How to Bake a Quantum Π

JACQUES CARETTE, McMaster University, Canada
CHRIS HEUNEN, University of Edinburgh, United Kingdom
ROBIN KAARSGAARD, University of Southern Denmark, Denmark
AMR SABRY, Indiana University, USA

We construct a computationally universal quantum programming language QuantumΠ from two copies of Π, the internal language of rig groupoids. The first step constructs a pure (measurement-free) term language by interpreting each copy of Π in a generalisation of the category **Unitary** in which every morphism is "rotated" by a particular angle, and the two copies are amalgamated using a free categorical construction expressed as a computational effect. The amalgamated language only exhibits quantum behaviour for specific values of the rotation angles, a property which is enforced by imposing a small number of equations on the resulting category. The second step in the construction introduces measurements by layering an additional computational effect.

CCS Concepts: • **Theory of computation** → **Categorical semantics**; **Quantum computation theory**; • **Software and its engineering** → *General programming languages.*

Additional Key Words and Phrases: quantum programming language, unitary quantum computing, reversible computing, rig category

ACM Reference Format:
Jacques Carette, Chris Heunen, Robin Kaarsgaard, and Amr Sabry. 2024. How to Bake a Quantum Π. *Proc. ACM Program. Lang.* 8, ICFP, Article 236 (August 2024), 29 pages. https://doi.org/10.1145/3674625

1 Introduction

A distinguishing and well-established aspect of quantum theory is the concept of *complementarity*. Roughly speaking, an observation in one experimental setting excludes the possibility of gaining any information in a *complementary* setting. This phenomenon has been formalised as an equation that relates a particular collection of "classical morphisms" in the category of finite-dimensional Hilbert spaces, giving axiomatic models of quantum theory [16].

From a programming language perspective, the natural question is whether it is possible to extend a classical language to witness complementarity *within* the language. The most important practical significance of such a construction would be that some forms of reasoning about quantum programs would reduce to classical reasoning. Foundationally, this construction would turn around the prevalent view of quantum computing [21, 49, 50, 53], potentially shedding light on a long-standing foundational question in physics about the relationship between quantum and classical theories [8].

We give just such a recipe, namely constructing a computationally universal quantum programming language from two copies of a (particular) universal classical reversible language Π.

Authors' Contact Information: Jacques Carette, McMaster University, Hamilton, Canada, carette@mcmaster.ca; Chris Heunen, University of Edinburgh, Edinburgh, United Kingdom, Chris.Heunen@ed.ac.uk; Robin Kaarsgaard, University of Southern Denmark, Odense, Denmark, kaarsgaard@imada.sdu.dk; Amr Sabry, Indiana University, Bloomington, USA, sabry@iu.edu.

ACM 2475-1421/2024/8-ART236
https://doi.org/10.1145/3674625

Technically, in one copy of Π, every term is given a conventional interpretation where boolean negation is modeled by $\left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$ and controlled operations like the Toffoli gate are modeled using classical conditional expressions. In the other copy of Π, every term is interpreted as follows: rotate by $\pi/8$, apply the standard interpretation, and rotate back. This conjugation by rotations interprets boolean negation as $\frac{1}{\sqrt{2}}\left(\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right)$, which is the Hadamard gate. Since the combined language provides both the the Toffoli and Hadamard gates, it is computationally universal for quantum computing [3, 51]. Complementarity is quite a bit more subtle and requires more details of the construction beyond rotating by $\pi/8$ and is the topic of Section 7.

If our goal was merely to design some computationally universal quantum programming language, the recipe above would suffice. However, we insist that the resulting programming language is equipped with a sound theory for reasoning about program equivalences, which we achieve as follows. First, we do not fix an arbitrary rotation of $\pi/8$ but assume the rotation is given by some angle ϕ. We then impose equations to define two Frobenius structures [28, Chapter 5], and force them to be complementary [16][28, Chapter 6]). Thm. 7.3 then proves that the equations force ϕ to be chosen such that the two copies of Π combine to form a computationally universal language, each with well-established reasoning principles, and extended with the additional equations in Fig. 12 and Def. 7.1.

The mathematical formalism is expressed using free categorical constructions, and makes heavy use of Hughes' arrows [29, 31]. To show the recipe in action and explore its pragmatics in programming and reasoning about quantum circuits, we apply it to a canonical reversible programming language Π yielding the computationally universal quantum programming language QuantumΠ, and implement the entire project in Agda. [1]

Related work. Quantum programming languages [7, 10, 21, 43, 47, 48, 50, 53], classical reversible languages [12, 15, 32, 33, 56, 57], their categorical semantics [14, 20, 25, 26, 30, 34, 44, 46, 52, 54], complementarity of classical structures [16, 17], and amalgamation of categories [40] have all been individually studied before.

Existing quantum programming languages are really "circuit description languages", on which this article improves by exhibiting QuantumΠ's canonical status. The difference between QuantumΠ and languages like Quipper and Qunity is the language design itself: QuantumΠ by construction combines classical reversible languages, whereas the latter layer explicit quantum constructs on top of classical ones. Similarly, complementarity is central to the ZX- [16] and ZH-calculi [6, 19], though completeness needs more axioms. Those calculi concern general quantum theory (all complex matrices), whereas QuantumΠ concerns quantum computation (only unitary matrices). Hence QuantumΠ need not be able to emulate these calculi, and entirely circumvents the accompanying circuit synthesis problem. Our main contribution is an infrastructure organising these established ideas so that quantum behaviour emerges from classical programming languages using computational effects. The quest for such a computational 'quantum effect' also underlies [4], and this article improves on that very early work. We focus on reversible quantum computing, and in Sec. 8.3 add measurement in a modular way following Heunen and Kaarsgaard [25].

Outline. Fig. 1 summarises the technical development in two parallel threads: the design of a computationally universal quantum programming language on the left and the corresponding categorical models on the right. Sec. 2 starts with a review of relevant background on the categorical semantics of quantum computing, and Sec. 3 reviews the core classical reversible programming language Π and its semantics in rig groupoids. The first step of our construction in Sec. 4 is to take two copies of Π, called Π_Z and Π_ϕ, and embed their semantics, in two different ways, in the category **Unitary** of finite-dimensional Hilbert spaces and unitaries. These are then amalgamated

[1] Available from https://github.com/JacquesCarette/QuantumPi and artifact on Zenod.

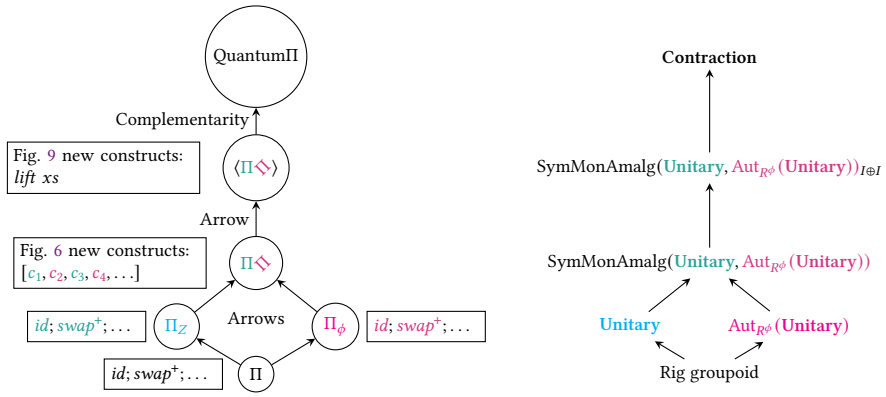


Fig. 1. Progression of languages and their categorical semantics.

in Sec. 5 to produce a language Π◇ where expressions from Π_Z and Π_ϕ can be freely interleaved. Π◇is then itself extended in Sec. 6 to the language ⟨Π◇⟩ which exposes the classical structures explicitly. Sec. 7 proves our main result, the canonicity theorem. Sec. 8 introduces QuantumΠ, the user-level interface of ⟨Π◇⟩. There we provide an Agda implementation of executable circuits *and* machine checkable proofs of various circuit equivalences. Our proofs show that some forms of reasoning about quantum programs in QuantumΠ indeed reduce to classical reasoning augmented with complementarity. It also shows that QuantumΠ can model gates with complex numbers and be extended with measurement to model complete quantum algorithms. A concluding section summarises results and discusses possible future directions.

2 Categories; Quantum Computing; Computational Universality

To fix notation and the background knowledge assumed, we briefly discuss the types of categories that are useful in reversible programming: dagger categories and rig categories. Then we will discuss quantum computing in categorical terms, complementarity, and computational universality. For the basics of category theory, we refer to Leinster [37].

2.1 Dagger Categories and Groupoids

A morphism $f : A \to B$ is *invertible*, or an *isomorphism*, when there exists a morphism $f^{-1} : B \to A$ such that $f^{-1} \circ f = \mathrm{id}_A$ and $f \circ f^{-1} = \mathrm{id}_B$. This inverse f^{-1} is necessarily unique. A category where every morphism is invertible is a *groupoid*.

At first sight, groupoids form the perfect semantics for reversible computing. But every step in a computation being reversible is slightly less restrictive than it being invertible. For each step $f : A \to B$, there must still be a way to 'undo' it, given by $f^\dagger : B \to A$. This should also still respect composition, in that $(g \circ f)^\dagger = f^\dagger \circ g^\dagger$ and $\mathrm{id}_A^\dagger = \mathrm{id}_A$. Moreover, a 'cancelled undo' should not change anything: $f^{\dagger\dagger} = f$. Therefore every morphism f has a partner $f^\dagger$. A category equipped with such a choice of partners is called a *dagger category*.

A groupoid is an example of a dagger category, where every morphism is *unitary*, that is, $f^\dagger = f^{-1}$. Think, for example, of the category **FinBij** with finite sets for objects and bijections for morphisms. But not every dagger category is a groupoid. For example, the dagger category **PInj** has sets as objects, and partial injections as morphisms. Here, the dagger satisfies $f \circ f^\dagger \circ f = f$, but not necessarily $f^\dagger \circ f = \mathrm{id}$ because f may only be partially defined. In a sense, the dagger category **PInj** is the universal model for reversible computation [24, 35]. When a category has a dagger, it

makes sense to demand that every other structure on the category respects the dagger, and we will do so. The theory of dagger categories is similar to the theory of categories in some ways, but very different in others [27].

2.2 Monoidal Categories and Rig Categories

Programming becomes easier when the programmer can express more programs natively. For example, it is handy to have type combinators like sums and products. Semantically, this is modelled by considering not mere categories, but monoidal ones. A *monoidal category* is a category equipped with a type combinator that turns two objects A and B into an object $A \otimes B$, and a term combinator that turns two morphisms $f : A \to B$ and $f' : A' \to B'$ into a morphism $f \otimes f' : A \otimes A' \to B \otimes B'$. This has to respect composition and identities. Moreover, there has to be an object I that acts as a unit for $\otimes$, and isomorphisms $\alpha : A \otimes (B \otimes C) \to (A \otimes B) \otimes C$ and $\lambda : I \otimes A \to A$ and $\rho : A \otimes I \to A$. In a *symmetric monoidal category*, there are additionally isomorphisms $\sigma : A \otimes B \to B \otimes A$. All these isomorphisms have to respect composition and satisfy certain coherence conditions, see [39] or [28, Chapter 1]. We speak of a *(symmetric) monoidal dagger category* when the coherence isomorphisms are unitary. Intuitively, $g \circ f$ models sequential composition, and $f \otimes g$ models parallel composition. For example, **FinBij** and **PInj** are symmetric monoidal dagger categories under cartesian product.

A *rig category* is monoidal in two ways in a distributive fashion. More precisely, it has two monoidal structures $\oplus$ and $\otimes$, such that $\oplus$ is symmetric monoidal but $\otimes$ not necessarily, and there are isomorphisms $\delta_L : A \otimes (B \oplus C) \to (A \otimes B) \oplus (A \otimes C)$ and $\delta_0 : A \otimes 0 \to 0$. These isomorphisms again have to respect composition and certain coherence conditions [36]. For example, **FinBij** and **PInj** are not only monoidal under cartesian product, but also under disjoint union, and the appropriate distributivity holds. Intuitively, $f \oplus g$ models a choice between f and g.

2.3 Quantum Computing (categorically)

Quantum computing with pure states is a specific kind of reversible computing. Good references are Nielsen and Chuang [41], Yanofsky and Mannucci [55]. A quantum system is modelled by a finite-dimensional Hilbert space A. The category giving semantics to finite-dimensional pure state quantum theory is therefore **FHilb**, whose objects are finite-dimensional Hilbert spaces, and whose morphisms are linear maps. Categorical semantics for pure state quantum computing is the groupoid **Unitary** of finite-dimensional Hilbert spaces as objects with unitaries as morphisms. Both are rig categories under direct sum $\oplus$ and tensor product $\otimes$.

The pure *states* of a quantum system modelled by a Hilbert space A are the vectors of unit norm, conventionally denoted by a *ket* $|y\rangle \in A$. These are equivalently given by morphisms $\mathbb{C} \to A$ in **FHilb** that map $z \in \mathbb{C}$ to $z\,|y\rangle \in A$. Dually, the functional $A \to \mathbb{C}$ maps $y \in A$ to the inner product $\langle x|y\rangle$ is conventionally written as a *bra* $\langle x|$. Morphisms $A \to \mathbb{C}$ are also called *effects*.

FHilb is a dagger rig category. The *dagger* of linear map $f : A \to B$ is uniquely determined via the inner product by $\langle f(x)|y\rangle = \langle x|f^\dagger(y)\rangle$. The dagger of a state is an effect, and vice versa. In quantum computing, pure states evolve along unitary gates. These are exactly the morphisms that are unitary in the dagger categories sense: $f^\dagger \circ f = \mathrm{id}$ and $f \circ f^\dagger = \mathrm{id}$, exhibiting the groupoid **Unitary** as a dagger subcategory of **FHilb**.

Once orthonormal bases $\{|i\rangle\}$ and $\{|j\rangle\}$ for finite-dimensional Hilbert spaces A and B are fixed, we can express morphisms $f : A \to B$ as a matrix with entries $\langle i|f|j\rangle$. The dagger then becomes the complex conjugate transpose, the tensor product becomes the Kronecker product of matrices, and the direct sum becomes a block diagonal matrix. The Hilbert spaces $\mathbb{C}^n$ come with the canonical *computational basis* consisting of the n vectors with a single entry 1 and otherwise 0, also called the *Z-basis* and denoted $\{|0\rangle, \ldots, |n-1\rangle\}$.

We can embed the category **FPInj** of finite sets and partial injections in to **FHilb**, that sends $\{0, \ldots, n-1\}$ to $\mathbb{C}^n$ preserving composition, identities, tensor product, direct sum, and dagger; we get a dagger rig functor $\ell^2 \colon \mathbf{FPInj} \to \mathbf{FHilb}$, that restricts to a dagger rig functor $\mathbf{FinBij} \to \mathbf{Unitary}$. Thus reversible computing (**FinBij**) is to classical reversible theory (**FPInj**) as quantum computing (**Unitary**) is to quantum theory (**FHilb**). In particular, the *Toffoli gate*, which is universal for reversible computing, transfers to a quantum gate that acts on vectors.

There are many such embeddings, one for every uniform choice of computational basis [24]. If we only care about computation with qubits (rather than qutrits or the more general qudits), we could also send a bijection f to $(H^{\otimes n})^\dagger \circ \ell^2(f) \circ H^{\otimes n}$, where H is the Hadamard matrix, to compute in the X basis rather than the computational (Z) basis.

2.4 Complementarity

A choice of basis $\{|i\rangle\}$ on an n-dimensional Hilbert space A defines a morphism $\delta \colon A \to A \otimes A$ in **FHilb** that maps $|i\rangle$ to $|ii\rangle = |i\rangle \otimes |i\rangle$. In fact, the morphisms arising this way are characterised by certain equational laws that make them into so-called *classical structures*, or *commutative special dagger Frobenius structures* [28, Chapter 5].

$$(\delta \otimes \mathrm{id}_A) \circ \delta = (\mathrm{id}_A \otimes \delta) \circ \delta \qquad \sigma_{A,A} \circ \delta = \delta \tag{1}$$

$$(\mathrm{id}_A \otimes \delta^\dagger) \circ (\delta \otimes \mathrm{id}_A) = (\delta^\dagger \otimes \mathrm{id}_A) \circ (\mathrm{id}_A \otimes \delta) \qquad \delta^\dagger \circ \delta = \mathrm{id}_A \tag{2}$$

As A is finite-dimensional, the basis vectors determine a state $\sum_{i=1}^n |i\rangle$ that is in *uniform superposition*. For the computational basis, this state is also denoted $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. Similarly, we shorthand $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. Now $\{|+\rangle, |-\rangle\}$ forms an orthogonal basis for $\mathbb{C}^2$, called the *X-basis*, different from the Z-basis.

