A HoTT Quantum Equational Theory

Jennifer Paykin
Galois Inc.
Portland, OR, USA
jpaykin@gmail.com

Steve Zdancewic
University of Pennsylvania
Philadelphia, PA, USA
stevez@cis.upenn.edu

This paper presents an equational theory for the QRAM model of quantum computation, formulated as an embedded language inside of homotopy type theory. The embedded language approach is highly expressive, and reflects the style of state-of-the art quantum languages like Quipper and QWIRE. The embedding takes advantage of features of homotopy type theory to encode unitary transformations as higher inductive paths, simplifying the presentation of an equational theory. We prove that this equational theory is sound and complete with respect to established models of quantum computation.

1 Introduction

One of the most prominent models of quantum computation today is known as the QRAM model, in which a quantum computer works alongside a classical computer to manipulate both quantum and classical data [11]. Programming languages for the QRAM model must provide good access to both quantum data, like qubits, as well as easy-to-use classical abstractions such as booleans, recursion, and functions. In addition, the two sorts of data should interact—it should be possible to measure a qubit and probabilistically obtain a classical boolean value.

Inspired by the QRAM model, several state-of-the-art quantum programming languages are implemented as embedded languages, where domain-specific features for quantum data are added to an existing general-purpose host language. Quipper, embedded in Haskell, utilizes Haskell type classes, monadic programming, and meta-programming [10]. Other embedded quantum languages include QISKit[1] and pyQuil, both embedded in Python.[2]

Unfortunately, reasoning about embedded quantum languages has largely been seen as futile; such reasoning would have to account for the quantum behavior of the embedded language as well as the classical behavior of the host language and the interactions between the two. Attempts to formalize Quipper have been restricted to a standalone language in the style of Quipper as opposed to the actual Haskell implementation [21, 20, 12].

Given a powerful enough host language, however, it is possible to study the meta-theory of an embedded language inside the host language itself. This is the approach is taken by $\mathcal{Q}$WIRE, a language implemented in Coq that uses its host language both for programming and for verification [16, 19].

This paper extends that technique to study the equational theory of an embedded quantum language. We build on an equational theory proposed by Staton [25], an algebraic account of the relationship between quantum data and classical control. Other works axiomatize particular sets of unitary transformations [4, 13, 14], but, like Staton, we focus on the relationships between quantum and classical features. Staton's algebraic framework includes measurement-based branching, but it does not contain classical data, functions, or other features we would expect from a QRAM-style language.

[1] https://github.com/QISKit
[2] http://pyquil.readthedocs.io

Submitted to:
QPL 2019

Because our goal is to extend Staton's equational theory to a richer embedded language, we choose a host language that specializes in equality—homotopy type theory (HoTT). In HoTT, proofs of equality, also called *paths*, can contain extra computational content. In the past few years, a variety of applications have used paths in HoTT to represent data structures such as containers [1], version control patches [5], and SQL queries [8]. These data structures all form groupoids, which are generalized by paths in HoTT.

This work builds on the observation that unitary transformations, a core component of quantum computing, form a groupoid. We exploit this structure to encode unitaries in the paths between quantum types. This simplifies Staton's theory because many of the structural rules in his presentation can be derived in our presentation. As such, this paper makes the following contributions:

- We present a quantum programming language embedded in homotopy type theory (Sections 3 and 4). The language is both expressive, because users have access to classical higher-order functions, data structures, and more through the host language; and sound, thanks to its use of linear types for quantum data. We have formalized some of the underlying HoTT data structures in Coq.
- We define an equational theory for the embedded quantum language with two axioms that describe how unitary transformations interact with initialization and measurement (Section 5).
- We prove the equational theory is sound with respect to a standard semantics in terms of superoperators over density matrices (Section 6).
- Throughout, we use Staton's algebraic equational theory (Section 3.4) as a specification of the equations we expect to hold in our theory, and we prove that a fragment of our language is sound and complete with respect to Staton's axioms (Section 7).

An extended version of this work with full details and proofs is available online [18]. A prior version of this work, presented in Chapters 6 and 7 of the first author's dissertation [15], did not include the formal results in Coq or the connection in Section 7 with Staton's algebraic theory.

2 Background and main ideas

2.1 Quantum computing

In this paper we consider quantum programs that operate on qubits—superpositions of classical bits of the form $c_0\,|0\rangle + c_1\,|1\rangle$, where $c_0, c_1 : \mathbb{C}$ are amplitudes satisfying $|c_0|^2 + |c_1|^2 = 1$. A qubit can be measured, resulting in the bit 0 with probability $|c_0|^2$, or the bit 1 with probability $|c_1|^2$. Qubits e can also be manipulated by applying one of a set of unitary matrices U, with application written $U \# e$. For example, the X (pronounced "not") unitary swaps the amplitudes of $|0\rangle$ and $|1\rangle$, so measuring a qubit of the form $X \# e$ is the same as[3] negating the measurement of e: $\texttt{meas}(X \# e) = \neg(\texttt{meas}\ e)$.

Quantum programs, therefore, do four main things: they initialize qubits, apply unitary gates, measure qubits, and invoke classical programs to process classical measurement results. This means that quantum programs are low-level—they often lack basic abstractions like data structures and parametricity—and effectful—measuring a qubit can non-locally affect a different part of the quantum state.

Researchers realized early on that quantum programming languages would benefit from formal study, including type systems [23, 3], verification [28, 19], and denotational semantics [2]. For example, this work uses linear types to enforce the *no-cloning* principle of quantum mechanics: unknown quantum states cannot be duplicated. This means that a program that uses quantum data twice, like $\lambda x.(x,x)$, might not correspond to a valid quantum computation. Linear types enforce the fact that quantum variables occur exactly once in a term, so that well-typed programs have a sound denotational semantics [23, 22]

[3] By "is the same" we mean "results in the same probability distribution."

This work also focuses on equational theories, which characterize when two quantum programs are equivalent. Equational theories help programmers understand the meaning of their programs and help compiler writers ensure that their optimizations are sound. Unfortunately, developing sound equational theories for effectful languages in general, and quantum languages in particular, is notoriously difficult.

2.2 Homotopy type theory (HoTT)

Homotopy type theory is, in many ways, a theory of equivalence. In HoTT, proofs of equality $p : a = b$, called *paths*, may have computational content. That is, there may be other proofs of equality besides the trivial reflexivity path $1_a : a = a$.

In HoTT, we write $a = b$ for the type of *propositional* proofs of equality; that is, a type with a single constructor $1_a : a = a$. Propositional equality is distinguished from *judgmental* equality $a \equiv b$, which asserts that a and b are equal by definition. The judgment $a \equiv b$ is not a type; it is only valid in the meta-theory and has no computational content. For more intuition on the difference between propositional and judgmental equality, see the HoTT book [27, Chapter 1].

Homotopy type theory was developed as a type-theoretic alternative to set theory, but it has applications in a wide variety of computational domains [5, 8, 1]. When a domain is difficult to characterize equationally, but uses data in the shape of an equivalence relation or groupoid, HoTT can help.

Consider a type A that you want to quotient by an equivalence relation R. For every element $a : A$, the equivalence class of a is written $[a]_R : A/R$, and whenever $R(a,b)$, it should be the case that $[a]_R = [b]_R$. In set theory it is possible to define the equivalence class as a set $[a]_R = \{x : A \mid R(a,x)\}$, so that $[a]_R$ contains the same elements as $[b]_R$. In programming environments, where the representation of data structures matters, sets are often implemented as lists or arrays, so $[a]_R$ must be carefully constructed so that it has the same representation as $[b]_R$.

Homotopy type theory sidesteps this representation problem with *higher inductive types*: inductive definitions with constructors for both terms and paths.

Definition 1 (Informal)**.** *The quotient A/R of a type A by an equivalence relation R on A is a higher inductive type generated by the following constructors: (1) for $a : A$, there is a term $[a]_R : A/R$; and (2) for $a, b : A$ and $r : R(a,b)$, there is a proof $[r]_R : [a]_R = [b]_R$.*

Notice that if r_1 and r_2 are two different witnesses (proofs) of $R(a,b)$, then $[r_1]_R$ is different from $[r_2]_R$—the structure of the relation R is preserved in the paths of A/R. Note also that Definition 1 is informal; see Sojakova [24] for a formal definition of set quotients with types and induction principles specified. In that work, Sojakova formalizes higher inductive types as homotopy-intial algebras.

