ}$ rule defines a well-typed quantum circuit. In practice, we often
The following lemma shows that a value can only have modality 1 and, in particular, that the free variables of a parameter must come from a parameter context.

**Lemma 3.3.** If $\Gamma \vdash_{\alpha} V : B$, then $\alpha = 1$. Moreover, if $\Gamma \vdash_{\alpha} V : P$, then $\alpha = 1$ and $\Gamma = \Phi$.

The following lemma shows that the type system has the usual substitution property.

**Lemma 3.4 (Substitution).** If $\Gamma_1, x : A, \Gamma'_1 \vdash_{\alpha} M : B$ and $\Gamma_2 \vdash_{\alpha} V : A$, then $\Gamma_1, \Gamma'_1, \Gamma_2 \vdash_{\alpha} [V/x]M : B$.

### 3.1 Interpretation of the typing rules

The modality is a syntactic device to track the dynamic lifting monad $T$. We will interpret $\Gamma \vdash_{\alpha} M : A$ as a map $[\Gamma] \rightarrow [A]$ in $A$, and $\Gamma \vdash_{0} M : A$ as a map $[\Gamma] \rightarrow T[A]$. The modalities in types such as $\alpha - \sigma_{\alpha} B$ and $!_{\alpha}A$ also indicate occurrences of the dynamic lifting monad $T$.

**Definition 3.5.** We interpret types as objects in $A$.

\[
\begin{align*}
[A \rightarrow_{1} B] & = [A] \rightarrow [B] \\
[A \rightarrow_{0} B] & = [A] \rightarrow T[B] \\
[A \otimes B] & = [A] \otimes [B] \\
[!_{1}A] & = p[A] \\
[!_{0}A] & = pT[A] \\
[\text{Circ}(S, U)] & = pA([S], [U]) \\
[\text{Bool}] & = p(1 + 1) \\
[\text{Bit}] & = \text{Bit} \\
[\text{Qubit}] & = \text{Qubit}
\end{align*}
\]

For a parameter type $P$, there exists $X \in V$ such that $[P] = pX$. For a simple type $S$, there exists $Y \in M$ such that $[S] = \psi Y$. We call objects of the form $\psi Y$ *simple objects*. We write $\alpha[A]$ to mean $T[A]$ if $\alpha = 0$, otherwise it is $[A]$. We interpret a context $\Gamma$ as a tensor product of all objects in $\Gamma$ (denoted by $[[\Gamma]]$). The interpretation of parameter context $[[\Phi]]$ is a parameter object and the interpretation of a label context $[[\Sigma]]$ is a simple object. Without loss of generality, we assume that if $[[\Sigma]] = [[\Sigma']]$, then $\Sigma = \Sigma'$ (this condition can always be ensured by making additional isomorphic copies of objects, if necessary).

The interpretation of typing judgements is defined as follows.

**Definition 3.6 (Interpretation).** To each valid typing judgement $\Gamma \vdash_{\alpha} M : A$, we associate a map $[M] : [\Gamma] \rightarrow \alpha[A]$ in $A$, called its *interpretation*. Note that $[[M]]$ here is an abbreviation for $[[\Gamma] \vdash_{\alpha} M : A]$.

The interpretation is defined by induction on the derivation of $\Gamma \vdash_{\alpha} M : A$. Here we show a few cases, the rest are in Appendix C.

- **Case**

\[
\frac{\Gamma \vdash_{\alpha} M : \text{Bit}}{\Gamma \vdash_{0} \text{dynlift } M : \text{Bool}}.
\]

By induction hypothesis, we have $[M] : [\Gamma] \rightarrow \alpha[\text{Bit}]$. If $\alpha = 1$, we define $[\text{dynlift } M]$ by

\[
[\Gamma] \xrightarrow{[M]} [\text{Bit}] \xrightarrow{\text{dynlift}} T[\text{Bool}].
\]

If $\alpha = 0$, we define $[\text{dynlift } M]$ by

\[
[\Gamma] \xrightarrow{[M]} T[\text{Bit}] \xrightarrow{T \text{dynlift}} T[T[\text{Bool}]] \rightarrow T[\text{Bool}].
\]
• Case

\[
\Gamma, x : A \vdash \alpha M : B \\
\Gamma \vdash \lambda x. M : A \to \alpha B.
\]

By induction hypothesis, we have \([M] : \Gamma \otimes [A] \to \alpha B\). Using monoidal closedness, we define \([\lambda x. M] := \text{curry}\([M]\) : \Gamma \to [A] \to [B] \to [B]\).

• Case

\[
\Phi, \Gamma_1 \vdash_{\alpha_1} M : A \to \alpha_B B \\
\Phi, \Gamma_2 \vdash_{\alpha_2} N : A \\
\Phi, \Gamma_1, \Gamma_2 \vdash_{\alpha_1 \cdot \alpha_2 \beta} MN : B.
\]

Here we only consider the case where \(\alpha_1 = \alpha_2 = \beta = 0\). The other cases are similar. By induction hypothesis, we have morphisms \([M] : \Phi \otimes [\Gamma_1] \to T([A] \to T[B])\) and \([N] : \Phi \otimes [\Gamma_2] \to T[A]\). Thus we define \([MN]\) to be the following.

\[
\Phi \otimes [\Gamma_1] \otimes [\Gamma_2] \xrightarrow{\text{dup} \otimes [\Gamma_1] \otimes [\Gamma_2]} \Phi \otimes \Phi \otimes [\Gamma_1] \otimes [\Gamma_2] \xrightarrow{[M] \otimes [N]} T([A] \to T[B]) \otimes T[A]
\]

\[
\xrightarrow{T} TT([A] \to T[B]) \otimes [A] \xrightarrow{T} TT[B] \xrightarrow{\mu} T[B].
\]

• Case

\[
\Phi \vdash_{\alpha} M : A \\
\Phi \vdash \text{lift } M : !_{\alpha} A.
\]

By induction hypothesis, we have \([M] : [\Phi] = pX \to \alpha [A]\) for some \(X \in \mathcal{V}\). By the \(\mathcal{V}\)-adjunction \(\rho \vdash \phi\), we have \(\delta[M] : X \to \beta \alpha [A]\). So we define \([\text{lift } M]\) := \(\rho \delta[M] : pX \to \rho \delta \alpha [A]\).

• Case

\[
\Gamma \vdash_{\beta} M : !_{\alpha} A \\
\Gamma \vdash_{\alpha \cdot \beta} \text{force } M : A.
\]

We only consider the case where \(\alpha = \beta = 0\), the other cases are similar. By induction hypothesis, we have a map \([M] : \Gamma \to T \rho \delta T[A]\). Since there is a \(\mathcal{V}\)-natural transformation \(\text{force} : \rho \delta T[A] \to T[A]\), we define \([\text{force } M]\) by

\[
\Gamma \xrightarrow{[M]} T \rho \delta T[A] \xrightarrow{T \text{force}} TT[A] \xrightarrow{\mu} T[A].
\]

• Case

\[
\Gamma \vdash_{\alpha} M : !_1 (S \to U) \\
\Gamma \vdash_{\alpha} \text{box } S M : \text{Circ}(S, U).
\]

Here we only consider the case \(\alpha = 1\). By induction hypothesis, we have \([M] : \Gamma \to \rho \phi ([S] \to \rho \phi [U])\). We define \([\text{box } SM]\) by

\[
\Gamma \xrightarrow{[M]} \rho \phi ([S] \to \rho \phi [U]) \xrightarrow{\text{box}} \rho \phi ([S], [U]).
\]

• Case

\[
\Phi, \Gamma_1 \vdash_{\alpha} M : \text{Circ}(S, U) \\
\Phi, \Gamma_2 \vdash_{\beta} N : S \\
\Phi, \Gamma_1, \Gamma_2 \vdash_{\alpha \cdot \beta} \text{apply}(M, N) : U.
\]
Here we only consider the case $\alpha = \beta = 0$. By induction hypothesis, we have $[[M]] : [[\Gamma_1]] \rightarrow TpA([[S]], [[U]])$ and $[[N]] : [[\Gamma_2]] \rightarrow T[[S]]$. Thus we define $[[\text{apply}(M, N)]]$ by

$$[[\Phi] \otimes [[\Gamma_1]] \otimes [[\Gamma_2]] \xrightarrow{\text{dup} \otimes [[\Gamma_1]] \otimes [[\Gamma_2]]} [[\Phi] \otimes [[\Phi]] \otimes [[\Gamma_1]] \otimes [[\Gamma_2]] \xrightarrow{[M] \otimes [N]} TpA([[S]], [[U]]) \otimes T[[S]]$$

$$\xrightarrow{T} T(TpA([[S]], [[U]]) \otimes [[S]])$$

$$\xrightarrow{TT} TT(pA([[S]], [[U]]) \otimes [[S]])$$

$$\xrightarrow{T((\text{force} \circ \text{unbox}) \otimes [[S]])} T((([[S]] \rightarrow [[U]]) \otimes [[S]])$$

$$\xrightarrow{T_{\epsilon}} T[[U]]$$.

Our interpretation of the typing rules satisfies the usual semantics substitution theorem. The details of the proof are in Appendix [D].

**Theorem 3.7** (Substitution). If $\Phi, \Gamma_1, x : A, \Gamma_2 \vdash_\alpha M : B$ and $\Phi, \Gamma_3 \vdash_1 V : A$, then

$$[[V/x]M] = [[M]] \circ ([[\Phi] \otimes [[\Gamma_1]] \otimes [[V]] \otimes [[\Gamma_2]]) \circ (\text{dup} \otimes [[\Gamma_1]] \otimes [[\Gamma_2]] \otimes [[\Gamma_3]]) : [\Phi, \Gamma_1, \Gamma_2, \Gamma_3] \rightarrow \alpha[B]$$

The next two theorems show that values of parameter type are in the image of functor $p$, and that values of simple types are isomorphisms.

**Theorem 3.8.** If $\Phi \vdash_1 V : P$, then $[[V]] = pf : pX \rightarrow pY$ for some $f : X \rightarrow Y \in \mathcal{V}$ such that $[[\Phi]] = pX, [[P]] = pY$.

**Theorem 3.9.** Suppose $\Sigma \vdash_1 V : S$, then $[[V]] : [[\Sigma]] \rightarrow [[S]]$ is an isomorphism in $A$.

Since the embedding $\psi : M \hookrightarrow V(A)$ is fully faithful, $[[V]] : [[\Sigma]] \rightarrow [[S]]$ is also an isomorphism in $M$.

### 4 Operational semantics and soundness

In this section, we will specify an operational semantics for Proto-Quipper-Dyn and show that it is sound with respect to the $\mathcal{V}$-enriched categorical model $A$ for dynamic lifting.

We distinguish two kinds of evaluation in order to model Quipper’s two runtimes. The evaluation rules for circuit generation time will work with morphisms in $M$, i.e., quantum circuits. On the other hand, the evaluation rules for circuit execution time will work with morphisms in $Q$, i.e., quantum operations. Because of the embeddings $\psi : M \hookrightarrow V(A)$ and $\phi : Q \hookrightarrow Kl_{VT}(V(A))$, we are able to interpret the configurations for these two runtimes as maps in the $\mathcal{V}$-enriched category $A$.

#### 4.1 Operational semantics for circuit generation time

First of all, we specify the meaning of appending circuits in the category $M$.

**Definition 4.1** (Circuit append). Suppose $C : \Sigma \rightarrow \Sigma_1, \Sigma_2$ and $D : \Sigma_1' \rightarrow \Sigma_3$ are morphisms in $M$ and there are typing judgments $\Sigma_1 \vdash_1 V : S$ and $\Sigma_1' \vdash_1 V' : S$. We define $\text{append}(D, C, V', V)$ to be the following morphism in $M$.

$$((D \circ [V']^{-1} \circ [V]) \otimes [\Sigma_2]) \circ C : \Sigma \rightarrow \Sigma_3, \Sigma_2$$

Thus $\text{append}(D, C, V', V)$ is the result of appending the circuit $D$ to $C$ by connecting the interfaces $V'$ and $V$. The following are evaluation rules for circuit generation time, where the underlying states are given by morphisms in $M$. 

13
Definition 4.2 (Circuit generation time evaluation).

\[
\begin{align*}
(C, M) \Downarrow (C_2, \lambda x. M') & \quad (C_1, M) \Downarrow (C_2, (a, D, b)) \\
(C_2, N) \Downarrow (C_3, V) & \quad (C_2, N) \Downarrow (C_3, V) \\
(C_2. [V/x] M') \Downarrow (C_2', V') & \quad \text{append}(D, C_3, a, V) = C' \quad \text{apply} \\
(C_1, M N) \Downarrow (C_4, V') & \quad (C_1, \text{apply}(M, N)) \Downarrow (C_2', b) \\
(C', \text{lift} M') & \quad (C', M') \Downarrow (C'', V) \\
(C, \text{force} M) \Downarrow (C'', V) & \\
\end{align*}
\]

In the rule \textit{box}, we use \text{gen}(S) = \text{a} to mean that the \text{a} is a fresh simple term of type \text{S}. Note that the evaluation of \((C, M) \Downarrow (C', V)\) does not account for dynamic lifting, and the underlying states are circuits. So it is the same set of evaluation rules as in [RS18]. The evaluation comes with the following notion of configuration.