As far as picking a basis to treat as 'the' computational basis is concerned, all bases are created equal. But once that arbitrary choice is fixed, some other bases are more equal than others. The Z-basis and the X-basis are *mutually unbiased*, meaning that a state of the one basis and an effect of the other basis always give the same inner product: $\langle 0|+\rangle = \langle 1|+\rangle = \langle 0|-\rangle = \langle 1|-\rangle$. That is, measuring in one basis a state prepared in the other gives no information at all. This can also be expressed by an equation between the associated Frobenius structures $\delta_1, \delta_2 \colon A \to A \otimes A$ (see [16] or [28, Chapter 6]):

$$(\delta_1^\dagger \otimes \mathrm{id}_A) \circ (\mathrm{id}_A \otimes \delta_2) \circ (\mathrm{id}_A \otimes \delta_2^\dagger) \circ (\delta_1 \otimes \mathrm{id}_A) = \mathrm{id}_{A \otimes A} \tag{3}$$

(To be precise, we adopt a simplified version using Heunen and Vicary [28, Prop. 6.7], and the fact that in finite dimension any injective morphism $A \to A$ is an isomorphism.)

Two complementary classical structures $A \to A \otimes A$ determine a unitary gate $A \to A$, corresponding to the linear map that turns the basis corresponding to one classical structure into the basis corresponding to the other. In the case of the Z and X bases, this is the Hadamard gate. Notice that the Hadamard gate is involutive: $H \circ H = \mathrm{id}$.

2.5 Computational Universality

The inner product of a Hilbert space A lets us measure how close two vectors $|x\rangle, |y\rangle \in A$ are by looking at the norm of their difference $\|x - y\|^2 = \langle x - y \mid x - y\rangle$. This leads to the dagger rig category **Contraction** of finite-dimensional Hilbert spaces and contractions: linear maps $f \colon A \to B$ satisfying $\|f(a)\| \leq \|a\|$ for all $a \in A$. In **Contraction**, the notion of state is relaxed from a vector of unit length to a vector of *at most* unit length (these are sometimes called *subnormalised states* or simply *substates*). This categorical model adds to pure state quantum computation the ability to

$$
\begin{array}{lr}
b ::= 0 \mid 1 \mid b + b \mid b \times b & \text{(value types)} \\
t ::= b \leftrightarrow b & \text{(combinator types)} \\
i ::= id \mid swap^{+} \mid assocr^{+} \mid assocl^{+} \mid unite^{+} l \mid uniti^{+} l & \text{(isomorphisms)} \\
\quad \mid swap^{\times} \mid assocr^{\times} \mid assocl^{\times} \mid unite^{\times} l \mid uniti^{\times} l & \\
\quad \mid dist \mid factor \mid absorbl \mid factorzr & \\
c ::= i \mid c \fatsemi c \mid c + c \mid c \times c \mid inv\ c & \text{(combinators)}
\end{array}
$$

Fig. 2. Π syntax

terminate without a useful outcome, where the norm $\|x\|$ of a state x signifies the probability of a nondegenerate outcome when measured; interpreting a state of norm 0 as complete failure,

A finite set of unitary gates $\{U_1, \ldots, U_k\}$ on qubits is *strictly universal* when for any unitary U and any $\varepsilon > 0$ there is a sequence of gates $U_{i_1} \circ \cdots \circ U_{i_p}$ with distance at most ε to U. This means that any computation whatsoever can be approximated by a circuit from the given set of gates up to arbitrary accuracy. The set is *computationally universal* when it can be used to simulate, possibly using ancillae and/or encoding, to accuracy within $\varepsilon > 0$ any quantum circuit on n qubits and t gates from a strictly universal set with only polylogarithmic overhead in n, t, and $\frac{1}{\varepsilon}$. This means that the gate set can perform general quantum computation without too much overhead.

Theorem 2.1. *[3, 51] The Toffoli and Hadamard gate set is computationally universal. In fact, Toffoli is computationally universal in conjunction with any real basis-changing single-qubit unitary gate.*

Notice that this theorem only needs sequential composition $\circ$ and parallel composition $\otimes$, and not sum types $\oplus$. Correspondingly, it only applies to Hilbert spaces of dimension 2^n.

3 The Classical Core: Π

Our eventual goal is to define a computationally universal quantum programming language from two copies of a classical reversible language. In this section, we review the syntax and semantics of Π [33], a language that is universal for reversible computing over finite types and whose semantics is expressed in the rig groupoid of finite sets and bijections.

3.1 Syntax and Types

In reversible boolean circuits, the number of input bits matches the number of output bits. Thus, a key insight for a programming language of reversible circuits is to ensure that each primitive operation preserves the number of bits, which is just a natural number. The algebraic structure of natural numbers as the free commutative semiring (or, commutative rig), with $(0, +)$ for addition, and $(1, \times)$ for multiplication then provides sequential, vertical, and horizontal circuit composition. Generalizing these ideas, a typed programming language for reversible computing should ensure that every primitive expresses an isomorphism of finite types, *i.e.*, a permutation. The syntax of the language Π in Fig. 2 captures this concept. Type expressions b are built from the empty type (0), the unit type (1), the sum type (+), and the product type ($\times$). A type isomorphism $c : b_1 \leftrightarrow b_2$ models a reversible circuit that permutes the values in b_1 and b_2. These type isomorphisms are built from the primitive identities and their compositions. These isomorphisms correspond exactly to the laws of a *rig* operationalised into invertible transformations [11, 12] which have the types in Fig. 3. Each line in the top part of the figure has the pattern $c_1 : b_1 \leftrightarrow b_2 : c_2$ where c_1 and c_2 are duals; c_1 has type $b_1 \leftrightarrow b_2$ and c_2 has type $b_2 \leftrightarrow b_1$.

$$
\begin{array}{rcrcll}
id & : & b & \leftrightarrow & b & : \ id \\
swap^{+} & : & b_1 + b_2 & \leftrightarrow & b_2 + b_1 & : \ swap^{+} \\
assocr^{+} & : & (b_1 + b_2) + b_3 & \leftrightarrow & b_1 + (b_2 + b_3) & : \ assocl^{+} \\
unite^{+}l & : & 0 + b & \leftrightarrow & b & : \ uniti^{+}l \\
swap^{\times} & : & b_1 \times b_2 & \leftrightarrow & b_2 \times b_1 & : \ swap^{\times} \\
assocr^{\times} & : & (b_1 \times b_2) \times b_3 & \leftrightarrow & b_1 \times (b_2 \times b_3) & : \ assocl^{\times} \\
unite^{\times}l & : & 1 \times b & \leftrightarrow & b & : \ uniti^{\times}l \\
dist & : & (b_1 + b_2) \times b_3 & \leftrightarrow & (b_1 \times b_3) + (b_2 \times b_3) & : \ factor \\
absorbl & : & b \times 0 & \leftrightarrow & 0 & : \ factorzr
\end{array}
$$

$$
\frac{c_1 : b_1 \leftrightarrow b_2 \quad c_2 : b_2 \leftrightarrow b_3}{c_1 \fatsemi c_2 : b_1 \leftrightarrow b_3}
\qquad
\frac{c : b_1 \leftrightarrow b_2}{inv\ c : b_2 \leftrightarrow b_1}
$$

$$
\frac{c_1 : b_1 \leftrightarrow b_3 \quad c_2 : b_2 \leftrightarrow b_4}{c_1 + c_2 : b_1 + b_2 \leftrightarrow b_3 + b_4}
\qquad
\frac{c_1 : b_1 \leftrightarrow b_3 \quad c_2 : b_2 \leftrightarrow b_4}{c_1 \times c_2 : b_1 \times b_2 \leftrightarrow b_3 \times b_4}
$$

Fig. 3. Types for Π combinators

$$
\begin{aligned}
\textsc{ctrl}\ c &= dist \fatsemi id + (id \times c) \fatsemi factor : (1 + 1) \times x \to (1 + 1) \times x \\
\textsc{x} &= swap^{+} : 1 + 1 \to 1 + 1 \\
\textsc{cx} &= \textsc{ctrl}\ swap^{+} : (1 + 1) \times (1 + 1) \to (1 + 1) \times (1 + 1) \\
\textsc{ccx} &= \textsc{ctrl}\ \textsc{cx} : (1 + 1) \times ((1 + 1) \times (1 + 1)) \to (1 + 1) \times ((1 + 1) \times (1 + 1))
\end{aligned}
$$

Fig. 4. Derived Π constructs.

To see how to express reversible circuits, we first define n-bit words, i.e. $\mathbb{2}^n$. We define $\mathbb{2}$ as the type $1 + 1$, with the left injection representing false and the right injection representing true. Boolean negation (the X-gate) is then the primitive combinator $swap^{+}$. Then n-bit words are an n-ary product of values of $\mathbb{2}$. To express the CX- and CCX-gates we need to encode a notion of conditional expression. Such conditionals turn out to be expressible using the distributivity and factoring identities of rigs as shown in Fig. 4. An input value of type $\mathbb{2} \times b$ is processed by the *dist* operator, which converts it into a value of type $(1 \times b) + (1 \times b)$. Only in the right branch, which corresponds to the case when the boolean is true, is the combinator c applied to the value of type b. The inverse of *dist*, namely *factor* is applied to get the final result. Using this conditional, CX is defined as CTRL X and the Toffoli CCX is defined as CTRL CX.

Theorem 3.1 (Π Expressivity). *Π is universal for classical reversible circuits, i.e., boolean bijections $\mathbb{2}^n \to \mathbb{2}^n$ (for any natural number n).*

3.2 Semantics

By design, Π has a natural model in *rig groupoids* [12, 15]. Indeed, every atomic isomorphism of Π corresponds to a coherence isomorphism in a rig category, while sequencing corresponds to composition, and the two parallel compositions are handled by the two monoidal structures. Inversion corresponds to the canonical dagger structure of groupoids. This interpretation is summarised in Fig. 5. The denotational semantics directly suggests a big-step operational semantics where each combinator $c : b_1 \leftrightarrow b_2$ maps to a (bijective) function $[\![b_1]\!] \to [\![b_2]\!]$ [15, 33]. With a

Types

$$\begin{array}{rcl@{\qquad}rcl} [\![0]\!] & = & O & [\![1]\!] & = & I \\ [\![b_1 + b_2]\!] & = & [\![b_1]\!] \oplus [\![b_2]\!] & [\![b_1 \times b_2]\!] & = & [\![b_1]\!] \otimes [\![b_2]\!] \end{array}$$

Terms

$$\begin{array}{rcl@{\qquad}rcl@{\qquad}rcl} [\![id]\!] & = & \mathrm{id} & [\![inv\ c]\!] & = & [\![c]\!]^\dagger & [\![c_1 \fatsemi c_2]\!] & = & [\![c_2]\!] \circ [\![c_1]\!] \\ [\![assocr^+]\!] & = & \alpha_\oplus & [\![assocl^+]\!] & = & \alpha_\oplus^{-1} & [\![swap^+]\!] & = & \sigma_\oplus \\ [\![uniti^+ l]\!] & = & \lambda_\oplus^{-1} & [\![unite^+ l]\!] & = & \lambda_\oplus & & & \\ [\![assocr^\times]\!] & = & \alpha_\otimes & [\![assocl^\times]\!] & = & \alpha_\otimes^{-1} & [\![swap^\times]\!] & = & \sigma_\otimes \\ [\![uniti^\times l]\!] & = & \lambda_\otimes^{-1} & [\![unite^\times l]\!] & = & \lambda_\otimes & & & \\ [\![dist]\!] & = & \delta_R & [\![factor]\!] & = & \delta_R^{-1} & [\![c_1 + c_2]\!] & = & [\![c_1]\!] \oplus [\![c_2]\!] \\ [\![absorbl]\!] & = & \delta_0 & [\![factorzr]\!] & = & \delta_0^{-1} & [\![c_1 \times c_2]\!] & = & [\![c_1]\!] \otimes [\![c_2]\!] \end{array}$$

Fig. 5. The semantics of Π in rig groupoids with monoidal structures $(O, \oplus)$ and $(I, \otimes)$.

little more effort, it is possible to derive an equivalent presentation using a small-step abstract machine [13].

4 Models of Π from Automorphisms

When Π is used as a stand-alone classical language, **FinBij** is the canonical choice for the semantics. As we aim to recover quantum computation from two copies of Π, we need more structure. We begin by explaining the categorical construction needed to embed a rig groupoid in the category $\mathrm{Aut}_a(\mathbf{Unitary})$ parameterised by a family of automorphisms a. We then use this construction to give two models for Π embedded in $\mathrm{Aut}_a(\mathbf{Unitary})$ (for different a).

4.1 $\mathrm{Aut}_a(\mathbf{Unitary})$

We generalize the category **Unitary** to a family of categories parameterised by automorphisms that are pre- and post-composed with every morphism.

Definition 4.1. Let $\mathbf{C}$ be a category, and for each object C let $a_C : C \to C$ be an automorphism (that is not necessarily natural in any sense). Form a new category $\mathrm{Aut}_a(\mathbf{C})$ with:

- **Objects:** objects of $\mathbf{C}$.
- **Morphisms:** morphisms are those of the form $a_B^{-1} \circ f \circ a_A$ for every $f : A \to B$ of $\mathbf{C}$.
- **Composition:** as in $\mathbf{C}$.

We note that $\mathbf{C}$ and $\mathrm{Aut}_a(\mathbf{C})$ are equivalent as categories, but **not** as rig categories – their additive monoidal structure differ.

Proposition 4.2. *When* $\mathbf{C}$ *is a rig groupoid, so is* $\mathrm{Aut}_a(\mathbf{C})$.

Proof. To see that $\mathrm{Aut}_a(\mathbf{C})$ is a category, observe that, since $\mathrm{Aut}_a(\mathbf{C})$ inherits composition from $\mathbf{C}$, identities are those from $\mathbf{C}$ since $a_A^{-1} \circ \mathrm{id}_A \circ a_A = a_A^{-1} \circ a_A = \mathrm{id}_A$, and composition is conjugated composition of morphisms from $\mathbf{C}$ since $a_C^{-1} \circ g \circ a_B \circ a_B^{-1} \circ f \circ a_A = a_C^{-1} \circ g \circ f \circ a_A$. Associativity and unitality of composition in $\mathrm{Aut}_a(\mathbf{C})$ follow directly. That $\mathrm{Aut}_a(\mathbf{C})$ is a groupoid when $\mathbf{C}$ is follows since for every isomorphism f:

$$a_B^{-1} \circ f \circ a_A \circ a_A^{-1} \circ f^{-1} \circ a_B = a_B^{-1} \circ f \circ f^{-1} \circ a_B = a_B^{-1} \circ a_B = \mathrm{id}_B$$

and analogously $a_A^{-1} \circ f^{-1} \circ a_B \circ a_B^{-1} \circ f \circ a_A = \mathrm{id}_A$.

Supposing now **C** is symmetric monoidal, define a symmetric monoidal structure on objects as in **C** (with unit I as in **C**), and on morphisms $a_B^{-1} \circ f \circ a_A$ and $a_B'^{-1} \circ f' \circ a_A'$ by

$$a_{B\otimes B'}^{-1} \circ (a_B \circ (a_B^{-1} \circ f \circ a_A) \circ a_A^{-1}) \otimes (a_B' \circ (a_B'^{-1} \circ f' \circ a_A') \circ a_A'^{-1}) \circ a_{A\otimes A'}$$

in **C**, which simplifies to $a_{B\otimes B'}^{-1} \circ (f \otimes f') \circ a_{A\otimes A'}$. In other words, monoidal products of morphisms in $\mathrm{Aut}_a(\mathbf{C})$ are merely monoidal products of morphisms from **C** conjugated by the appropriate automorphisms. Coherence isomorphisms are those from **C**, but conjugated by the appropriate automorphisms.

The rest of the proof (bifunctoriality, naturality of coherence isomorphisms, coherence conditions and rig structure) proceed in much the same way. The details are omitted for length, but are found in the extended version. □

4.2 Models of Π_Z and Π_ϕ: Unitary and Aut_{R^ϕ}(Unitary)

The choice of semantics for Π_Z is easy to justify: it will use the family of identity automorphisms, i.e, it will use the canonical category **Unitary** itself. The semantics for Π_ϕ will be "rotated" by some angle with respect to that of Π_Z. By that, we mean that the semantics of Π_ϕ will use a family of automorphisms that is parameterised by a rotation matrix $\mathbf{r}^\phi = \left(\begin{smallmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{smallmatrix}\right)$ for some yet-to-be determined angle ϕ.