Despite the extra computational content of equality types in HoTT, the usual properties still hold for paths generated by higher inductive types. The principle of *path induction* states that, given a property $P : \prod_{a,b:\alpha} a = b \to \mathtt{Type}$ on paths, if P holds on the reflexivity path, then P holds on any path. That is:

$$\mathtt{path\_ind}_P : \left(\prod(x : \alpha), P_{x,x}(1_x)\right) \to \prod(x\ y : \alpha)(p : x = y),\ P_{x,y}(p).$$

If $p : a = b$ for $a, b : \alpha$ and $x : Q(a)$ for some property $Q : \alpha \to \mathtt{Type}$, then it is possible to *transport* the path p over x to obtain a proof $\mathtt{transport}_Q\ p\ x : Q(b)$. If a and b are types and $x : a$, we write $\mathtt{coerce}\ p\ x \equiv \mathtt{transport}_{\lambda x.x}\ p\ x : b$.

Functions $f : \alpha \to \beta$ in HoTT are *functorial*, meaning that paths $p : a = b$ on α can be promoted to paths $\mathtt{ap}_f\ p : fa = fb$. An *equivalence* $f : \alpha \cong \beta$ between types α and β consists of a pair of functions $f : \alpha \to \beta$ and $f^{-1} : \beta \to \alpha$, along with proofs $\eta : \prod_a f^{-1}(fa) = a$ and $\varepsilon : \prod_b f(f^{-1}b) = b$ such that $\prod_a \mathtt{ap}_f\ \eta_a = \varepsilon_{f\ a}$ [27, Definition 4.2.1]. The *univalence axiom* states that an equivalence $f : \alpha \cong \beta$ between types can be coerced to a path $\mathtt{univ}\ f : \alpha = \beta$, such that $\mathtt{coerce}\ (\mathtt{univ}\ f)\ a = fa$.

2.3 Main idea—unitaries as paths

The core idea of this work is to encode the unitaries in the higher inductive structure of quantum types.

We start by defining unitary matrices in homotopy type theory. Let $\mathbb{C}$ be a type of complex numbers in HoTT.[4] For any finite types $\alpha, \beta : \mathtt{FinType}$, let $\mathtt{Matrix}(\alpha,\beta)$ be the set of complex-valued $2^{|\alpha|} \times 2^{|\beta|}$ matrices.[5] We write I for the identity matrix, AB for matrix multiplication, $A^\dagger$ for the conjugate transpose of A, $A \otimes B$ for the tensor product, and $A \oplus B$ for the block matrix $\left(\begin{smallmatrix} A & 0 \\ 0 & B \end{smallmatrix}\right)$.

A unitary matrix $\mathtt{UMatrix}(\alpha,\beta)$ is a matrix $U : \mathtt{Matrix}(\alpha,\beta)$ such that its conjugate transpose is its own inverse: $U^\dagger U = UU^\dagger = I$. Unitary transformations `UMatrix` form a groupoid—a category whose objects are finite types, and whose morphisms $\mathtt{UMatrix}(\alpha,\beta)$ are all convertible.

In this paper we take the types of our quantum language to be $\mathtt{QType} \equiv \mathtt{FinType}/_1\mathtt{UMatrix}$, the *groupoid quotient* of `UMatrix`.[6] Qubits are represented as the type $[\mathtt{Bool}]_{\mathtt{UMatrix}}$ of two-dimensional vector spaces, and unitary transformations $U : \mathtt{UMatrix}(\alpha,\beta)$ are encoded as paths of type $[\alpha]_{\mathtt{UMatrix}} = [\beta]_{\mathtt{UMatrix}}$. For quantum types σ and τ, we write $\mathscr{U}(\sigma,\tau)$ for the unitary path type $\sigma = \tau$.

Encoding unitaries as paths has two important consequences. First, there is no need for explicit syntax for applying a unitary transformation; unitary application $U \# e$ is defined to be `transport` $U\ e$, the result of transporting the path $U : \sigma = \tau$ over a term $\Gamma \vdash e : \sigma$. Second, many of the structural axioms on unitaries can now be proven by path induction. For example:

Proposition 2. *Suppose $\Gamma \vdash e : \sigma$. Then, for $U : \sigma = \tau$ and $V : \tau = \rho$ we have $V \# (U \# e) = (V \circ U) \# e$.*

Proof. By path induction over V. If V is the trivial path by reflexivity on σ, written 1_σ, then $V \circ U = U$. By the definition of `transport`, for all x we have $1 \# x = x$. So $1 \# (U \# e) = U \# e = (1 \circ U) \# e$. □

Crucially, it is *not* possible to prove the following false statement:

Proposition 3 (False)**.** *Let $\Gamma \vdash e : \sigma$ and $U : \sigma = \sigma$. Then $U \# e = e$.*

Path induction only applies on proofs $a = b$ when at least one of a or b is a free variable, so it does not apply here. In fact, the statement is false—Proposition 11 will show that $X \# |0\rangle = |1\rangle$, but it is not the case that $|0\rangle = |1\rangle$ due to the soundness of the denotational semantics (Theorem 22).

3 A quantum term calculus

This section presents a specification of an embedded linear quantum language. The embedding, inspired by the linearity monad embedding [17] and linear/non-linear type theory [7], lets us use non-linear data in a natural way. In Section 4 we implement this language in HoTT by encoding unitaries as paths.

Quantum types $\sigma, \tau : \mathtt{QType}$ consist of linear pairs $\sigma \otimes \tau$ and sums $\sigma \oplus \tau$, as well as host language (also called *classical*) types $\mathtt{Lower}\ \alpha$, which represent superpositions of classical values of type α. Our semantics is restricted to finite-dimensional vector spaces, so we insist that host types α be finite.

A typing context $\Gamma : \mathtt{Ctx}$ is a finite map from linear variables `Var` to `QTypes`. We write $\emptyset$ for the empty typing context, $x : \sigma$ for the singleton typing context, and Γ_1, Γ_2 for the disjoint merge.

[4] The precise formulation of the complex numbers in HoTT is not relevant for this work; for concreteness we can pick a representation based on the Dedekind reals [27, Chapter 11]. But we never need to define functions out of $\mathbb{C}$, and so all that we require is that basic computational properties, such as arithmetic and the complex conjugate, are valid.

[5] Here we use $|\alpha|$ to refer to the size of the finite type α. The fact that $\mathtt{Matrix}(\alpha,\beta)$ is a set means that for any two paths $p_1, p_2 : A = B$ between matrices $A, B : \mathtt{Matrix}(\alpha,\beta)$, it is the case that $p_1 = p_2$.

[6] Section 4.1 extends the set quotient type on equivalence relations α/R to groupoid quotients $\alpha/_1 G$.

$$\frac{\Gamma = x : \sigma}{x : \texttt{QExp}\ \Gamma\ \sigma}\ \text{VAR} \qquad \frac{e : \texttt{QExp}\ \Gamma\ \sigma \qquad e' : \texttt{QExp}\ (\Gamma', x : \sigma)\ \tau}{\texttt{let}\ x := e\ \texttt{in}\ e' : \texttt{QExp}\ (\Gamma, \Gamma')\ \tau}\ \text{LET}$$

$$\frac{e_1 : \texttt{QExp}\ \Gamma_1\ \sigma_1 \qquad e_2 : \texttt{QExp}\ \Gamma_2\ \sigma_2}{(e_1, e_2) : \texttt{QExp}\ (\Gamma_1, \Gamma_2)\ (\sigma_1 \otimes \sigma_2)}\ \otimes\text{-I} \qquad \frac{e : \texttt{QExp}\ \Gamma\ (\sigma_1 \otimes \sigma_2) \qquad e' : \texttt{QExp}\ (\Gamma', x_1 : \sigma_1, x_2 : \sigma_2)\ \tau}{\texttt{let}\ (x_1, x_2) := e\ \texttt{in}\ e' : \texttt{QExp}\ (\Gamma, \Gamma')\ \tau}\ \otimes\text{-E}$$

$$\frac{e_i : \texttt{QExp}\ \Gamma\ \sigma_i}{\iota_i e_i : \texttt{QExp}\ \Gamma\ (\sigma_1 \oplus \sigma_2)}\ \oplus\text{-I} \qquad \frac{e : \texttt{QExp}\ \Gamma\ (\sigma_1 \oplus \sigma_2) \quad e_1 : \texttt{QExp}\ (\Gamma', x_1 : \sigma_1)\ \tau \quad e_2 : \texttt{QExp}\ (\Gamma', x_2 : \sigma_2)\ \tau}{\texttt{case}\ e\ \texttt{of}\ (\iota_1 x_1 \to e_1 \mid \iota_2 x_2 \to e_2) : \texttt{QExp}\ (\Gamma, \Gamma')\ \tau}\ \oplus\text{-E}$$

$$\frac{a : \alpha}{\texttt{put}\ a : \texttt{QExp}\ \emptyset\ (\texttt{Lower}\ \alpha)}\ \text{LOWER-I} \qquad \frac{e : \texttt{QExp}\ \Gamma\ (\texttt{Lower}\ \alpha) \qquad f : \alpha \to \texttt{QExp}\ \Gamma'\ \tau}{e >!\ f : \texttt{QExp}\ (\Gamma, \Gamma')\ \tau}\ \text{LOWER-E}$$

Figure 1: An embedded linear/non-linear type system.