Definition 4.3 (Well-typed circuit rules). We write \(\Sigma \vdash (C, M) : A; \Sigma'\) to mean there exists \(\Sigma''\) such that \(C : \Sigma \rightarrow \Sigma', \Sigma''\) and \(\Sigma'' \vdash_1 M : A\).

A well-typed circuit configuration requires a typed term with modality 1, i.e., \(\vdash_1 M : A\). It is a runtime error if a term with dynamic lifting is encountered when using the evaluation rules in Definition 4.2. Our type system and the following type preservation theorem ensures that this cannot happen.

Theorem 4.4. If \(\Sigma \vdash (C, M) : A; \Sigma'\) and \((C, M) \Downarrow (C', V)\), then \(\Sigma \vdash (C', V) : A; \Sigma'\).

In the following we define the interpretation \([C, M]\) as a map in the \(\mathcal{V}\)-category \(A\).

Definition 4.5. Suppose \(\Sigma \vdash (C, M) : A; \Sigma'\). We have maps \(\psi_C : [\Sigma] \rightarrow [\Sigma'] \otimes [\Sigma'']\) and \([M] : [\Sigma''] \rightarrow [A]\) in \(A\). We define \([C, M]\) as follows:

\[
[\Sigma] \xrightarrow{\psi_C} [\Sigma'] \otimes [\Sigma''] \xrightarrow{[\Sigma'] \otimes [M]} [\Sigma'] \otimes [A].
\]

The following theorem shows that the evaluation rules for circuit generation time are sound with respect to the categorical model \(A\). Since in this case dynamic lifting cannot occur, the proof is similar to the one in [RS18].

Theorem 4.6. If \(\Sigma \vdash (C, M) : A; \Sigma'\) and \((C, M) \Downarrow (C', V)\), then \([C, M] = [C', V]\).

4.2 Operational semantics for circuit execution time

Since dynamic lifting requires the ability to access the states in \(Q\), we first define the concepts of state and addresses.

Definition 4.7 (State and addresses). For any object \(S \in Q\), a state is a morphism \(Q : I \rightarrow S \in Q\). We write \(\text{addr}(Q) = \Sigma\) if \(\phi(S) = [\Sigma]\), we call \(\Sigma\) the addresses of \(Q\). (Recall that we have, for convenience and without loss of generality, assumed that the interpretation function \([-\] is one-to-one on label contexts).

We often write \(Q : I \rightarrow \Sigma \in Q\) for \(Q : I \rightarrow S\), where \(\phi(S) = [\Sigma]\). The following \text{read} operation will be used to define the operational semantics for dynamic lifting.

Definition 4.8 (Read operation). Suppose \(\text{addr}(Q) = \Sigma, \ell : \text{Bit}\) and \(Q = p_1(Q_1 \otimes \text{inj}_1) + p_2(Q_2 \otimes \text{inj}_2)\), where \(\text{addr}(Q_1) = \text{addr}(Q_2) = \Sigma\) and \(p_1, p_2 \in [0, 1]\) and \(p_1 + p_2 = 1\). We define a formal sum \(\text{read}(Q, \ell) = p_1(Q_1, \text{False}) + p_2(Q_2, \text{True})\), where \(\text{False, True : Bool}\).
Note that by the last condition in Assumption 2.5, we know that \( Q = p_1(Q_1 \otimes \text{inj}_1) + p_2(Q_2 \otimes \text{inj}_2) : I \rightarrow \Sigma \otimes \text{Bit} \) for some essentially uniquely determined \( Q_1, Q_2 : I \rightarrow \Sigma \), and \( p_1, p_2 \in [0,1] \) such that \( p_1 + p_2 = 1 \). The only time \( Q_i \) is not uniquely determined is when \( p_i = 0 \), but in this case, it will turn out that the \( Q_i \) does not matter since it corresponds to a branch of computation taken with probability zero. In this case, we can just make some fixed but arbitrary choice for \( Q_i \). So the read operation makes the information of the probabilities \( p_1, p_2 \) and the states \( Q_1, Q_2 \) available.

In the following, we define the circuit execution time counterpart of Definition 4.1. It specifies the meaning of updating a quantum state by applying a quantum circuit, where the identity-on-object interpretation functor \( J : M \rightarrow Q \) is needed for the definition.

**Definition 4.9.** Suppose \( Q : I \rightarrow \Sigma_1, \Sigma_2 \) is a morphism in \( Q \), and \( C : \Sigma'_1 \rightarrow \Sigma_3 \) is a morphism in \( M \), and there are typing judgements \( \Sigma_1 \vdash V : S \) and \( \Sigma'_1 \vdash V' : S' \). We define \( \text{operate}(C, Q, V', V) \) to be the following map in \( Q \):

\[
(J(C \circ [V']^{-1} \circ [V]) \otimes [\Sigma_2]) \circ Q : I \rightarrow \Sigma_3, \Sigma_2.
\]

We now define the operational semantics for circuit execution time. The underlying states of the evaluation are the states in \( Q \). The evaluation is of the form \( (Q, M) \downarrow \sum_{i \in [n]} p_i(Q_i, V_i) \). Its intuitive meaning is that the configuration \((Q, M)\) can be reduced to \((Q_i, V_i)\) with probability \( p_i \). The notation \( \sum_{i \in [n]} p_i(Q_i, V_i) \) is a short hand for the formal sum \( p_1(Q_1, V_1) + \ldots + p_n(Q_n, V_n) \), and we assume \( \sum_{i \in [n]} p_i = 1 \). Write \( [n] = \{1, \ldots, n\} \).

**Definition 4.10** (Operational semantics for circuit execution time).

\[
\begin{align*}
(Q, M) & \downarrow \sum_{i \in [n]} p_i(Q_i, \lambda x. M'_i) & (Q, N) & \downarrow \sum_{j \in [m]} q_{i,j}(Q'_{i,j}, V_{i,j}) & (Q, M') & \downarrow \sum_{i \in [n]} p_i(Q_i, \text{lift } M'_i) & (Q, N) & \downarrow \sum_{j \in [m]} q_{i,j}(Q'_{i,j}, V_{i,j}) \\
(Q, M N) & \downarrow \sum_{(i,j,k) \in [n] \times [m] \times [k]} \prod_{i,j,k} p_{i,j,k}(Q''_{i,j,k}, V'_{i,j,k}) & (Q, M') & \downarrow \sum_{j \in [m]} q_{i,j}(Q'_{i,j}, V_{i,j}) & (Q, N) & \downarrow \sum_{j \in [m]} q_{i,j}(Q'_{i,j}, V_{i,j}) & \text{apply} & (Q, S M) & \downarrow \sum_{i \in [n]} p_i(Q_i, (a, D_i, b_i)) \\
(Q, M) & \downarrow \sum_{i \in [n]} p_i(Q_i, (a, D_i, b_i)) & (Q, N) & \downarrow \sum_{j \in [m]} q_{i,j}(Q'_{i,j}, V_{i,j}) & (Q, M') & \downarrow \sum_{i \in [n]} p_i(Q_i, \text{lift } M'_i) & (Q, N) & \downarrow \sum_{j \in [m]} q_{i,j}(Q'_{i,j}, V_{i,j}) & \text{box} & \text{apply} & (Q, N) & \downarrow \sum_{i \in [n]} p_i(Q_i, \lambda x. M'_i) \\
(Q, M) & \downarrow \sum_{i \in [n]} p_i(Q_i, \text{force } M'_i) & (Q, \text{gen}(S)) & = a & (Q, N) & \downarrow \sum_{i \in [n]} p_i(Q_i, \lambda x. M'_i) & (Q, M) & \downarrow \sum_{i \in [n]} p_i(Q_i, \lambda x. M'_i) & (Q, M') & \downarrow \sum_{i \in [n]} p_i(Q_i, \text{force } M'_i) \\
(Q, \text{dynlift } M) & \downarrow \sum_{(i,j) \in [n] \times [m]} \prod_{i,j} p_{i,j}(Q'_{i,j}, a_{i,j}) & (Q, \text{let } (x, y) = (N \text{ in } M)) & \downarrow \sum_{(i,j) \in [n] \times [m]} (Q''_{i,j}, V''_{i,j}) & (Q, \text{let } (x, y) = (N \text{ in } M)) & \downarrow \sum_{(i,j) \in [n] \times [m]} (Q''_{i,j}, V''_{i,j}) & (Q, \text{let } (x, y) = (N \text{ in } M)) & \downarrow \sum_{(i,j) \in [n] \times [m]} (Q''_{i,j}, V''_{i,j})
\end{align*}
\]

In the apply rule, we use \( \text{operate} \) instead of \( \text{append} \), which allows a quantum circuit to be applied as a quantum operation. In the \( \text{dynlift} \) rule, for each \( (Q_i, \ell_i) \), we apply the operation \( \text{read}(Q_i, \ell_i) \), which gives rise to two possible outcomes \( (Q'_{i,1}, a_{i,1}), (Q'_{i,2}, a_{i,2}) \) with probabilities \( q_{i,1}, q_{i,2} \), where \( a_{i,1}, a_{i,2} : \text{Bool} \) and \( a_{i,1} \neq a_{i,2} \). This is the only rule that gives rise to probabilistic results in the evaluation. In the box rule, the evaluation of \( (\text{Id}_S, M'_i a) \) uses the rules defined in Definition 4.2 so it is performed at circuit generation time.

We now define a well-typed configuration for evaluating a term under a quantum state.

**Definition 4.11** (Well-typed configuration). We write \( \vdash Q, M : A ; \Sigma' \) to mean there exists \( \Sigma'' \) such that \( \Sigma'' \vdash_\alpha M : A \), and \( \text{addr}(Q) = \Sigma'', \Sigma' \).

Since the evaluation rules in Definition 4.10 account for dynamic lifting, the above configuration allows the term \( M \) to have modality 0. The operational semantics defined in Definition 4.10 is type-safe in the following sense.
Theorem 4.12. If \( \vdash_\alpha (Q, M) : A; \Sigma' \) and \( (Q, M) \Downarrow \sum_{i \in [n]} p_i(Q_i, V_i) \), then \( \vdash_1 (Q_i, V_i) : A; \Sigma' \) for all \( i \in [n] \).

Theorem 4.13. If \( \vdash_1 (Q, M) : A; \Sigma' \) and \( (Q, M) \Downarrow \sum_{i \in [n]} p_i(Q_i, V_i) \), then \( n = 1 \). In other words, we actually have \( (Q, M) \Downarrow (Q', V) \).

In the following we interpret a well-typed configuration \( \vdash_\alpha (Q, M) : A; \Sigma' \) as a map in the Kleisli category \( \mathcal{K}_T(A) \).

Definition 4.14. Suppose \( \vdash_\alpha (Q, M) : A; \Sigma' \). We have \( \phi Q : \alpha \rightarrow T([\Sigma'_1] \otimes [\Sigma''] \otimes [\Sigma'_2]) \) and \( M : [\Sigma''] \rightarrow \alpha[A] \) in \( A \). We define \([Q, M]\) by:

- If \( \alpha = 1 \), then
  \[
  I \xrightarrow{\phi Q} T([\Sigma'_1] \otimes [\Sigma''] \otimes [\Sigma'_2]) \xrightarrow{T(M \otimes [\Sigma'_2])} T([\Sigma'_1] \otimes [A] \otimes [\Sigma'_2]).
  \]

- If \( \alpha = 0 \), then
  \[
  I \xrightarrow{\phi Q} T([\Sigma'_1] \otimes [\Sigma''] \otimes [\Sigma'_2]) \xrightarrow{T(M \otimes [\Sigma'_2])} T([\Sigma'_1] \otimes [A] \otimes [\Sigma'_2]) \xrightarrow{T(T \otimes [\Sigma'_2])} T(T([\Sigma'_1] \otimes [A] \otimes [\Sigma'_2])) \xrightarrow{T(T \otimes [\Sigma'_2])} T([\Sigma'_1] \otimes [A] \otimes [\Sigma'_2]).
  \]

The following theorem shows that the operational semantics in Definition 4.10 is sound with respect to the semantic model \( A \).