Definition 4.3. The canonical model of Π_Z is the rig groupoid **Unitary** of finite-dimensional Hilbert spaces and unitaries.

We recall that as a rig category, **Unitary** is *semi-simple* in the sense that all of its objects are generated by the rig structure (this is a direct consequence of the fact that each finite-dimensional Hilbert space is isometrically isomorphic to $\mathbb{C}^n$ for some n [38]). In other words, every object in **Unitary** can be written (up to isomorphism) using the two units O and I as well as the monoidal product $\otimes$ and sum $\oplus$. We will use this fact to define a family of automorphisms R_A^ϕ in **Unitary** which will be used to form a model of Π_ϕ.

Definition 4.4. Given an angle ϕ, we define a family R_A^ϕ of automorphisms in **Unitary** as follows:

$$\begin{array}{rclrcl} R_O^\phi & = & \mathrm{id}_O & R_I^\phi & = & \mathrm{id}_I \\ R_{A\otimes B}^\phi & = & R_A^\phi \otimes R_B^\phi & & & \\ R_{A\oplus B}^\phi & = & (\theta_A^{-1} \oplus \theta_B^{-1}) \circ \mathbf{r}^\phi \circ (\theta_A \oplus \theta_B) & \multicolumn{3}{l}{(\text{when } A \simeq I \simeq B)} \\ R_{A\oplus B}^\phi & = & R_A^\phi \oplus R_B^\phi & \multicolumn{3}{l}{(\text{when } A \not\simeq I \text{ or } B \not\simeq I)} \end{array}$$

The morphisms θ_A and θ_B in this definition refer to the isomorphisms witnessing $A \simeq I$ and $B \simeq I$ respectively. In particular, this definition requires one to decide isomorphism with I. This sounds potentially difficult, but is fortunately very simple: an object is isomorphic to I iff it can be turned into I by eliminating additive units O and multiplicative units I using the unitors $\lambda_\oplus$, $\rho_\oplus$, $\lambda_\otimes$, and $\rho_\otimes$ as well as the associators $\alpha_\oplus$ and $\alpha_\otimes$ as necessary.

Essentially, Aut_{R^ϕ}(**Unitary**) consists of unitaries in which qubit (sub)systems are conjugated by the unitary $\mathbf{r}^\phi$. This is significant, because it means that the additive symmetry $\sigma_\oplus$ on $I \oplus I$ is no longer $\left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$ as usual, but instead the potentially much more interesting gate:

$$\begin{pmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix} = \begin{pmatrix} \sin 2\phi & \cos 2\phi \\ \cos 2\phi & -\sin 2\phi \end{pmatrix} .$$

This leads us to the family of models of Π_ϕ.

Definition 4.5. Given a value for ϕ, a model of Π_ϕ is the rig groupoid $\mathrm{Aut}_{R^\phi}(\mathbf{Unitary})$ of finite-dimensional Hilbert spaces and unitaries of the form $(R^\phi_B)^{-1} \circ U \circ R^\phi_A$.

5 $\Pi\diamondsuit$ from Amalgamation

The aim of this section is to define the language $\Pi\diamondsuit$ that combines the separate definitions of Π_Z and Π_ϕ into a combined language that interleaves expressions from each. We begin by explaining the amalgamations of categories in Secs. 5.1, 5.2, and 5.3. We use these constructions to define categorical models of $\Pi\diamondsuit$ in Sec. 5.4. These models justify the definition of $\Pi\diamondsuit$ as an arrow over the individual sublanguages as shown in Sec. 5.5.

5.1 Amalgamation of Categories

Programs in $\Pi\diamondsuit$ are formal compositions of Π_Z programs and Π_ϕ programs that respects product types. To account for this semantically, we need to combine *models* of Π_Z and Π_ϕ in a way that preserves the monoidal product. This construction is known as the *amalgamation of categories* (see, *e.g.*, MacDonald and Scull [40]). We now recall this construction in the slightly simpler case where the two categories have the same objects, and go on to extend it to the symmetric monoidal case.

Definition 5.1. Given two categories $\mathbf{C}$ and $\mathbf{D}$ with the same objects, form a new category $\mathrm{Amalg}(\mathbf{C},\mathbf{D})$ as follows:

- **Objects**: Objects of $\mathbf{C}$ (equivalently $\mathbf{D}$).
- **Morphisms:** Morphisms $A_1 \longrightarrow A_{n+1}$ are equivalence classes of finite lists $[f_n,\ldots,f_1]$ of morphisms $f_i : A_i \longrightarrow A_{i+1}$ of $\mathbf{C}$ or $\mathbf{D}$ tagged with their category of origin, under the equivalence $\sim$ below.
- **Identities:** Empty lists $[]$.
- **Composition:** Concatenation of lists, $[g_n,\ldots,g_1] \circ [f_m,\ldots,f_1] = [g_n,\ldots,g_1,f_m,\ldots,f_1]$.

When the origin category is important we will write, *e.g.*, $f^{\mathbf{C}}$ to mean that f is tagged with $\mathbf{C}$ and so originated from this category. Let $\sim$ denote the least equivalence satisfying

$$[\mathrm{id}] \sim [] \tag{4}$$

$$[f^{\mathbf{A}}, g^{\mathbf{A}}] \sim [f^{\mathbf{A}} \circ g^{\mathbf{A}}] \tag{5}$$

as well as the congruence:

$$\frac{[f_n,\ldots,f_1] \sim [f'_{n'},\ldots,f'_1] \qquad [g_m,\ldots,g_1] \sim [g'_{m'},\ldots,g'_1]}{[f_n,\ldots,f_1] \circ [g_m,\ldots,g_1] \sim [f'_{n'},\ldots,f'_1] \circ [g'_{m'},\ldots,g'_1]} \tag{6}$$

Note that the inner composition in (5) refers to composition in the category $\mathbf{A}$, which in turn refers to either $\mathbf{C}$ or $\mathbf{D}$. To verify that this forms a category, we notice that concatenation of lists is associative and has the empty list as unit; however, since these are not lists per se but equivalence classes of lists, we must check that composition is well-defined. To see this, consider the normalisation procedure that repeats the following two steps until a fixed point is reached:

(1) Remove all identities using (4) and (6).
(2) Compose all composable adjacent morphisms using (5) and (6).

That a fixed point is always reached follows by the fact that both of these steps are monotonically decreasing in the length of the list, which is always finite.

As we would hope, there are straightforward embeddings $\mathbf{C} \longrightarrow \mathrm{Amalg}(\mathbf{C},\mathbf{D}) \longleftarrow \mathbf{D}$.

Proposition 5.2. *There are embeddings* $\mathscr{E}_L : \mathbf{C} \longrightarrow \mathrm{Amalg}(\mathbf{C},\mathbf{D})$ *and* $\mathscr{E}_R : \mathbf{D} \longrightarrow \mathrm{Amalg}(\mathbf{C},\mathbf{D})$ *given on objects by* $X \mapsto X$ *and on morphisms by* $f \mapsto [f]$.

Proof. $\mathscr{E}_L(\mathrm{id}) = [\mathrm{id}] \sim []$ and $\mathscr{E}_L(g \circ f) = [g \circ f] \sim [g,f] = \mathscr{E}_L(g) \circ \mathscr{E}_L(f)$, likewise for $\mathscr{E}_R$. □

Since Π_Z and Π_ϕ are both reversible, we would expect Π◇ to be so, by taking inverses pointwise. We show that the amalgamation of *groupoids* is, again, a groupoid.

Proposition 5.3. $\mathrm{Amalg}(\mathbf{C}, \mathbf{D})$ *is a groupoid when* $\mathbf{C}$ *and* $\mathbf{D}$ *are.*

Proof. Define $[f_n, \ldots, f_1]^{-1} = [f_1^{-1}, \ldots, f_n^{-1}]$ (where f_i^{-1} is the inverse to f_i in the origin category), and proceed by induction on n. When $n = 0$, $[] \circ []^{-1} = [] \circ [] = []$. Assuming the inductive hypothesis on all lists of length n we see on lists of length $n+1$ that

$$\begin{aligned}
&[f_{n+1}, \ldots, f_1] \circ [f_{n+1}, \ldots, f_1]^{-1} = [f_{n+1}, \ldots, f_1, f_1^{-1}, \ldots, f_{n+1}^{-1}] \sim [f_{n+1}, \ldots, f_1 \circ f_1^{-1}, \ldots, f_{n+1}^{-1}] \\
&= [f_{n+1}, \ldots, f_2, \mathrm{id}, f_2^{-1}, \ldots, f_{n+1}^{-1}] \sim [f_{n+1}, \ldots, f_2, f_2^{-1}, \ldots, f_{n+1}^{-1}] \\
&= [f_{n+1}, \ldots, f_2] \circ [f_2^{-1}, \ldots, f_{n+1}^{-1}] = [f_{n+1}, \ldots, f_2] \circ [f_{n+1}, \ldots, f_2]^{-1} = []
\end{aligned}$$

where the last identity follows by the inductive hypothesis. □

5.2 Amalgamation of Symmetric Monoidal Categories

Categorically, the amalgamation of categories (with the same objects) has a universal property as a pushout of (identity-on-objects) embeddings in the category **Cat** of (small) categories and functors between them [40]. While a good first step towards a model of Π◇, it is not enough. This is because it is only a pushout of mere functors between unstructured categories, so it will not necessarily respect *structure* present in the categories being amalgamated, such as symmetric monoidal structure. Thus we extend the amalgamation of categories to one for symmetric monoidal categories, and later that this yields an arrow over the symmetric monoidal categories involved.

Definition 5.4. Given two symmetric monoidal categories $\mathbf{C}$ and $\mathbf{D}$ with the same objects, such that their symmetric monoidal products agree on objects (specifically, their units are the same), form a new category $\mathrm{SymMonAmalg}(\mathbf{C}, \mathbf{D})$ as follows:

- **Objects**, **Morphisms**, **Identities**, and **Composition** as in $\mathrm{Amalg}(\mathbf{C}, \mathbf{D})$ (Def. 5.1).
- **Monoidal unit:** I, the monoidal unit of $\mathbf{C}$ and $\mathbf{D}$.
- **Monoidal product:** On objects, define $A \otimes B$ to be as in $\mathbf{C}$ and $\mathbf{D}$. On morphisms, define $[f_n, \ldots, f_1] \otimes [g_m, \ldots, g_1] = [f_n \otimes \mathrm{id}, \ldots, f_1 \otimes \mathrm{id}, \mathrm{id} \otimes g_m, \ldots, \mathrm{id} \otimes g_1]$ where $f_i \otimes \mathrm{id}$ and $\mathrm{id} \otimes g_j$ are formed in the origin category of f_i and g_j respectively, up to the extended equivalence below.
- **Coherence isomorphisms:** The coherence isomorphisms α, σ, λ, ρ, and their inverses are given by equivalence classes of lifted coherence isomorphisms from $\mathbf{C}$ and $\mathbf{D}$ (*e.g.*, $[\alpha^{\mathbf{C}}]$) up to the extended equivalence below.

The extended equivalence is the least one containing (4), (5), and (6) from Def. 5.1 as well as

$$[f \otimes \mathrm{id}, \mathrm{id} \otimes g] \sim [\mathrm{id} \otimes g, f \otimes \mathrm{id}] \tag{7}$$

$$[\alpha^{\mathbf{C}}] \sim [\alpha^{\mathbf{D}}] \qquad [\sigma^{\mathbf{C}}] \sim [\sigma^{\mathbf{D}}] \qquad [\lambda^{\mathbf{C}}] \sim [\lambda^{\mathbf{D}}] \qquad [\rho^{\mathbf{C}}] \sim [\rho^{\mathbf{D}}] \tag{8}$$

in addition to the congruence:

$$\frac{[f_n, \ldots, f_1] \sim [f'_{n'}, \ldots, f'_1] \qquad [g_m, \ldots, g_1] \sim [g'_{m'}, \ldots, g'_1]}{[f_n, \ldots, f_1] \otimes [g_m, \ldots, g_1] \sim [f'_{n'}, \ldots, f'_1] \otimes [g'_{m'}, \ldots, g'_1]} \tag{9}$$

Note that (7) above holds even when f and g originate from different categories, such that this is not simply a consequence of (5) and bifunctoriality in the origin category.

It follows that this defines a category, but we also need:

Proposition 5.5. $\mathrm{SymMonAmalg}(\mathbf{C}, \mathbf{D})$ *is symmetric monoidal.*

For reasons of space, we omit this proof, which contains no ideas not already seen in previous proofs. An extended version of this paper has all the details.

As with the amalgamation of mere categories, one can show that this extends to a pushout of monoidal embeddings. Further, the embeddings we presented earlier into the amalgamation of mere categories extends to well-behaved ones in the symmetric monoidal case well:

Proposition 5.6. *There are strict monoidal embeddings* $\mathscr{E}_L : \mathbf{C} \to \mathrm{SymMonAmalg}(\mathbf{C},\mathbf{D})$ *and* $\mathscr{E}_R : \mathbf{D} \to \mathrm{SymMonAmalg}(\mathbf{C},\mathbf{D})$ *given by* $X \mapsto X$ *on objects and* $f \mapsto [f]$ *on morphisms.*

Proof. We show the case for $\mathscr{E}_L$, as $\mathscr{E}_R$ is entirely analogous. $\mathscr{E}_L$ was shown to be functorial in Prop. 5.2, so suffices to show that it preserves coherence isomorphisms and the monoidal product exactly on objects and morphisms. On objects $\mathscr{E}_L(A \otimes B) = A \otimes B = \mathscr{E}_L(A) \otimes \mathscr{E}_L(B)$. On morphisms $\mathscr{E}_L(f \otimes g) = [f \otimes g] = [f \otimes \mathrm{id} \circ \mathrm{id} \otimes g] \sim [f \otimes \mathrm{id}, \mathrm{id} \otimes g] = [f] \otimes [g] = \mathscr{E}_L(f) \otimes \mathscr{E}_L(g)$. Finally, on coherence isomorphisms β, $\mathscr{E}_L(\beta) = [\beta^{\mathbf{C}}] \sim [\beta^{\mathbf{D}}] = \mathscr{E}_R(\beta)$, as desired. □

From the construction of inverses in Prop. 5.3, it follows that amalgamation preserves symmetric monoidal *groupoids* as well:

Corollary 5.7. $\mathrm{SymMonAmalg}(\mathbf{C},\mathbf{D})$ *is a symmetric monoidal groupoid when* $\mathbf{C}$ *and* $\mathbf{D}$ *are.*

Lastly we show that whenever a functor *out of* a symmetric monoidal amalgamation is needed, it is sufficient to consider functors out of each of the underlying categories. The lemma will be used to prove the existence of a computationally universal model of $\Pi\diamondsuit$ in Thm. 5.11.

Lemma 5.8. *Let* $\mathbf{C}$ *and* $\mathbf{D}$ *be symmetric monoidal categories with the same objects such that their monoidal structures agree. For any other symmetric monoidal category* $\mathbf{E}$*, to give a strict monoidal identity-on-objects functor* $\mathrm{SymMonAmalg}(\mathbf{C},\mathbf{D}) \to \mathbf{E}$ *is to give strict monoidal identity-on-objects functors* $\mathbf{C} \to \mathbf{E}$ *and* $\mathbf{D} \to \mathbf{E}$*.*

Proof. Given a strict monoidal identity-on-objects functor $F : \mathrm{SymMonAmalg}(\mathbf{C},\mathbf{D}) \to \mathbf{E}$, we compose with the (strict monoidal identity-on-objects) functors $\mathscr{E}_L$ and $\mathscr{E}_R$ to obtain the required functors $F \circ \mathscr{E}_L : \mathbf{C} \to \mathbf{E}$ and $F \circ \mathscr{E}_R : \mathbf{D} \to \mathbf{E}$.