Quantum expressions are given by the type $\texttt{QExp}\ \Gamma\ \sigma$, defined inductively by the rules in Figure 1. LExp represents a typing judgment; we sometimes write $\Gamma \vdash e : \sigma$ for $e : \texttt{QExp}\ \Gamma\ \sigma$. Most of the constructions are standard for a linear lambda calculus, except for $\texttt{Lower}\ \alpha$, which we describe here.

The introduction rule for $\texttt{Lower}\ \alpha$ says that for any $a : \alpha$ in the host type theory, there is a linear expression $\texttt{put}\ a$ of quantum type $\texttt{Lower}\ \alpha$ that does not use any linear variables. The elimination rule for $\texttt{Lower}\ \alpha$ has the form $e >!\ f$,[7] where e is a quantum term of type $\texttt{Lower}\ \alpha$ and f is a host-level function from values of type α to quantum expressions. Intuitively, the expression $e >!\ f$ *measures*[8] the quantum expression e, resulting in a value of type α, and then applies f to that value.

3.1 Equational theory

The behavior of this calculus is given as an equational theory, shown in Figure 2. We write $e\{e_1/x_1, \ldots, e_n/x_n\}$ for the simultaneous capture-avoiding substitution of the e_i's for the linear variables x_i in e, and $e_1 \sim_\alpha e_2$ (respectively $e_1 \sim_\beta e_2$) for the usual notion of α (respectively β) equivalence.

Eta equivalence, written $e_1 \sim_\eta e_2$, is allowed for product types but not in general for sums $\sigma \oplus \tau$ or classical types $\texttt{Lower}\ \alpha$. Semantically, case analysis for sums and $>!$ for $\texttt{Lower}\ \alpha$ perform quantum measurement (see Section 3.3), so a term e of type $\sigma \oplus \tau$ is *not* semantically equivalent to its η-expanded version, $\texttt{case}\ e\ \texttt{of}\ (\iota_1 x \to \iota_1 x \mid \iota_2 x \to \iota_2 x)$. However, η expansion for the multiplicative product and unit type $\texttt{Lower}\ ()$ are admissible. In fact, *any* two values of unit type are equivalent, because the unit type is terminal in the category of density matrices in which we define a denotational semantics in Section 6.

Commuting conversions, written $e \sim_{cc} e'$, describe how elimination forms can move within an expression [9, Chapter 10]. Rules of this form are common for linear lambda calculi [26].

3.2 Linear functions

The quantum language does not include higher-order functions, but we can encode first-order functions in our host language in a data type $\sigma \multimap \tau$ (pronounced σ "lolli" τ) representing computations with input σ and output τ, defined inductively by the constructor $\texttt{suspend} : \prod_x \texttt{QExp}\ (x : \sigma)\ \tau \to (\sigma \multimap \tau)$. To apply $g : \sigma \multimap \tau$ to an argument $\Gamma \vdash e : \sigma$, we define $\Gamma \vdash \texttt{force}\ g\ e : \tau$ so that $\texttt{force}(\texttt{suspend}\ x\ e')\ e \equiv e'\{e/x\}$.[9]

[7] Pronounced e "let-bang" f, in reference to the linear logic operation !, pronounced "bang". We sometimes write $\texttt{let}\ !x := e\ \texttt{in}\ e'$ for $e >!\ \lambda x.e'$ when it is clear that x is a host-level variable, not a linear variable bound by the linear typing context.

[8] The basis in which this measurement occurs corresponds to the fixed order of the finite type α.

[9] The eliminator says that for any $k : \prod_x \texttt{QExp}\ (x : \sigma)\ \tau \to \beta$ there is a function $\texttt{rec}^k_{\multimap} : (\sigma \multimap \tau) \to \beta$ such that $\texttt{rec}^k_{\multimap}(x\ e) \equiv k\ x\ (fx)$. The induction principle says that for any $P : (\sigma \multimap \tau) \to \texttt{Type}$, given a proof $p : \prod_x \prod_{e : \texttt{QExp}\ (x:\sigma)\ \tau} P\ (\texttt{suspend}\ x\ e)$,

$$\texttt{let}\ x \coloneqq e\ \texttt{in}\ e' \sim_\beta e'\{e/x\} \qquad (\beta\text{-LET})$$
$$\texttt{let}\ (x_1,x_2) \coloneqq (e_1,e_2)\ \texttt{in}\ e' \sim_\beta e'\{e_1/x_1, e_2/x_2\} \qquad (\beta\text{-}\otimes)$$
$$\texttt{case}\ \iota_i e\ \texttt{of}\ (\iota_1 x_1 \to e_1 \mid \iota_2 x_2 \to e_2) \sim_\beta e_i\{e/x_i\} \qquad (\beta\text{-}\oplus)$$
$$\texttt{put}\ a >!\ f \sim_\beta fa \qquad (\beta\text{-LOWER})$$

$$e \sim_\eta \texttt{let}\ (x_1,x_2) \coloneqq e\ \texttt{in}\ (x_1,x_2) \qquad \text{for } \Gamma \vdash e : \sigma_1 \otimes \sigma_2 \qquad (\eta\text{-}\otimes)$$
$$e_1 \sim_\eta e_2 \qquad \text{for } \Gamma \vdash e_1, e_2 : \texttt{Lower}\ () \qquad (\eta\text{-}())$$

$$e_0\{\texttt{let}\ y \coloneqq e\ \texttt{in}\ e'/x\} \sim_{cc} \texttt{let}\ y \coloneqq e\ \texttt{in}\ e_0\{e'/y\} \qquad (\text{CC-LET})$$
$$e_0\{\texttt{let}\ (y_1,y_2) \coloneqq e\ \texttt{in}\ e'/x\} \sim_{cc} \texttt{let}\ (y_1,y_2) \coloneqq e\ \texttt{in}\ e_0\{e'/x\} \qquad (\text{CC-}\otimes)$$
$$e_0\{\texttt{case}\ e\ \texttt{of}\ (\iota_1 y_1 \to e_1 \mid \iota_2 y_2 \to e_2)/x\} \sim_{cc} \texttt{case}\ e\ \texttt{of}\ (\iota_1 y_1 \to e_0\{e_1/x\} \mid \iota_2 y_2 \to e_0\{e_2/x\}) \qquad (\text{CC-}\oplus)$$
$$e_0\{e >!\ f/x\} \sim_{cc} e >!\ \lambda a.e_0\{fa/x\} \qquad (\text{CC-LOWER})$$

Figure 2: β, η, and commuting conversion equivalences. Recall that we write $\Gamma \vdash e : \tau$ for $e : \texttt{QExp}\ \Gamma\ \tau$.

3.3 Quantum data, and measurement as case analysis

A qubit is a quantum type with two basis elements, $|0\rangle$ and $|1\rangle$, so we encode $\texttt{Qubit} \equiv \texttt{Lower Bool}$. Initialization, of type $\texttt{Bool} \to \texttt{QExp}\ \emptyset\ \texttt{Qubit}$, is defined as $\lambda b.\ \texttt{put}\ b$, and measurement, of type $\texttt{Qubit} \multimap \texttt{Lower Bool}$, is $\texttt{suspend}(\lambda x.\ \texttt{let}\ !b \coloneqq x\ \texttt{in}\ \texttt{put}\ b)$. On first glance, these definitions appear not to be doing anything—$\texttt{force meas}\ e$ in particular is just the η expansion of e. But this highlights a critical semantic fact of our system: *case analysis performs quantum measurement*. This is the reason why η expansion is not sound in general: a measured qubit is *not* equivalent to an unmeasured one.

3.4 Unitary transformations.

The remaining components of the language are unitary transformations $\mathscr{U}(\sigma,\tau)$. A unitary $U : \mathscr{U}(\sigma,\tau)$ can be applied to an expression $e : \texttt{QExp}\ \Gamma\ \sigma$ to produce an expression $U\ \#\ e : \texttt{QExp}\ \Gamma\ \tau$. Section 4 will show how to derive the unitaries in the lambda calculus described so far; this section will spell out a specification of the properties we expect to hold of the quantum fragment.

