Quantum Information Effects

Quantum Information Effects

CHRIS HEUNEN, University of Edinburgh, United Kingdom
ROBIN KAARSGAARD, University of Edinburgh, United Kingdom

We study the two dual quantum information effects to manipulate the amount of information in quantum computation: hiding and allocation. The resulting type-and-effect system is fully expressive for irreversible quantum computing, including measurement. We provide universal categorical constructions that semantically interpret this arrow metalanguage with choice, starting with any rig groupoid interpreting the reversible base language. Several properties of quantum measurement follow in general, and we translate (noniterative) quantum flow charts into our language. The semantic constructions turn the category of unitaries between Hilbert spaces into the category of completely positive trace-preserving maps, and they turn the category of bijections between finite sets into the category of functions with chosen garbage. Thus they capture the fundamental theorems of classical and quantum reversible computing of Toffoli and Stinespring.

CCS Concepts: • **Theory of computation → Quantum computation theory**; **Categorical semantics**.

Additional Key Words and Phrases: quantum computation, reversible computation, information effects, measurement, effects, arrows, categorical semantics

ACM Reference Format:
Chris Heunen and Robin Kaarsgaard. 2022. Quantum Information Effects. *Proc. ACM Program. Lang.* 6, POPL, Article 2 (January 2022), 27 pages. https://doi.org/10.1145/3498663

ACKNOWLEDGMENTS

This material is based upon work supported by the Engineering and Physical Sciences Research Council Fellowship No. EP/R044759/1, and the Independent Research Fund Denmark under DFF-International Postdoc Fellowship No. 0131-00025B. We thank Pablo Andrés-Martínez and the anonymous reviewers for their comments, corrections, and suggestions for this paper, and Martti Karvonen for early discussions.

1 INTRODUCTION

Something is rotten in the state of quantum computing. It subsumes classical computing, which is generally irreversible, yet it is most often formulated as a reversible quantum circuit, with an irreversible quantum measurement as an afterthought. The conceptual status of this irreversible measurement remains mysterious. This is known as the measurement problem.

Classical computing itself is most often formulated as composed of irreversible operations. However, by the seminal works of Toffoli [Toffoli 1980] and Bennett [Bennett 1973], and more recently by James and Sabry [James and Sabry 2012], we know that it can also be phrased in terms of reversible operations, as long as we consider systems to be open and interact with an environment that is eventually disregarded. This final part is important, as reversible computations alone (be they classical or quantum) cannot change the *amount* of information (as measured by an

Authors' addresses: Chris Heunen, School of Informatics, University of Edinburgh, 10 Crichton Street, Edinburgh, EH8 9AB, United Kingdom, chris.heunen@ed.ac.uk; Robin Kaarsgaard, School of Informatics, University of Edinburgh, 10 Crichton Street, Edinburgh, EH8 9AB, United Kingdom, robin.kaarsgaard@ed.ac.uk.

2475-1421/2022/1-ART2
https://doi.org/10.1145/3498663

appropriate notion of *entropy*). To understand the nature of quantum measurement in computation requires us to close two conceptual gaps:

(i) from reversible classical computing to reversible quantum computing; and
(ii) from reversible quantum computing to irreversible quantum computing.

It may seem that much has to be added to a reversible language to make it suitable for quantum computing. Similarly, it may seem that much less can be expressed in purely reversible quantum computations than in arbitrary quantum computations with measurements. We argue, however, that both gaps are smaller than they may appear.

To do this, we start with the reversible combinator language Π, which governs classical reversible computation, and extend it with combinators for quantum phases and the quantum *Hadamard gate*. We call the result $\mathcal{U}\Pi$ ("yuppie"), because it is already approximately universal for reversible quantum computing with unitary gates.

To address (ii) we introduce two *quantum information effects* – computational effects that manipulate the amount of information – through two arrows [Hughes 2005]. The first computational effect allows *allocation* of auxiliary space on a hidden heap, leading to the arrow metalanguage $\mathcal{U}\Pi_a$ ("yuppie-a"). This calculus is approximately universal for quantum computing with *isometries* rather than unitaries. The second computational effect dually allows *hiding*, by disregarding specifically marked *garbage output*, leading to $\mathcal{U}\Pi_a^\chi$ ("yuppie-chi-a"). We prove that this calculus is approximately universal for *arbitrary quantum computations*, including measurement.

Thus we have an arrow metalanguage that, with two simple computational effects on top of a pure reversible model, is fully expressive for irreversible quantum computing. All the allocation and hiding is tracked by the type system, and so allows us to compile an irreversible quantum program into an explicit reversible quantum circuit.

We provide matching categorical semantics via surprisingly simple concrete constructions that have very general universal properties. Vanilla Π may be interpreted in rig categories: categories with two monoidal structures $(\otimes, \oplus)$, where the product $(\otimes)$ distributes over the sum $(\oplus)$. We will interpret $\mathcal{U}\Pi$ in the category **Unitary** of Hilbert spaces and unitaries. This is a choice of canonical model: all that is needed is a rig category with morphisms to interpret phase gates and the Hadamard gate (which we will see is equivalent to having a notion of *superposition*).

For $\mathcal{U}\Pi_a$ we provide a free construction that turns a rig category $\mathbf{C}$ into a new one $R[\mathbf{C}]$ where the unit for the sum becomes initial. Then $R[\mathbf{Unitary}]$ is the category **Isometry** of Hilbert spaces and isometries. Dually, we interpret $\mathcal{U}\Pi_a^\chi$ via a free construction L making a monoidal unit terminal. Now $L[\mathbf{Isometry}]$ is the category **CPTP** of arbitrary (irreversible) quantum channels. Classically, R transforms the category of bijections between sets into that of injections, and in turn L transforms that into arbitrary functions with chosen garbage. This lets us reformulate Toffoli's *fundamental theorem of reversible computing* [Toffoli 1980] as a purely categorical statement.