In the other direction, given strict monoidal identity-on-objects functors $G : \mathbf{C} \to \mathbf{E}$ and $H : \mathbf{D} \to \mathbf{E}$, we define a functor $F_{G,H} : \mathrm{SymMonAmalg}(\mathbf{C},\mathbf{D}) \to \mathbf{E}$ on objects by $F_{G,H}(A) = G(A) = H(A) = A$. Given some morphism $[f_n,\ldots,f_1]$, assume without loss of generality that each f_i originates in $\mathbf{C}$ for all even i, and in $\mathbf{D}$ for all odd n. We define $F_{G,H}([\,]) = \mathrm{id}$ and

$$F_{G,H}([f_n,\ldots,f_1]) = G(f_n) \circ H(f_{n-1}) \circ \cdots \circ H(f_1)$$

This is immediately functorial. To see that it is strict monoidal, $F_{G,H}(A \otimes B) = F_{G,H}(A) \otimes F_{G,H}(B)$ follows trivially, while

$$\begin{aligned}
F_{G,H}([f_n,\ldots,f_1] \otimes [g_m,\ldots,g_1]) &= F_{G,H}([f_n \otimes \mathrm{id},\ldots,f_1 \otimes \mathrm{id}, \mathrm{id} \otimes g_m,\ldots,\mathrm{id} \otimes g_1]) \\
&= G(f_n \otimes \mathrm{id}) \circ \cdots \circ H(f_1 \otimes \mathrm{id}) \circ G(\mathrm{id} \otimes g_m) \circ \cdots \circ H(\mathrm{id} \otimes g_1) \\
&= G(f_n) \otimes \mathrm{id} \circ \cdots \circ H(f_1) \otimes \mathrm{id} \circ \mathrm{id} \otimes G(g_m) \circ \cdots \circ \mathrm{id} \otimes H(g_1) \\
&= (G(f_n) \circ \cdots \circ H(f_1)) \otimes \mathrm{id} \circ \mathrm{id} \otimes (G(g_m) \circ \cdots \circ H(g_1)) \\
&= (G(f_n) \circ \cdots \circ H(f_1)) \otimes (G(g_m) \circ \cdots \circ H(g_1)) \\
&= F_{G,H}([f_n,\ldots,f_1]) \otimes F_{G,H}([g_m,\ldots,g_1])
\end{aligned}$$

That this preserves coherence isomorphisms such as the associator $[\alpha^{\mathbf{C}}] \sim [\alpha^{\mathbf{D}}]$ follows by $G(\alpha) = F_{G,H}([\alpha^{\mathbf{C}}]) = F_{G,H}([\alpha^{\mathbf{D}}]) = H(\alpha) = \alpha$, and similarly for the unitors λ, ρ and symmetry σ. Further, $F_{G,H}$ is clearly uniquely determined by G and H, i.e., $F_{G,H} \circ \mathscr{E}_L = G$ and $F_{G,H} \circ \mathscr{E}_R = H$. □

5.3 The Amalgamation Arrow

Semantically, arrows correspond to (identity-on-objects) strict *premonoidal* functors between *premonoidal* categories [31, 45], a special case of these being the more well-behaved (identity-on-objects) strict *monoidal* functors between *monoidal* categories. In this way, the strict monoidal functors $\mathbf{Unitary} \to \mathrm{SymMonAmalg}(\mathbf{Unitary}, \mathrm{Aut}_{R^\phi}(\mathbf{Unitary})) \leftarrow \mathrm{Aut}_{R^\phi}(\mathbf{Unitary})$ of Prop. 5.6 provide a semantics for the arrow combinators.

Proposition 5.9. *The strict monoidal functors $\mathscr{E}_L$ and $\mathscr{E}_R$ are arrows over the categories* **Unitary** *and* $\mathrm{Aut}_{R^\phi}(\mathbf{Unitary})$.

5.4 Model of Π◇: $\mathrm{SymMonAmalg}(\mathbf{Unitary}, \mathrm{Aut}_{R^\phi}(\mathbf{Unitary}))$

Given models of Π_Z (Def. 4.3) and Π_ϕ (Def. 4.5), using Def. 5.4 we can give a model of Π◇:

Definition 5.10. Given a value for ϕ, a model of Π◇ is the symmetric monoidal groupoid

$$\mathrm{SymMonAmalg}(\mathbf{Unitary}, \mathrm{Aut}_{R^\phi}(\mathbf{Unitary}))$$

with **Unitary** and $\mathrm{Aut}_{R^\phi}(\mathbf{Unitary})$ considered as symmetric monoidal groupoids equipped with their monoidal products $(\otimes, I)$.

In other words, $\mathrm{SymMonAmalg}(\mathbf{Unitary}, \mathrm{Aut}_{R^\phi}(\mathbf{Unitary}))$ identifies the monoidal products $(\otimes, I)$ in **Unitary** and $\mathrm{Aut}_{R^\phi}(\mathbf{Unitary})$, but leaves their respective monoidal sums $(\oplus, O)$ alone. This may seem like a very curious choice—perhaps even a wrong one!—but is done for very deliberate reasons, which we describe here.

First, identifying the two monoidal products is entirely reasonable, since $R^\phi_{A\otimes B} = R^\phi_A \otimes R^\phi_B$, so the monoidal product on morphisms in $\mathrm{Aut}_{R^\phi}(\mathbf{Unitary})$ is really

$$\begin{aligned}(R^\phi_{A'\otimes B'})^{-1} \circ (f \otimes g) \circ R^\phi_{A\otimes B} &= (R^\phi_{A'})^{-1} \otimes (R^\phi_{B'})^{-1} \circ (f \otimes g) \circ R^\phi_A \otimes R^\phi_B \\ &= ((R^\phi_{A'})^{-1} \circ f \circ R^\phi_A) \otimes ((R^\phi_{B'})^{-1} \circ g \otimes R^\phi_B)\end{aligned}$$

i.e., the monoidal product in **Unitary** of morphisms from $\mathrm{Aut}_{R^\phi}(\mathbf{Unitary})$ (on objects, the two monoidal products agree on the nose). From this it also follows that the coherence isomorphisms for the monoidal product (*i.e.*, the associator $\alpha_\otimes$, unitors $\lambda_\otimes$ and $\rho_\otimes$, and symmetry $\sigma_\otimes$) in $\mathrm{Aut}_{R^\phi}(\mathbf{Unitary})$ all coincide with those in **Unitary** by naturality, since, *e.g.*

$$\begin{aligned}(R^\phi_{A\otimes(B\otimes C)})^{-1} \circ \alpha \circ R^\phi_{(A\otimes B)\otimes C} &= (R^\phi_A)^{-1} \otimes ((R^\phi_B)^{-1} \otimes (R^\phi_C)^{-1}) \circ \alpha \circ (R^\phi_A \otimes R^\phi_B) \otimes R^\phi_C \\ &= (R^\phi_A)^{-1} \otimes ((R^\phi_B)^{-1} \otimes (R^\phi_C)^{-1}) \circ R^\phi_A \otimes (R^\phi_B \otimes R^\phi_C) \circ \alpha \\ &= \alpha\end{aligned}$$

and likewise for the unitors and symmetry. Thus all of the morphisms identified by the amalgamation $\mathrm{SymMonAmalg}(\mathbf{Unitary}, \mathrm{Aut}_{R^\phi}(\mathbf{Unitary}))$ are ones which were equal to begin with.

Second, one may wonder why we do not go further and identify the monoidal *sums* in **Unitary** and $\mathrm{Aut}_{R^\phi}(\mathbf{Unitary})$ as well. In short, this is because it would confine Π◇ to being a classical language! We saw in Sec. 4.2 that the symmetry of the monoidal sum $\sigma_\oplus$ in $\mathrm{Aut}_{R^\phi}(\mathbf{Unitary})$ was $\left(\begin{smallmatrix}\sin 2\phi & \cos 2\phi \\ \cos 2\phi & -\sin 2\phi\end{smallmatrix}\right)$ whereas in **Unitary** it is the usual swap $\left(\begin{smallmatrix}0 & 1 \\ 1 & 0\end{smallmatrix}\right)$, and, indeed, the fact that we have both of these is central to our approach. However, identifying the monoidal sums would force us to identify these as well, destroying any hope of Π◇ being more expressive than Π_Z or Π_ϕ on their own. One cannot even hope to identify the monoidal sums as mere monoidal structures (as opposed to as *symmetric* monoidal structures), since the bifunctoriality clause of the equivalence (*i.e.*, clause (7) of Def. 5.4) fails for **Unitary** and $\mathrm{Aut}_{R^\phi}(\mathbf{Unitary})$ on $I \oplus I$ in all nontrivial cases.

$$b ::= 0 \mid 1 \mid b + b \mid b \times b \qquad \text{(value types)}$$
$$t ::= b \leftrightsquigarrow_{Z\phi} b \qquad \text{(combinator types)}$$
$$m ::= [] \mid c_Z :: m \mid c_\phi :: m \qquad \text{(amalgamations)}$$

$$\frac{}{[] : b \leftrightsquigarrow_{Z\phi} b} \qquad \frac{c : b_1 \leftrightarrow_Z b_2 \quad cs : b_2 \leftrightsquigarrow_{Z\phi} b_3}{c :: cs : b_1 \leftrightsquigarrow_{Z\phi} b_3} \qquad \frac{c : b_1 \leftrightarrow_\phi b_2 \quad cs : b_2 \leftrightsquigarrow_{Z\phi} b_3}{c :: cs : b_1 \leftrightsquigarrow_{Z\phi} b_3}$$

Fig. 6. Syntax and type rules of $\Pi\diamondsuit$. The combinators c_Z and c_ϕ are Π combinators (Fig. 2) tagged with their sublanguage of origin.

$$m ::= \ldots \mid m \mathbin{@} m \qquad \text{(derived amalgamations)}$$
$$d ::= m \mid arr_Z\ c_Z \mid arr_\phi\ c_\phi \mid d \ggg d \qquad \text{(derived combinators)}$$
$$\mid id : b \leftrightsquigarrow_{Z\phi} b \mid swap^\times : b_1 \times b_2 \leftrightsquigarrow_{Z\phi} b_2 \times b_1 \mid assocr^\times \mid assocl^\times$$
$$\mid unite^\times \mid uniti^\times \mid first\ d \mid second\ d \mid d \ast\!\!\ast\!\!\ast d \mid inv\ d$$

Fig. 7. Derived $\Pi\diamondsuit$ constructs. The combinators $first : b_1 \times b_2 \leftrightsquigarrow_{Z\phi} b_1$ and $second : b_1 \times b_2 \leftrightsquigarrow_{Z\phi} b_2$ are instances of the standard arrow constructors [29].

5.5 $\Pi\diamondsuit$: Syntax, Arrow Combinators, and Computational Universality

As the two languages Π_Z and Π_ϕ share the same syntax, their syntactic amalgamation in Fig. 6 is rather straightforward. We simply build sequences of expressions coming from either language. To disambiguate amalgamations, we will annotate terms from Π_Z and Π_ϕ by their language of origin and write, *e.g.*, x_ϕ for the x isomorphism from Π_ϕ. Further, we will consider the cons operator $::$ to be right associative, and use list notation such as $[swap^+_\phi, swap^+_Z]$ as syntactic sugar for the amalgamation $swap^+_\phi :: swap^+_Z :: []$.

For convenience, we introduce the meta-operation $\cdot@\cdot$ that takes two amalgamations and forms the amalgamation given by their concatenation, *e.g.*, $[c_1, c_2] \mathbin{@} [c_3, c_4] = [c_1, c_2, c_3, c_4]$. As such, any amalgamation can be uniquely described as a finite heterogeneous list of terms from Π_ϕ and Π_Z. But there is no need to reason about raw lists since $\Pi\diamondsuit$ is an *arrow* over both Π_Z and Π_ϕ that lifts the underlying multiplicative structure to the combined language (see Fig. 7 for the derived arrow constructs). We recall that the construction only involves lifting *products* (via the arrow combinators *first*, *second*, and $\ast\!\!\ast\!\!\ast$) and not also *sums*; in other words, we merely define an *arrow* and not an *arrow with choice* [29]. The reason is that doing so in any meaningful way would require semantically identifying the sum structures of Π_Z and Π_ϕ, and that this, in turn, would prevent quantum behaviours from emerging from the construction.

The semantics of $\Pi\diamondsuit$ (in SymMonAmalg(**Unitary**, Aut_{R^ϕ}(**Unitary**))) is given in Fig. 8. First the lifting of combinators builds singleton lists $arr_Z(c_Z) = [c_Z]$ and $arr_\phi(c_\phi) = [c_\phi]$, and composition of amalgamations is given by their concatenation: $cs_1 \ggg cs_2 = cs_1 \mathbin{@} cs_2$. To define *first* we need to make use of meta-level recursion in order to traverse amalgamations. To do this, we notice that both Π_Z and Π_ϕ are trivially arrows, with arrow lifting given by the identity, arrow composition given by composition, and *first c* given by $first\ c = c \times id$. As such, *first* can be defined in $\Pi\diamondsuit$ by mapping this underlying combinator over the list, *i.e.*, $first\ xs = map\ first\ xs$. To derive *second*, we note that we can define all of the combinators relating to $\times$ (precisely: $swap^\times$, $assocr^\times$, $assocl^\times$, $unite^\times$, $uniti^\times$) by lifting them from either Π_Z or Π_ϕ. It turns out not to matter which we choose, as they are equivalent.

Semantics of new constructs

$$[\![[c_1,\ldots,c_n]]\!] = [[\![c_1]\!],\ldots,[\![c_n]\!]]$$

Derived identities

$$\begin{array}{rclrclrcl}
[\![arr_Z\ c]\!] & = & \mathscr{E}_L([\![c]\!]) & [\![arr_\phi\ c]\!] & = & \mathscr{E}_R([\![c]\!]) & & & \\
[\![\mathrm{id}]\!] & = & [] & [\![d_1 \ggg d_2]\!] & = & [\![d_2]\!] \circ [\![d_1]\!] & [\![inv\ d]\!] & = & [\![d]\!]^\dagger \\
[\![assocr^\times]\!] & = & [\alpha_\otimes] & [\![assocl^\times]\!] & = & [\alpha_\otimes^{-1}] & [\![swap^\times]\!] & = & [\sigma_\otimes] \\
[\![unite^\times]\!] & = & [\rho_\otimes] & [\![uniti^\times]\!] & = & [\rho_\otimes^{-1}] & & & \\
[\![first\ d]\!] & = & [\![d]\!] \otimes \mathrm{id} & [\![second\ d]\!] & = & \mathrm{id} \otimes [\![d]\!] & [\![d_1 \ast\!\!\ast\!\!\ast d_2]\!] & = & [\![d_1]\!] \otimes [\![d_2]\!]
\end{array}$$

Fig. 8. The arrow semantics of Π◇.

Arbitrarily, we define their liftings in Π◇ to be those from Π_Z, *e.g.*, $swap^\times = arr_Z(swap_Z^\times)$ and so on for the remaining ones. We can then derive *second* and $\ast\!\!\ast\!\!\ast$ in the usual way as:

$$second = swap^\times \ggg first \ggg swap^\times \quad \text{and} \quad xs \ast\!\!\ast\!\!\ast ys = first\ xs \ggg second\ ys$$

We conclude this section by showing that there exists a particular model in which Π◇ is computationally universal for quantum circuits.

Theorem 5.11. *If ϕ is chosen to be $\pi/8$, the model of Π◇ is computationally universal for quantum circuits, i.e., unitaries on Hilbert spaces of dimension 2^n (for any natural number n).*

Proof. Define a functor $\mathrm{Aut}_{R^{\pi/8}}(\mathbf{Unitary}) \to \mathbf{Unitary}$ given by $A \mapsto A$ and $(R_B^{\pi/8})^{-1} \circ f \circ R_A^{\pi/8} \mapsto (R_B^{\pi/8})^{-1} \circ f \circ R_A^{\pi/8}$; this is strict monoidal and identity-on-objects. By Lem. 5.8, this functor, along with the identity $\mathbf{Unitary} \to \mathbf{Unitary}$ uniquely define a (strict monoidal identity-on-objects) functor $[\![\cdot]\!] : \mathrm{SymMonAmalg}(\mathbf{Unitary}, \mathrm{Aut}_{R^{\pi/8}}(\mathbf{Unitary})) \to \mathbf{Unitary}$ sending Π_Z programs to their usual interpretation lifted into unitaries, and Π_ϕ programs to the meaning of the corresponding Π_Z program conjugated by appropriate $R_A^{\pi/8}$'s. Under this interpretation, $[\![arr_Z\ \mathrm{ccx}]\!]$ is the Toffoli gate, and $[\![arr_\phi\ \mathrm{x}]\!]$ is the Hadamard gate, while at the same time $[\![c_1 \ast\!\!\ast\!\!\ast c_2]\!] = [\![c_1]\!] \otimes [\![c_2]\!]$ and $[\![c_1 \ggg c_2]\!] = [\![c_2]\!] \circ [\![c_1]\!]$, allowing parallel and sequential composition of gates. But then it follows by Thm. 2.1 that Π◇ is computationally universal. □

6 ⟨Π◇⟩ from States and Effects

In the previous section, we arbitrarily chose a value of ϕ to induce quantum behaviour. We will now demonstrate that the "right" values of ϕ emerge from requiring the categorical model to satisfy one complementarity equation relating states and effects. Towards that goal, we generalise in this section Π◇ with the notions of states $|\cdot\rangle$ and effects $\langle\cdot|$ yielding the language ⟨Π◇⟩.