Following Staton [25], we focus on four main ways to combine unitaries: if $U : \mathscr{U}(q,r)$ and $V : \mathscr{U}(q',r')$, then $U \otimes V : \mathscr{U}(q \otimes q', r \otimes r')$ is the tensor product of U by V, and $U \oplus V : \mathscr{U}(q \oplus q', r \oplus r')$ is the block matrix. In addition, unitaries form a groupoid: there is an identify unitary, written 1; unitaries are subject to composition, written $V \circ U$; and they are invertible, written $U^\dagger$. Staton proves that all unitary matrices can be constructed from 1-qubit unitaries with the direct sum and tensor product [25].

The equational theory of unitaries, shown in Figure 3, is divided into three classes. First, the "structural" axioms characterize how unitaries interact with syntactic forms of the language. For example, Equation U-$\otimes$-INTRO describes how the tensor product $U_1 \otimes U_2$ distributes over pairs. The groupoid axioms characterize that unitaries form a groupoid.

The third set of axioms describe how certain unitaries interact with initialization and measurement. Such unitaries are completely defined by isomorphisms on their basis sets, which we call *unitary equiv-*

there is a function $\texttt{ind}^p_{\multimap} : \prod_{f:\sigma\multimap\tau} P\ f$ such that $\texttt{ind}^p_{\multimap}(\texttt{suspend}\ x\ e) \equiv p\ x\ e$. $\texttt{force}$ is defined as $\texttt{rec}^{\lambda x\ e_0.\ e_0\{e/x\}}_{\multimap}$.

Structural Axioms

$$(U_1 \otimes U_2)\,\#\,(e_1,e_2) \approx (U_1\,\#\,e_1, U_2\,\#\,e_2) \qquad \text{(U-}\otimes\text{-INTRO)}$$
$$\texttt{let}\ (x_1,x_2) \coloneqq (U_1 \otimes U_2)\,\#\,e\ \texttt{in}\ e' \approx \texttt{let}\ (y_1,y_2) \coloneqq e\ \texttt{in}\ e'\{U_1\,\#\,y_1/x_1, U_2\,\#\,y_2/x_2\} \qquad \text{(U-}\otimes\text{-ELIM)}$$
$$U\,\#\,(\texttt{let}\ (x_1,x_2) \coloneqq e\ \texttt{in}\ e') \approx \texttt{let}\ (x_1,x_2) \coloneqq e\ \texttt{in}\ U\,\#\,e' \qquad \text{(U-}\otimes\text{-COMM)}$$

$$(U_1 \oplus U_2)\,\#\,(\iota_1 e) \approx U_1\,\#\,e \qquad \text{(U-}\oplus\text{-INTRO}_1\text{)}$$
$$(U_1 \oplus U_2)\,\#\,(\iota_2 e) \approx U_2\,\#\,e \qquad \text{(U-}\oplus\text{-INTRO}_2\text{)}$$
$$\begin{aligned}&\texttt{case}\ (U_1 \oplus U_2)\,\#\,e\ \texttt{of}\ (\iota_1 x_1 \to e_1 \mid \iota_2 x_2 \to e_2)\\ &\quad\approx \texttt{case}\ e\ \texttt{of}\ (\iota_1 y_1 \to e_1\{U_1\,\#\,y_1/x_1\} \mid \iota_2 y_2 \to e_2\{U_2\,\#\,y_2/x_2\})\end{aligned} \qquad \text{(U-}\oplus\text{-ELIM)}$$
$$U\,\#\,(\texttt{case}\ e\ \texttt{of}\ (\iota_1 x_1 \to e_1 \mid \iota_2 x_2 \to e_2)) \approx \texttt{case}\ e\ \texttt{of}\ (\iota_1 x_1 \to U\,\#\,e_1 \mid \iota_2 x_2 \to U\,\#\,e_2) \qquad \text{(U-}\oplus\text{-COMM)}$$

$$U\,\#\,(e >!\, f) \approx e >!\, \lambda x \to U\,\#\,(fx) \qquad \text{(U-LOWER-COMM)}$$
$$U\,\#\,e >!\, \lambda \_.e' \approx e >!\, \lambda \_.e' \qquad \text{(U-LOWER-ELIM)}$$

Groupoid Axioms

$$U\,\#\,(V\,\#\,e) \approx (U \circ V)\,\#\,e \qquad \text{(U-COMPOSE)}$$
$$I\,\#\,e \approx e \qquad \text{(U-I)}$$
$$U^\dagger\,\#\,U\,\#\,e \approx e \qquad \text{(U-}\dagger\text{)}$$

Semantic Axioms

$$X\,\#\,\texttt{init}\ b \approx \texttt{init}(\neg b) \qquad \text{(X-INTRO)}$$
$$\texttt{let}\ !x \coloneqq \texttt{meas}(X\,\#\,e)\ \texttt{in}\ e' \approx \texttt{let}\ !y \coloneqq \texttt{meas}\ e\ \texttt{in}\ e'\{\neg y/x\} \qquad \text{(X-ELIM)}$$

$$\texttt{SWAP}\,\#\,(e_1,e_2) \approx (e_2,e_1) \qquad \text{(SWAP-INTRO)}$$
$$\texttt{let}\ (x,y) \coloneqq \texttt{SWAP}\,\#\,e\ \texttt{in}\ e' \approx \texttt{let}\ (y,x) \coloneqq e\ \texttt{in}\ e' \qquad \text{(SWAP-ELIM)}$$

$$\texttt{DISTR}\,\#\,(\texttt{init}\ b, e) \approx \texttt{if}\ b\ \texttt{then}\ \iota_2\ e\ \texttt{else}\ \iota_1\ e \qquad \text{(DISTR-INTRO)}$$
$$\texttt{case}(\texttt{DISTR}\,\#\,e)\ \texttt{of}\ (\iota_1 z_1 \to e_1 \mid \iota_2 z_2 \to e_2) \approx \texttt{let}\ (!b, y) \coloneqq e\ \texttt{in}\ (\texttt{init}\ b, e) \qquad \text{(DISTR-ELIM)}$$

Figure 3: Structural, groupoid, and semantic axioms. The relation $e \approx e'$ is the union of α, β, η, and commuting conversion relations, along with the quantum-specific axioms from Section 5. For the semantic axioms, $X : \mathscr{U}(\texttt{Qubit},\texttt{Qubit})$, $\texttt{SWAP} : \mathscr{U}(\sigma_1 \otimes \sigma_2, \sigma_2 \otimes \sigma_1)$, and $\texttt{DISTR} : \mathscr{U}(\texttt{Qubit} \otimes \tau, \tau \oplus \tau)$.

alences $\sigma \Longleftrightarrow \tau$, and define formally in Section 5. Every unitary equivalence f can be lifted to a unitary transformation $\widetilde{f} : \mathscr{U}(\sigma,\tau)$. For example, for the equivalence $\texttt{swap} : \prod_{\sigma_1,\sigma_2} \sigma_1 \otimes \sigma_2 \Longleftrightarrow \sigma_2 \otimes \sigma_1$, it should be the case that $\widetilde{\texttt{swap}}\,\#\,(e_1,e_2) \approx (e_2,e_1)$ as shown in Figure 3. We call (e_1,e_2) the *partial initialization* of the quantum system $\Pi_{\sigma_1,\sigma_2}\sigma_1 \otimes \sigma_2$. Partial initialization and its counterpart, partial measurement, precisely characterize the behavior of unitary equivalences $f : \sigma \Longleftrightarrow \tau$:

$$\widetilde{f}\,\#\,\texttt{init}_\sigma\ b \approx \texttt{init}_\tau(fb) \quad \text{(U-INTRO)} \qquad \texttt{match}_\tau\ (\widetilde{f}\,\#\,e)\ \texttt{with}\ g \approx \texttt{match}_\sigma\ e\ \texttt{with}\ g \circ f. \quad \text{(U-ELIM)}$$

4 Deriving equational rules in HoTT

The goal of this section is to encode unitaries in quantum types, to minimize the number of axioms needed to recover the equational theory described in the previous section. As described in Section 2.3, we do this by encoding unitaries in the groupoid quotient $\texttt{QType} \equiv \texttt{FinType}/_1\texttt{UMatrix}$.

Section 4.1 have been formalized in Coq using the HoTT library.[10]

4.1 `QType` as a groupoid quotient

Definition 4 (Sojakova [24], Section 4.3)**.** *If G is a groupoid with objects α, then the* groupoid quotient *of G, written $\alpha/_1 G$, is a higher inductive type with the following constructors:*

$$\texttt{point} : \alpha \to \alpha/_1 G \qquad \texttt{cell\_compose} : \Pi_{f,g}\texttt{cell}(g \circ f) = \texttt{cell}\ g \circ \texttt{cell}\ f$$
$$\texttt{cell} : \Pi_{a,b} G(a,b) \to \texttt{point}\ a = \texttt{point}\ b \qquad \texttt{is-1-type} : \texttt{1-type}(\alpha/_1 G)$$

The fact that $\alpha/_1 G$ is a 1-type means that, for $x, y : \alpha/_1 G$, if $f, g : x = y$ and $p, q : f = g$, then $p = q$.