Surprising mileage is obtained from these simple constructions, as we prove in general several properties of measurement that can be expressed entirely as semantic equivalences between program fragments. For example, we show that measurement commutes with injections and projections, and that measurement is idempotent. More generally, we show that our setting can interpret (noniterative) *quantum flow charts* [Selinger 2004]. This highlights the potential of $\mathcal{U}\Pi_a^\chi$ not only as a vehicle for theoretical studies, here the study of measurement, but also as an intermediate language with a strong equational theory owing to its categorical semantics. All constructions and translations in this paper are formalised in (heavily extended) Glasgow Haskell.

Related Work. Classical information effects are due to [James and Sabry 2012], while the characterisation of **Isometry** and **CPTP** as successive completions of **Unitary** was first given by [Huot and Staton 2018, 2019] (see also [Heunen and Kaarsgaard 2021]). Quantum programming languages are

an active research topic [Bichsel et al. 2020; Green et al. 2013; Paykin et al. 2017; Sabry et al. 2018; Selinger 2004]. In particular, quantum measurement has been studied extensively as a computational effect [Altenkirch and Green 2010; Green and Altenkirch 2008; Vizzotto et al. 2006, 2009; Westerbaan 2017] too. While proven practically useful, the precise meaning of *measurement-as-an-effect* has remained unclear, perhaps partly because of the wide-spread conscription to the view "quantum data, classical control" [Selinger 2004].

We generalise this previous work by providing the missing origin story of measurement-as-an-effect as a sequence of arrow constructions (semantically, categorical completions) that can be applied (and given precise meaning) to *any* rig groupoid. In other words, where previous work shows *that* measurement arises in this way, using facts specific to quantum theory, we show *how*, using only the language of rig categories. With measurement thus defined in the abstract, our categorical constructions eliminate the need for involved functional-analytic semantics using operator algebras [Cho and Westerbaan 2016; Péchoux et al. 2020; Rennela and Staton 2020].

The $R[-]$-construction on small categories was studied as a special case of freely adjoining monoidal indeterminates in [Hermida and Tennent 2012]. An unquotiented, bicategorical version of this construction was also given here, later rediscovered in [Cruttwell et al. 2021; Fong et al. 2019] as the Para-construction. Though quotienting will be important here, a bicategorical construction could see use to study the structure and order of dilations (see also [Houghton-Larsen 2021]).

Overview. Section 2 recalls background material. Next, Section 3 discusses Π, introduces the languages $\mathcal{U}\Pi$, $\mathcal{U}\Pi_a$, and $\mathcal{U}\Pi_a^\chi$, and proves expressivity theorems. Section 4 deals with categorical semantics: it recalls rig groupoids, introduces the L and R constructions, and proves that they respect $\otimes$ and give the appropriate setting to interpret $\oplus$ as an arrow with choice. Section 5 concerns universal properties of the L and R constructions, and shows that they encompass two fundamental results: Toffoli's fundamental theorem, and Stinespring's dilation theorem. In Section 6 we derive extra properties in the arrow metalanguage, generically in $L[R[\mathbf{C}]]$. Finally, Section 7 concludes and lists interesting directions for future work.

2 BACKGROUND

This section recalls the basics of quantum theory, monoidal categories, and information effects.

2.1 Quantum Theory

For more details we refer to [Nielsen and Chuang 2002; Yanofsky and Mannucci 2008].

2.1.1 Pure Quantum Theory and Bra-Ket Notation. A quantum system is captured by a complex Hilbert space H. For example, qubits are modelled by $H = \mathbb{C}^2$. The *pure states* ψ of H are the vectors of unit norm: $\|\psi\| = 1$. By convention, vectors are denoted as a *ket* $|\psi\rangle$. This is handy, because then the functional $H \to \mathbb{C}$ that maps ϕ to the inner product $\langle\psi|\phi\rangle$ can be denoted as the *bra* $\langle\psi|$. As a further consequence, the outer product of ϕ with ψ (with signature $H \to H$) becomes $|\phi\rangle\langle\psi|$.

Pure dynamics of a quantum system are reversible. Evolution is given by a *unitary* linear map $U : H \to H$, meaning that U is a bijection that is *isometric*: $\langle U\phi|U\psi\rangle = \langle\phi|\psi\rangle$. More generally, any continuous linear function $f : H \to K$ has an *adjoint* $f^\dagger : K \to H$ satisfying $\langle f\phi|\psi\rangle = \langle\phi|f^\dagger\psi\rangle$. An isometry then satisfies $f^\dagger \circ f = \mathrm{id}$, and a unitary furthermore satisfies $f \circ f^\dagger = \mathrm{id}$.

Pure quantum theory subsumes reversible classical computation. Any finite set I generates a Hilbert space $\mathbb{C}^I$ of linear combinations of elements of I. Thus $\{|i\rangle \mid i \in I\}$ forms a basis of $\mathbb{C}^I$ that is moreover orthonormal: $\langle i|j\rangle$ is 1 when $i = j$ and vanishes otherwise. We call this basis of $\mathbb{C}^n$ induced by the set $\{1, 2, \ldots, n\}$ the *computational basis*. Any bijection of $\{1, 2, \ldots, n\}$ induces a unitary on $\mathbb{C}^n$ that preserves the computational basis.

If two quantum systems are modelled by Hilbert spaces H and K, the compound system is given by their tensor product $H \otimes K$. For example, a 3-qubit system is modelled by $\mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \mathbb{C}^2 \simeq \mathbb{C}^8$. Similarly, if H and K evolve along unitaries U and V, then $H \otimes K$ evolves along $U \otimes V$.