6.1 Classical Structures

The no-cloning theorem states that it is not possible to clone an arbitrary quantum state. However, it is possible to clone the subset of quantum states that are "classical." For example:

$$\begin{array}{rcl}
|0\rangle & \mapsto & |00\rangle \\
|1\rangle & \mapsto & |11\rangle \\
1/\sqrt{2}\ (|0\rangle + |1\rangle) & \mapsto & 1/\sqrt{2}\ (|00\rangle + |11\rangle)
\end{array}$$

If partial operations are allowed, then these maps are reversible, i.e, the classical clone maps are injective but not surjective and their inverses are only *partial* injective functions. An important property of these classical clone maps is that their behaviour is *basis dependent*. In particular, the

above maps assume the clone operation is defined in the computational Z-basis. If we instead use the X-basis $= \{|+\rangle, |-\rangle\}$, we get that cloning $|+\rangle = 1/\sqrt{2}\ (|0\rangle + |1\rangle)$ produces $|{+}{+}\rangle = 1/2(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$ which is quite different from cloning the same state in the Z-basis.

As we will establish, the cloning operations in Π_Z and Π_ϕ each satisfy the properties of classical structures necessary for quantum behaviour to emerge in the next section.

6.2 Monoidal Indeterminates

At a categorical level, the only missing ingredient is to allow for morphisms to manipulate ancilla systems $\mathcal{J}_N(U)$ generated by a single object N. The first step in this construction involves (strong monoidal) functors out of the free symmetric monoidal category on a single distinguished object $\star$.

Definition 6.1. Let $\mathrm{Gen}_\star$ denote the free symmetric monoidal category of one generator, which we denote $\star$. Given an object N of a symmetric monoidal category $\mathbf{C}$, let $\mathcal{J}_N : \mathrm{Gen}_\star \to \mathbf{C}$ denote the evident strong monoidal functor such that $\mathcal{J}_N(\star) = N$.

See, e.g., [1] for an explicit description of the free symmetric monoidal category. The functor $\mathcal{J}_N$ allows us to form a new category $\mathbf{C}_N$.

Definition 6.2. Define a symmetric monoidal category $\mathbf{C}_N$ as follows:

- **Objects:** As in $\mathbf{C}$.
- **Morphisms:** Morphisms $A \to B$ are equivalence classes of triples $[U, f, V]$ consisting of two objects U and V of $\mathrm{Gen}_N(\mathbf{C})$ and a morphism $f : A \otimes \mathcal{J}_N(U) \to B \otimes \mathcal{J}_N(V)$ under the equivalence $\sim$ below.
- **Identities:** The identity $A \to A$ is the equivalence class of $\mathrm{id}_{A\otimes I}$ (since $\mathcal{J}_N(I) = I$).
- **Composition**: The composition of $[U, f, V]$ and $[W, g, X]$ with $f : A \otimes \mathcal{J}_N(U) \to B \otimes \mathcal{J}_N(V)$ and $g : B \otimes \mathcal{J}_N(W) \to C \otimes \mathcal{J}_N(X)$ is the equivalence class of the representative $A \otimes \mathcal{J}_N(U \otimes W) \to C \otimes \mathcal{J}_N(V \otimes X)$ in $\mathbf{C}$ given by

$$[U \otimes W, \alpha_\otimes \circ g \otimes \mathrm{id}_{\mathcal{J}_N(V)} \circ \alpha_\otimes^{-1} \circ \mathrm{id}_B \otimes \sigma_\otimes \circ \alpha_\otimes \circ f \otimes \mathrm{id}_{\mathcal{J}_N(W)} \circ \alpha_\otimes^{-1}, V \otimes X].$$

- **Monoidal structure**: On objects as in $\mathbf{C}$. On morphisms, the monoidal product of $[U, f, V]$ and $[W, g, X]$ with $f : A \otimes \mathcal{J}_N(U) \to B \otimes \mathcal{J}_N(V)$ and $g : C \otimes \mathcal{J}_N(W) \to D \otimes \mathcal{J}_N(X)$ is $[U \otimes W, \vartheta^{-1} \circ f \otimes g \circ \vartheta, V \otimes X]$, where $\vartheta : (A \otimes C) \otimes (B \otimes D) \to (A \otimes B) \otimes (C \otimes D)$ is the evident isomorphism. Coherence isomorphisms β are given by $[I, \beta \otimes \mathrm{id}_I, I]$.

Define the equivalence relation $\sim$ as the least such satisfying (for all U and V)

$$f \otimes \mathcal{J}_N(\mathrm{id}_U) \sim f \otimes \mathcal{J}_N(\mathrm{id}_V) \tag{10}$$

and $f \sim g$ if there exist mediators $m : U \to U'$ and $n : V \to V'$ in $\mathrm{Gen}_N(\mathbf{C})$ making the square below commute:

$$\begin{array}{ccc} A \otimes \mathcal{J}_N(U) & \xrightarrow{f} & B \otimes \mathcal{J}_N(V) \\ {\scriptstyle \mathrm{id}\otimes\mathcal{J}_N(m)}\downarrow & & \downarrow{\scriptstyle \mathrm{id}\otimes\mathcal{J}_N(n)} \\ A \otimes \mathcal{J}_N(U') & \xrightarrow[g]{} & B \otimes \mathcal{J}_N(V') \end{array} \tag{11}$$

This construction is a dual pair of *monoidal indeterminates* constructions of Hermida and Tennent [23] (using the simplified form for the equivalence due to Andrés-Martínez et al. [5, Def. 8]). This describes a symmetric monoidal category [23]. Note that the two clauses in the equivalence relation are necessary precisely to ensure uniqueness of identities and associativity of composition. For a given morphism $[X, f, Y]$, we will collectively denote the objects X and Y as the *ancilla system* of $[X, f, Y]$.

Importantly, this also defines an arrow over **C** in the form of a strict monoidal functor:

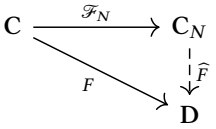
Proposition 6.3. *There is a strict monoidal functor $\mathscr{F}_N : \mathbf{C} \to \mathbf{C}_N$ (for any choice of N) given by $A \mapsto A$ on objects and $f \mapsto [I, f \otimes \mathrm{id}_I, I]$ on morphisms.*

Proof. See Hermida and Tennent [23, Remark 2.4]. □

The category $\mathbf{C}_N$ contains all morphisms f of **C** by lifting them into the trivial ancilla system $[I, f \otimes \mathrm{id}_I, I]$. However, it also adds a state $I \to N$ (as the equivalence class of $\sigma_\otimes : I \otimes N \to N \otimes I$, *i.e.*, $[N, \sigma_\otimes, I]$) and an effect $N \to I$ (as the equivalence class of $\sigma_\otimes : N \otimes I \to I \otimes N$, *i.e.*, $[I, \sigma_\otimes, N]$).

One may reasonably wonder whether this construction adds more than this unique state and effect: the answer, informally, is no. The trivial ancilla system (*e.g.*, $[I, f, I]$) is needed to account for morphisms that do not use states or effects at all, while ancilla systems involving $N \otimes \cdots \otimes N$ represent multiple uses of the unique state or effect. The more formal answer to this question is that the functor $\mathscr{F}_N$ is universal with the following property:

Proposition 6.4. *Given any symmetric monoidal category **D** outfitted with distinguished morphisms $I \to N$ and $N \to I$, and strict monoidal functor $F : \mathbf{C} \to \mathbf{D}$, there is a unique strict monoidal functor $\widehat{F} : \mathbf{C}_N \to \mathbf{D}$ satisfying $F(N) = N$ and making the triangle on the right commute.*



Proof. See Hermida and Tennent [23, Theorem 2.9], noting that this straightforwardly extends to strict monoidal functors when all functors involved are strict monoidal. □

Note that the morphisms $I \to N$ and $N \to I$ are considered part of the *structure* of **D**. While **D** has other choices of morphisms $I \to N$ and $N \to I$, the theorem states that for any such choice of morphisms and strict monoidal functor $\mathbf{C} \to \mathbf{D}$, there is a unique strict monoidal functor $\mathbf{C}_N \to \mathbf{D}$. Observe further that $\mathbf{C}_N$ is a dagger category when **C** is, with $[X, f, Y]^\dagger = [Y, f^\dagger, X]$. With this dagger structure, the adjoint of the state $I \to N$ is the effect $N \to I$ and vice versa.

6.3 Models of $\langle \Pi\diamondsuit \rangle$: $\mathrm{SymMonAmalg}(\mathbf{Unitary}, \mathrm{Aut}_{R^\phi}(\mathbf{Unitary}))_{I \oplus I}$ and Contraction

In order to model quantum computation, we specialize the free model $\mathbf{C}_N$ over an arbitrary model **C** built in the last section by fixing the semantics of the state and effect to correspond to the traditional basis vector $|0\rangle$ and dual vector $\langle 0|$. This is achieved by choosing $N = I \oplus I$ making $\mathrm{SymMonAmalg}(\mathbf{Unitary}, \mathrm{Aut}_{R^\phi}(\mathbf{Unitary}))_{I \oplus I}$ a model of $\langle \Pi\diamondsuit \rangle$. This model embeds in **Contraction**, the universal dagger rig category containing all unitaries, states, and effects [5] as follows.

Recall how we gave semantics to $\Pi\diamondsuit$ via an (identity-on-objects) strict monoidal functor $[\![-]\!] : \mathrm{SymMonAmalg}(\mathbf{Unitary}, \mathrm{Aut}_{R^\phi}(\mathbf{Unitary})) \to \mathbf{Unitary}$. Recall further that the category **Contraction** of finite-dimensional Hilbert spaces and contractions contains all states as morphisms $I \to A$ (as these are isometries), all effects as morphisms $A \to I$ (as these are coisometries). Since all unitaries are contractions we get an inclusion functor **Unitary** $\to$ **Contraction** (which is easily seen to be a strict monoidal dagger functor), and precomposing with the strict monoidal functor $[\![-]\!] : \mathrm{SymMonAmalg}(\mathbf{Unitary}, \mathrm{Aut}_{R^\phi}(\mathbf{Unitary})) \to \mathbf{Unitary}$ yields a functor:

$$\mathrm{SymMonAmalg}(\mathbf{Unitary}, \mathrm{Aut}_{R^\phi}(\mathbf{Unitary})) \to \mathbf{Contraction}$$

Now, choosing $|0\rangle$ as the distinguished state $I \to I \oplus I$, and $\langle 0|$ as the distinguished effect $I \oplus I \to I$, by Prop. 6.4 we get a unique extended functor

$$[\![-]\!] : \mathrm{SymMonAmalg}(\mathbf{Unitary}, \mathrm{Aut}_{R^\phi}(\mathbf{Unitary}))_{I \oplus I} \to \mathbf{Contraction}$$

which assigns concrete semantics to $\langle \Pi\diamondsuit \rangle$ in **Contraction**.

$$
\begin{array}{lr}
b ::= 0 \mid 1 \mid b + b \mid b \times b & \text{(value types)} \\
n ::= 1 \mid 1 + 1 \mid n \times n & \text{(ancilla types)} \\
t ::= b \leftrightsquigarrow b & \text{(combinator types)} \\
p ::= \mathit{lift}\ m & \text{(primitives)}
\end{array}
$$

$$
\frac{xs : b_1 \times n_1 \leftrightsquigarrow_{Z\phi} b_2 \times n_2}{\mathit{lift}\ xs : b_1 \leftrightsquigarrow b_2}
$$

Fig. 9. $\langle \Pi\diamondsuit \rangle$ syntax and type rules. Amalgamations m are defined in Figs. 6 and 7.

$$
\begin{array}{lr}
d ::= p \mid \mathit{arr}\ m \mid d \ggg d \mid \mathit{first}\ d \mid \mathit{second}\ d \mid d \ast\!\!\!\ast\!\!\!\ast\ d & \text{(derived combinators)} \\
\quad \mid \mathit{id} \mid \mathit{swap}^{\times} \mid \mathit{assocr}^{\times} \mid \mathit{assocl}^{\times} \mid \mathit{unite}^{\times} \mid \mathit{uniti}^{\times} & \\
\quad \mid \mathit{inv}\ d \mid \mathit{zero} \mid \mathit{assertZero} &
\end{array}
$$

Fig. 10. Derived $\langle \Pi\diamondsuit \rangle$ constructs. The combinators *first* and *second* are instances of the standard arrow constructors [29].

Semantics of new constructs

$$
[\![\mathit{lift}\ xs]\!] = [[\![n_1]\!], [\![xs]\!], [\![n_2]\!]]
$$

Derived identities

$$
\begin{array}{rclrclrcl}
[\![\mathrm{id}]\!] & = & [I, \mathrm{id}, I] & [\![\mathit{inv}\ d]\!] & = & [\![d]\!]^{\dagger} & [\![d_1 \ggg d_2]\!] & = & [\![d_2]\!] \circ [\![d_1]\!] \\
[\![\mathit{assocr}^{\times}]\!] & = & [I, \alpha_{\otimes} \otimes \mathrm{id}, I] & [\![\mathit{assocl}^{\times}]\!] & = & [I, \alpha_{\otimes}^{-1} \otimes \mathrm{id}, I] & & & \\
[\![\mathit{unite}^{\times}]\!] & = & [I, \rho_{\otimes} \otimes \mathrm{id}, I] & [\![\mathit{uniti}^{\times}]\!] & = & [I, \rho_{\otimes}^{-1} \otimes \mathrm{id}, I] & [\![\mathit{swap}^{\times}]\!] & = & [I, \sigma_{\otimes} \otimes \mathrm{id}, I] \\
[\![\mathit{first}\ d]\!] & = & [\![d]\!] \otimes \mathrm{id} & [\![\mathit{second}\ d]\!] & = & \mathrm{id} \otimes [\![d]\!] & [\![d_1 \ast\!\!\!\ast\!\!\!\ast\ d_2]\!] & = & [\![d_1]\!] \otimes [\![d_2]\!] \\
[\![\mathit{zero}]\!] & = & [I \oplus I, \sigma_{\otimes}, I] & [\![\mathit{assertZero}]\!] & = & [I, \sigma_{\otimes}, I \oplus I] & [\![\mathit{arr}\ m]\!] & = & \mathscr{F}_{I \oplus I}([\![m]\!])
\end{array}
$$

Fig. 11. The arrow semantics of $\langle \Pi\diamondsuit \rangle$.

The only difference between $\mathrm{SymMonAmalg}(\mathbf{Unitary}, \mathrm{Aut}_{R^{\phi}}(\mathbf{Unitary}))_{I \oplus I}$ and **Contraction** is that the latter has more morphisms, which can only be approximated with morphisms from the former – which matches our main theorem that $\langle \Pi\diamondsuit \rangle$ is (only) computationally (and not exactly) universal.

6.4 Syntax, States, and Effects

In $\langle \Pi\diamondsuit \rangle$, we allow the creation and discarding of a restricted set of values of ancilla types. As given in Fig. 9, the ancilla types are restricted to be collections of bits. The construct *lift* allows the discarding of some ancilla n_1 and the creation of some ancilla n_2. The new language defines an arrow over $\Pi\diamondsuit$ with the derived combinators in Fig. 10. Most notably, the language includes two new derived constructs *zero* and *assertZero* whose semantics are $|0\rangle$ and $\langle 0|$ respectively.