The induction principle (in the extended version of this paper [18]) states that for a predicate P on $\alpha/_1 G$, there is a proof $\texttt{ind}_P : \prod x, Px$, provided (1) for all x, Px is a 1-type; (2) for all $a : \alpha$, there is a proof $\texttt{P\_point}_a : P(\texttt{point}\ a)$; and (3) for all $f : G(a,b)$, there is a proof $\texttt{P\_cell}_f : \texttt{transport}_P\ (\texttt{cell}\ f)\ (\texttt{P\_point}_a) = \texttt{P\_point}_b$ satisfying certain conditions.

We define `QType` to be the groupoid quotient of `UMatrix`: $\texttt{QType} \equiv \texttt{FinType}/_1\texttt{UMatrix}$. Intuitively, for $\sigma, \tau : \texttt{QType}$, the type $\sigma = \tau$ corresponds to unitary transformations from σ to τ. The groupoid quotient ensures that the identity and inverse of paths correspond to the appropriate operations on matrices.

Proposition 5. *Let $I : \texttt{UMatrix}(\alpha, \alpha)$ be the identity matrix on α. Then $\texttt{cell}\ I = 1_{\texttt{point}\ \alpha}$.*

Proposition 6. *Let $U : \texttt{UMatrix}(\alpha, \beta)$. Then $(\texttt{cell}\ U)^{-1} = \texttt{cell}\ U^\dagger$.*

Next we need to define the type formers `Lower`, $\otimes$, and $\oplus$. We take $\texttt{Lower}\ \alpha$ to be $(\texttt{point}\ \alpha) : \texttt{QType}$, and $\otimes$ and $\oplus$ are defined by quotient induction. To define $\otimes : \texttt{QType} \to \texttt{QType} \to \texttt{QType}$, we apply a variant of the quotient recursion principle on two variables, which can be found in the extended version of this paper [18], and in the Coq formalization. For $U : \sigma = \tau$ and $U' : \sigma' = \tau'$, we lift the tensor product to $U \otimes U' : \sigma \otimes \sigma' = \tau \otimes \tau'$. The computation principle states that $\texttt{cell}\ U \otimes \texttt{cell}\ U' = \texttt{cell}(U \otimes U')$.

4.2 Deriving the groupoid axioms

The fact that unitaries are paths means that Figure 3's groupoid axioms can be derived for free by path induction and facts of linear algebra.

Proposition 7 (U-COMPOSE)**.** *Let $V : \sigma = \tau$ and $U : \tau = \rho$. Then $U \# (V \# e) = (U \circ V) \# e$.*

Proposition 8 (U-I)**.** *If $e : \texttt{QExp}\ \Gamma\ \sigma$ then $\texttt{cell}\ I \# e = e$.*

Proposition 9 (U-†)**.** *If $U : \sigma = \tau$ and $e : \texttt{QExp}\ \Gamma\ \sigma$ then $U^\dagger \# U \# e = e$.*

4.3 Deriving the structural axioms

With one exception, the structural axioms from Figure 3 are trivial by path induction, and we omit their proofs here. The exception is the behavior of U-LOWER-ELIM: when a qubit is measured but the result is not relevant to the rest of the computation.

Proposition 10. *If $U : \texttt{Qubit} = \texttt{Qubit}$, then $\texttt{let}\ !\_ := \texttt{meas}(U \# e)\ \texttt{in}\ e' \approx \texttt{let}\ !\_ := \texttt{meas}\ e\ \texttt{in}\ e'$.*

Proof. It is not possible to do induction on U here, since its endpoints are both fixed. However, Proposition 10 follows from the η rule for the unit type. $e_1 \sim_\eta e_2$. □

[10] The formalization is available on github: `https://github.com/jpaykin/GroupoidQuotient`. It uses the HoTT library, an extention to Coq that supports homotopy type theory and higher inductive types [6].

$$[X]^m \equiv mX \qquad [\texttt{Lower}\ \alpha]^m \equiv \alpha \qquad [\sigma_1 \otimes \sigma_2]^m \equiv [\sigma_1]^m \times [\sigma_2]^m \qquad [\sigma_1 \oplus \sigma_2]^m \equiv [\sigma_1]^m + [\sigma_2]^m$$

$$\gamma^m_X(x : \texttt{Var}) \equiv x : \texttt{point}(mX) \qquad \gamma^m_{\texttt{Lower}\ \alpha}(a : \alpha) \equiv \emptyset$$
$$\gamma^m_{\sigma_1 \otimes \sigma_2}(b_1, b_2) \equiv \gamma^m_{\sigma_1}(b_1), \gamma^m_{\sigma_2}(b_2) \qquad \gamma^m_{\sigma_1 \oplus \sigma_2}(\texttt{inl}\ b_1) \equiv \gamma^m_{\sigma_1}(b_1) \qquad \gamma^m_{\sigma_1 \oplus \sigma_2}(\texttt{inr}\ b_2) \equiv \gamma^m_{\sigma_2}(b_2)$$

$$\texttt{init}^m_X\ x \equiv x \qquad \texttt{init}^m_{\texttt{Lower}\ \alpha}\ a \equiv \texttt{put}\ a$$
$$\texttt{init}^m_{\sigma_1 \otimes \sigma_2}\ (b_1, b_2) \equiv (\texttt{init}^m_{\sigma_1}\ b_1, \texttt{init}^m_{\sigma_2}\ b_2) \qquad \texttt{init}^m_{\sigma_0 \oplus \sigma_1}(\texttt{inl}\ b_0) \equiv \iota_0(\texttt{init}^m_{\sigma_0}\ b_0) \qquad \texttt{init}^m_{\sigma_0 \oplus \sigma_1}(\texttt{inr}\ b_1) \equiv \iota_1(\texttt{init}^m_{\sigma_1}\ b_1)$$

$$\frac{\Gamma \vdash e : \texttt{point}\ [\sigma]^m \qquad bs : \prod_{b : [\sigma]^{\underline{\texttt{Var}}}} \gamma^m_\sigma(b), \Gamma' \vdash - : \tau}{\Gamma, \Gamma' \vdash \texttt{match}_\sigma\ e\ \texttt{with}\ bs : \tau}$$

$$\texttt{match}_X\ e\ \texttt{with}\ bs \equiv (bs\ x)\{e/x\} \qquad \text{where } x \text{ is fresh}$$
$$\texttt{match}_{\texttt{Lower}\ \alpha}\ e\ \texttt{with}\ bs \equiv e >\!!\ bs$$
$$\texttt{match}_{\sigma_1 \otimes \sigma_2}\ e\ \texttt{with}\ bs \equiv \texttt{let}\ (x_1, x_2) \coloneqq e\ \texttt{in}\ \texttt{match}_{\sigma_1}\ x_1\ \texttt{with}\ \lambda b_1.\ \texttt{match}_{\sigma_2}\ x_2\ \texttt{with}\ \lambda b_2.\ bs(b_1, b_2)$$
$$\texttt{match}_{\sigma_0 \oplus \sigma_1}\ e\ \texttt{with}\ bs \equiv \texttt{case}\ e\ \texttt{of}\ \begin{cases} \iota_0 x_0 \to \texttt{match}_{\sigma_0}\ x_0\ \texttt{with}\ \lambda b_0.\ bs(\texttt{inl}\ b_0) \\ \iota_1 x_1 \to \texttt{match}_{\sigma_1}\ x_1\ \texttt{with}\ \lambda b_1.\ bs(\texttt{inr}\ b_1) \end{cases}$$

Figure 4: Operations on open quantum types

5 Equivalence of unitaries

This section establishes the soundness of the semantic axioms from Figure 3. First, the "not" unitary:

Proposition 11 (X-INTRO and X-ELIM)**.**
$\texttt{cell}\ X\ \#\ \texttt{put}\ b = \texttt{put}(\neg b)$ *and* $(\texttt{cell}\ X\ \#\ e) >\!!\ f = e >\!!\ \lambda b. f(\neg b)$.

The proof of this proposition relies on the following two lemmas, both proved by path induction:

Lemma 12. *For any* $f : \alpha = \beta$ *and* $a : \alpha$*:*
$\texttt{ap}_{\texttt{point}}\ f\ \#\ \texttt{put}\ a = \texttt{put}(\texttt{coerce}\ f\ a)$ *and* $\texttt{ap}_{\texttt{point}}\ f\ \#\ e) >\!!\ g = e >\!!\ \lambda x.\ g(\texttt{coerce}\ f\ x)$.