2.1.2 Mixed Quantum Theory. A quantum computation (in the quantum circuit model) consists of a composition of tensor products of unitary gates, which is entirely reversible. However, reading out the result of the computation requires a measurement, which is an irreversible operation. The standard model therefore considers *mixed states*. These are given by a *density matrix*, which is a linear function $\rho : H \to H$ such that $0 \leq \langle \rho(\psi) | \psi \rangle \leq 1$ for all $|\psi\rangle$. Thus any pure state $|\psi\rangle \in H$ is also a mixed state $|\psi\rangle\langle\psi| : H \to H$.

Mixed states no longer have reversible dynamics. Any unitary $U : H \to H$ still induces a map that takes a mixed state ρ to a mixed state $U^\dagger \circ \rho \circ U$. But now the appropriate dynamics allow more possibilities, generally given by so-called *completely positive trace-preserving (CPTP)* maps, also known as *quantum channels*. It is not important here to set out their definition. What is important is *Stinespring's dilation theorem*, which says that any CPTP map $H \to K$ may be factored as a pure evolution $H \to K \otimes G$, given by $\rho \mapsto V^\dagger \circ \rho \circ V$ for an isometry V, followed by a map $K \otimes G \to K$. That is, irreversible (mixed) quantum theory is contained within reversible (pure) quantum theory, as long as you allow an environment to play the role of auxiliary state space but disregard it.

2.1.3 Superposition and Measurement. Superposition is the ability of a quantum state $|\rho\rangle$ to occupy several classical states $|b_1\rangle \ldots |b_n\rangle$ at once, so long as no measurement occurs. Each classical state in a superposed state is weighted by a complex number α_i known as an *amplitude*. Once a system in superposition $|\psi\rangle = \sum_{i=1}^n \alpha_i |b_i\rangle$ is measured, it collapses to one of the classical states $|b_k\rangle$. The outcome of such a measurement is probabilistic, with the probability of observing $|b_i\rangle$ given by $|\langle b_i|\psi\rangle|^2$; this is called the *Born rule*. Using density matrices, measurement with respect to $\{|b_i\rangle\}_{i \in I}$ is represented by the *measurement instrument channel*

$$\rho \mapsto \sum_{i \in I} |b_i\rangle\langle b_i| \, \rho \, |b_i\rangle\langle b_i|$$

that sends quantum states to their post-measurement (mixed classical) states. For example, measuring a qubit $|\psi\rangle\langle\psi|$ for $|\psi\rangle = \alpha |0\rangle + \beta |1\rangle$ with respect to $\{|0\rangle, |1\rangle\}$ results in the mixed state $|\alpha|^2 |0\rangle\langle 0| + |\beta|^2 |1\rangle\langle 1|$.

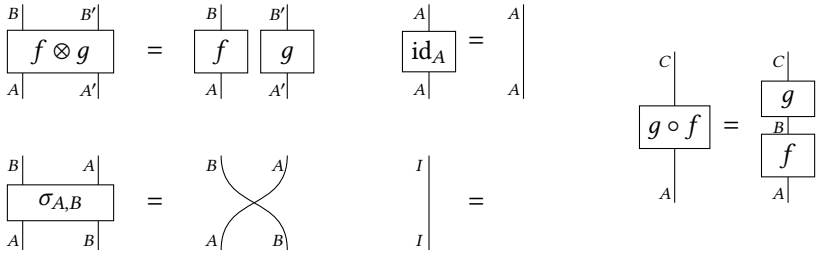
2.1.4 Global and Relative Phase. Recall that the complex conjugate of a complex number $\varphi = a + bi$ is $\overline{\varphi} = a - bi$. A *phase* is a complex number satisfying $\varphi \cdot \overline{\varphi} = 1$; equivalently, φ has norm 1. Quantum states that differ only by a *global phase*, $|\psi'\rangle = \varphi |\psi\rangle$, are indistinguishable, in that they have the same measurement statistics. But the phase difference between *parts* of states can be incredibly important. For example, the states $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ differ only by the phase -1 in their amplitude for $|1\rangle$. This difference in *relative phase* makes $|+\rangle$ and $|-\rangle$ orthogonal.

2.2 Monoidal Categories

For our semantics, we will assume that the reader is familiar with the basic notions of categories and functors [Leinster 2016]. To set notation, recall that a category $\mathbf{C}$ is symmetric monoidal when it comes equipped with a tensor product $\otimes : \mathbf{C} \times \mathbf{C} \to \mathbf{C}$, a unit object I, isomorphisms $\lambda_A : I \otimes A \to A$ and $\rho_A : A \otimes I \to A$ called unitors, isomorphisms $\alpha_{A,B,C} : A \otimes (B \otimes C) \to (A \otimes B) \otimes C$ called associators, and isomorphisms $\sigma_{A,B} : A \otimes B \to B \otimes A$ called symmetries for all objects A, B, C, that satisfy certain coherence laws [Heunen and Vicary 2019].

A category lets one compose morphisms 'in sequence'; a monoidal category additionally lets one compose morphisms 'in parallel'. This is expressed satisfyingly in the graphical calculus for

monoidal categories. We draw a morphism as a box with an incoming wire labelled by its domain and an outgoing wire labelled by its codomain. Composition becomes stacking boxes vertically, whereas we draw the tensor product of boxes side by side. In particular, objects $A \otimes B$ may be drawn as a single wire labelled $A \otimes B$, or as two parallel wires labelled by A and B, and the nullary case of a wire labelled I is simply not drawn. The special morphisms $\sigma_{A,B}$ are drawn as crossing two wires. The coherence laws simply say that one may ignore the coherence isomorphisms graphically.



Of special interested are symmetric monoidal categories whose tensor unit is initial or terminal. If there is a unique morphism $A \longrightarrow I$ for any object A, the monoidal category is called *affine*, and if there is a unique morphism $I \longrightarrow A$ for any object A, it is called *coaffine*.