We summarise the arrow semantics of $\langle \Pi\diamondsuit \rangle$ in Fig. 11. To see that this is an arrow, we must define *arr*, $\ggg$, and *first*. Bringing combinators from $\Pi\diamondsuit$ into $\langle \Pi\diamondsuit \rangle$ is straightforwardly done by adding the trivial ancilla 1 to both the input and output, $\mathit{arr}\ m = \mathit{lift}(\mathit{unite}^{\times} \ggg m \ggg \mathit{uniti}^{\times})$. This allows us to lift the isomorphisms *id*, $\mathit{swap}^{\times}$, $\mathit{assocr}^{\times}$, $\mathit{assocl}^{\times}$, $\mathit{unite}^{\times}$, and $\mathit{uniti}^{\times}$ of $\Pi\diamondsuit$ by applying *arr* to them. To compose lifted $\Pi\diamondsuit$ terms $m : b_1 \times n_1 \leftrightsquigarrow_{Z\phi} b_2 \times n_2$ and $p : b_2 \times n_3 \leftrightsquigarrow_{Z\phi}$

$$
\begin{aligned}
copy_Z \ggg (id \ast\!\!\ast\!\!\ast\ copy_Z) &= copy_Z \ggg (copy_Z \ast\!\!\ast\!\!\ast\ id) \ggg assoc^\times \\
copy_Z \ggg swap^\times &= copy_Z \\
copy_Z \ggg (inv\ copy_Z) &= \mathrm{id} \\
(copy_Z \ast\!\!\ast\!\!\ast\ id) \ggg (id \ast\!\!\ast\!\!\ast\ inv\ copy_Z) &= (id \ast\!\!\ast\!\!\ast\ copy_Z) \ggg (inv\ copy_Z \ast\!\!\ast\!\!\ast\ id)
\end{aligned}
$$

$$
\begin{aligned}
copy_X \ggg (id \ast\!\!\ast\!\!\ast\ copy_X) &= copy_X \ggg (copy_X \ast\!\!\ast\!\!\ast\ id) \ggg assoc^\times \\
copy_X \ggg swap^\times &= copy_X \\
copy_X \ggg (inv\ copy_X) &= \mathrm{id} \\
(copy_X \ast\!\!\ast\!\!\ast\ id) \ggg (id \ast\!\!\ast\!\!\ast\ inv\ copy_X) &= (id \ast\!\!\ast\!\!\ast\ copy_X) \ggg (inv\ copy_X \ast\!\!\ast\!\!\ast\ id)
\end{aligned}
$$

$$
\begin{aligned}
zero \ggg assertZero &= id \\
zero \ast\!\!\ast\!\!\ast\ id \ggg \textsc{ctrl}\ c &= zero \ast\!\!\ast\!\!\ast\ id \\
one \ast\!\!\ast\!\!\ast\ id \ggg \textsc{ctrl}\ c &= one \ast\!\!\ast\!\!\ast\ c \\
zero \ggg \mathrm{x}_\phi \ggg assertOne &= one \ggg \mathrm{x}_\phi \ggg assertZero
\end{aligned}
$$

Fig. 12. Equations satisfied in $\langle \Pi\diamondsuit \rangle$.

$b_3 \times n_4$, since ancillae are closed under products, we can form this as the lifting of a term of type $b_1 \times (n_1 \times n_3) \leftrightsquigarrow_{Z\phi} b_3 \times (n_4 \times n_2)$, namely

$$
\begin{aligned}
(lift\ m) \ggg (lift\ p) = lift(assocl^\times \ggg first\ m \ggg assocr^\times \ggg second\ swap^\times \ggg \\
assocl^\times \ggg first\ p \ggg assocr^\times)\ .
\end{aligned}
$$

Then, *first* can be defined using *first* in $\Pi\diamondsuit$, since this allows us to extend a lifted term of type $b_1 \times n_1 \leftrightsquigarrow_{Z\phi} b_2 \times n_2$ to one of type $(b_1 \times n_1) \times b_3 \leftrightsquigarrow_{Z\phi} (b_2 \times n_2) \times b_3$, so we need only swap the ancillae back into the rightmost position from there, *i.e.*,

$$
\begin{aligned}
first(lift\ m) = lift(assocr^\times \ggg second\ swap^\times \ggg assocl^\times \ggg first\ m \ggg \\
assocr^\times \ggg second\ swap^\times \ggg assocr^\times)\ .
\end{aligned}
$$

In turn, *second* and $\ast\!\!\ast\!\!\ast$ are derived exactly as in $\Pi\diamondsuit$. Inversion is simple since lifted terms are symmetric in having an ancilla type on both their input and output, so we have $inv(lift\ m) = lift(inv\ m)$. Finally, the state *zero* and effect *assertZero* exist as the lifting of $swap^\times : 1 \times (1+1) \leftrightsquigarrow_{Z\phi} (1+1) \times 1$ and $swap^\times : (1+1) \times 1 \leftrightsquigarrow_{Z\phi} 1 \times (1+1)$, *i.e.*,

$$
zero = lift(swap^\times) : 1 \leftrightsquigarrow 1 + 1 \qquad \text{and} \qquad assertZero = lift(swap^\times) : 1 + 1 \leftrightsquigarrow 1\ ,
$$

bringing the state into and out of focus respectively. A pleasant consequence of these definitions is that $inv(zero) = assertZero$ and vice versa. More generally, states and effects in $\langle \Pi\diamondsuit \rangle$ satisfy the following properties.

Proposition 6.5 (Classical Structures for Π_Z and Π_ϕ and their execution laws). *To avoid clutter, we will implicitly lift Π_Z and Π_ϕ gates to $\langle \Pi\diamondsuit \rangle$, writing c_Z for arr $(arr_Z\ c)$ and c_ϕ for arr $(arr_\phi\ c)$. Introduce the following abbreviations:*

$$
\begin{aligned}
copy_Z &= uniti^\times \ggg id \ast\!\!\ast\!\!\ast\ zero \ggg cx_Z & \qquad copy_X &= x_\phi \ggg copy_Z \ggg x_\phi \ast\!\!\ast\!\!\ast\ x_\phi \\
one &= zero \ggg x_Z & \qquad assertOne &= x_Z \ggg assertZero
\end{aligned}
$$

The equations in Fig. 12 are satisfied in $\langle \Pi \diamondsuit \rangle$.

Proof. The first two groups state that $copy_Z$ and $copy_X$ are each a classical cloning map for the relevant basis: the Z-basis in the case of $copy_Z$ and some rotated basis depending on ϕ for $copy_X$. Because we fixed the semantics of Π_Z to be the standard semantics in **Unitary** without any rotation, the action of $copy_Z$ on input v is to produce the pair (v, v). The rotated version $copy_X$ has the same semantics but in another basis. The last group of equations are the *execution equations*, which are so named as they describe how the states (and, by duality, effects) interact with program execution. The first shows that preparing the zero state and then asserting it does nothing at all. The remaining equations define how states (and, by dualising the equations, effects) must interact with control, and with one another: e.g., the second equation shows that passing $|0\rangle$ to a control line prevents the controlled program from being executed, while the third shows that passing $|1\rangle$ on a control line executes the controlled program. □

6.5 Computational Universality

Like in the previous section, we can conclude that there exists a particular model in which $\langle \Pi \diamondsuit \rangle$ is computationally universal for quantum circuits equipped with arbitrary states and effects.

Theorem 6.6 (Expressivity). *If ϕ is chosen to be $\pi/8$, the model of $\langle \Pi \diamondsuit \rangle$ is computationally universal for quantum circuits equipped with arbitrary states and effects.*

The more technical presentation of this theorem is the following. Say that a *preparation of states* on a 2^n-dimensional Hilbert space is a tensor product $s_1 \otimes \cdots \otimes s_n$, where each s_i is either a state or an identity. Dually, a *preparation of effects* is the adjoint $s_1^\dagger \otimes \cdots \otimes s_n^\dagger$ to a preparation of states $s_1 \otimes \cdots \otimes s_n$. The theorem then states that $\langle \Pi \diamondsuit \rangle$ is approximately universal for contractions $A \longrightarrow B$ between Hilbert spaces A (of dimension 2^n) and B (of dimension 2^m) of the form $S\mathcal{U}E$, where S is a preparation of states, $\mathcal{U}$ is unitary, and E is a preparation of effects.

Proof. Let $s_1 \otimes \cdots \otimes s_n$ be some state preparation, $\mathcal{U}$ be some unitary, and $t_1^\dagger \otimes \cdots \otimes t_n^\dagger$ be some effect preparation. In the state preparation $s_1 \otimes \cdots \otimes s_n$, for each non-identity s_i, choose some unitary S_i mapping $|0\rangle$ to s_i. Likewise, in the effect preparation $t_1^\dagger \otimes \cdots \otimes t_n^\dagger$, choose for each non-identity $t_i^\dagger$ a unitary $T_i^\dagger$ mapping $\langle 0|$ to $t_i^\dagger$. Produce now a state preparation $s_1' \otimes \cdots \otimes s_n'$ where $s_i' = |0\rangle$ if s_i is a state, and $s_i' = \mathrm{id}$ if s_i is an identity. Produce an effect preparation $t_1' \otimes \cdots \otimes t_n'$ similarly. Notice that $(S_1 \otimes \cdots \otimes S_n)(s_1' \otimes \cdots \otimes s_n') = s_1 \otimes \cdots \otimes s_n$ and $(T_1 \otimes \cdots \otimes T_n)(t_1' \otimes \cdots \otimes t_n') = t_1 \otimes \cdots \otimes t_n$. However, the state preparation $s_1' \otimes \cdots \otimes s_n'$ involves only identities and $|0\rangle$, and so has a direct representation in $\Pi \diamondsuit$ as a product of a number of *id* and *zero* terms—call the resulting term p. Likewise, $t_1' \otimes \cdots \otimes t_n'$ has a direct representation in $\Pi \diamondsuit$ as a product of a number of *id* and *assertZero* terms—call the resulting term q. Finally, by Thm. 5.11 we can approximate the unitary $(S_1 \otimes \cdots \otimes S_n)\mathcal{U}(T_1^\dagger \otimes \cdots \otimes T_n^\dagger)$ by some $\Pi \diamondsuit$ term u. But then $p \ggg u \ggg q$ approximates $(s_1 \otimes \cdots \otimes s_n)\mathcal{U}(t_1^\dagger \otimes \cdots \otimes t_n^\dagger)$. □

Though the proof above may at first glance appear to be non-constructive (as it involves choice among unitaries), we note that a unitary in finite dimension mapping $|0\rangle$ to some $|v\rangle$ (of norm 1) can be constructed using the Gram-Schmidt process.

7 Canonicity and Quantum Computational Universality

This section proves the main result of the paper. So far, we have built a particular model of $\langle \Pi \diamondsuit \rangle$ in **Contraction** and proved that by imposing $\phi = \pi/8$, we get a computationally universal quantum programming language. In fact, it turns out that we have a much stronger result. *Any model of*

$\langle \Pi \diamondsuit \rangle$ *in* **Contraction** *satisfying the equations for classical structures and their execution laws defined in Prop. 6.5 as well as the complementarity equation in Def. 7.1 is computationally universal!*

Definition 7.1 (Complementarity). Using the same lifting notation as before, the complementarity law requires the following identity:

$$\begin{aligned} id \;=\; & (copy_Z \mathbin{***} id) \ggg assocr^{\times} \ggg \\ & (id \mathbin{***} (inv\ copy_X)) \ggg (id \mathbin{***} copy_X) \ggg assocl^{\times} \ggg \\ & ((inv\ copy_Z) \mathbin{***} id) \end{aligned}$$

Recall that the complementarity law is equivalent to the two bases being mutually unbiased. In particular, it holds in $\mathrm{SymMonAmalg}(\mathbf{Unitary}, \mathrm{Aut}_{R^\phi}(\mathbf{Unitary}))_{I\oplus I}$ if and only if $\phi = \pm\pi/8$.

To show the canonicity theorem, we rely on the following characterisation of orthonormal bases complementary to the Z-basis where the unitary change of basis is involutive:

Proposition 7.2. *Every orthonormal basis* $\{|b_1\rangle, |b_2\rangle\}$ *on* $\mathbb{C}^2$ *which is complementary to the* Z*-basis, and for which the associated change of basis unitary is involutive, is either of the form* $|b_1\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ e^{-i\theta} \end{pmatrix}, |b_2\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix} e^{i\theta} \\ -1 \end{pmatrix}$ *or of the form* $|b_1\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix} -1 \\ e^{-i\theta} \end{pmatrix} |b_2\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix} e^{i\theta} \\ 1 \end{pmatrix}$ *where* $\theta \in [0, 2\pi)$.

The proof is obtained via a straightforward matrix computation and is omitted for space (but can be found in the extended version of the paper.)

Theorem 7.3 (Canonicity). *If a categorical semantics* $[\![-]\!]$ *for* $\langle \Pi \diamondsuit \rangle$ *in* **Contraction** *satisfies the classical structure laws and the execution laws (defined in Prop. 6.5) and the complementarity law (Def. 7.1), then it is computationally universal. Specifically, it must be the semantics of Sec. 6.3 with the semantics of* x_ϕ *being the Hadamard gate (up to conjugation by* X *and* Z*) and:* $[\![copy_Z]\!] \colon |i\rangle \mapsto |ii\rangle$, $[\![zero]\!] = |0\rangle$, $[\![copy_X]\!] \colon |\pm\rangle \mapsto |\pm\pm\rangle$, *and* $[\![assertZero]\!] = \langle 0|$ *up to a global unitary.*

Proof. Observe that $[\![I + I]\!] = \mathbb{C}^2$ must be the qubit. Without loss of generality, we may assume that Π_Z has the usual semantics in the computational (Z) basis—this is the freedom that the global unitary affords.

The execution equations ensure that $\{[\![zero]\!], [\![one]\!]\}$ and $\{[\![plus]\!], [\![minus]\!]\}$ are copyable by $[\![copy_Z]\!]$ and $[\![copy_X]\!]$ respectively. The complementarity equations further ensure that $\{[\![zero]\!], [\![one]\!]\}$ and $\{[\![plus]\!], [\![minus]\!]\}$ form complementarity orthonormal bases for $\mathbb{C}^2$. By assumption $[\![zero]\!]$ and $[\![one]\!]$ form the Z-basis, so the only possibility for $\{[\![plus]\!], [\![minus]\!]\}$ is as an orthonormal basis complementary to the Z-basis.

Since $[\![arr_\phi\ swap^+]\!]$ is the symmetry of a symmetric monoidal category it is involutive, and by the complementarity equations it is the change of basis unitary between orthonormal bases. It follows then by Proposition 7.2 that

$$[\![arr_\phi\ swap^+]\!] = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & e^{i\theta} \\ e^{-i\theta} & -1 \end{pmatrix} \qquad \text{or} \qquad [\![arr_\phi\ swap^+]\!] = \frac{1}{\sqrt{2}}\begin{pmatrix} -1 & e^{i\theta} \\ e^{-i\theta} & 1 \end{pmatrix}$$

for some $\theta \in [0, 2\pi)$. The execution equation $zero \ggg \mathrm{x}_\phi \ggg assertOne = one \ggg \mathrm{x}_\phi \ggg assertZero$ translates to the requirement that $e^{i\theta} = \langle 0|\,[\![arr_\phi\ swap^+]\!]\,|1\rangle = \langle 1|\,[\![arr_\phi\ swap^+]\!]\,|0\rangle = e^{-i\theta}$ in turn implying $e^{i\theta} = \pm 1$. This leaves:

$$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \qquad \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -1 \\ -1 & -1 \end{pmatrix}, \qquad \frac{1}{\sqrt{2}}\begin{pmatrix} -1 & 1 \\ 1 & 1 \end{pmatrix}, \quad \text{and} \quad \frac{1}{\sqrt{2}}\begin{pmatrix} -1 & -1 \\ -1 & 1 \end{pmatrix}$$

as the only possibilities for $[\![arr_\phi\ swap^+]\!]$, which are precisely Hadamard up to conjugation by Z and/or X. Either way, this is a real basis-changing single-qubit unitary, so computationally universal in conjunction with the Toffoli gate (which is expressible in Π_Z) by Thm. 2.1. □

Observe that the Hadamard gate could already be expressed in $\Pi\diamondsuit$ with the appropriate choice of ϕ, without states and effects. The latter were only needed to impose equations on $\Pi\diamondsuit$ which essentially forces ϕ to be $\pi/8$.

8 QuantumΠ: Examples and Reasoning

Having shown that quantum behaviour emerges from two copies of a classical reversible programming language mediated by the complementarity equation, we now illustrate that developing quantum programs similarly only needs classical principles augmented with the complementarity equation, and some forms of reasoning can similarly be expressed. Before we proceed however, we present a sanitised version of $\langle\Pi\diamondsuit\rangle$ that fixes $\phi = \pi/8$, that hides some of the constructs that were only needed for the intermediate steps, and that uses Agda syntax for ease of experimentation and for providing machine-checked proofs of equivalences. The code shown elides some of the routine definitions which will be made publicly available.

8.1 QuantumΠ: Syntax and Terms

The public interface of QuantumΠ consists of two layers: the core reversible classical language Π (of Fig. 2) and the arrow layer. We reproduce these below using the notation in our Agda specification.