Lemma 13. *If* $U : \texttt{UMatrix}(\alpha_1, \alpha_2)$ *and* $H : \alpha_2 = \alpha_3$*, then* $\texttt{cell}(\texttt{transport}\ H\ U) = \texttt{ap}_{\texttt{point}}\ H \circ \texttt{cell}\ U$.

Proof of Proposition 11. Instantiating Lemma 12 with $(\texttt{univ}\ \neg)$, it suffices to check that $\texttt{cell}\ X = \texttt{ap}_{\texttt{point}}(\texttt{univ}\ \neg)$. Notice that, as matrices, $X = \texttt{transport}_{\texttt{UMatrix}(\texttt{Bool},-)}\ (\texttt{univ}\ \neg)\ I$. Then

$$\begin{aligned} \texttt{cell}(\texttt{transport}_{\texttt{UMatrix}(\texttt{Bool},-)}\ (\texttt{univ}\ \neg)\ I) &= \texttt{ap}_{\texttt{point}}(\texttt{univ}\ \neg) \circ \texttt{cell}\ I && (\textit{Lemma}\ 13) \\ &= \texttt{ap}_{\texttt{point}}(\texttt{univ}\ \neg) && (\textit{Proposition}\ 5) \quad \square \end{aligned}$$

This technique does not extend to polymorphic equivalences such as $\texttt{swap} : \prod \alpha\beta, \alpha \times \beta = \beta \times \alpha$. Lemma 12 tells us how $\texttt{SWAP} \equiv \texttt{ap}_{\texttt{point}}\ \texttt{swap}$ behaves on classical states: $\texttt{SWAP}\ \#\ \texttt{put}\ (a, b) = \texttt{put}\ (b, a)$. But Equation SWAP-INTRO is an even stronger statement: that $\texttt{SWAP}\ \#\ (e_1, e_2) \sim_q (e_2, e_1)$ for any e_1 and e_2. Similarly, the elimination form of Lemma 12 tells us that measuring both components of $\texttt{SWAP}\ \# e$, where e is a pair of qubits, is the same as measuring e and then swapping its arguments.

We can think of SWAP's behavior as acting on a state whose structure is only *partially* known, corresponding to the polymorphism of its underlying function swap. Our solution is to define a sort of *partial initialization* and *partial measurement* that generalizes this notion for swap and other polymorphic paths.

Consider quantum types with the addition of type variables $X : \texttt{TVar}$: $\sigma ::= X \mid \texttt{Lower}\ \alpha \mid \sigma_1 \otimes \sigma_2 \mid \sigma_1 \oplus \sigma_2$. We call these *open quantum types*. Given a map $m : \texttt{TVar} \to \texttt{Type}$, we can define a basis set corresponding to σ, written $[\sigma]^m$, as shown in Figure 4.

Let $m : \texttt{TVar} \to \texttt{Type}$ and let $\underline{\texttt{Var}}$ be the constant map $\lambda\_.\texttt{Var}$. Then every $b : [\sigma]^{\underline{\texttt{Var}}}$ gives rise to a typing context $\gamma^m_\sigma(b)$ as well as a term using these variables: if $\Gamma = \gamma^m_\sigma(b)$ then $\Gamma \vdash \texttt{init}^m_\sigma\ b : \texttt{point}\ [\sigma]^m$ is called *partial initialization*, as defined in Figure 4. Similarly, *partial measurement* eliminates terms of type $\texttt{point}\ [\sigma]^m$. A *unitary equivalence* $\sigma \leftrightarrows \tau$ of open quantum types is a proof that $\prod_m [\sigma]^m \cong [\tau]^m$.

Lemma 14. *If $f : \sigma \leftrightarrows \tau$ then for every $b : [\sigma]^{\underline{\mathtt{Var}}}$ there is a path $\gamma^m_\tau(fb) = \gamma^m_\sigma(b)$.*

The proof of Lemma 14 is quite involved and can be found in the extended version of this paper [18].

We can finally complete the equational theory for our quantum language by defining two axioms, written $\sim_q$, about the behavior of partial initialization and partial measurement.

For $f : \sigma \leftrightarrows \tau$ and $m : \mathtt{TVar} \to \mathtt{Type}$, let us write $[f]^m : \mathtt{point}\ [\sigma^m] = \mathtt{point}\ [\tau]^m$ for $\mathtt{ap}_{\mathtt{point}}(\mathtt{univ}\ f_m)$. Let $b : [\sigma]^{\underline{\mathtt{Var}}}$, let $e : \mathtt{QExp}\ \Gamma\ (\mathtt{point}\ [\sigma]^m)$ and let $bs : \prod_{b' : [\tau]^{\underline{\mathtt{Var}}}} \mathtt{QExp}\ (\Gamma', \gamma^m_\tau(b'))\ (\tau)$.

Axiom 15.

$$[f]^m\ \#\ \mathtt{init}^m_\sigma\ b \sim_q \mathtt{init}^m_\tau(f_{\underline{\mathtt{Var}}} b) \qquad \text{(U-INTRO)}$$

$$\mathtt{match}_\tau\ [f]^m\ \#\ e\ \mathtt{with}\ bs \sim_q \mathtt{match}_\sigma\ e\ \mathtt{with}\ (bs \circ f_{\underline{\mathtt{Var}}}) \qquad \text{(U-ELIM)}$$

Definition 16. *Define the relation $e_1 \approx_q e_2$ on expressions as $\approx_q \equiv \sim_\alpha \cup \sim_\beta \cup \sim_\eta \cup \sim_{cc} \cup \sim_q$. We write $e_1 \approx e_2$ for equality modulo $\approx_q$,* i.e., *the type $[e_1]_{\approx_q} = [e_2]_{\approx_q}$.*

Proposition 17 (SWAP-INTRO and SWAP-ELIM)**.** *Let* SWAP *be the unitary $[\mathtt{swap}]^m$, where* swap *is the equivalence $\lambda(x,y).(y,x)$ of type $X \otimes Y \leftrightarrows Y \otimes X$. Then*

$$\mathtt{SWAP}\ \#\ (e_1, e_2) \approx (e_2, e_1) \qquad \text{(SWAP-INTRO)}$$

$$\mathtt{let}\ (y,x) \coloneqq \mathtt{SWAP}\ \#\ e\ \mathtt{in}\ e' \approx \mathtt{let}\ (x,y) \coloneqq e\ \mathtt{in}\ e' \qquad \text{(SWAP-ELIM)}$$

Proof. For the introduction rule, it suffices to show that $\mathtt{SWAP}\ \#\ (x,y) \approx (y,x)$ for free variables x and y.

$$\mathtt{SWAP}\ \#\ (x,y) \equiv \mathtt{SWAP}\ \#\ \mathtt{init}_{X\otimes Y}\ (x,y) \sim_q \mathtt{init}_{Y\otimes X}(\mathtt{swap}(x,y)) \equiv \mathtt{init}_{Y\otimes X}\ (y,x) \equiv (y,x)$$

Elimination is similarly straightforward from Axiom U-ELIM. □

Proposition 18. *Let* CNOT *be the unitary $\mathtt{ap}_{\mathtt{point}}\ \mathtt{cnot}$, where $\mathtt{cnot} \equiv \lambda(b,b').\ (b, \mathtt{if}\ b\ \mathtt{then}\ \neg b'\ \mathtt{else}\ b')$.*

$$\mathtt{CNOT}\ \#\ (\mathtt{put}\ b, e) \approx\ (\mathtt{put}\ b, \mathtt{if}\ b\ \mathtt{then}\ X\ \#\ e\ \mathtt{else}\ e) \qquad \text{(CNOT-INTRO)}$$

$$\mathtt{let}\ (!\_, y) \coloneqq \mathtt{CNOT}\ \#\ e\ \mathtt{in}\ e' \approx \mathtt{let}\ (!b, y') \coloneqq e\ \mathtt{in}\ \mathtt{if}\ b\ \mathtt{then}\ e'\{X\ \#\ y'/y\}\ \mathtt{else}\ e'\{y'/y\} \qquad \text{(CNOT-ELIM)}$$

Proof. As a matrix, CNOT is equal to $\mathtt{DISTR}^{-1} \circ (I \oplus X) \circ \mathtt{DISTR}$, where DISTR is the unitary $[\mathtt{distr}]^m$ for $\mathtt{distr} : \mathtt{Lower\ Bool} \otimes X \leftrightarrows X \oplus X$. From Equation U-INTRO we can derive that

$$\mathtt{DISTR}\ \#\ (\mathtt{put}\ b, e) \sim_q \mathtt{if}\ b\ \mathtt{then}\ \iota_1\ e\ \mathtt{else}\ \iota_0\ e \quad \text{and} \quad \begin{array}{l} \mathtt{DISTR}^{-1}\ \#\ (\iota_0 e) \sim_q (\mathtt{put}\ \mathtt{false}, e) \\ \mathtt{DISTR}^{-1}\ \#\ (\iota_1 e) \sim_q (\mathtt{put}\ \mathtt{true}, e) \end{array}$$

The result for CNOT follows from this, and the proof of Equation CNOT-ELIM follows similarly. □

6 Soundness

This section sketches a denotational semantics for the quantum term calculus with respect to superoperators over density matrices; the details can be found in the extended version of this paper [18].