A functor $F \colon \mathbf{C} \longrightarrow \mathbf{D}$ between symmetric monoidal categories is (strong) monoidal when it is equipped with isomorphisms $F(A) \otimes F(B) \simeq F(A \otimes B)$ and $F(I) \simeq I$ that respect the coherence isomorphisms of $\mathbf{C}$ and $\mathbf{D}$. It is *strict* monoidal when these isomorphisms are in fact identities. Monoidal functors between (co)affine categories automatically preserve the terminal (initial) object.

2.3 Information Effects

Classical computation, embodied by functions on finite sets, is irreversible, because applying a function in general loses information. This can be made precise via the Shannon entropy $H = -\sum p_i \log p_i$ that measures how surprising it is when a variable takes the value i with probability p_i. The functions that preserve Shannon entropy are precisely bijections. This direct connection between information preservation and reversibility is a consequence of *Landauer's principle* [Landauer 1961].

The central idea of information effects [James and Sabry 2012] is that this irreversible model of computation arises from a reversible (bijective) model of computation, together with computational effects for duplicating and erasing information. Thus irreversible programs are reversible instructions governed by an arrow metalanguage that tracks interaction with a global environment.

Quantum theory, embodied by quantum channels between finite-dimensional Hilbert spaces, is also irreversible. The information content of a quantum state ρ can be made precise by von Neumann entropy $S = -\mathrm{tr}(\rho \log \rho)$. In this case, the information-preserving maps are also the reversible ones: those of the form $\rho \mapsto U^\dagger \circ \rho \circ U$ for unitary U. This mirrors the classical connection between information preservation and reversibility.

3 THREE GENERATIONS OF YUPPIE

Π is a reversible combinator language introduced in [Bowman et al. 2011; James and Sabry 2012] to study strongly typed reversible *classical* programming. Many extensions exist, such as partiality and iteration [Bowman et al. 2011; James and Sabry 2012], fractional types [Chen et al. 2020; Chen and Sabry 2021], negative types [Chen and Sabry 2021], and higher combinators [Carette and Sabry 2016; Kaarsgaard and Veltri 2019]. This section introduces a quantum extension to Π, and shows it to be approximately universal for unitaries, the canonical model of pure quantum computation (without measurement). We then use two arrow constructions to extend this with the quantum

Syntax

$$
\begin{aligned}
b &::= 0 \mid 1 \mid b + b \mid b \times b && \text{(base types)}\\
t &::= b \leftrightarrow b && \text{(combinator types)}\\
a &::= id \mid swap^{+} \mid unit^{+} \mid uniti^{+} \mid assoc^{+} \mid associ^{+}\\
&\quad \mid swap^{\times} \mid unit^{\times} \mid uniti^{\times} \mid assoc^{\times} \mid associ^{\times}\\
&\quad \mid distrib \mid distribi \mid distribo \mid distriboi && \text{(primitive combinators)}\\
d &::= midswap^{+} \mid midswap^{\times} && \text{(derived combinators)}\\
c &::= a \mid c \fatsemi c \mid c + c \mid c \times c && \text{(combinators)}
\end{aligned}
$$

Typing rules

$$
\begin{array}{ccccc}
id & : & b \leftrightarrow b & : & id\\
swap^{+} & : & b_1 + b_2 \leftrightarrow b_2 + b_1 & : & swap^{+}\\
unit^{+} & : & b + 0 \leftrightarrow b & : & uniti^{+}\\
assoc^{+} & : & (b_1 + b_2) + b_3 \leftrightarrow b_1 + (b_2 + b_3) & : & associ^{+}\\
swap^{\times} & : & b_1 \times b_2 \leftrightarrow b_2 \times b_1 & : & swap^{\times}\\
unit^{\times} & : & b \times 1 \leftrightarrow b & : & uniti^{\times}\\
assoc^{\times} & : & (b_1 \times b_2) \times b_3 \leftrightarrow b_1 \times (b_2 \times b_3) & : & associ^{\times}\\
distrib & : & b_1 \times (b_2 + b_3) \leftrightarrow (b_1 \times b_2) + (b_1 \times b_3) & : & distribi\\
distribo & : & b \times 0 \leftrightarrow 0 & : & distriboi\\
midswap^{+} & : & (b_1 + b_2) + (b_3 + b_4) \leftrightarrow (b_1 + b_3) + (b_2 + b_4) & : & midswap^{+}\\
midswap^{\times} & : & (b_1 \times b_2) \times (b_3 \times b_4) \leftrightarrow (b_1 \times b_3) \times (b_2 \times b_4) & : & midswap^{\times}
\end{array}
$$

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

Fig. 1. The syntax and type system of Π.

information effects of *allocation* and *hiding* to arrive at an arrow metalanguage which we prove approximately universal for quantum channels, the canonical model of full quantum computation (with measurement).

3.1 Reversible Classical Combinators: Π

The syntax and type system of the unextended calculus Π is shown in Fig. 1. It comprises a small set of invertible, first-order, strongly typed polymorphic combinators on data constructed from (classical) sum and products types, as well as their units 0 and 1. These combinators enable data of sum and product type to be swapped (sending inl x to inr x and vice versa for sums, and (x, y) to (y, x) for products), reassociated, and have their respective units added and removed in the usual way. Products can also be distributed over sums (and back again) as usual. Finally, these combinators can be composed in sequence $c_1 \fatsemi c_2$ and in parallel using both + and ×. That is, $c_1 \times c_2$ takes a pair (x, y) and produces the pair $(c_1\, x, c_2\, x)$, while $c_1 + c_2$ takes inl x to inl $(c_1 x)$ and inr y to inr $(c_2 y)$.