Theorem 4.15 (Soundness). If \( \vdash_\alpha (Q, M) : A; \Sigma', \) and \( (Q, M) \Downarrow \sum_{i \in [n]} p_i(Q_i, V_i) \), then \([Q, M] = \sum_{i \in [n]} p_i[[Q_i, V_i]] : I \rightarrow T([\Sigma'] \otimes [\Sigma''])\).

Proof sketch. The proof is by induction on the evaluation rules. Here we focus on the case for dynamic lifting. Please see Appendix E for the proofs of the other cases.

Suppose \( \text{addr}(Q) = \Sigma'', \Sigma' \), and \( \Sigma'' \vdash_0 M : A \), and

\[
\Sigma'' \vdash_0 \text{dynlift} \; M : \text{Bit}.
\]

Consider the following.

\[
(Q, M) \Downarrow (Q', \ell)
\]

\[
\text{read}(Q', \ell) = q_1(Q'_1, \text{False}) + q_2(Q'_2, \text{True})
\]

\[
(Q, \text{dynlift} \; M) \Downarrow q_1(Q'_1, \text{False}) + q_2(Q'_2, \text{True})
\]

Since \( \text{read}(Q', \ell) = q_1(Q'_1, \text{False}) + q_2(Q'_2, \text{True}) \) implies that \( Q' = q_1(Q'_1 \otimes \text{inj}_1) + q_2(Q'_2 \otimes \text{inj}_2) \) in \( Q \), we have the following in \( A \).

\[
\phi Q' = q_1(\mu \circ Tt \circ s \circ (\phi Q'_1 \otimes \phi(\text{inj}_1))) + q_2(\mu \circ Tt \circ s \circ (\phi Q'_2 \otimes \phi(\text{inj}_2))),
\]

where \( \phi Q' : I \rightarrow T(\text{Bit} \otimes [\Sigma'']) \), and \( \phi Q'_1, \phi Q'_2 : I \rightarrow T[\Sigma'] \), and \( \phi(\text{inj}_1), \phi(\text{inj}_2) : I \rightarrow T\text{Bit} \). Note that by condition (g), we have \( \phi(\text{inj}_1) = \eta \circ \text{init} \circ \text{[False]} \) and \( \phi(\text{inj}_2) = \eta \circ \text{init} \circ \text{[True]} \). We need to show that

\[
\llbracket Q, \text{dynlift} \; M \rrbracket = q_1(T([\text{False}] \otimes [\Sigma'']) \circ \phi Q'_1) + q_2(T([\text{True}] \otimes [\Sigma'']) \circ \phi Q'_2).
\]

By induction hypothesis, we have \([Q, M] = [Q', \ell]\), i.e., \( T([M] \otimes [\Sigma']) \circ \phi Q = \phi Q' \). Thus

\[
\llbracket Q, \text{dynlift} \; M \rrbracket = \mu \circ Ts \circ T((\text{dynlift} \circ [M]) \otimes [\Sigma']) \circ \phi Q.
\]
\[ T(\text{Bit} \otimes [\Sigma']) \stackrel{t}{\longrightarrow} T(\text{Bit} \otimes [\Sigma']) \]

\[ T(\text{Bit} \otimes [\Sigma']) \longrightarrow T(T\text{Bit} \otimes [\Sigma']) \longrightarrow T(T\text{Bool} \otimes [\Sigma']) \]

\[ T([\text{False}] \otimes T[\Sigma']) \longrightarrow T([\text{False}] \otimes T[\Sigma']) \]

We just need to show

\[ T([\text{False}] \otimes [\Sigma']) \circ \phi Q' \]

This is true because of the commutative diagram in Figure 3.

**Remark.** In practice, a closed term \( M \) is always evaluated with the initial configuration \((\text{Id}_{1}, M)\), where \( \text{Id}_{1} : I \rightarrow I \) is a state in \( \mathbb{Q} \). When \( M \) has modality 0, we would need access to a quantum computer/simulator in order to evaluate \((\text{Id}_{1}, M)\) and each run of \((\text{Id}_{1}, M)\) could give a different value. When \( M \) has modality 1, the evaluation of \((\text{Id}_{1}, M)\) is deterministic, i.e., the top-level quantum state is updated in a deterministic fashion. In this case, instead of performing the quantum operations, we could also just generate a list of gates, which can be done entirely in a classical computer.
5 Dynamic lifting in Proto-Quipper

While typing judgments and certain types \((A \multimap \alpha B \text{ and } !\alpha A)\) are annotated with a modality, information about this modality is meant to be hidden from the programmer unless an error occurs. For example, if one attempt to box a function which uses dynamic lifting, the type checker will raise a modality error. As a result, such programming errors are caught at compile time in Proto-Quipper-Dyn, whereas they are only caught at runtime in Quipper.

The modality inference can be readily integrated into bi-directional type checking \([PT00]\), which features a pair of recursively defined functions for type checking and type inference. To work with modalities, the type checking function not only takes a term and a type as inputs, but also the current modality of the typing judgment. For example, when checking a term \(\lambda x.M\) against a type \(A \multimap \alpha B\) with current modality \(\beta\), the type checking function first ensures that the current modality \(\beta\) is 1, then extends the current typing environment with \(x : A\) and recursively checks the term \(M\) against the type \(B\), with the modality \(\alpha\). The type inference function takes a term as input and outputs the inferred type as well as the inferred modality. For example, when inferring the type for a term \(MN\), the type inference function first infers a type \(A \multimap \alpha B\) and a modality \(\beta\) for \(M\), and then it infers a type \(A\) and a modality \(\gamma\) for the term \(N\), so the type inference function will return the type \(B\) and the inferred modality \(\alpha \& \beta \& \gamma\).

We now discuss several Proto-Quipper-Dyn programs that make use of dynamic lifting.

5.1 Quantum teleportation

The following circuit implements a one-qubit quantum teleportation protocol.

```
0  H  Meas  
0  H  Meas  
X  Z
```

This circuit is generated by the following Proto-Quipper-Dyn programs.

```
Listing 1: Alice and Bob circuits
alice1 : !(Qubit -> Qubit -> Bit * Bit)
alice1 a q =
  let (a, q) = CNot a q
  q = H q
in (Meas a, Meas q)

bob1 : !(Qubit -> Bit -> Bit -> Qubit)
bob1 q x y =
  let (q, x) = C_X q x
  (q, y) = C_Z q y
  = Discard x
  = Discard y
in q

Listing 2: Teleportation circuit
bell00 : !(Unit -> Qubit * Qubit)
bell00 =
  let a = Init0 ()
b = Init0 ()
in CNot b (H a)
tele1 : !(Qubit -> Qubit)
tele1 q =
  let (b, a) = bell00 ()
  (x, y) = alice1 a q
  z = bob1 b x y
  in z

boxTele : Circ(Qubit, Qubit)
boxTele = box Qubit tele1
```

As can be seen in Listings 1 and 2, the modality information is not visible to the programmer. Because the programs in Listings 1 and 2 do not use dynamic lifting, the modalities in the fully annotated types are all 1. For example, the fully annotated type of `tele1` is \(l_1(Qubit \multimap l_1 Qubit)\). We can therefore box `tele1` into a quantum circuit. The evaluation of `boxTele` occurs on a classical computer and generates the circuit diagram above.

For comparison, let us consider the following Proto-Quipper-Dyn programs that implement quantum teleportation using dynamic lifting.
As before, the code in Listings 3 and 4 contains no modality annotations. In the `alice2` function, dynamic lifting is used right after the measurement gate `Meas : Qubit → Bit`. Accordingly, the fully annotated type of `alice2` is `!(Qubit → Qubit → Bool × Bool)`. The `bob2` function then uses if-then-else expressions to decide whether to apply the gates `QNot` and `ZGate`, rather than applying the bit-controlled gates `C_0X` and `C_0Z`, as in the `bob1` function in Listing 1.

The `tele2` function calls the `bob2` function with the booleans provided by the `alice2` function. Hence, the `tele2` function implicitly uses dynamic lifting. Its fully annotated type is `!(Qubit → Bool → Bool → Qubit)`. Because of the modality inference, the type checker will issue a typing error for the `boxAttempt` function. According to the typing rule for `box`, the `box Qubit` function requires an argument of type `!(Qubit ⊸ Qubit)`, which is distinct from the type of `tele2`. This error is sensible because the `tele2` function does not correspond to a circuit.

The `test` function applies `tele2` to an input qubit in the `|0⟩` state. The output value of `test` should then be `False` with probability 1. Note that the evaluation of `test` requires access to a quantum computer or a simulator.

### 5.2 Magic state distillation

Magic states are quantum states that can be used, in conjunction with Clifford gates, to perform universal quantum computing fault tolerantly [BK05]. For example, there is a standard method to implement a $T$ gate using the magic state $(|0⟩ + e^{\pi i/4} |1⟩)/\sqrt{2}$, along with Clifford gates and measurements. This enables the application of any operation from the Clifford+$T$ gate set, a well-known universal set of quantum gates [NC02].

The process of producing a magic state such as $(|0⟩ + e^{\pi i/4} |1⟩)/\sqrt{2}$ from several imperfect states is called **magic state distillation** [BK05]. In order to distill a magic state $|M⟩$, one first prepares several qubits in a state that approximates $|M⟩$ up to an error rate $\epsilon$. A carefully designed quantum circuit is then applied to these qubits and some of them are measured. If all of the measurement results are 0, then the remaining qubits are guaranteed to be in a state that approximates $|M⟩$ up to an improved error rate $\epsilon' < \epsilon$. If any one of the measurement results is 1, then all of the qubits are discarded and the entire process is restarted. In practice, several rounds of distillation are required to obtain a state that approximates $|M⟩$ up to an acceptable error rate.

A Proto-Quipper-Dyn implementation of Bravyi and Kitaev’s distillation algorithm is given in Listing 5. In the `distill` function, we first apply a five-qubit error correction circuit `fiveQubits` to the inputs, then measure the qubits and, through dynamic lifting, promote the resulting bits to booleans. If all of the booleans are `False`, the distillation was successful and we return the remaining qubit. Otherwise the distillation failed, so we discard the unmeasured qubit and return nothing. Dynamic lifting is essential for defining the `distill` function because in this case the if-then-else expression cannot be implemented as a circuit.
The distillation function performs \( n \) rounds of magic state distillation. The function prepMixedState prepares an initial imperfect state. The distillation function is a recursive function that assumes five successful distillations from the previous round and then applies the distill function to the resulting qubits. If that function returns a qubit, the \( n \)-th round of distillation was successful, otherwise it will restart the whole process.

### 5.3 Repeat-Until-Success

The repeat-until-success paradigm provides a technique to apply a unitary that cannot be implemented exactly, at the cost of potentially running the same circuit multiple times. In order to apply a non-Clifford+T gate \( N \) to a target qubit \( |\phi\rangle \), one first initializes several ancillary qubits before applying a well-chosen Clifford+T circuit \( C \) to the target and the ancillas and measuring the ancillas. If all of the measurement results are 0, the target qubit is guaranteed to be in the state \( N|\phi\rangle \). Otherwise, a correction is applied to the target to return it to its initial state and the process is repeated.

Consider the following circuit used in [PS13] to illustrate the implementation of the gate \( V_3 = \frac{I + 2iZ}{\sqrt{5}} \) using the repeat-until-success method.

The top wire is the target qubit, while the wires below it are the ancillas. We apply a sequence of gates \( (H, H, T^*, CNOT, T, \text{ and } H) \) to the ancillas before measuring the first ancilla. If, as we assume here, the measurement result is 0, then we apply a sequence of gates \( (T, Z, CNOT, T, \text{ and } H) \) to the target qubit and the second ancilla before measuring the second ancilla. Assuming, again, that the measurement result is 0, we then know that the target qubit is in the desired state. Note that the circuit above is not a representation of the entire repeat-until-success protocol. Instead, it is the circuit constructed in the event that both
measurement results are 0 (which can be shown to occur with probability 5/8). If the measurements yield different results, the circuit constructed by the repeat-until-success protocol is different. For example, if the result of the second measurement is 1, a $Z$ gate must be applied to the target qubit to return it to its initial state.

Listing 6 gives a precise description of the implementation of $V_3$ in Proto-Quipper-Dyn.

### Listing 6: A repeat-until-success example

```haskell
v3 : !(Qubit -> Qubit)
v3 q =
  let a1 = tgate_inv (H (Init0 ()))
  a2 = H (Init0 ())
  (a1, a2) = CNot a1 a2
  a1 = H (TGate a1)
  in if dynlift (Meas a1)
    then
      let _ = Discard (Meas a2)
      in v3 q
    else let q = ZGate (TGate q)
      (a2, q) = CNot a2 q
      a2 = H (TGate a2)
      in if dynlift (Meas a2)
        then v3 (ZGate q)
        else q
```

Once again, dynamic lifting plays an essential role here. Note that the $v3$ function has type $!_1(Qubit \rightarrow_0 Qubit)$; it is a quantum computation, rather than a quantum circuit.