The QuantumΠ types are directly collected in an Agda datatype:

```
data U : Set where
  0     : U
  I     : U
  _+ᵤ_  : U → U → U
  _×ᵤ_  : U → U → U
```

Since commutative monoids are used multiple times, their definition is abstracted in a structure `CMon` that is instantiated twice as `M×` and `M+`. The Π-combinators are encoded in a type family:

```
data _↔_ : U → U → Set where
  id↔      : t ↔  t
  add      : t₁ M+.⇔ t₂ → t₁ ↔ t₂
  mult     : t₁ M×.⇔ t₂ → t₁ ↔ t₂
  dist     : (t₁ +ᵤ t₂) ×ᵤ t₃ ↔ (t₁ ×ᵤ t₃) +ᵤ (t₂ ×ᵤ t₃)
  factor   : {t₁ t₂ t₃ : U} → (t₁ ×ᵤ t₃) +ᵤ (t₂ ×ᵤ t₃) ↔ (t₁ +ᵤ t₂) ×ᵤ t₃
  absorbl  : t ×ᵤ 0 ↔ 0
  factorzr : 0 ↔  t ×ᵤ 0
  _⨟_      : (t₁ ↔ t₂) → (t₂ ↔ t₃) → (t₁ ↔ t₃)
  _⊕_      : (t₁ ↔ t₃) → (t₂ ↔ t₄) → (t₁ +ᵤ t₂ ↔ t₃ +ᵤ t₄)
  _⊗_      : (t₁ ↔ t₃) → (t₂ ↔ t₄) → (t₁ ×ᵤ t₂ ↔ t₃ ×ᵤ t₄)
```

Finally, the syntax and types of the QuantumΠ combinators are encoded in another type family that uses another instance `M` of our commutative monoid.

```
data _⇔_ : U → U → Set where
  arrZ       : (t₁ ↔ t₂) → (t₁ ⇔ t₂)
  arrϕ       : (t₁ ↔ t₂) → (t₁ ⇔ t₂)
  mult       : t₁ M.⇔ t₂ → t₁ ⇔ t₂
  id⇔         : t ⇔ t
  _>>>_      : (t₁ ⇔ t₂) → (t₂ ⇔ t₃) → (t₁ ⇔ t₃)
  _***_      : (t₁ ⇔ t₃) → (t₂ ⇔ t₄) → (t₁ ×ᵤ t₂ ⇔ t₃ ×ᵤ t₄)
  zero       : I ⇔ 2
  assertZero : 2 ⇔ I
```

In the following, we will refer to common gates and states which we collect here. The definitions are a straightforward transcription into Agda of the ones in previous sections. Below Π refers to the module that (abstractly) defines Π-combinators.

```
X H Z : 𝟚 ⇔ 𝟚
X = arrZ Π.swap₊
H = arrϕ Π.swap₊
Z = H >>> X >>> H

one plus minus : I ⇔ 𝟚
one = zero >>> X
plus = zero >>> H
minus = plus >>> Z
```

```
ctrlZ : (t ↔ t) → 𝟚 ×ᵤ t ⇔ 𝟚 ×ᵤ t
ctrlZ c = arrZ (ΠT.ctrl c)

cx cz : 𝟚 ×ᵤ 𝟚 ⇔ 𝟚 ×ᵤ 𝟚
cx = ctrlZ Π.swap₊
cz = id⇔ *** H >>> cx >>> id⇔ *** H

ccx : 𝟚 ×ᵤ 𝟚 ×ᵤ 𝟚 ⇔ 𝟚 ×ᵤ 𝟚 ×ᵤ 𝟚
ccx = arrZ ΠT.ccx
```

And so, as expected, the X gate and the H gate are both lifted versions of $swap^+$ from the underlying definition of Π. The classical gates of Π and their controlled versions are lifted using arrZ. Evidently QuantumΠ does not include complex numbers. However, the language, being computationally universal, can express them by encoding $a + ib$ as $\left(\begin{smallmatrix} a & -b \\ b & a \end{smallmatrix}\right)$ [3] where an extra qubit distinguishes the real part from the imaginary part. Under this encoding, we can express the controlled-S gate as:

```
ctrlS : 𝟚 ×ᵤ 𝟚 ×ᵤ 𝟚 ⇔ 𝟚 ×ᵤ 𝟚 ×ᵤ 𝟚
ctrlS = (id⇔ *** id⇔ *** H) >>>
        ccx >>>
        (id⇔ *** id⇔ *** H) >>>
        ccx
```

8.2 Proving Simple Equivalences

The laws of classical structures, the execution equations, and the complementarity law, combined with the conventional laws for arrows and monoidal and rig categories, allow us to reason about QuantumΠ programs at an abstract extensional level that eschews complex numbers, vectors, and matrices. As a first demonstration, we can prove that the X and H gates are both involutive:

```
xInv : (X >>> X) ≡ id⇔
xInv =
  begin
    (X >>> X)                    ≡⟨ arrZR ⟩
    (arrZ (Π.swap₊ ⨟ Π.swap₊))   ≡⟨ classicalZ linv⨟l ⟩
    (arrZ id↔)                   ≡⟨ arrZidL ⟩
    id⇔                          ■

hadInv : (H >>> H) ≡ id⇔
hadInv = arrϕR ⊙ classicalϕ linv⨟l ⊙ arrϕidL
```

The first proof uses Agda's equational style where each step is justified by one of the QuantumΠ equivalences (see full code). The proof starts by moving "under the arrow" exposing the underlying $swap^+$ gate, using the fact that $swap^+$ is an involution, and then lifting the equivalence back through the arrow. The proof for H, presented as a sequence of equivalences, applies the same strategy to the other arrow. In later examples, we also use some additional reasoning combinators that help manage congruences, associativity and sequencing. For a more interesting example, we prove that the Z gate sends $[\![minus]\!]$ to $[\![plus]\!]$:

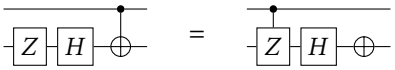
```
minusZ≡plus : (minus >>> Z) ≡ plus
minusZ≡plus =
  begin
    (minus >>> Z)
      ≡⟨ id≡ ⟩
    ((plus >>> H >>> X >>> H) >>> H >>> X >>> H)
      ≡⟨ ((assoc>>>l ⊙ assoc>>>l) ⟩⨟⟨id ) ⊙ pullʳ assoc>>>l ⟩
    (((plus >>> H) >>> X) >>> (H >>> H) >>> X >>> H)
      ≡⟨ id⟩⨟⟨ ((hadInv ⟩⨟⟨id) ⊙ idl>>>l) ⟩
    (((plus >>> H) >>> X) >>> X >>> H)
      ≡⟨ pullʳ assoc>>>l ⟩
```

```
    ((plus >>> H) >>> (X >>> X) >>> H)
      ≡⟨ id⟩⨟⟨ (xInv ⟩⨟⟨id ⊙ idl>>>l) ⟩
    ((plus >>> H) >>> H)
      ≡⟨ cancelʳ hadInv ⟩
    plus ■
```

The execution equations allow us to extend this reasoning to programs involving control, *e.g.*:

```
oneMinusPlus : ((one *** minus) >>> cz) ≡ (one *** plus)
oneMinusPlus = begin
  (one *** minus) >>> (id⇔ *** H) >>> cx >>> (id⇔ *** H)
    ≡⟨ assoc>>>l ⊙ (homL*** ⟩⨟⟨id) ⟩
  ((one >>> id⇔) *** (minus >>> H)) >>> cx >>> (id⇔ *** H)
    ≡⟨ idr>>>l ⟩⊗⟨id ⟩⨟⟨id ⟩
  (one *** (minus >>> H))>>> cx >>> (id⇔ *** H)
    ≡⟨ idl>>>r ⟩⊗⟨ idr>>>r ⟩⨟⟨id ⟩
  ((id⇔ >>> one) *** ((minus >>> H) >>> id⇔)) >>> cx >>> (id⇔ *** H)
    ≡⟨ homR*** ⟩⨟⟨id ⊙ assoc>>>r ⟩
  (id⇔ *** (minus >>> H)) >>> (one *** id⇔) >>> cx >>> (id⇔ *** H)
    ≡⟨ id⟩⨟⟨ (assoc>>>l ⊙ e3L ⟩⨟⟨id) ⟩
  (id⇔ *** (minus >>> H)) >>> (one *** X) >>> (id⇔ *** H)
    ≡⟨ id⟩⨟⟨ (homL*** ⊙ (idr>>>l ⟩⊗⟨id)) ⟩
  (id⇔ *** (minus >>> H)) >>> (one *** (X >>> H))
    ≡⟨ homL*** ⊙ (idl>>>l ⟩⊗⟨ assoc>>>r ) ⟩
  one *** (minus >>> H >>> X >>> H)
    ≡⟨ id⟩⊗⟨ minusZ≡plus  ⟩
  (one *** plus) ■
```

We can string together more involved proofs to establish more involved identities.




```
xcxA : id⇔ *** X >>> cx ≡ cx >>> id⇔ *** X
xcxA = begin
  id⇔ *** X >>> cx                    ≡⟨ arrZidR ⟩⊗⟨id ⟩⨟⟨id ⊙ class*>R ⟩
  arrZ ((id↔ Π.⊗ Π.swap₊) Π.⨟ ΠT.cx)  ≡⟨ classicalZ xcx ⟩
  arrZ (ΠT.cx Π.⨟ (id↔ Π.⊗ Π.swap₊))  ≡⟨ class>*L ⊙ id⟩⨟⟨ arrZidL ⟩⊗⟨id ⟩
  cx >>> id⇔ *** X                    ■

zhcx : (id⇔ *** Z) >>> (id⇔ *** H) >>> cx ≡ cz >>> (id⇔ *** H) >>> (id⇔ *** X)
zhcx = begin
  (id⇔ *** Z) >>> (id⇔ *** H) >>> cx
    ≡⟨ id≡ ⟩
  (id⇔ *** (H >>> X >>> H)) >>> (id⇔ *** H) >>> cx
    ≡⟨ assoc>>>l ⊙ (homL*** ⊙ (idl>>>l ⟩⊗⟨id)) ⟩⨟⟨id ⟩
  (id⇔ *** ((H >>> X >>> H) >>> H)) >>> cx
    ≡⟨ id⟩⊗⟨ pullʳ (cancelʳ hadInv) ⟩⨟⟨id ⟩
  id⇔ *** (H >>> X) >>> cx
    ≡⟨ (idl>>>r ⟩⊗⟨id ⊙ homR***) ⟩⨟⟨id ⊙ assoc>>>r ⟩
  (id⇔ *** H) >>> (id⇔ *** X) >>> cx
    ≡⟨ id⟩⨟⟨ xcxA ⟩
  (id⇔ *** H) >>> cx >>> (id⇔ *** X)
    ≡⟨ id⟩⨟⟨ id⟩⨟⟨ insertᶠ 1*HInv ⟩
  (id⇔ *** H) >>> cx >>> (id⇔ *** H) >>> (id⇔ *** H) >>> (id⇔ *** X)
    ≡⟨ assoc>>>l ⊙ assoc>>>l ⊙ assoc>>>r ⟩⨟⟨id ⟩
  (id⇔ *** H >>> cx >>> id⇔ *** H) >>> (id⇔ *** H) >>> (id⇔ *** X)
    ≡⟨ id≡ ⟩
  cz >>> (id⇔ *** H) >>> (id⇔ *** X)      ■
```

8.3 Postulating Measurement

Heunen and Kaarsgaard [25] derive quantum measurement as a computational effect layered on top of a language of unitaries, by extending the language with effects for *classical cloning* and *hiding*. Since QuantumΠ already has notions of classical cloning given by the $copy_Z$ and $copy_X$ combinators, we only need to extend it with hiding to obtain measurement of qubit systems.

We can extend QuantumΠ with hiding using the exact same arrow construction as the one used to introduce hiding in $\mathcal{U}\Pi_a^\chi$ by Heunen and Kaarsgaard [25], with two subtle differences. The first difference is that, since the model of QuantumΠ is a (dagger) symmetric monoidal category and not a rig category, this will yield a mere *arrow* over QuantumΠ, and not an *arrow with choice*:

$$\frac{c : b_1 \leftrightsquigarrow b_2 \times b_3 \quad b_3 \text{ inhabited}}{\mathit{lift}\ c : b_1 \rightsquigarrow b_2}$$

All available arrow combinators, including the crucial $\mathit{discard} : b \rightsquigarrow 1$ and the derived projections $\mathit{fst} : b_1 \times b_2 \rightsquigarrow b_1$ and $\mathit{snd} : b_1 \times b_2 \rightsquigarrow b_2$, are defined precisely as Heunen and Kaarsgaard [25] define them. The second difference concerns partiality of the model. Since the model of QuantumΠ is one of *partial* maps (whereas the model of $\mathcal{U}\Pi_a$ is one of *total* maps), we need to accommodate for this in the categorical model. This is precisely what is done by the $L^t_\otimes$-construction of Andrés-Martínez et al. [5]. In short, the resulting model will not satisfy $[\![c \ggg \mathit{discard}]\!] = [\![\mathit{discard}]\!]$ for *all* programs c, though it will satisfy it those c for which $[\![c \ggg \mathit{inv}\ c]\!] = [\![\mathit{id}]\!]$.

With this notion of hiding, we can derive measurement in the two bases as

$$\mathit{measure}_Z = \mathit{copy}_Z \ggg \mathit{fst} \qquad \text{and} \qquad \mathit{measure}_\phi = \mathit{copy}_X \ggg \mathit{fst} \tag{12}$$

exactly as done in previous work [18, 25]. Note that, by commutativity of copying, we could equivalently have chosen the second projection *snd* instead of *fst*. This can all be expressed in the Agda formalisation as follows:

```
postulate
  discard : t ⇔ I
  discardL : (d : t₁ ⇔ t₂) → d >>> discard ≡ discard
```

This postulate is *dangerous*, as it does not enforce that it is only applied to total maps (though we are careful to only do so in the examples here). We hope to patch this loophole in future work.

```
  fst : (t₁ ×ᵤ t₂) ⇔ t₁
  fst = (id⇔ *** discard) >>> unite⋆r

  snd : (t₁ ×ᵤ t₂) ⇔ t₂
  snd = swap⋆ >>> fst
```

```
measureZ measureϕ : 2 ⇔ 2
measureZ = copyZ >>> fst
measureϕ = copyϕ >>> fst
```

From this observation, measurements in the ϕ-basis are nothing more than measurement in the Z-basis conjugated by H. Following the same principle, we can define measurement in more exotic bases. For example, measurement in the 2-qubit Bell basis can be defined by conjugating a pair of Z-measurements by the unitary CX $\ggg$ H $***$ *id*.

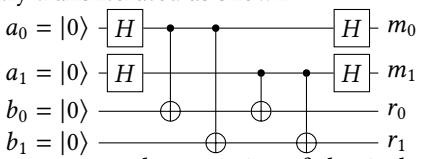
```
measure : measureϕ ≡ (H >>> measureZ >>> H)
measure = begin
  measureϕ                ≡⟨ id≡ ⟩ -- definition
  copyϕ >>> fst           ≡⟨ id≡ ⟩ -- definitions
  (H >>> copyZ >>> (H *** H)) >>> (id⇔ *** discard) >>> unite⋆r
    ≡⟨ assoc>>>l ⟩⨟⟨id ⊙ assoc>>>r ⊙ id⟩⨟⟨ assoc>>>l ⟩
  (H >>> copyZ) >>> ((H *** H) >>> (id⇔ *** discard)) >>> unite⋆r
    ≡⟨ id⟩⨟⟨ homL*** ⟩⨟⟨id ⟩
  (H >>> copyZ) >>> ((H >>> id⇔) *** (H >>> discard)) >>> unite⋆r
    ≡⟨ id⟩⨟⟨ idr>>>l ⟩⊗⟨ discardL H ⟩⨟⟨id ⟩
  (H >>> copyZ) >>> H *** discard >>> unite⋆r
```

```
  ≡⟨ id⟩⨟⟨ seq21*** ⟩⨟⟨id ⟩
(H >>> copyZ) >>> (id⇔ *** discard >>> H *** id⇔) >>> unite⋆r
  ≡⟨ assoc>>>r ⊙ id⟩⨟⟨ (assoc>>>l ⊙ assoc>>>l )⨟⟨id ⊙ assoc>>>r) ⟩
H >>> (copyZ >>> id⇔ *** discard) >>> (H *** id⇔) >>> unite⋆r
  ≡⟨ id⟩⨟⟨ id⟩⨟⟨ uniter⋆≡r ⟩
H >>> (copyZ >>> id⇔ *** discard) >>> (unite⋆r >>> H)
  ≡⟨ id⟩⨟⟨ (assoc>>>l ⊙ assoc>>>r )⨟⟨id) ⟩
H >>> (copyZ >>> id⇔ *** discard >>> unite⋆r) >>> H
  ≡⟨ id≡ ⟩
(H >>> measureZ >>> H) ■
```

8.4 Quantum Algorithms: Simon and Grover

The language QuantumΠ can easily model textbook quantum algorithms. The circuit on the right solves an instance of Simon's problem; it can be directly transliterated as shown:

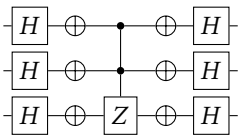
```
simon : I ×u I ×u I ×u I ⇔ 2 ×u 2 ×u 2 ×u 2
simon = map4 zero >>>
        H *** H *** id⇔ *** id⇔ >>>
        arrZ cxGroup >>>
        H *** H *** id⇔ *** id⇔
```



The four CX-gates, and more generally an arbitrary quantum oracle consisting of classical gates, can be implemented in the underlying classical language and lifted to QuantumΠ.