Aside from the base combinators, a pair of useful derived combinators $midswap^{+} : (b_1 + b_2) + (b_3 + b_4) \leftrightarrow (b_1 + b_3) + (b_2 + b_4)$ and $midswap^{\times} : (b_1 \times b_2) \times (b_3 \times b_4) \leftrightarrow (b_1 \times b_3) \times (b_2 \times b_4)$ can

$$
\begin{array}{rcl@{\qquad}rcl}
inv(id) & = & id & inv(c_1 \fatsemi c_2) & = & inv(c_2) \fatsemi inv(c_1) \\
inv(c_1 + c_2) & = & inv(c_1) + inv(c_2) & inv(c_1 \times c_2) & = & inv(c_1) \times inv(c_2) \\
inv(swap^+) & = & swap^+ & inv(swap^\times) & = & swap^\times \\
inv(unit^+) & = & uniti^+ & inv(uniti^+) & = & unit^+ \\
inv(assoc^+) & = & associ^+ & inv(associ^+) & = & assoc^+ \\
inv(unit^\times) & = & uniti^\times & inv(uniti^\times) & = & unit^\times \\
inv(assoc^\times) & = & associ^\times & inv(associ^\times) & = & assoc^\times \\
inv(distrib) & = & distribi & inv(distribi) & = & distrib \\
inv(distribo) & = & distriboi & inv(distriboi) & = & distribo \\
inv(phase_\varphi) & = & phase_{\overline{\varphi}} & inv(hadamard) & = & hadamard
\end{array}
$$

Fig. 2. The inversion meta-combinator *inv* in $(\mathcal{U})\Pi$.

Syntax

$$
\begin{array}{lr}
a ::= \cdots \mid phase_\varphi \mid hadamard & \text{(primitive combinators)} \\
d ::= \cdots \mid superposition & \text{(derived combinators)}
\end{array}
$$

Typing rules

$$
\begin{array}{ccccc}
phase_\varphi & : & 1 \leftrightarrow 1 & : & phase_{\overline{\varphi}} \\
hadamard & : & 1 + 1 \leftrightarrow 1 + 1 & : & hadamard \\
superposition & : & b + b \leftrightarrow b + b & : & superposition
\end{array}
$$

Fig. 3. The syntax and typing rules of $\mathcal{U}\Pi$ in addition to those in Π (see Fig. 1).

be defined as

$$
\begin{aligned}
midswap^+ &= assoc^+ \fatsemi (id + associ^+) \fatsemi (id + (swap^+ + id)) \fatsemi (id + assoc^+) \fatsemi associ^+ \\
midswap^\times &= assoc^\times \fatsemi (id \times associ^\times) \fatsemi (id \times (swap^\times \times id)) \fatsemi (id \times assoc^\times) \fatsemi associ^\times \quad .
\end{aligned}
$$

The definition and use of derived combinators, which we will make heavy use of, should be taken as no more than aliasing, or macro definition and expansion. Recursive systems (mutually or otherwise) of derived combinators are *not* permitted.

Π takes semantics in *rig groupoids* (see Section 4), the canonical choice being the category **FinBij** of finite sets and bijective functions. Indeed, Π is universal for finite bijective functions; Fig. 4 shows the implementations of the universal gate set $\{PX, CNOT, TOFFOLI\}$ [Toffoli 1980].

Inversion. Our presentation of Π differs slightly from [James and Sabry 2012]: our syntax does not include an inversion combinator *inv c*; instead we derive it as a metacombinator (in Fig. 2). This avoids some superfluous syntax – e.g., $inv(c_1 + c_2)$ and $(inv\ c_1) + (inv\ c_2)$ are equivalent, as are $inv(id)$ and id – but results in a higher number of base combinators. Some basic well-behavedness properties can be straightforwardly shown by induction, summarised as follows.

Proposition 1. *Let c be a* $(\mathcal{U})\Pi$ *combinator. Then:*

(i) $c : b_1 \leftrightarrow b_2$ *implies* $inv(c) : b_2 \leftrightarrow b_1$, *and*
(ii) $inv(inv(c)) = c$.

3.2 Reversible Quantum Combinators: $\mathcal{U}\Pi$

$\mathcal{U}\Pi$ ("yuppie") extends Π with notions of phase and superposition, in the form of the $phase_\varphi$ and *hadamard* combinators. Fig. 3 shows the syntax and types of this small extension. While the types

of $\mathcal{U}\Pi$ remain the same as in Π, in $\mathcal{U}\Pi$ they are entirely quantum rather than (as in Π) entirely classical. For example, where $1 + 1$ in Π is the type of *bits*, in $\mathcal{U}\Pi$ it is the type of *qubits* (with no way of forming the type of bits). $\mathcal{U}\Pi$ canonically takes semantics in the category **Unitary** of finite dimensional Hilbert spaces and unitaries. The full treatment of these semantics is given in Section 4, but later in this section we will show that $\mathcal{U}\Pi$ is *approximately universal* for unitaries.

Phases correspond with unitaries $\mathbb{C} \to \mathbb{C}$. Since $\mathbb{C}$ is the tensor unit in **Unitary**, we can express an arbitrary phase φ through the combinator $phase_\varphi : 1 \leftrightarrow 1$. This will allow us to form quantum *phase gates* like S and T, and to multiply a combinator $c : b_1 \leftrightarrow b_2$ by an arbitary phase φ as:

$$\varphi \bullet c = uniti^{\times} \fatsemi c \times phase_\varphi \fatsemi unit^{\times}$$

We include *all* phases, yielding an uncountable number of phase combinators, even though a finite number of phases suffice for approximate universality up to a global phase. We find including all of them to be the more principled solution, especially when an appropriate finite subset (such as $\{\pm i, \pm 1, \pm\cos\left(\frac{\pi}{4}\right) \pm i\sin\left(\frac{\pi}{4}\right)\}$) can be chosen in a concrete implementation without detriment.