### 6 Conclusion

We have given an axiomatization of an enriched categorical semantics for Proto-Quipper with dynamic lifting. We defined a type system with a modality to keep track of functions that use dynamic lifting. The main benefit of our type system is that it statically prevents a class of run-time errors caused by applying the boxing operation to a function that uses dynamic lifting. We also gave an operational semantics for dynamic lifting. The operational semantics models both circuit generation and circuit execution. We also defined an abstract categorical semantics for this language and proved that the type system and the operational semantics are sound with respect to it. Lastly, we gave some examples of quantum algorithms that rely on dynamic lifting.

### Acknowledgements

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


A Diagrams for strong monoidal \(\mathcal{V}\)-functors

Definition A.1. Suppose \(\mathcal{A}, \mathcal{B}\) are \(\mathcal{V}\)-categories. A \(\mathcal{V}\)-functor \(F : \mathcal{A} \to \mathcal{B}\) is strong monoidal if the following hold.

- In \(\mathcal{B}\), there is an isomorphism \(e : I \to FI\) and a \(\mathcal{V}\)-natural isomorphism \(m : FA \otimes FB \to F(A \otimes B)\).
- For all \(A, B, C \in \mathcal{A}\), the following diagrams commute.

\[
\begin{array}{ccc}
(FA \otimes FB) \otimes FC & \xrightarrow{\alpha} & FA \otimes (FB \otimes FC) \\
\downarrow m \otimes FC & & \downarrow F \otimes m \\
F(A \otimes B) \otimes FC & \xrightarrow{m} & FA \otimes F(B \otimes C) \\
\downarrow m & & \downarrow m \\
F((A \otimes B) \otimes C) & \xrightarrow{F \alpha} & F(A \otimes (B \otimes C)) \\
\end{array}
\]

\[
\begin{array}{ccc}
I \otimes FA & \xrightarrow{\ell \otimes FA} & FI \otimes FA \\
\downarrow l & & \downarrow m \\
FA & \xrightarrow{FI} & F(I \otimes A) \\
\end{array}
\]

\[
\begin{array}{ccc}
FA \otimes I & \xrightarrow{FA \otimes \ell} & FA \otimes FI \\
\downarrow r & & \downarrow m \\
FA & \xleftarrow{Fr} & F(A \otimes I) \\
\end{array}
\]

B Convexity

Let \([0,1]\) denote the real unit interval.

Definition B.1. A convexity structure on a set \(X\) is an operation that assigns to all \(p, q \in [0,1]\) with \(p + q = 1\) and all \(x, y \in X\) an element \(px + qy \in X\), subject to the following properties. Throughout, we assume \(p + q = 1\).

(a) \(px + qx = x\) for all \(x \in X\).
(b) \(px + qy = qy + px\) for all \(x, y \in X\).
(c) \(0x + 1y = y\) for all \(x, y \in X\).
In the literature, we sometimes find a different, but equivalent condition of the form

\[ \text{Definition C.1.} \]

\[ \text{Case Φ} \]

Suppose \( \alpha \llbracket \)

\[ \text{We define} \]

\[ \text{Note that} \]

\[ \text{morphism in} \]

\[ \text{We often expand the binary + operation to a multi-arity operation, i.e.,} \sum_i p_i x_i, \text{where} \sum_i p_i = 1 \text{and all denominators are non-zero.} \]

We show a few non-trivial cases.

- Case

\[ \Phi, x : A \vdash_1 x : A. \]

We define \([x]\) as

\[ \Phi \otimes [A] \xrightarrow{\text{discard} \otimes [A]} [A]. \]

- Case

\[ \Phi, \Gamma_1 \vdash_1 M : A \quad \Phi, \Gamma_2 \vdash_1 N : B \]

\[ \Phi, \Gamma_1, \Gamma_2 \vdash_1 M, N : A \otimes B. \]

Suppose \( \alpha_1 = \alpha_2 = 0 \). The other cases are similar. By induction hypothesis, we have \([M] : [\Phi] \otimes [\Gamma_1] \rightarrow T[A]\) and \([N] : [\Phi] \otimes [\Gamma_2] \rightarrow T[B]\). So we define \([M, N]\) as the following composition

\[ [\Phi] \otimes [\Gamma_1] \otimes [\Gamma_2] \xrightarrow{\text{dup} \otimes [\Gamma_1] \otimes [\Gamma_2]} [\Phi] \otimes [\Phi] \otimes [\Gamma_1] \otimes [\Gamma_2] \xrightarrow{[(M) \otimes [N]]} T[A] \otimes T[B] \]

\[ \xrightarrow{\text{let}} T(T[A] \otimes [B]) \xrightarrow{Tx} TT([A] \otimes [B]) \xrightarrow{\text{let}} T([A] \otimes [B]). \]

- Case

\[ \Sigma_1 \vdash_1 a : S \quad \Sigma_2 \vdash_1 b : U \]

\[ C : \Sigma_1 \rightarrow \Sigma_2 \]

\[ \Phi \vdash_1 (a, C, b) : \text{Circ}(S, U) \]

Note that \([a] : [\Sigma_1] \rightarrow [S]\) and \([b] : [\Sigma_2] \rightarrow [U]\) are isomorphisms. Moreover, \(C : [\Sigma_1] \rightarrow [\Sigma_2]\) is a morphism in \(A\). We write \(\theta_C = [\theta] \circ C \circ [a]^{-1} : [S] \rightarrow [U]\). Thus we have \(\text{curry}(\theta_C) : I \rightarrow [S] \rightarrow [U]\), and \(\delta(\text{curry}(\theta_C)) : 1 \rightarrow \delta([S] \rightarrow [U])\) for the adjunct mate of \(\text{curry}(\theta_C)\). Thus we define \([\langle a, C, b \rangle]\) as

\[ [\Phi] \xrightarrow{\text{discard}} I = p1 \xrightarrow{p\delta(\text{curry}(\theta_C))} p0([S] \rightarrow [U]) \xrightarrow{\text{box}} pA([S], [U]). \]

- Case “let”. We only consider the following; the other cases are similar.

\[ \Phi, \Gamma_1, x : P, y : B \vdash_0 M : C \quad \Phi, \Gamma_2 \vdash_0 N : P \otimes B \]

\[ \Phi, \Gamma_1, \Gamma_2 \vdash_0 \text{let} (x, y) = N \text{ in } M : C \]

Note that \(a, b, c, d \in [0, 1]\) with \(a + b + c + d = 1\) and all denominators are non-zero.

**Remark.** Property (d) can best be understood by realizing that both sides of the equation are equal to \(ax + by + cz + dw\), decomposed in two different ways into convex sums of two elements at a time. In the literature, we sometimes find a different, but equivalent condition of the form \(s(px + qy) + rz = spx + (qs + r)(\frac{qs}{qs + r} + \frac{r}{qs + r})z\). The latter axiom is arguably shorter, but harder to read.
By induction hypothesis, we have \([M] : [\Phi] \otimes [\Gamma_1] \otimes [P] \otimes [B] \rightarrow T[C]\) and \([N] : [\Phi] \otimes [\Gamma_2] \rightarrow T([P] \otimes [B]).\) Thus we define \([\text{let } (x, y) = N \text{ in } M]\) as the following composition.

\[
\begin{align*}
[\Phi] & \otimes [\Gamma_1] \otimes [\Gamma_2] & \xrightarrow{\text{dup}} & [\Phi] \otimes [\Gamma_1] \otimes [\Gamma_2] \\
& \otimes [\Gamma_1, \otimes [\Gamma_2] & \xrightarrow{T} & [\Phi] \otimes [\Gamma_1] \otimes T([P] \otimes [B]) \\
& \xrightarrow{T} & T([\Phi] \otimes [\Gamma_1] \otimes [P] \otimes [B]) & \overset{T[M]}{\rightarrow} T[C].
\end{align*}
\]

## D Proof of Theorem 3.7

### Theorem D.1 (Substitution)

If \(\Phi, \Gamma_1, x : A, \Gamma_2 \vdash_{\alpha} M : B\) and \(\Phi, \Gamma_3 \vdash_{1} V : A\), then

\([[[V/x]M] = [M] \circ ([\Phi] \otimes [\Gamma_1] \otimes [V] \otimes [\Gamma_2]) \circ (\text{dup} \otimes [\Gamma_1] \otimes [\Gamma_2] \otimes [\Gamma_3]) : [\Phi, \Gamma_1, \Gamma_2, \Gamma_3] \rightarrow \alpha[B].\)

**Proof.** By induction on \(\Phi, \Gamma_1, x : A, \Gamma_2 \vdash_{\alpha} M : B\). Here we show a few nontrivial cases.

- **Case**

  \[
  \Phi, \Gamma_1, x : A, \Gamma_2 \vdash_{\alpha} M : \text{Bit}
  \]

  Suppose \(\Phi, \Gamma_2 \vdash_{1} V : A\) and \(\alpha = 0\). By induction hypothesis, we have

  \[
  [[[V/x]M] = [M] \circ ([\Phi] \otimes [\Gamma_1] \otimes [V] \otimes [\Gamma_2]) \circ (\text{dup} \otimes [\Gamma_1] \otimes [\Gamma_2])
  \]

  \[
  : [\Phi] \otimes [\Gamma_1] \otimes [\Gamma_2] \rightarrow T[\text{Bit}].
  \]

  Since \(\Phi, \Gamma_1, \Gamma_2, \Gamma_3 \vdash_{0} \text{dynlift}[V/x]M : \text{Bool},\) we have

  \[
  [\text{dynlift}[V/x]M] = \mu \circ T \text{dynlift} \circ [[[V/x]M]
  \]

  \[
  = (\mu \circ T \text{dynlift} \circ [M]) \circ ([\Phi] \otimes [\Gamma_1] \otimes [V] \otimes [\Gamma_2]) \circ (\text{dup} \otimes [\Gamma_1] \otimes [\Gamma_2])
  \]

  \[
  : [\Phi] \otimes [\Gamma_1] \otimes [\Gamma_2] \otimes [\Gamma_3] \rightarrow T[\text{Bool}].
  \]

- **Case**

  \[
  \Phi, x : P, \Phi' \vdash_{\alpha} M : A
  \]

  \[
  \Phi, x : P, \Phi' \vdash_{1} \text{lift } M : !_{\alpha} A.
  \]

  By Lemma 3.3 we have \(\Phi \vdash_{1} V : P\). Since \(\Phi, \Phi' \vdash_{\alpha} [V/x]M : A\), by induction hypothesis, we have

  \[
  [[[V/x]M] = [M] \circ ([\Phi] \otimes [V] \otimes [\Phi']) \circ (\text{dup} \otimes [\Phi'])
  \]

  \[
  : [\Phi] \otimes [\Phi'] \rightarrow \alpha[A].
  \]

  Since \(\Phi, \Phi' \vdash_{1} \text{lift}[V/x]M : !_{\alpha} A\), we have

  \[
  [\text{lift}[V/x]M] = p\delta[[[V/x]M] : [\Phi] \otimes [\Phi'] \rightarrow \alpha[A].
  \]

  Note that \(\delta[[[V/x]M]\) is the unique morphism such that \(\text{force} \circ p\delta[[[V/x]M] = [[[V/x]M].\) On the other hand, we have

  \[
  \text{force} \circ p\delta[M] \circ ([\Phi] \otimes [V] \otimes [\Phi']) \circ (\text{dup} \otimes [\Phi'])
  \]

  \[
  = [M] \circ ([\Phi] \otimes [V] \otimes [\Phi']) \circ (\text{dup} \otimes [\Phi']) = [[[V/x]M].
  \]

  Thus

  \[
  [\text{lift}[V/x]M] = p\delta[[[V/x]M] = p\delta[M] \circ ([\Phi] \otimes [V] \otimes [\Phi']) \circ (\text{dup} \otimes [\Phi'])
  \]

  \[
  = [\text{lift } M] \circ ([\Phi] \otimes [V] \otimes [\Phi']) \circ (\text{dup} \otimes [\Phi']).
  \]
Case

\[
\Gamma_1, x : C, \Gamma'_1, y : A \vdash_\alpha M : B \\
\Gamma_1, x : C, \Gamma'_1 \vdash_\lambda y. M : A \to_\alpha B.
\]

Suppose \(\Gamma_2 \vdash_1 V : C\). By induction hypothesis, we have

\[
[[V/x]M] = [M] \circ ([\Gamma_1] \otimes [V] \otimes [\Gamma'_1, y : A])
\]

\[
: [\Gamma_1] \otimes [\Gamma_2] \otimes [\Gamma'_1, y : A] \to \alpha[B].
\]

Since \(\Gamma_1, \Gamma_2, \Gamma'_1 \vdash \lambda y. [V/x]M : A \to_\alpha B\), we have

\[
[[\lambda y. [V/x]M] = \text{curry}([[V/x]M]) : ([\Gamma_1] \otimes [\Gamma_2] \otimes [\Gamma'_1]) \to [A] \to \alpha[B],
\]

which is the unique morphism such that

\[
\epsilon \circ (\text{curry}([[V/x]M]) \otimes [A]) = [[V/x]M].
\]

On the other hand, we have

\[
\epsilon \circ (\text{curry}[M]) \circ ([\Gamma_1] \otimes [V] \otimes [\Gamma'_1, y : A]) \circ [A])
\]

\[
= \epsilon \circ (\text{curry}[M] \otimes [A]) \circ ([\Gamma_1] \otimes [V] \otimes [\Gamma'_1, y : A])
\]

\[
= [[M] \circ ([\Gamma_1] \otimes [V] \otimes [\Gamma'_1, y : A]) = [[V/x]M].
\]

Thus \([\lambda y. [V/x]M] = \text{curry}[M] \circ ([\Gamma_1] \otimes [V] \otimes [\Gamma'_1, y : A]) = [\lambda y. M] \circ ([\Gamma_1] \otimes [V] \otimes [\Gamma'_1, y : A])\).