For a quantum type $\sigma : \mathtt{QType}$, we define the type $\mathtt{Density}\ \sigma$ of density matrices of type σ by quotient induction. First, define $\mathtt{Density}(\mathtt{point}\ \alpha)$ to be the collection of density matrices of type $\mathtt{Matrix}(\alpha,\alpha)$. Next, we must show that for any unitary $U : \mathtt{UMatrix}(\alpha,\beta)$, we have $\mathtt{Density}(\mathtt{point}\ \alpha) = \mathtt{Density}(\mathtt{point}\ \beta)$. This path can be obtained through univalence from the equivalence $U^* \equiv \lambda\rho.U\rho U^\dagger$. The function U^* is a superoperator when U is unitary, and it is invertible via the function $(U^\dagger)^*$.

A reader might be concerned that because we defined $\mathtt{Density}\ \tau$ by quotient induction, we have inadvertently collapsed all two density matrices ρ_1, ρ_2 such that $\rho_2 = U^*\rho_1$. This is not true. Just

$$\frac{}{\Gamma, x:\sigma_1\otimes\cdots\otimes\sigma_n,\Gamma' \mid \overrightarrow{a_i:\sigma_i}\vdash x(\overrightarrow{a_i})}\ \text{VAR} \qquad \frac{\Gamma \mid \Delta, a_1:\sigma_1, a_2:\sigma_2 \vdash t}{\Gamma \mid \Delta, a:\sigma_1\otimes\sigma_2 \vdash a(a_1,a_2).t}\otimes \qquad \frac{x:\sigma \mid \Delta\vdash t \qquad \Gamma \mid \Theta, a:\sigma\vdash u}{\Gamma \mid \Delta,\Theta\vdash t[x(a)\mapsto u]}\ \text{SUBST}$$

$$\frac{\Gamma \mid \Delta, a:\texttt{Qubit}\vdash t}{\Gamma \mid \Delta\vdash \texttt{new}(a.t)}\ \text{NEW} \qquad \frac{\Gamma \mid \Delta\vdash t \qquad \Gamma \mid \Delta\vdash u}{\Gamma \mid \Delta, a:\texttt{Qubit}\vdash \texttt{meas}(a,t,u)}\ \text{MEAS} \qquad \frac{U:\mathscr{U}(\sigma,\tau) \qquad \Gamma \mid \Delta, b:\tau\vdash t}{\Gamma \mid \Delta, a:\sigma\vdash U(a,b.t)}\ \text{U}$$

Figure 5: Algebraic structure of quantum computation.

because $\rho : \texttt{Density}\ \sigma$ and $\texttt{Density}\ \sigma = \texttt{Density}\ \sigma$ does not mean that there is a path $\rho = U\rho U^\dagger$. In particular, notice that $\texttt{Density}\ \texttt{Qubit}$ is just a set of 2×2 matrices.

The denotational semantics of the quantum term language maps expressions $e : \texttt{QExp}\ \Gamma\ \sigma$ to a super-operator $[\![e]\!] : \texttt{Density}\ \Gamma \to \texttt{Density}\ q$ between density matrices. The soundness of β, η, and commuting conversion equivalences comes down to the fact that the category of density matrices is symmetric monoidal, has sums, and has a terminal object.

Lemma 19. *If $e_1 \sim_o e_2$ for $o \in \{\beta, \eta, cc\}$, then $[\![e_1]\!] = [\![e_2]\!]$.*

For any $U : \sigma = \tau$, define $[\![U]\!] : \texttt{Density}\ \sigma \to \texttt{Density}\ \tau$ by path induction such that $[\![1]\!] = \lambda x.x$.

Lemma 20. *If $U : \sigma = \tau$ and $\Gamma \vdash e : \sigma$, then $[\![U\ \#\ e]\!] = [\![U]\!] \circ [\![e]\!]$.*

We can now prove the soundness of the axioms regarding the behavior of unitary equivalences

Theorem 21 (Soundness of Axiom 15)**.** *Let $f : \sigma \leftrightarrows \tau$ and $b : [\sigma]^{\underline{\texttt{Var}}}$. Then $[\![\widetilde{f_m}\ \#\ \texttt{init}^m_\sigma\ b]\!] = [\![\texttt{init}^m_\tau(f_{\underline{\texttt{Var}}}b)]\!]$. If $e : \texttt{QExp}\ \Delta\ (\texttt{point}\ [\sigma]^m)$ and $bs : \prod_{bs:[\tau]^{\underline{\texttt{Var}}}} \texttt{QExp}\ (\gamma^m_\tau(b), \Delta')\ q$, then $[\![\texttt{match}_\tau\ (\widetilde{f_m}\ \#\ e)\ \texttt{with}\ bs]\!] = [\![\texttt{match}_\sigma\ e\ \texttt{with}\ bs \circ f_{\underline{\texttt{Var}}}]\!]$.*

Theorem 22 (Soundness)**.** *If $e_1 \approx e_2$, then $[\![e_1]\!] = [\![e_2]\!]$.*

7 Completeness

This section will draw a formal connection between the HoTT calculus defined in this paper and Staton's algebraic presentation, shown in Figure 5. The algebric calculus judgment $\Gamma \mid \Delta \vdash t$ is presented in continuation-passing-style. The intuition is that Γ contains continuation variables, to which quantum variables in Δ can be passed. In Staton's original presentation, variables could hold only qubits, but here we allow variables to hold arbitrary tuples. However, we do restrict types that occur in the algebraic calculus to those of the form $\texttt{Qubit}^n$. We say that a quantum type is *binary* if it either has the form $\texttt{Lower}\ \texttt{Bool}$, or has the form $\sigma_1 \otimes \sigma_2$ for binary types σ_1 and σ_2.

We choose to encode the unitary application rule U using `transport` as in Section 4: $U(a, b.t) \equiv \texttt{transport}\ U\ t\{a/b\}$. In addition, the substitution rule SUBST is defined by induction on t.

The equational theory for this algebra, $t \approx\!\!\approx u$, is a straightforward translation of the equational theory in Staton's presentation. We omit them here for space constraints, but details can be found in [18]. Roughly, they are broken down into "interesting" axioms corresponding to our semantic axioms (Figure 3), and "administrative" axioms corresponding to our structural and groupoid axioms. In addition, to account for the behavior of the $\otimes$ rule, which was not present in Staton's original presentation, we add three commuting conversion rules to our calculus, shown in Figure 6.

7.1 Algebraic to HoTT calculus translation

Let $x : \sigma \mid \Delta \vdash t$ be a term with exactly one continuation. We define $\langle t \rangle : \texttt{QExp}\ \Delta\ \sigma$ in Figure 7.

Theorem 23. *If $t \approx\!\!\approx u$ then $\langle t \rangle \approx \langle u \rangle$.*

$$a(a_1,a_2).b(b_1,b_2).t \approxeq b(b_1,b_2).a(a_1,a_2).t$$
$$a(a_1,a_2).\mathtt{new}(b.t) \approxeq \mathtt{new}(b.a(a_1,a_2).t)$$
$$a(a_1,a_2).\mathtt{meas}(b,t_1,t_2) \approxeq \mathtt{meas}(b,a(a_1,a_2).t_1,a(a_1,a_2).t_2)$$

Figure 6: Commuting conversion rules for pairs

$$\langle x(a_1,\ldots,a_n)\rangle \equiv (a_1,\ldots,a_n)$$
$$\langle \mathtt{new}(a.t)\rangle \equiv \mathtt{let}\ a := \mathtt{init}\ 0\ \mathtt{in}\ \langle t\rangle$$
$$\langle \mathtt{meas}(a,t,u)\rangle \equiv \mathtt{let}\ !x := a\ \mathtt{in}\ \mathtt{if}\ x\ \mathtt{then}\ \langle u\rangle\ \mathtt{else}\ \langle v\rangle$$