Superpositions are introduced by means of the *hadamard* combinator (of type $1 + 1 \leftrightarrow 1 + 1$, or $Qbit \leftrightarrow Qbit$), named after the Hadamard gate from which it takes its semantics:

$$|0\rangle \mapsto \tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \qquad\qquad |1\rangle \mapsto \tfrac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

It introduces uniform superpositions of states in the computational basis. Though effective, it can be argued that this combinator is not conceptually clean: all of the Π combinators are parametrically polymorphic and pertain to *structure* rather than *behaviour*, but *hadamard* is monomorphic, and pertains specifically to the behaviour of qubits. To mend this, we could instead introduce a parametrically polymorphic combinator $superposition : b + b \leftrightarrow b + b$ with semantics:

$$\mathrm{inl}(|\psi\rangle) \mapsto \tfrac{1}{\sqrt{2}}(\mathrm{inl}(|\psi\rangle) + \mathrm{inr}(|\psi\rangle)) \qquad\qquad \mathrm{inr}(|\psi\rangle) \mapsto \tfrac{1}{\sqrt{2}}(\mathrm{inl}(|\psi\rangle) - \mathrm{inr}(|\psi\rangle))$$

Now *hadamard* is just the *superposition* combinator on the type $1 + 1 \leftrightarrow 1 + 1$. Interestingly, the two are equivalent in the presence of the other combinators, as *superposition* can be derived:

$$superposition = (uniti^{\times} + uniti^{\times}) \fatsemi distribi \fatsemi (id \times hadamard) \fatsemi distrib \fatsemi (unit^{\times} + unit^{\times})$$

It also follows from this definition that *superposition*, like *hadamard*, is self-inverse. Whether the *hadamard* or the *superposition* combinator is taken as primal thus comes down to preference; there is no difference in expressivity, and one is easily derived from the other.

Expressiveness. We have taken an established combinator calculus for reversible *classical* computing, and extended it only slightly with two quantum combinators modelling phase and superposition. One may ask whether this extension is sufficient to express all of reversible *quantum* computing. This question contains a number of subtleties, not least because there are systems of quantum computing which do include concepts of both phase and superposition, but can nevertheless be efficiently simulated by purely classical means (e.g., the Clifford gate set *without* T).

Fig. 4 shows the implementation of a variety of reversible quantum (and classical) gates in $\mathcal{U}\Pi$. Note the meta-combinator *ctrl*, which produces a combinator for the usual *controlled gate* for a combinator corresponding to a gate c. Briefly, *ctrl* c takes $(|0\rangle, |\psi\rangle)$ to $(|0\rangle, |\psi\rangle)$, and $(|1\rangle, |\psi\rangle)$ to $(|1\rangle, c(|\psi\rangle))$. Though $\mathcal{U}\Pi$ cannot distinguish at the type level between classical and quantum data, it is computationally universal for reversible classical computing, with $\{px, cnot, toffoli\}$ as one example of a universal gate set [Toffoli 1980].

Returning to the quantum case, using these representations of quantum gates, it can be shown that $\mathcal{U}\Pi$ is *approximately universal* for reversible quantum computing as well: it can approximate any unitary (on a space of dimension 2^n) to arbitrary precision (measured by the operator norm).

$$
\begin{array}{rcl}
px & : & Qbit \leftrightarrow Qbit \\
px & = & swap^{+}
\end{array}
\qquad
\begin{array}{rcl}
py & : & Qbit \leftrightarrow Qbit \\
py & = & swap^{+} \fatsemi (phase_{-i} + phase_{i})
\end{array}
\qquad
\begin{array}{rcl}
pz & : & Qbit \leftrightarrow Qbit \\
pz & = & id + phase_{-1}
\end{array}
$$

$$
\begin{array}{rcl}
s & : & Qbit \leftrightarrow Qbit \\
s & = & id + phase_{i}
\end{array}
\qquad
\begin{array}{rcl}
t & : & Qbit \leftrightarrow Qbit \\
t & = & id + phase_{e^{\frac{i\pi}{4}}}
\end{array}
$$

$$
\begin{array}{rcl}
ctrl\ c & : & b \leftrightarrow b \rightarrow Qbit \times b \leftrightarrow Qbit \times b \\
ctrl\ c & = & swap^{\times} \fatsemi distrib \fatsemi (unit^{\times} + unit^{\times}) \fatsemi (id + c) \fatsemi (uniti^{\times} + uniti^{\times}) \fatsemi distribi \fatsemi swap^{\times}
\end{array}
$$

$$
\begin{array}{rcl}
cnot & : & Qbit^2 \leftrightarrow Qbit^2 \\
cnot & = & ctrl\ px
\end{array}
\qquad
\begin{array}{rcl}
toffoli & : & Qbit^3 \leftrightarrow Qbit^3 \\
toffoli & = & ctrl\ cnot
\end{array}
\qquad
\begin{array}{rcl}
fredkin & : & Qbit^3 \leftrightarrow Qbit^3 \\
fredkin & = & ctrl\ swap^{\times}
\end{array}
$$

Fig. 4. The implementation of a variety of quantum gates in $\mathcal{U}\Pi$. We use $Qbit^n$ as shorthand for the n-fold product of the qubit type $Qbit = 1 + 1$ with itself.

Theorem 2. *$\mathcal{U}\Pi$ is approximately universal for unitaries: For any $2^n \times 2^n$ unitary U and $\delta > 0$ there exists a $\mathcal{U}\Pi$ combinator u such that $\left\| U - [\![u]\!] \right\|_{\text{op}} < \delta$.*

3.3 Quantum Combinators with Allocation: $\mathcal{U}\Pi_a$

Next we extend $\mathcal{U}\Pi$ with an *allocation effect* $alloc : 0 \rightarrowtail a$, yielding the language of $\mathcal{U}\Pi_a$ (*"yuppie-a"*). This effect is introduced by letting combinators $b_1 \rightarrowtail b_2$ in $\mathcal{U}\Pi_a$ be given by invertible combinators with a *heap* of type b_3: that is, as $\mathcal{U}\Pi$ combinators of type $b_1 + b_3 \leftrightarrow b_2$. Analogous to [James and Sabry 2012], this enables the type system to track the information effects.