Case

\[
\Phi, \Gamma_1, x : C, \Gamma'_1 \vdash_{\alpha_1} M : A \to_{\beta} B \\
\Phi, \Gamma_1, x : C, \Gamma'_1 \vdash_{\alpha_2} N : A \\
\Phi, \Gamma_1, x : C, \Gamma'_1, \Gamma_2 \vdash_{\alpha_1, \alpha_2, \beta} MN : B
\]

Suppose \(\Phi, \Gamma_3 \vdash_1 V : C\) and \(\alpha_1 = \alpha_2 = \beta = 0\). By induction hypothesis, we have

\[
[[V/x]M] = [M] \circ ([\Phi] \otimes [\Gamma_1] \otimes [V] \otimes [\Gamma'_1]) \circ (\text{dup} \otimes [\Gamma_1] \otimes [\Gamma'_1] \otimes [\Gamma_3])
\]

\[
: [\Phi] \otimes [\Gamma_1] \otimes [\Gamma_3] \otimes [\Gamma'_1] \to T([\alpha_1 \to T[B]).
\]

Moreover,

\[
[[V/x]MN] = \mu \circ T_e \circ \mu \circ T_s \circ t \circ ([M] \otimes [N]) = \mu \circ T_e \circ \mu \circ T_s \circ [N].
\]

On the other hand,

\[
[MN] \circ ([\Phi] \otimes [\Gamma_1] \otimes [V] \otimes [\Gamma'_1] \otimes [\Gamma_2]) \circ (\text{dup} \otimes [\Gamma_1] \otimes [\Gamma'_1] \otimes [\Gamma_2] \otimes [\Gamma_3])
\]

\[
= \mu \circ T_e \circ \mu \circ T_s \circ ([M] \otimes [N])
\]

\[
\circ ([\Phi] \otimes [\Gamma_1] \otimes [V] \otimes [\Gamma'_1] \otimes [\Gamma_2]) \circ (\text{dup} \otimes [\Gamma_1] \otimes [\Gamma'_1] \otimes [\Gamma_2] \otimes [\Gamma_3]).
\]

Thus

\[
[[V/x]MN] = [[M] \circ ([\Phi] \otimes [\Gamma_1] \otimes [V] \otimes [\Gamma'_1] \otimes [\Gamma_2])
\]

\[
\circ (\text{dup} \otimes [\Gamma_1] \otimes [\Gamma'_1] \otimes [\Gamma_2] \otimes [\Gamma_3]).
\]

26
E Proof of Theorem 4.15

Case Φ

\[ \Gamma_1, x : C, \Gamma'_1 \vdash_\beta M : \text{!}_\alpha A \]
\[ \Gamma_1, x : C, \Gamma'_1 \vdash_{\alpha \& \beta} \text{force } M : A. \]

Suppose \( \Gamma_2 \vdash_1 V : C \) and \( \alpha = \beta = 0 \). By induction hypothesis, we have
\[
[[V/x]M] = [M] \circ ([\Gamma_1] \otimes [V] \otimes [\Gamma'_1])
\]
\[ : [\Gamma_1] \otimes [\Gamma_2] \otimes [\Gamma'_1] \rightarrow Tp\phi T[A]. \]

Since \( \Gamma_1, \Gamma_2, \Gamma'_1 \vdash_0 \text{force}[V/x]M : A \), we have
\[
[[\text{force}[V/x]M]] = \mu \circ T\text{force} \circ [[V/x]M] = \mu \circ T\text{force} \circ [M] \circ ([\Gamma_1] \otimes [V] \otimes [\Gamma'_1])
\]
\[ : [\Gamma_1] \otimes [\Gamma_2] \otimes [\Gamma'_1] \rightarrow T[A]. \]

On the other hand, we have
\[
[[\text{force}[V/x]M]] = \mu \circ T\text{force} \circ ([\Gamma_1] \otimes [V] \otimes [\Gamma'_1])
\]
\[ = \mu \circ T\text{force} \circ [M] \circ ([\Gamma_1] \otimes [V] \otimes [\Gamma'_1]). \]

So \( [[\text{force}[V/x]M]] = [[\text{force}M]] \circ ([\Gamma_1] \otimes [V] \otimes [\Gamma'_1]). \)

Case

\[ \Phi, x : P \vdash_0 M : A \quad \Phi, x : P \vdash_0 N : B \]
\[ \Phi, x : P \vdash_0 (M, N) : A \otimes B \]

Suppose \( \Phi \vdash_1 V : P \). By induction hypothesis, we have \( [[V/x]M] = [M] \circ ([\Phi] \otimes [V]) \circ \text{dup} : [\Phi] \rightarrow T[A] \) and \( [[V/x]N] = [N] \circ ([\Phi] \otimes [V]) \circ \text{dup} : [\Phi] \rightarrow T[B] \). We have
\[
[[[V/x]M, [V/x]N]] = \mu \circ T\sigma \circ t \circ ([[V/x]M] \otimes [[V/x]N]) \circ \text{dup}
\]
\[ = \mu \circ T\sigma \circ t \circ ([M] \circ ([\Phi] \otimes [V]) \circ \text{dup}) \circ ([N] \circ ([\Phi] \otimes [V]) \circ \text{dup} \circ \text{dup}) \circ \text{dup}. \]

On the other hand,
\[
[[M, N]] \circ ([\Phi] \otimes [V]) \circ \text{dup}
\]
\[ = \mu \circ T\sigma \circ t \circ ([M] \otimes [N]) \circ \text{dup}_{[\Phi] \otimes [V]} \circ ([\Phi] \otimes [V]) \circ \text{dup}_{[\Phi] \otimes [V]} \circ \text{dup}_{[\Phi] \otimes [V]}
\]
\[ = \mu \circ T\sigma \circ t \circ ([M] \circ ([\Phi] \otimes [V]) \circ \text{dup}_{[\Phi] \otimes [V]} \circ ([N] \circ ([\Phi] \otimes [V]) \circ \text{dup}_{[\Phi] \otimes [V]} \circ \text{dup}_{[\Phi] \otimes [V]} \circ \text{dup}_{[\Phi] \otimes [V]}. \]

Thus \( [[[V/x]M, [V/x]N]] = [[M, N]] \circ ([\Phi] \otimes [V]) \circ \text{dup}. \)

\[ \Box \]

E Proof of Theorem 4.15

Theorem E.1 (Soundness). If \( \vdash_\alpha (Q, M) : A; \Sigma' \), and \( (Q, M) \downarrow \sum_{i \in [n]} p_i (Q_i, V_i) \), then
\[
[[Q, M]] = \sum_{i \in [n]} p_i [[Q_i, V_i]] : I \rightarrow T[[A] \otimes [\Sigma']]. \]

27
\textbf{Proof.} In the case where $\alpha = 1$, there is no dynamic lifting, and therefore the proof is mostly similar to the proof of Theorem 4.6. We consider just one typical case:

\[ (Q, M) \Downarrow (Q_1, \lambda x.M_1') \]
\[ (Q_1, N) \Downarrow (Q_1', V_1) \]
\[ (Q_1', [V_1/x]M_1') \Downarrow (Q_1'', V_1') \]
\[ (Q, MN) \Downarrow (Q_1'', V_1') \]

Suppose $\text{addr}(Q) = \Sigma''$, $\Sigma'$ and $\Sigma'' \vdash_1 MN : A$. By the typing rule for $MN$, we have

\[ \Sigma'' \vdash_1 M : B \vdash_0 A \quad \Sigma'' \vdash_1 N : B \]

where $\Sigma'' = \Sigma_1'', \Sigma_2''$. By Theorem 4.12 we have $\text{addr}(Q_1) = \Sigma_1''', \Sigma_2'''', \Sigma'$, and $\Sigma'' \vdash_1 \lambda x.M_1' : B \vdash_0 A$, and $\text{addr}(Q_1') = \Sigma_1''', \Sigma_2''', \Sigma'$, and $\Sigma'' \vdash_1 V_1 : B$, and $\text{addr}(Q_1'') = \Sigma'''', \Sigma'$, and $\Sigma''' \vdash_1 V_1' : B$.

By induction hypothesis, we have

\[ [Q, M] = T([M] \otimes [\Sigma_2'''] \otimes [\Sigma']) \circ \phi Q = T(\langle \lambda x.M_1' \rangle \otimes [\Sigma_2'''] \otimes [\Sigma']) \circ \phi Q_1 \]
\[ : I \rightarrow T([B \rightarrow A] \otimes [\Sigma_2'''] \otimes [\Sigma']) \]

and

\[ [Q_1, N] = T([\Sigma''''] \otimes [N] \otimes [\Sigma']) \circ \phi Q_1 = T([\Sigma''''] \otimes [V_1] \otimes [\Sigma']) \circ \phi Q_1' \]
\[ : I \rightarrow T([\Sigma''''] \otimes [B] \otimes [\Sigma']) \]

and

\[ [Q_1', [V_1/x]M_1'] = T([V_1/x]M_1' \otimes [\Sigma']) \circ \phi Q_1' \]
\[ = T([M_1'] \otimes ([\Sigma''''] \otimes [V_1])) \circ \phi Q_1' \]
\[ = T([V_1'] \otimes [\Sigma']) \circ \phi Q_1' : I \rightarrow T([A] \otimes [\Sigma']) \]

We need to show that

\[ [Q, MN] = T([MN] \otimes [\Sigma']) \circ \phi Q \]
\[ = T((\epsilon \circ ([M] \otimes [N])) \otimes [\Sigma']) \circ \phi Q \]
\[ = T(([M_1'] \otimes ([V_1] \otimes [\Sigma''''])) \otimes [\Sigma']) \circ \phi Q_1' \]

This is the case because

\[ [Q, MN] = T([MN] \otimes [\Sigma']) \circ \phi Q \]
\[ = T((\epsilon \circ ([M] \otimes [N])) \otimes [\Sigma']) \circ \phi Q \]
\[ = T(\epsilon \circ [\Sigma']) \circ T([B \rightarrow_0 A] \otimes [N] \otimes [\Sigma']) \circ T([\lambda x.M_1'] \otimes [\Sigma_2'''] \otimes [\Sigma']) \circ \phi Q_1 \]
\[ = T(\epsilon \circ [\Sigma']) \circ T([B \rightarrow_0 A] \otimes [N] \otimes [\Sigma']) \circ T([\lambda x.M_1'] \otimes [\Sigma_2'''] \otimes [\Sigma']) \circ \phi Q_1 \]
\[ = T(\epsilon \circ [\Sigma']) \circ T([\lambda x.M_1'] \otimes [B] \otimes [\Sigma']) \circ T([\Sigma''''] \otimes [V_1] \otimes [\Sigma']) \circ \phi Q_1' \]
\[ = T(\epsilon \circ [\Sigma']) \circ T([\Sigma''''] \otimes [V_1] \otimes [\Sigma']) \circ \phi Q_1' \]
\[ = T(([M_1'] \otimes ([\Sigma''''] \otimes [V_1])) \otimes [\Sigma']) \circ \phi Q_1' \]