Having access to measurement allows us to express end-to-end algorithms as we illustrate with an implementation of a small instance of Grover's search [22], which, with high probability, is able to find a particular element x_0 in an unstructured data store of size n by probing it only $O(\sqrt{n})$ times. The algorithm works by preparing a particular quantum state, and then repeating a subprogram—the *Grover iteration*, consisting of an *oracle* stage and an *amplification* stage—a fixed number of times proportional to $\sqrt{n}$, before finally measuring the output. The data store of size n is implemented as a unitary $U : [2^n] \to [2^n]$ such that $U|x\rangle = -|x\rangle$ if $|x\rangle$ is the element $|x_0\rangle$ being searched for, and $U|x\rangle = |x\rangle$ otherwise. Though this uses nontrivial phases, it is still a classical program in disguise, and one could also use a classical function instead (though this presentation is slightly more economical). We assume that this unitary is given to us in the form of a QuantumΠ program. The final part of the algorithm is the amplification stage, which guides the search towards $|x_0\rangle$. This part is the same for every oracle, depending only on the number of qubits. On three qubits, the amplifier is given by the circuit on the right. Using the fact that the Z gate is actually negation conjugated by Hadamard, we see that this 3-qubit amplifier is expressible as the QuantumΠ program on the left:

```
amp : 2 ×u 2 ×u 2 ⇔ 2 ×u 2 ×u 2
amp = map3 H >>>
      map3 X >>>
      id⇔ *** id⇔ *** H >>>
      ccx >>>
      id⇔ *** id⇔ *** H >>>
      map3 X >>>
      map3 H
```



To complete the implementation, suppose that we are given a unitary of the form $u : 2^3 \leftrightsquigarrow_{Z\phi} 2^3$ described above. The initial state before iteration should be $|+++\rangle$, and we need to repeat the Grover iteration $\lceil \frac{\pi}{4}\sqrt{2^3} \rceil = 3$ times before measuring the output in the computational basis.

All together, this yields the following QuantumΠ program implementing 3-qubit Grover search:

```
grover₃ : I ×ᵤ I ×ᵤ I ⇔ 2 ×ᵤ 2 ×ᵤ 2
grover₃ = map3 plus >>>
          repeat 3 (u >>> amp) >>>
          map3 measureZ
```

8.5 The Formalization

Figure 5 and Theorem 3.1 were previously formalized [12, 15]. We additionally formalized [9] everything shown in Figures 2–4, and 6–11, Definitions 4.4–5.1, Proposition 5.2, the arrows of Proposition 5.9, the construction of Definitions 4.5 and 6.1 as well as all examples and proofs in this section.

We have not formalized the Aharonov-Shi Theorem 2.1, nor Proposition 6.5, Theorem 5.11, Theorem 7.3, although all the constructions used in theorems have been implemented.

9 Future Work

The more Π*, the better precision.* We have shown that two copies of a classical language, when aligned just right, are sufficient to yield computationally universal quantum computation. However, encoding general rotation gates, such as the ones needed for the quantum Fourier transform, is awkward and inefficient using just Toffoli and Hadamard. Can additional copies of the classical base language, when aligned carefully, improve this—and, if so, by how much?

Completeness. An equational theory is *sound and complete* when any two objects in the semantic domain are equal iff they are provably equal using the rules of the equational theory. While it is clear that reasoning in QuantumΠ is sound, completeness is wholly unclear. A complete equational theory for the Clifford+T gate set beyond 2 qubits is already unknown.

Formal quantum experiments. Programming languages have served as tools in thought experiments about physical theories [2, 42]. (An extension of) QuantumΠ could be used similarly, to formulate contextuality scenarios or quantum protocols, and use the reasoning capabilities of QuantumΠ to answer questions about them.

Approximate reasoning. The categorical semantics of QuantumΠ only give a way to prove that two programs are *exactly* equal. How can we extend the model of QuantumΠ to account for reasoning that two programs are equal not on the nose, but up to a given error?

Acknowledgments

This material is based upon work supported by the United States' National Science Foundation under Grant No. 1936353, and Canada's National Science and Engineering Council under Grant RGPIN-2018-05812. We are grateful to Tiffany Duneau for her comments and suggestions on an earlier version of this paper.

References

[1] S. Abramsky. 2005. Abstract scalars, loops, and free traced and strongly compact closed categories. In *International Conference on Algebra and Coalgebra in Computer Science*. Springer, 1–29.

[2] S. Abramsky and D. Horsman. 2015. DEMONIC programming: a computational language for single-particle equilibrium thermodynamics, and its formal semantics.. In *Proceedings 12th International Workshop on Quantum Physics and Logic (Electronic Proceedings in Theoretical Computer Science, 195)*. 1–16.

[3] D. Aharonov. 2003. A simple proof that Toffoli and Hadamard are quantum universal. (2003). arXiv:quant-ph/0301040.

[4] T. Altenkirch and A. Green. 2009. The Quantum IO Monad. In *Semantic Techniques in Quantum Computation*, S. Gay and I. Mackie (Eds.). Cambridge University Press, 173–205.

[5] P. Andrés-Martínez, C. Heunen, and R. Kaarsgaard. 2022. Universal Properties of Partial Quantum Maps. (2022). arXiv:2206.04814.

[6] M. Backens and A. Kissinger. 2019. ZH: A complete graphical calculus for quantum computations involving classical non-linearity. In *Quantum Physics and Logic (Electronic Proceedings in Theoretical Computer Science, 287)*. 23–42.

[7] B. Bichsel, M. Baader, T. Gehr, and M. Vechev. 2020. Silq: A High-Level Quantum Language with Safe Uncomputation and Intuitive Semantics. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM. https://doi.org/10.1145/3385412.3386007

[8] G. A. Briggs, J. N. Butterfield, and A. Zeilinger. 2013. The Oxford Questions on the foundations of quantum physics. *Proceedings. Mathematical, physical, and engineering sciences* 469, 2157 (2013).

[9] J. Carette, C. Heunen, R. Kaarsgaard, and A. Sabry. 2024. Code for How to Bake a Quantum Π. https://doi.org/10.5281/zenodo.11491613

[10] J. Carette, C. Heunen, R. Kaarsgaard, and A. Sabry. 2024. With a Few Square Roots, Quantum Computing is as Easy as Pi. *Proceedings of the ACM on Programming Languages* 8, POPL, Article 19 (2024), 546–574 pages.

[11] J. Carette, R. P. James, and A. Sabry. 2022. Embracing the laws of physics: Three reversible models of computation. Advances in Computers, Vol. 126. Elsevier, 15–63. https://doi.org/10.1016/bs.adcom.2021.11.009

[12] J. Carette and A. Sabry. 2016. Computing with Semirings and Weak Rig Groupoids. In *Programming Languages and Systems*, P. Thiemann (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 123–148.

[13] C.-H. Chen and A. Sabry. 2021. A Computational Interpretation of Compact Closed Categories: Reversible Programming with Negative and Fractional Types. *Proc. ACM Program. Lang.* 5, POPL, Article 9 (1 2021), 29 pages. https://doi.org/10.1145/3434290

[14] K. Cho and A. Westerbaan. 2016. Von Neumann Algebras Form a Model for the Quantum Lambda Calculus. (2016). arXiv:1603.02133 [cs.LO]

[15] V. Choudhury, J. Karwowski, and A. Sabry. 2022. Symmetries in Reversible Programming: From Symmetric Rig Groupoids to Reversible Programming Languages. *Proc. ACM Program. Lang.* 6, POPL, Article 6 (1 2022), 32 pages. https://doi.org/10.1145/3498667

[16] B. Coecke and R. Duncan. 2011. Interacting quantum observables: categorical algebra and diagrammatics. *New Journal of Physics* 13 (2011), 043016.

[17] B. Coecke, R. Duncan, A. Kissinger, and Q. Wang. 2012. Strong complementarity and non-locality in categorical quantum mechanics. In *Logic in Computer Science*. IEEE.

[18] B. Coecke and S. Perdrix. 2012. Environment and Classical Channels in Categorical Quantum Mechanics. *Logical Methods in Computer Science* 8 (2012), 1–24. Issue 4.

[19] C. Comfort. 2019. Circuit relations for real stabilizers: towards TOF+H. (2019). arXiv:1904.10614.

[20] R. Glück, R. Kaarsgaard, and T. Yokoyama. 2019. Reversible programs have reversible semantics. In *Formal Methods. FM 2019 International Workshops (Lecture Notes in Computer Science, Vol. 12232)*. Springer, 413–427.

[21] A. Green, P. LeFanu Lumsdaine, N. J. Ross, P. Selinger, and B. Valiron. 2013. Quipper: a Scalable Quantum Programming Language. In *Proceedings of the 34th Annual ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2013, Seattle (ACM SIGPLAN Notices, Vol. 48(6))*. 333–342. https://doi.org/10.1145/2499370.2462177

[22] L. K. Grover. 1996. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth annual ACM Symposium on Theory of Computing (STOC '96)*. ACM, 212–219.

[23] C. Hermida and R. D. Tennent. 2012. Monoidal Indeterminates and Categories of Possible Worlds. *Theoretical Computer Science* 430 (2012), 3–22. https://doi.org/10.1016/j.tcs.2012.01.001

[24] C. Heunen. 2013. On the functor ℓ^2. In *Computation, Logic, Games, and Quantum Foundations*. Springer, 107–121.

[25] C. Heunen and R. Kaarsgaard. 2022. Quantum Information Effects. *Proceedings of the ACM on Programming Languages* 6, POPL (2022), 1–27.

[26] C. Heunen, R. Kaarsgaard, and M. Karvonen. 2018. Reversible effects as inverse arrows. In *Proceedings of the Thirty-Fourth Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXIV) (Electronic Notes in Theoretical Computer Science, Vol. 341)*. Elsevier, 179–199.

[27] C. Heunen and M. Karvonen. 2016. Monads on dagger categories. *Theory and Applications of Categories* 31 (2016), 1016–1043.

[28] C. Heunen and J. Vicary. 2019. *Categories for quantum theory*. Oxford University Press.

[29] J. Hughes. 2005. Programming with Arrows. In *Advanced Functional Programming (Lecture Notes in Computer Science, Vol. 3622)*. Springer, 73–129. https://doi.org/10.1007/11546382_2

[30] M. Huot and S. Staton. 2019. Quantum Channels as a Categorical Completion. In *Proceedings of the ACM/IEEE Symposium on Logic in Computer Science*, Vol. 35. 1–13.

[31] B. Jacobs, C. Heunen, and I. Hasuo. 2009. Categorical Semantics for Arrows. *Journal of Functional Programming* 19, 3–4 (2009), 403–438. https://doi.org/10.1017/S0956796809007308

[32] P. A. H. Jacobsen, R. Kaarsgaard, and M. K. Thomsen. 2018. CoreFun: A Typed Functional Reversible Core Language. In *International Conference on Reversible Computation (RC 2018)*. Springer, 304–321.

[33] R. P. James and A. Sabry. 2012. Information Effects. In *POPL '12: Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of programming languages*. ACM, 73–84. https://doi.org/10.1145/2103656.2103667
[34] X. Jia, A. Kornell, B. Lindenhovius, M. Mislove, and V. Zamdzhiev. 2022. Semantics for Variational Quantum Programming. *Proc. ACM Program. Lang.* 6, POPL, Article 26 (2022), 31 pages. https://doi.org/10.1145/3498687
[35] J. Kastl. 1979. *Algebraische Modelle, Kategorien und Gruppoide*. Studien zur Algebra und ihre Anwendungen, Vol. 7. Akademie-Verlag Berlin, Chapter Inverse categories, 51–60.
[36] M. L. Laplaza. 1972. Coherence for distributivity. In *Coherence in categories (Lecture Notes in Mathematics, 281)*. Springer, 29–65.
[37] T. Leinster. 2014. *Basic category theory*. Cambridge University Press.
[38] M. Loaiza. 2017. A short introduction to Hilbert space theory. In *Journal of Physics: Conference Series*, Vol. 839. IOP Publishing, 012002.
[39] S. Mac Lane. 1963. Natural Associativity and Commutativity. *Rice University Studies* 49, 4 (1963).
[40] J. MacDonald and L. Scull. 2009. Amalgamations of categories. *Canad. Math. Bull.* 52, 2 (2009), 273–284.
[41] M. A. Nielsen and I. Chuang. 2002. *Quantum Computation and Quantum Information*. Cambridge University Press.
[42] N. Nurgalieva, S. Mathis, L. Del Rio, and R. Renner. 2022. Quanundrum – a platform to simulate thought experiments with quantum agents. Software package, https://github.com/jangnur/Quanundrum.
[43] J. Paykin, R. Rand, and S. Zdancewic. 2017. QWIRE: A Core Language for Quantum Circuits. *POPL 2017: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, 846–858. https://doi.org/10.1145/3009837.3009894
[44] R. Péchoux, S. Perdrix, M. Rennela, and V. Zamdzhiev. 2020. Quantum Programming with Inductive Datatypes: Causality and Affine Type Theory. In *Foundations of Software Science and Computation Structures (FOSSACS 2020) (Lecture Notes in Computer Science, Vol. 12077)*. 562–581. https://doi.org/10.1007/978-3-030-45231-5_29
[45] J. Power and E. Robinson. 1997. Premonoidal Categories and Notions of Computation. *Mathematical Structures in Computer Science* 7, 5 (1997). https://doi.org/10.1017/S0960129597002375
[46] M. Rennela and S. Staton. 2020. Classical Control, Quantum Circuits and Linear Logic in Enriched Category Theory. *Logical Methods in Computer Science* 16 (2020), 6192. https://doi.org/10.23638/LMCS-16(1:30)2020
[47] A. Sabry, B. Valiron, and J. K. Vizzotto. 2018. From symmetric pattern-matching to quantum control. In *International Conference on Foundations of Software Science and Computation Structures (FOSSACS 2018)*. Springer, 348–364.
[48] P. Selinger. 2004. Towards a Quantum Programming Language. *Mathematical Structures in Computer Science* 14, 4 (2004), 527–586. https://doi.org/10.1017/S0960129504004256
[49] P. Selinger and B. Valiron. 2005. A Lambda Calculus for Quantum Computation with Classical Control. In *Proceedings of the 7th International Conference on Typed Lambda Calculi and Applications, TLCA 2005, Nara, Japan (Lecture Notes in Computer Science, Vol. 3461)*. Springer, 354–368. https://doi.org/10.1007/11417170_26
[50] P. Selinger and B. Valiron. 2009. Quantum Lambda Calculus. In *Semantic Techniques in Quantum Computation*, S. Gay and I. Mackie (Eds.). Cambridge University Press, Chapter 4, 135–172.
[51] Y. Shi. 2003. Both Toffoli and Controlled-NOT Need Little Help to Do Universal Quantum Computing. *Quantum Info. Comput.* 3, 1 (1 2003), 84–92.
[52] S. Staton. 2015. Algebraic Effects, Linearity, and Quantum Programming Languages. In *Proceedings of the ACM Symposium on Principles of Programming Languages*. 395–406.
[53] F. Voichick, R. Rand, and M. Hicks. 2022. Qunity: A Unified Language for Quantum and Classical Computing. https://doi.org/arxiv.2204.12384 arXiv:2204.12384 [cs.PL]
[54] A. Westerbaan. 2017. Quantum Programs as Kleisli Maps. In *Proceedings 13th International Conference on Quantum Physics and Logic (QPL 2016) (Electronic Proceedings in Theoretical Computer Science, Vol. 236)*. 215–228. https://doi.org/10.4204/EPTCS.236.14
[55] N. Yanofsky and M. A. Mannucci. 2008. *Quantum Computing for Computer Scientists*. Cambridge University Press.
[56] T. Yokoyama, H. B. Axelsen, and R. Glück. 2011. Towards a reversible functional language. In *International Workshop on Reversible Computation*. Springer, 14–29.
[57] T. Yokoyama and R. Glück. 2007. A reversible programming language and its invertible self-interpreter. In *Proceedings of the 2007 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*. 144–153.

Received 2024-02-28; accepted 2024-06-18