This small extension will allow us to define a *classical cloning* combinator that clones classical states exactly, and sends quantum states $|\psi\rangle$ to $\sqrt{|\psi\rangle} \otimes \sqrt{|\psi\rangle}$; this will be crucial later on in deriving a combinator for measurement in $\mathcal{U}\Pi_a^{\chi}$.

$\mathcal{U}\Pi_a$ canonically takes semantics in the category **Isometry** of Hilbert spaces and isometries: in Section 4, we will see how a categorical model of $\mathcal{U}\Pi$ can be extended universally to model of $\mathcal{U}\Pi_a$, and Section 5 shows how this construction connects the canonical model of $\mathcal{U}\Pi$ to that of $\mathcal{U}\Pi_a$. Now we show that the approximate universality theorem for $\mathcal{U}\Pi$ with its unitary semantics extends to an approximate universality for $\mathcal{U}\Pi_a$ with its semantics in isometries.

Fig. 5 gives an over view of $\mathcal{U}\Pi_a$. It is an arrow metalanguage [Hughes 2005; James and Sabry 2012; Power and Robinson 1997] built atop $\mathcal{U}\Pi$: it has the same base types as $\mathcal{U}\Pi$, but introduces a new, irreversible combinator type $b \rightarrowtail b$ (reflecting the fact that combinators in $\mathcal{U}\Pi_a$ are no longer invertible). All combinators in $\mathcal{U}\Pi_a$ are constructed from combinators in $\mathcal{U}\Pi$ by means of the single primitive *lift* combinator, following the type rule:

$$\frac{u : b_1 + b_3 \leftrightarrow b_2}{lift\ u : b_1 \rightarrowtail b_2}$$

So a combinator in $\mathcal{U}\Pi_a$ corresponds to a combinator in $\mathcal{U}\Pi$ with a hidden *heap* of type b_3. Section 4 will discuss that some quotienting is needed for this construction to behave; we defer further details about the semantics until then, including the arrow laws.

For this to constitute an arrow, we must produce meta-combinators *arr*, $\ggg$, and *first*. To start, *arr* must lift a $\mathcal{U}\Pi$ combinator u to a *pure* $\mathcal{U}\Pi_a$ one, free of effects. To do this, we assign it the trivial heap 0 and remove it before proceeding with u:

$$arr\ u = lift(unit^{+} \fatsemi u)$$

In Fig. 5, *iso* refers to atomic combinators of $\mathcal{U}\Pi$ brought into $\mathcal{U}\Pi_a$ by applying *arr* to them. We write, for example, $swap^{+}$ in $\mathcal{U}\Pi_a$ to refer to $arr(swap^{+})$, and so on.

Syntax

$$
\begin{aligned}
b &::= 0 \mid 1 \mid b + b \mid b \times b && \text{(base types)}\\
t &::= b \rightarrowtail b && \text{(combinator types)}\\
c &::= \mathit{lift}\ u && \text{(primitive combinators)}\\
d &::= \mathit{iso} \mid \mathit{arr}\ u \mid c \ggg c \mid \mathit{first}\ c \mid \mathit{second}\ c \mid \mathit{left}\ c \mid \mathit{right}\ c\\
&\quad \mid c \mathbin{***} c \mid c \mathbin{+++} c \mid \mathit{inhab} \mid \mathit{inl} \mid \mathit{inr} \mid \mathit{alloc} \mid \mathit{clone} && \text{(derived combinators)}
\end{aligned}
$$

Typing rules

$$\frac{u : b_1 + b_3 \leftrightarrow b_2}{\mathit{lift}\ u : b_1 \rightarrowtail b_2} \qquad \frac{u : b_1 \leftrightarrow b_2}{\mathit{arr}\ u : b_1 \rightarrowtail b_2} \qquad \frac{c_1 : b_1 \rightarrowtail b_2 \quad c_2 : b_2 \rightarrowtail b_3}{c_1 \ggg c_2 : b_1 \rightarrowtail b_3}$$

$$\frac{c : b_1 \rightarrowtail b_2}{\mathit{first}\ c : b_1 \times b_3 \rightarrowtail b_2 \times b_3} \qquad \frac{c : b_1 \rightarrowtail b_2}{\mathit{second}\ c : b_3 \times b_1 \rightarrowtail b_3 \times b_2}$$

$$\frac{c : b_1 \rightarrowtail b_2}{\mathit{left}\ c : b_1 + b_3 \rightarrowtail b_2 + b_3} \qquad \frac{c : b_1 \rightarrowtail b_2}{\mathit{right}\ c : b_3 + b_1 \rightarrowtail b_3 + b_2}$$

$$\frac{c_1 : b_1 \rightarrowtail b_3 \quad c_2 : b_2 \rightarrowtail b_4}{c_1 \mathbin{+++} c_2 : b_1 + b_2 \rightarrowtail b_3 + b_4} \qquad \frac{c_1 : b_1 \rightarrowtail b_3 \quad c_2 : b_2 \rightarrowtail b_4}{c_1 \mathbin{***} c_2 : b_1 \times b_2 \rightarrowtail b_3 \times b_4} \qquad \frac{b \text{ inhabited}}{\mathit{inhab} : 1 \rightarrowtail b}$$

$$\frac{}{\mathit{alloc} : 0 \rightarrowtail a} \qquad \frac{}{\mathit{inl} : a \rightarrowtail a + b} \qquad \frac{}{\mathit{inr} : b \rightarrowtail a + b} \qquad \frac{}{\mathit{clone} : a \rightarrowtail a \times a}$$

$$\frac{}{1 \text{ inhabited}} \qquad \frac{b_1 \text{ inhabited} \quad b_2 \text{ inhabited}}{b_1 \times b_2 \text{ inhabited}} \qquad \frac{b_1 \text{ inhabited}}{b_1 + b_2 \text{ inhabited}} \qquad \frac{b_2 \text{ inhabited}}{b_1 + b_2 \text{ inhabited}}$$

Fig. 5. The syntax and type system of the arrow metalanguage $\mathcal{U}\Pi_a$.