Now we consider the case where $\alpha = 0$. Suppose $\text{addr}(Q) = \Sigma''$, $\Sigma'$ and $\Sigma'' \vdash_0 M : A$. We proceed by induction on $(Q, M) \Downarrow \sum_{i \in [n]} p_i(Q_i, V_i)$. 28
• Case

\[
(Q, M) \downarrow \sum_{i \in [n]} p_i(Q_i, \ell_i)
\]

\[
\text{read}(Q_i, \ell_i) = \sum_{j \in [2]} q_{i,j}(Q'_{i,j}, a_{i,j})
\]

\[
(Q, \text{dynlift } M) \downarrow \sum_{(i,j) \in [n] \times [2]} p_{i,j}(Q'_{i,j}, a_{i,j}).
\]

Suppose we have

\[
\Sigma'' \vdash_1 M : \text{Bit}
\]

\[
\Sigma'' \vdash_0 \text{dynlift } M : \text{Bool}.
\]

In this case we have the following derivation.

\[
\frac{(Q, M) \downarrow (Q', \ell) \quad \text{read}(Q', \ell) = q_1(Q'_1, \text{False}) + q_2(Q'_2, \text{True})}{(Q, \text{dynlift } M) \downarrow q_1(Q'_1, \text{False}) + q_2(Q'_2, \text{True})}
\]

Since \(\text{read}(Q', \ell) = q_1(Q'_1, \text{False}) + q_2(Q'_2, \text{True})\), we have that \(Q' = q_1(Q'_1 \otimes \text{inj}_1) + q_2(Q'_2 \otimes \text{inj}_2)\) in \(Q\). So in \(A\), we have

\[
\phi Q' = q_1(\mu \circ T t \circ s \circ (\phi Q'_1 \otimes \phi (\text{inj}_1))) + q_2(\mu \circ T t \circ s \circ (\phi Q'_2 \otimes \phi (\text{inj}_2))),
\]

where \(\phi Q' : I \rightarrow T(\text{Bit} \otimes [\Sigma'])\), and \(\phi Q'_1, \phi Q'_2 : I \rightarrow T[\Sigma']\), and \(\phi(\text{inj}_1), \phi(\text{inj}_2) : I \rightarrow T\text{Bit}\). Note that by condition \(\Box\), we have \(\phi(\text{inj}_1) = \eta \circ \text{init} \circ \text{[False]}\) and \(\phi(\text{inj}_2) = \eta \circ \text{init} \circ \text{[True]}\). We need to show that

\[
[Q, \text{dynlift } M] = q_1(T([\text{False}] \otimes [\Sigma']) \circ \phi Q'_1) + q_2(T([\text{True}] \otimes [\Sigma']) \circ \phi Q'_2).
\]

By induction hypothesis, we have \([Q, M] = [Q', \ell]\), i.e., \(T([M] \otimes [\Sigma']) \circ \phi Q = \phi Q'\). Thus

\[
[Q, \text{dynlift } M] = \mu \circ T s \circ T((\text{dynlift} \circ [M]) \otimes [\Sigma']) \circ \phi Q
\]

\[
= \mu \circ T s \circ T(\text{dynlift} \otimes [\Sigma']) \circ T([M] \otimes [\Sigma']) \circ \phi Q
\]

\[
= \mu \circ T s \circ T(\text{dynlift} \otimes [\Sigma']) \circ \phi Q'
\]

\[
= \mu \circ T s \circ T(\text{dynlift} \otimes [\Sigma'])
\]

\[
\circ (q_1(\mu \circ T t \circ s \circ (\phi Q'_1 \otimes \phi (\text{inj}_1))) + q_2(\mu \circ T t \circ s \circ (\phi Q'_2 \otimes \phi (\text{inj}_2))))
\]

\[
= q_1(\mu \circ T s \circ T(\text{dynlift} \otimes [\Sigma']) \circ \mu \circ T t \circ s \circ (\phi Q'_1 \otimes \phi (\text{inj}_1)))
\]

\[
+ q_2(\mu \circ T s \circ T(\text{dynlift} \otimes [\Sigma']) \circ \mu \circ T t \circ s \circ (\phi Q'_2 \otimes \phi (\text{inj}_2))).
\]

We just need to show

\[
T([\text{False}] \otimes [\Sigma']) \circ \phi Q'_1 = \mu \circ T s \circ T(\text{dynlift} \otimes [\Sigma']) \circ \mu \circ T t \circ s \circ (\phi Q'_1 \otimes \phi (\text{inj}_1))
\]

\[
= \mu \circ T s \circ T(\text{dynlift} \otimes [\Sigma']) \circ \mu \circ T t \circ s \circ (\phi Q'_1 \otimes (\eta \circ \text{init} \circ \text{[False]})).
\]

This is true because of the following commutative diagram.
– Suppose we have

$$
\frac{\Sigma' \vdash_0 M : \text{Bit}}{\Sigma' \vdash_0 \text{dynlift} M : \text{Bool}}
$$

In this case \((Q, M) \downarrow \sum_{i \in [n]} p_i (Q_i, \ell_i)\) and \(\text{read}(Q_i, \ell_i) = q_{i,1}(Q_{i,1}, \text{False}) + q_{i,2}(Q_{i,2}, \text{True})\), where \(\text{addr}(Q_i) = \Sigma', \ell_i : \text{Bit}\) and \(\text{addr}(Q_{i,1}) = \text{addr}(Q_{i,2}) = \Sigma'\). Thus we have the following in \(A\): \(\phi Q_i = q_{i,1}(\mu \circ T t \circ s \circ (\phi Q_{i,1} \otimes \phi(\text{inj}_1))) + q_{i,2}(\mu \circ T t \circ s \circ (\phi Q_{i,2} \otimes \phi(\text{inj}_2)))\). Note that \(\phi(\text{inj}_1) = \eta \circ \text{init} \circ [\text{False}]\) and \(\phi(\text{inj}_2) = \eta \circ \text{init} \circ [\text{True}]\).

We need to show that

$$
[Q, \text{dynlift} M] = \sum_{i \in [n]} p_i q_{i,1}(T([\text{False}] \otimes [\Sigma']) \otimes \phi Q_{i,1}) + p_i q_{i,2}(T([\text{True}] \otimes [\Sigma']) \otimes \phi Q_{i,2}).
$$

By induction hypothesis, we have

$$
[Q, M] = \mu \circ T s \circ T([M] \otimes [\Sigma']) \circ \phi Q
$$

$$
= \sum_{i \in [n]} p_i (T([\ell_i] \otimes [\Sigma']) \circ \phi Q_i) = \sum_{i \in [n]} p_i \phi Q_i.
$$

Thus

$$
[Q, \text{dynlift} M] = \mu \circ T s \circ T((\mu \circ T \text{dynlift} \circ [M]) \otimes [\Sigma']) \circ \phi Q
$$

$$
= \mu \circ T s \circ T((\mu \circ T \text{dynlift}) \otimes [\Sigma']) \circ T([M] \otimes [\Sigma']) \circ \phi Q
$$

$$
= \mu \circ T s \circ T(\text{dynlift} \otimes [\Sigma']) \circ \mu \circ T s \circ T([M] \otimes [\Sigma']) \circ \phi Q
$$

$$
= \mu \circ T s \circ T(\text{dynlift} \otimes [\Sigma']) \circ (\sum_{i \in [n]} p_i \phi Q_i)
$$

$$
= \sum_{i \in [n]} p_i (\mu \circ T s \circ T(\text{dynlift} \otimes [\Sigma']) \circ \phi Q_i)
$$

$$
= \sum_{i \in [n]} p_i (\mu \circ T s \circ T(\text{dynlift} \otimes [\Sigma']) \circ (q_{i,1}(\mu \circ T t \circ s \circ (\phi Q_{i,1} \otimes \phi(\text{inj}_1)))))
$$
\[ + q_{i, 2} (\mu \circ Tt \circ s \circ (\phi Q'_{i, 2} \otimes \phi (\text{inj}_2))) \]
\[ = \sum_{i \in [n]} p_i q_{i, 1} (\mu \circ Ts \circ T(\text{dynlift} \otimes [\Sigma']) \circ \mu \circ Tt \circ s \circ (\phi Q'_{i, 1} \otimes \phi (\text{inj}_1))) \]
\[ + q_{i, 2} (\mu \circ Ts \circ T(\text{dynlift} \otimes [\Sigma']) \circ \mu \circ Tt \circ s \circ (\phi Q'_{i, 2} \otimes \phi (\text{inj}_2))). \]

So we just need to show
\[ T([\text{False}] \otimes [\Sigma']) \otimes \phi Q'_{i, 1} \]
\[ = \mu \circ Ts \circ T(\text{dynlift} \otimes [\Sigma']) \circ \mu \circ Tt \circ s \circ (\phi Q'_{i, 1} \otimes \phi (\text{inj}_1)). \]

We already showed this in the previous case.

The equality of (*) is by the following commutative diagram.

\[ T(T\text{Bit} \otimes [\Sigma']) \]
\[ T(T\text{Bit} \otimes [\Sigma']) \]
\[ T\text{Bit} \otimes [\Sigma'] \]
\[ T(\text{Dynlift} \otimes [\Sigma']) \]
\[ T(\text{Dynlift} \otimes [\Sigma']) \]
\[ (Q, M) \downarrow \sum_{i \in [n]} p_i (Q_i, \lambda x. M'_i) \]
\[ (Q_i, N) \downarrow \sum_{j \in [m]} q_{i,j} (Q'_{i,j}, V_{i,j}) \]
\[ (Q'_{i,j}, [V_{i,j}/x] M'_i) \downarrow \sum_{k \in [l]} s_{i,j,k} (Q''_{i,j,k}, V'_{i,j,k}) \]
\[ (Q, M N) \downarrow \sum_{(i,j,k) \in [n] \times [m] \times [l]} p_i q_{i,j} s_{i,j,k} (Q''_{i,j,k}, V'_{i,j,k}). \]

Here we only consider the following typing judgment (the other cases are proved similarly).
\[ \Sigma_1 \vdash_0 M : A \rightarrow_0 B \quad \Sigma_2 \vdash_0 N : A \]
\[ \Sigma_1, \Sigma_2 \vdash_0 MN : B \]

We assume that \( \text{addr}(Q) = \Sigma_1, \Sigma_2, \Sigma' \).

By induction hypothesis, we have
\[ \llbracket Q, M \rrbracket = \mu \circ Ts \circ T(s \otimes [\Sigma']) \circ T([M] \otimes [\Sigma_2] \otimes [\Sigma']) \circ \phi Q \]
\[ = \sum_{i \in [n]} p_i [Q_i, \lambda x. M'_i] = \sum_{i \in [n]} p_i (T([\lambda x. M'_i] \otimes [\Sigma_2] \otimes [\Sigma']) \circ \phi Q_i), \]

where \( \phi Q_i : I \rightarrow T([\Sigma'_i] \otimes [\Sigma_2] \otimes [\Sigma']) \) and \( \Sigma'_i \vdash_1 \lambda x. M'_i : A \rightarrow_0 B \). The induction hypothesis also gives
\[ [Q, N] = \mu \circ Ts \circ T(t \otimes [\Sigma']) \circ T([\Sigma'_i] \otimes [N] \otimes [\Sigma']) \circ \phi Q_i \]
\[ = \sum_{j \in [m]} q_{i,j} [Q'_{i,j}, V_{i,j}] = \sum_{j \in [m]} q_{i,j} (T([\Sigma'_i] \otimes [V_{i,j}] \otimes [\Sigma']) \circ \phi Q'_{i,j}), \]

31
where \( Q'_{i,j} : I \rightarrow T(\Sigma_i^j \otimes \Sigma_{i,j}^r \otimes [\Sigma']) \) and \( \Sigma_{i,j}^r \vdash V_{i,j} : A \). Moreover,

\[
[Q'_{i,j}, V_{i,j}/x] M'_i = \mu \circ Ts \circ T([V_{i,j}/x] M'_i) \otimes [\Sigma'] \circ \phi Q'_{i,j}
\]

\[
= \mu \circ Ts \circ T((M'_i \otimes (\Sigma'_i \otimes [V_{i,j}])) \otimes [\Sigma']) \circ \phi Q'_{i,j}
\]

\[
= \sum_{k \in [l]} s_{i,j,k} Q''_{i,j,k}, V_{i,j,k}'
\]

where \( \phi Q''_{i,j,k} : I \rightarrow T(\Sigma_{i,j,k}^r \otimes [\Sigma']) \) and \( \Sigma_{i,j,k}^r \vdash V_{i,j,k}' : B \).