Figure 7: Translation from algebraic to HoTT calculus

$$\overline{x : \mathtt{BExp}\ (x:\sigma)\ \sigma} \quad \Rightarrow \quad \overline{y : \sigma \mid x : \sigma \vdash y(x)}$$

$$\frac{b : \mathtt{Bool}}{\mathtt{put}\ b : \mathtt{BExp}\ \emptyset\ (\mathtt{Qubit})} \quad \Rightarrow \quad \frac{b = \mathtt{false}}{x : \mathtt{Qubit} \mid \emptyset \vdash \mathtt{new}(a.x(a))} \quad , \quad \frac{b = \mathtt{true}}{x : \mathtt{Qubit} \mid \emptyset \vdash \mathtt{new}(a.X(a,x(a)))}$$

$$\frac{e : \mathtt{BExp}\ \Gamma\ (\mathtt{Lower}\ \mathtt{Bool}) \qquad f : \mathtt{Bool} \to \mathtt{BExp}\ \Gamma'\ \tau}{e >\!! f : \mathtt{BExp}\ (\Gamma,\Gamma')\ \tau} \quad \Rightarrow \quad \frac{x : \mathtt{Lower}\ \mathtt{Bool} \mid \Gamma \vdash (\!|e|\!) \qquad \prod_b y : \tau \mid \Gamma' \vdash (\!|fb|\!)}{y : \tau \mid \Gamma,\Gamma' \vdash (\!|e|\!)[x(q) \mapsto \mathtt{meas}(q, f(\mathtt{false}), f(\mathtt{true}))]}$$

Figure 8: Select rules for encoding the quantum HoTT calculus into Staton's algebraic calculus.

7.2 HoTT to algebraic calculus translation

Since the algebraic calculus can only represent binary types, we omit the term constructors for sum types $\oplus$. Figure 8 sketches the translation of these *binary* expressions $e : \mathtt{BExp}\ \Gamma\ \tau$ to $y : \tau \mid \Gamma \vdash (\!|e|\!)^y$.

Theorem 24. *If $e_1, e_2 : \mathtt{BExp}\ \Gamma\ \sigma$ and $e_1 \approx e_2$, then $(\!|e_1|\!) \approxeq (\!|e_2|\!)$.*

Lemma 25. *For all $e : \mathtt{BExp}\ \Gamma\ \sigma$ and $x : \tau \mid \Delta \vdash t$: $\langle (\!|e|\!) \rangle \approx e$ and $(\!|\langle t \rangle|\!)^x \approxeq t$.*

Theorem 26. *The* BExp *HoTT calculus is sound and complete with respect to the algebraic calculus.*

Staton proves that the algebraic calculus is sound and fully complete with respect to ${C^*_{\mathrm{CPU}}}^2$, the category of C^* algebras of dimension 2^n with completely positive and unitary transformations [25]. As a consequence, the completeness of the BExp fragment of the HoTT calculus extends to C^*_{CPU}. We speculate but have yet to prove that the unrestricted HoTT calculus is sound and complete with respect to C^*_{CPU} algebras of arbitrary dimension.

8 Conclusion

This paper presents an equational theory for a linear quantum term calculus in a compact and elegant style using homotopy type theory. We justify these claims by deriving an equational theory known to be complete for a less expressive language, and by proving the semantics is sound with respect to a standard model of quantum computation. In doing so, we have both introduced a new tool to the study of quantum equational theory, and also demonstrated the application of homotopy type theory in a programming environment.

Acknowledgment

The authors are grateful to Matthew Weaver, Antoine Voizard, and Sam Staton for discussions regarding this work, as well as the anonymous reviewers who have provided feedback. This work is supported in part by the AFRL MURI No. FA9550-16-1-0082.

References

[1] M. Abbott, T. Altenkirch, N. Ghani, and C. McBride. Constructing polymorphic programs with quotient types. In D. Kozen, editor, *Mathematics of Program Construction*, pages 2–15, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[2] S. Abramsky and B. Coecke. Categorical quantum mechanics. *Handbook of quantum logic and quantum structures: quantum logic*, pages 261–324, 2008.

[3] T. Altenkirch and A. S. Green. *The Quantum IO Monad*, page 173–205. Cambridge University Press, Nov 2009.

[4] M. Amy, M. Roetteler, and K. M. Svore. Verified compilation of space-efficient reversible circuits. In R. Majumdar and V. Kunčak, editors, *Computer Aided Verification*, pages 3–21. Springer International Publishing, 2017.

[5] C. Angiuli, E. Morehouse, D. R. Licata, and R. Harper. Homotopical patch theory. In *Proceedings of the 19th ACM SIGPLAN International Conference on Functional Programming*, ICFP '14, pages 243–256, New York, NY, USA, 2014. ACM.

[6] A. Bauer, J. Gross, P. L. Lumsdaine, M. Shulman, M. Sozeau, and B. Spitters. The hott library: A formalization of homotopy type theory in coq. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs*, CPP 2017, pages 164–172, New York, NY, USA, 2017. ACM.

[7] N. Benton. A mixed linear and non-linear logic: Proofs, terms and models. In L. Pacholski and J. Tiuryn, editors, *Computer Science Logic*, volume 933 of *Lecture Notes in Computer Science*, pages 121–135. Springer Berlin Heidelberg, 1995.

[8] S. Chu, K. Weitz, A. Cheung, and D. Suciu. HoTTSQL: Proving query rewrites with univalent SQL semantics. *SIGPLAN Not.*, 52(6):510–524, 2017.

[9] J.-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and types*, volume 7. Cambridge University Press Cambridge, 1989.

[10] A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, and B. Valiron. Quipper: A scalable quantum programming language. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, pages 333–342, New York, NY, USA, 2013. ACM.

[11] E. H. Knill. Conventions for quantum pseudocode. Technical Report LAUR-96-2724, Los Alamos National Laboratory, 1996.

[12] M. Y. Mahmoud and A. P. Felty. Formal meta-level analysis framework for quantum programming languages. *Electronic Notes in Theoretical Computer Science*, 338:185 – 201, 2018. The 12th Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2017).

[13] K. Matsumoto and K. Amano. Representation of quantum circuits with Clifford and pi/8 gates, June 2008, arXiv:quant-ph/0806.3834.

[14] Y. Nam, N. J. Ross, Y. Su, A. M. Childs, and D. Maslov. Automated optimization of large quantum circuits with continuous parameters. *ArXiv e-prints*, Oct. 2017, 1710.07345.

[15] J. Paykin. *Linear/Non-Linear Types for Embedded Domain-Specific Languages*. PhD thesis, University of Pennsylvania, 2018.

[16] J. Paykin, R. Rand, and S. Zdancewic. QWIRE: A core language for quantum circuits. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, pages 846–858, New York, NY, USA, 2017. ACM.

[17] J. Paykin and S. Zdancewic. The linearity monad. In *Proceedings of the 10th ACM SIGPLAN International Symposium on Haskell*, Haskell 2017, pages 117–132, New York, NY, USA, 2017. ACM.

[18] J. Paykin and S. Zdancewic. A HoTT quantum equational theory (extended version), 2019, arXiv:cs.PL/1904.04371.

[19] R. Rand, J. Paykin, and S. Zdancewic. QWIRE practice: Formal verification of quantum circuits in Coq. In *Proceedings 14th International Conference on Quantum Physics and Logic, QPL 2017, Nijmegen, The Netherlands, 3-7 July 2017.*, pages 119–132, 2017.

[20] F. Rios and P. Selinger. A Categorical Model for a Quantum Circuit Description Language (Extended Abstract), 2017, arXiv:quant-ph/1706.02630.

[21] N. J. Ross. *Algebraic and Logical Methods in Quantum Computation*. PhD thesis, Dalhousie University, 2015.

[22] P. Selinger and B. Valiron. On a fully abstract model for a quantum linear functional language. In *Proceedings of the 4th International Workshop on Quantum Programming Languages, QPL 2006*, Electronic Notes in Theoretical Computer Science 210, pages 123–137. Elsevier, 2008.

[23] P. Selinger and B. Valiron. Quantum lambda calculus. In S. Gay and I. Mackie, editors, *Semantic Techniques in Quantum Computation*, pages 135–172. Cambridge University Press, 2009.

[24] K. Sojakova. *Higher Inductive Types As Homotopy-Initial Algebras*. PhD thesis, Carnegie Mellon University, New York, NY, USA, 2016.

[25] S. Staton. Algebraic effects, linearity, and quantum programming languages. In *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, pages 395–406, New York, NY, USA, 2015. ACM.

[26] S. Staton and P. B. Levy. Universal properties of impure programming languages. In *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '13, pages 179–192, New York, NY, USA, 2013. ACM.

[27] T. Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. `http://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.

[28] M. Ying. Floyd–Hoare logic for quantum programs. *ACM Trans. Program. Lang. Syst.*, 33(6):19:1–19:49, 2012.