We need to show

\[
[Q, MN] = \mu \circ Ts \circ T((MN \otimes [\Sigma']) \circ \phi Q
\]

\[
= \mu \circ Ts \circ T((\mu \circ Te \circ \mu \circ Ts \circ t \circ ([M] \otimes [N]) \otimes [\Sigma']) \circ \phi Q
\]

\[
= \sum_{i,j,k \in [n] \times [m] \times [l]} p_{i,j,k} s_{i,j,k} (T([V_{i,j,k}] \otimes [\Sigma']) \circ \phi Q''_{i,j,k}).
\]

On the right hand side, we have

\[
RHS = \sum_{i,j,k \in [n] \times [m] \times [l]} p_{i,j,k} s_{i,j,k} (T([V_{i,j,k}] \otimes [\Sigma']) \circ \phi Q''_{i,j,k})
\]

\[
\overset{IH}{=} \sum_{i,j \in [n] \times [m]} p_{i,j} (\mu \circ Ts \circ T((M'_i \otimes ([\Sigma'_i] \otimes [V_{i,j}])) \otimes [\Sigma']) \circ \phi Q_{i,j}).
\]

On the left hand side, we have

\[
LHS = \mu \circ Ts \circ T((\mu \circ Te \circ \mu \circ Ts \circ t \circ ([M] \otimes [N]) \otimes [\Sigma']) \circ \phi Q
\]

\[
= \mu \circ Ts \circ T((\mu \circ Te \circ \mu \circ Ts \circ t \circ (T([A] \rightarrow T[B]) \otimes [N]) \otimes ([M] \otimes [\Sigma_2]) \otimes [\Sigma']) \circ \phi Q
\]

\[
= \mu \circ Ts \circ T(\mu \otimes [\Sigma']) \circ T(\mu \circ Te \otimes ([\Sigma'] \otimes [\Sigma']) \circ T([A] \rightarrow T[B]) \otimes [N] \otimes [\Sigma'])
\]

\[
\overset{(1)}{=} \mu \circ Ts \circ T(\mu \otimes [\Sigma']) \circ T(\mu \circ Te \otimes (T([A] \rightarrow T[B]) \otimes [N]) \otimes [\Sigma'])
\]

\[
\overset{IH}{=} \mu \circ Ts \circ T(\mu \otimes [\Sigma']) \circ T(\mu \circ Te \otimes [\Sigma'] \otimes [T([A] \rightarrow T[B]) \otimes [N] \otimes [\Sigma'])
\]

\[
\sum_{i \in [n]} p_i (T(\lambda x. M'_i^r \otimes [\Sigma_2] \otimes [\Sigma']) \circ \phi Q_i)
\]

\[
= \sum_{i \in [n]} p_i (\mu \circ Ts \circ T(\mu \otimes [\Sigma']) \circ T(\mu \circ Te \otimes [\Sigma'] \otimes [T([A] \rightarrow T[B] \otimes [N] \otimes [\Sigma'])
\]

\[
T(\lambda x. M'_i^r \otimes [\Sigma_2] \otimes [\Sigma']) \circ \phi Q_i)
\]

\[
= \sum_{i \in [n]} p_i (\mu \circ Ts \circ T((\mu \circ Te \circ t \circ ([T([A] \rightarrow T[B]) \otimes [N]) \otimes (\lambda x. M'_i^r \otimes [\Sigma_2]) \otimes [\Sigma']) \circ \phi Q_i)
\]

\[
= \sum_{i \in [n]} p_i (\mu \circ Ts \circ T((\mu \circ Te \circ t \circ ([\lambda x. M'_i^r] \otimes [T[A]] \otimes ([\Sigma'_i] \otimes [N])) \otimes [\Sigma']) \circ \phi Q_i)
\]

\[
= \sum_{i \in [n]} p_i (\mu \circ Ts \circ T((\mu \circ Te \circ t \circ ([\lambda x. M'_i^r] \otimes [T[A]]) \otimes ([\Sigma'_i] \otimes [N])) \otimes [\Sigma']) \circ \phi Q_i)
\]
This is true because the following diagram commutes.

\[
\begin{align*}
&= \sum_{i \in [n]} p_i (\mu \circ T s \circ T (\mu \circ T \epsilon \circ t \circ ([\lambda x. M_i'] \otimes T[A]) \otimes [\Sigma'] \otimes [N] \otimes [\Sigma']) \circ \phi_{Q_i}) \\
&= \sum_{i \in [n]} p_i (\mu \circ T s \circ T (\epsilon \otimes [\Sigma']) \circ T([\lambda x. M_i'] \otimes [A] \otimes [\Sigma']) \circ \mu \circ T s \circ T (t \otimes [\Sigma']) \circ T ([\Sigma'] \otimes [N] \otimes [\Sigma']) \circ \phi_{Q_i}) \\
&= \sum_{i \in [n]} p_i (\mu \circ T s \circ T ([M_i'] \otimes [\Sigma']) \circ \bigoplus_{j \in [m]} q_i,j(T ([\Sigma_i'] \otimes [V_{i,j}] \otimes [\Sigma']) \circ \phi_{Q_{i,j}})) \\
&= \sum_{i,j \in [n] \times [m]} p_i q_{i,j} (\mu \circ T s \circ T ([M_i'] \otimes [\Sigma_i'] \otimes [V_{i,j}] \otimes [\Sigma']) \circ \phi_{Q_{i,j}}).
\end{align*}
\]

To show (1) holds, we just need to show

\[
T((\mu \circ T \epsilon \circ \mu \circ T s \circ t \circ T([A] \to T[B])) \otimes [N] \otimes [\Sigma'])
= T(\mu \otimes [\Sigma']) \circ T(T \epsilon \otimes [\Sigma']) \circ T(t \otimes [\Sigma']) \circ T([A] \to T[B]) \otimes [N] \otimes [\Sigma']) \circ \mu \circ T s \circ T(s \otimes [\Sigma']).
\]

This is true because the following diagram commutes.

To show (2) holds, we need to show

\[
\begin{align*}
&= T((\mu \circ T \epsilon \circ t \circ ([\lambda x. M_i'] \otimes T[A]) \otimes [\Sigma']) \\
&= T(\epsilon \otimes [\Sigma']) \circ T([\lambda x. M_i'] \otimes [A] \otimes [\Sigma']) \circ \mu \circ T s \circ T(t \otimes [\Sigma']).
\end{align*}
\]

This is true because the following diagram commutes.
• Case

\[
(Q, M) \Downarrow \sum_{i \in [n]} p_i(Q_i, \text{lift } M'_i) \\
\text{gen}(S) = a \\
(Id_{\Sigma'_a}, M'_i, a) \Downarrow (D_i, b_i) \\
(Q, \text{box } S \cdot M) \Downarrow \sum_{i \in [n]} p_i(Q_i, (a, D_i, b_i)).
\]

Suppose \(\text{addr}(Q) = \Sigma_1, \Sigma'\). Consider the following.

\[
\Sigma_1 \vdash_0 M : !_1(S \rightarrow_U U) \\
\Sigma_1 \vdash_0 \text{box } S \cdot M : \text{Circ}(S, U)
\]

By induction hypothesis, we have

\[
[Q, M] = \mu \circ Ts \circ T([M] \otimes [\Sigma']) \circ \phi Q = \sum_{i \in [n]} p_i[Q_i, \text{lift } M'_i]
\]

\[
= \sum_{i \in [n]} p_i T([\text{lift } M'_i] \otimes [\Sigma']) \circ \phi Q_i,
\]

where \(\phi Q_i : I \rightarrow T[\Sigma']\) and \(\vdash_1 \text{lift } M'_i : !_1(S \rightarrow_U U)\). Note that \(\Sigma_a \vdash a : S\). By Theorem 4.6, we have

\[
[\text{Id}_{\Sigma_a}, M'_i a] = \epsilon \circ ([M'_i] \otimes [a]) = [b_i] \circ D_i : [\Sigma_a] \rightarrow [U].
\]

We need to show

\[
[Q, \text{box } S \cdot M] = \mu \circ Ts \circ T([\text{box } S \cdot M] \otimes [\Sigma']) \circ \phi Q
\]

\[
= \mu \circ Ts \circ T((\text{box } \circ [M]) \otimes [\Sigma']) \circ \phi Q
\]

\[
= \sum_{i \in [n]} p_n[Q_i, (a, D_i, b)]]
\]

\[
= \sum_{i \in [n]} p_n(T((\text{box } \circ p\delta \text{curry}(\theta_{D_i}) \otimes [\Sigma']) \circ \phi Q_i),
\]

where \(\theta_{D_i} = [b_i] \circ D_i \circ [a]^{-1} : [S] \rightarrow [U]\), and \(\text{curry}(\theta_{D_i}) : I \rightarrow [S] \rightarrow [U]\) and \(\delta \text{curry}(\theta_{D_i}) : 1 \rightarrow b([S] \rightarrow [U])\).

This is true because we have the following.

\[
\mu \circ Ts \circ T((\text{box } \circ [M]) \otimes [\Sigma']) \circ \phi Q
\]

\[
= T((\text{box } \otimes [\Sigma']) \circ \mu \circ Ts \circ T([M] \otimes [\Sigma']) \circ \phi Q)
\]

\(\overset{\text{IH}}{=} T((\text{box } \otimes [\Sigma']) \circ \sum_{i \in [n]} p_i T([\text{lift } M'_i] \otimes [\Sigma']) \circ \phi Q_i)
\]

\[
= \sum_{i \in [n]} p_i T((\text{box } \otimes [\Sigma']) \circ T([\text{lift } M'_i] \otimes [\Sigma']) \circ \phi Q_i)
\]

\[
= \sum_{i \in [n]} p_n(T((\text{box } \circ p\delta [M'_i] \otimes [\Sigma']) \circ \phi Q_i)
\]

\[
= \sum_{i \in [n]} p_n(T((\text{box } \circ p\delta \text{curry}(\theta_{D_i}) \otimes [\Sigma']) \circ \phi Q_i).
\]

The last equality holds because

\[
\text{curry}(\theta_{D_i}) = \text{curry}([b_i] \circ D_i \circ [a]^{-1})
\]

\[
= \text{curry}((\epsilon \circ ([M'_i] \otimes [a])) \circ [a]^{-1}) = \text{curry}(\epsilon \circ ([M'_i] \otimes [S])) = [M'_i].
\]
Case

\[ (Q, M) \downarrow \sum_{i \in [n]} p_i(Q_i, \text{lift } M'_i) \]
\[ (Q_i, M'_i) \downarrow \sum_{j \in [m]} q_{i,j}(Q'_{i,j}, V_{i,j}) \]

\[ (Q, \text{force } M) \downarrow \sum_{(i,j) \in [n] \times [m]} p_{i,j}(Q'_{i,j}, V_{i,j}) \]

Suppose \( \text{addr}(Q) = \Sigma_1, \Sigma' \) and

\[ \Sigma_1 \vdash_0 M : !_0 A \]
\[ \Sigma_1 \vdash_0 \text{force } M : A. \]

By induction hypothesis, we have

\[ [Q, M] = \mu \circ Ts \circ ([M] \otimes [\Sigma']) \circ \phi_Q = \sum_{i \in [n]} p_i([Q_i, \text{lift } M'_i] \]
\[ = \sum_{i \in [n]} p_i(T([\text{lift } M'_i] \otimes [\Sigma']) \circ \phi_{Q_i}), \]

where \( \phi_{Q_i} : I \to T[\Sigma'] \) and \( \emptyset \vdash_1 \text{lift } M_i : !_0 A. \)

Also by induction hypothesis, we have

\[ [Q_i, M'_i] = \mu \circ Ts \circ ([M'_i] \otimes [\Sigma']) \circ \phi_{Q_i} = \sum_{j \in [m]} q_{i,j}([Q'_{i,j}, V_{i,j}] \]
\[ = \sum_{j \in [m]} q_{i,j}(T([V_{i,j}] \otimes [\Sigma']) \circ \phi_{Q'_{i,j}}), \]

where \( \phi_{Q'_{i,j}} : I \to T([\Sigma_2] \otimes [\Sigma']) \) and \( \Sigma_2 \vdash_{V_{i,j}} A. \)

We need to show

\[ [Q, \text{force } M] = \mu \circ Ts \circ T((\mu \circ T\text{force } [M] \otimes [\Sigma']) \circ \phi_Q \]
\[ = \sum_{i,j \in [n] \times [m]} p_{i,j}([Q'_{i,j}, V_{i,j}] = \sum_{i,j \in [n] \times [m]} p_{i,j}(T([V_{i,j}] \otimes [\Sigma']) \circ \phi_{Q'_{i,j}}. \]

This is true because

\[ \mu \circ Ts \circ T((\mu \circ T\text{force } [M] \otimes [\Sigma']) \circ \phi_Q \]
\[ = \mu \circ Ts \circ T((\mu \circ T\text{force } [\Sigma']) \circ T([M] \otimes [\Sigma']) \circ \phi_Q \]
\[ = \mu \circ Ts \circ T((\text{force } \otimes [\Sigma']) \circ \mu \circ Ts \circ T([M] \otimes [\Sigma']) \circ \phi_Q \]
\[ \overset{IH}{=} \mu \circ Ts \circ T((\text{force } \otimes [\Sigma']) \circ \sum_{i \in [n]} p_i(T([\text{lift } M'_i] \otimes [\Sigma']) \circ \phi_{Q_i}) \]
\[ = \sum_{i \in [n]} p_i(\mu \circ Ts \circ T(\text{force } \otimes [\Sigma']) \circ T([\text{lift } M'_i] \otimes [\Sigma']) \circ \phi_{Q_i}) \]
\[ = \sum_{i \in [n]} p_i(\mu \circ Ts \circ T([\text{lift } M'_i] \otimes [\Sigma']) \circ \phi_{Q_i}) \]
\[ \overset{IH}{=} \sum_{i \in [n]} p_i \sum_{j \in [m]} q_{i,j}(T([V_{i,j}] \otimes [\Sigma']) \circ \phi_{Q'_{i,j}}) \]
\[ = \sum_{i,j \in [n] \times [m]} p_{i,j}(T([V_{i,j}] \otimes [\Sigma']) \circ \phi_{Q'_{i,j}}. \]
Case

\[(Q, M) \downarrow \sum_{i \in [n]} p_i(Q_i, (a_i, D_i, b_i))\]
\[(Q, N) \downarrow \sum_{j \in [m]} q_{i,j}(Q''_{i,j}, V_{i,j})\]
operate\((D_i, Q''_{i,j}, a_i, V_{i,j}) = Q''_{i,j}\)

\[(Q, \text{apply}(M, N)) \downarrow \sum_{(i,j) \in [n] \times [m]} p_i q_{i,j}(Q''_{i,j}, b_i)\).

Suppose \(\text{addr}(Q) = \Sigma_1, \Sigma_2, \Sigma'\) and

\[\Sigma_1 \vdash_0 M : \text{Circ}(S, U) \quad \Sigma_2 \vdash_0 N : S\]
\[\Sigma_1, \Sigma_2 \vdash_0 \text{apply}(M, N) : U.\]

By induction hypothesis, we have

\[\llbracket Q, M \rrbracket = \mu \circ Ts \circ T(s \odot \llbracket \Sigma' \rrbracket) \circ T(\llbracket M \rrbracket \otimes \llbracket \Sigma_2 \rrbracket \otimes \llbracket \Sigma' \rrbracket) \circ \phi Q_i\]
\[= \sum_{i \in [n]} p_i \llbracket Q_i, (a_i, D_i, b_i) \rrbracket = \sum_{i \in [n]} p_i T(\llbracket a_i, D_i, b_i \rrbracket \otimes \llbracket \Sigma_2 \rrbracket \otimes \llbracket \Sigma' \rrbracket) \circ \phi Q_i\]
\[= \sum_{i \in [n]} p_i T((\text{box} \circ \rho \text{curry}(\theta_{D_i})) \otimes \llbracket \Sigma_2 \rrbracket \otimes \llbracket \Sigma' \rrbracket) \circ \phi Q_i,\]

where \(\phi Q_i : I \rightarrow T(\llbracket \Sigma_2 \rrbracket \otimes \llbracket \Sigma' \rrbracket)\), and \(\theta_{D_i} = [b_i \circ D_i \circ [a_i]^{-1} : [S] \rightarrow [U], \text{and } D_i : [\Sigma_{a_i}] \rightarrow [\Sigma_{b_i}]\), and \(\Sigma_{a_i} \vdash_1 a_i : S \text{ and } \Sigma_{b_i} \vdash_1 b_i : U\).

Moreover, by induction hypothesis we also have

\[\llbracket Q_i, N \rrbracket = \mu \circ Ts \circ T(s \odot \llbracket \Sigma_2 \rrbracket \otimes \llbracket \Sigma' \rrbracket) \circ \phi Q_i\]
\[= \sum_{j \in [m]} q_{i,j} \llbracket Q''_{i,j}, V_{i,j} \rrbracket = \sum_{j \in [m]} q_{i,j} T(\llbracket V_{i,j} \rrbracket \otimes \llbracket \Sigma_2 \rrbracket \otimes \llbracket \Sigma' \rrbracket) \circ \phi Q'_{i,j},\]

where \(\phi Q'_{i,j} : I \rightarrow T(\llbracket \Sigma_2 \rrbracket \otimes \llbracket \Sigma' \rrbracket)\) and \(\Sigma_2 \vdash_1 V_{i,j} : S\).

Since operate\((D_i, Q''_{i,j}, a_i, V_{i,j}) = Q''_{i,j}\), we have the following in Q.

\[Q''_{i,j} = ((J D_i \circ J [a_i]^{-1} \circ J [V_{i,j}]) \otimes \llbracket \Sigma' \rrbracket) \circ Q'_{i,j} : I \rightarrow \Sigma_{b_i}, \Sigma'.\]

Thus in A, we have

\[\phi Q''_{i,j} = \phi ((J D_i \circ J [a_i]^{-1} \circ J [V_{i,j}]) \otimes \llbracket \Sigma' \rrbracket) \circ Q'_{i,j}\]
\[= \mu \circ T(\mu \circ T s \circ \phi (J D_i \circ J [a_i]^{-1} \circ J [V_{i,j}]) \otimes \eta \llbracket \Sigma_2 \rrbracket) \circ \phi Q'_{i,j}\]
\[= \mu \circ T(\mu \circ T s \circ \phi (\eta [\Sigma_{a_i}] \circ D_i \circ [a_i]^{-1} \circ [V_{i,j}]) \otimes \eta \llbracket \Sigma_2 \rrbracket) \circ \phi Q'_{i,j}\]
\[\mapsto \mu \circ T(\mu \circ T s \circ \phi (\eta [\Sigma_{b_i}] \circ D_i \circ [a_i]^{-1} \circ [V_{i,j}]) \otimes \eta \llbracket \Sigma_2 \rrbracket) \circ \phi Q'_{i,j}\]
\[= \mu \circ T \eta [\Sigma_{b_i}] \circ D_i \circ [a_i]^{-1} \circ [V_{i,j}] \otimes [\Sigma'] \circ \phi Q'_{i,j}\]
\[= \mu \circ T(\eta [\Sigma_{b_i}] \otimes [\Sigma'] \circ ((D_i \circ [a_i]^{-1} \circ [V_{i,j}]) \otimes [\Sigma']) \circ \phi Q'_{i,j}\]
\[= T((D_i \circ [a_i]^{-1} \circ [V_{i,j}]) \otimes [\Sigma']) \circ \phi Q'_{i,j}\]

: \(I \rightarrow T(\llbracket \Sigma_{b_i} \rrbracket \otimes \llbracket \Sigma' \rrbracket)\).

Note that \((*)\) is by \(\phi \circ J = E \circ \psi\) in condition \#3
We need to show
\[
[Q, \text{apply}(M, N)] = \mu \circ Ts \circ T([\text{apply}(M, N)] \otimes [\Sigma']) \circ \phi Q
\]
\[
= \mu \circ Ts \circ T((T \circ T([\text{force } \circ \text{unbox}] \otimes [S]) \circ \mu \circ Ts \circ t \circ ([M] \otimes [N]) \otimes [\Sigma']) \circ \phi Q
\]
\[
= \sum_{i,j \in [n] \times [m]} p_i q_{i,j} [Q'_{i,j}, b_i]
\]
\[
= \sum_{i,j \in [n] \times [m]} p_i q_{i,j} (T([b_i] \otimes [\Sigma']) \circ \phi Q_{i,j}^1)
\]
\[
= \sum_{i,j \in [n] \times [m]} p_i q_{i,j} (T([b_i] \circ D_i \circ [a_i]^{-1} \circ [V_{i,j}]) \otimes [\Sigma']) \circ \phi Q_{i,j}^1.
\]
This is true because
\[
\mu \circ Ts \circ T((T \circ T([\text{force } \circ \text{unbox}] \otimes [S]) \circ \mu \circ Ts \circ t \circ ([M] \otimes [N]) \otimes [\Sigma']) \circ \phi Q
\]
\[
= \mu \circ Ts \circ T((T \circ T([\text{force } \circ \text{unbox}] \otimes [S]) \circ \mu \circ Ts \circ t \circ [\Sigma']) \circ T([\mu \circ Ts \circ t \circ ([M] \otimes [N]) \otimes [\Sigma']) \circ \phi Q
\]
\[
= \mu \circ Ts \circ T((T \circ T([\text{force } \circ \text{unbox}] \otimes [S]) \circ \mu \circ Ts \circ t \circ [\Sigma']) \circ T(Circ(S, U) \otimes [N] \otimes [\Sigma'])
\]
\[
\circ T([\Sigma_2] \otimes [\Sigma'] \otimes \phi Q
\]
\[
= \mu \circ Ts \circ T((T \circ T([\text{force } \circ \text{unbox}] \otimes [S]) \circ \mu \circ Ts \circ t \circ [\Sigma']) \circ T(Circ(S, U) \otimes [N] \otimes [\Sigma'])
\]
\[
\circ T([\Sigma_2] \otimes [\Sigma'] \otimes \phi Q_i)
\]
\[
= \sum_{i \in [n]} p_i (\mu \circ Ts \circ T((T \circ T([\text{force } \circ \text{unbox}] \otimes [S]) \circ \mu \circ Ts \circ t \circ [\Sigma']) \circ T(Circ(S, U) \otimes [N] \otimes [\Sigma']))
\]
\[
\circ T([\Sigma_2] \otimes [\Sigma'] \otimes \phi Q_i)
\]
\[
= \sum_{i \in [n]} p_i (\mu \circ Ts \circ T((T \circ T([\text{force } \circ \text{unbox}] \otimes [S] \circ t) \otimes [\Sigma']) \circ T([\text{force } \circ \text{unbox}] \otimes [\Sigma'] \otimes \phi Q_i))
\]
\[
\circ T([\Sigma_2] \otimes [\Sigma'] \otimes \phi Q_i)
\]
\[
= \sum_{i \in [n]} p_i T(\epsilon \circ [\Sigma']) \circ T(Circ(S, U) \otimes [N] \otimes [\Sigma']) \circ T(\mu \circ Ts \circ t \circ [\Sigma']) \circ T(\lambda^{-1} \otimes [\Sigma'])
\]
\[
\circ \mu \circ Ts \circ T([\Sigma_2] \otimes [\Sigma'] \otimes \phi Q_i)
\]
\[
= \sum_{i \in [n]} p_i T(\epsilon \circ [\Sigma']) \circ T(Circ(S, U) \otimes [N] \otimes [\Sigma']) \circ T(\mu \circ Ts \circ t \circ [\Sigma']) \circ T(\lambda^{-1} \otimes [\Sigma']) \circ T([\Sigma_2] \otimes [\Sigma'] \otimes \phi Q_i)
\]
\[
= \sum_{i \in [n]} p_i q_{i,j} (T(\theta_{D_i} \circ [\Sigma']) \circ T([V_{i,j}] \otimes [\Sigma']) \circ \phi Q_{i,j}^1)
\]
\[
= \sum_{i,j \in [n] \times [m]} p_i q_{i,j} (T([b_i] \circ D_i \circ [a_i]^{-1} \circ [V_{i,j}]) \otimes [\Sigma']) \circ T([V_{i,j}] \otimes [\Sigma']) \circ \phi Q_{i,j}^1.
\]
This is true because the following diagram commutes.

\[
= \sum_{i,j \in [n] \times [m]} p_i q_{i,j}(T(\{b_j\} \circ D_i \circ [a_i]^{-1} \circ [V_{ij}] \circ [\Sigma']) \circ \phi Q_{i,j}).
\]

To prove (1), we just need to prove

\[
T((\mu \circ Ts \circ t) \otimes [\Sigma']) \circ T(T \text{Circ}(S,U) \otimes [N] \otimes [\Sigma'])
\]

\[
= T((t \otimes [\Sigma']) \circ T(\text{Circ}(S,U) \otimes [N] \otimes [\Sigma']) \circ \mu \circ Ts \circ T(s \otimes [\Sigma']).
\]

This is true because the following diagram commutes.

To prove (2), we just need to show

\[
\mu \circ Ts \circ T((\text{force} \circ \text{unbox}) \otimes [S] \circ t) \otimes [\Sigma']) \circ T((\text{box} \circ \text{curry}(\theta_{D_j})) \otimes T[S] \otimes [\Sigma'])
\]

\[
= T(\epsilon \otimes [\Sigma']) \circ T((\text{force} \circ \text{unbox}) \otimes [\Sigma']) \circ T((\text{box} \circ \text{curry}(\theta_{D_j})) \otimes [S] \otimes [\Sigma']) \circ T(\lambda^{-1} \otimes [\Sigma']) \circ \mu \circ Ts.
\]

This is true because the following diagram commutes.

\[\square\]