To compose combinators $\mathit{lift}(u_1) : b_1 \rightarrowtail b_2$ and $\mathit{lift}(u_2) : b_2 \rightarrowtail b_3$ with heaps of type b_4 and b_4', we must track both heaps. The result will be a lifted $\mathcal{U}\Pi$ combinator of type $b_1 + (b_4 + b_4') \leftrightarrow b_3$ which permit u_1 and u_2 access to their parts of the heap accordingly:

$$\mathit{lift}(u_1) \ggg \mathit{lift}(u_2) = \mathit{lift}(\mathit{associ}^+ \fatsemi (u_1 + \mathit{id}) \fatsemi u_2)$$

The final combinator related to *lift* defining an arrow is *first*, allowing two arrows (each with their own information effects) to be executed in parallel. We define

$$\mathit{lift}(u_1) \mathbin{***} \mathit{lift}(u_2) = \mathit{lift}(\mathit{associ}^+ \fatsemi \mathit{distribi} + \mathit{distribi} \fatsemi \mathit{swap}^\times + \mathit{swap}^\times \fatsemi \mathit{distribi} \fatsemi \mathit{swap}^\times \fatsemi u_1 \times u_2)$$

and derive $\mathit{first}\ c = c \mathbin{***} \mathit{arr}(\mathit{id})$ and $\mathit{second}\ c = \mathit{arr}(\mathit{id}) \mathbin{***} c$ as usual. That is, given $\mathit{lift}(u_1)$ and $\mathit{lift}(u_2)$ with $u_1 : b_1 + b_3 \leftrightarrow b_2$ and $u_2 : b_1' + b_3' \leftrightarrow b_2'$, this defines their product by choosing the heap to be $(b_3 \times b_1') + ((b_1 \times b_2') + (b_3 \times b_3'))$, as we then have:

$$\begin{aligned}
(b_1 \times b_1') + ((b_1 \times b_3') + ((b_3 \times b_1') + (b_3 \times b_3'))) &\cong ((b_1 \times b_1') + (b_1 \times b_3')) + ((b_3 \times b_1') + (b_3 \times b_3'))\\
&\cong (b_1 \times (b_1' + b_3')) + (b_3 \times (b_1' + b_3'))\\
&\cong (b_1 + b_3) \times (b_1' + b_3')
\end{aligned}$$

Make *lift* an *arrow with choice* by defining a combinator $c_1 \mathbin{+++} c_2$ giving a choice between c_1 and c_2:

$$\mathit{lift}(u_1) \mathbin{+++} \mathit{lift}(u_2) = \mathit{lift}((u_1 + u_2) \fatsemi \mathit{midswap}^+)$$

From this we can derive $\mathit{left}\ c = c \mathbin{+++} \mathit{arr}(\mathit{id})$ and $\mathit{right}\ c = \mathit{arr}(\mathit{id}) \mathbin{+++} c$.

What can we do with this arrow metalanguage? Firstly, we can construct the promised allocation combinator $0 \rightarrowtail a$ by lifting the $\mathcal{U}\Pi$ map $0 + a \leftrightarrow a$ that removes the additive unit on the left, i.e.,

$alloc = lift(swap^+ \fatsemi unit^+)$. From this we can recover injections inl and inr as

$$inl = arr(uniti^+) \ggg right(alloc)$$

and analogously for inr, though we can also define them equivalently as $inl = lift(id)$ and $inr = lift(swap^+)$. Another crucial application of allocation is *classical cloning*.

3.3.1 Classical Cloning. Classical data can be copied, quantum data cannot. While there is a program that inputs a piece of classical data and outputs two copies of that data, no such program exists for quantum data; this is the *no cloning theorem* [Heunen and Vicary 2019; Nielsen and Chuang 2002].

In light of this, it may come as a bit of a surprise that we can derive a combinator $clone$ satisfying $[\![clone]\!](|0\rangle) = |00\rangle$ and $[\![clone]\!](|1\rangle) = |11\rangle$. After all, wouldn't that imply $[\![clone]\!](|\phi\rangle) = |\phi\rangle \otimes |\phi\rangle$ for a qubit $|\phi\rangle$? Fortunately not! To see this, consider a superposed state $|\phi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$.

$$\begin{aligned}[\![clone]\!](|\phi\rangle) &= [\![clone]\!](\alpha\,|0\rangle + \beta\,|1\rangle) = [\![clone]\!](\alpha\,|0\rangle) + [\![clone]\!](\beta\,|1\rangle) \\ &= \alpha\,[\![clone]\!](|0\rangle) + \beta\,[\![clone]\!](|1\rangle) = \alpha\,|00\rangle + \beta\,|11\rangle\end{aligned}$$

Now, $|00\rangle$ and $|11\rangle$ are shorthands for $|0\rangle \otimes |0\rangle$ and $|1\rangle \otimes |1\rangle$, and the tensor product of Hilbert spaces satisfies $s(|u\rangle \otimes |v\rangle) = (s\,|u\rangle) \otimes |v\rangle = |u\rangle \otimes (s\,|v\rangle)$ for all scalars s and vectors $|u\rangle \otimes |v\rangle$ in $U \otimes V$. This means for example that $\alpha\,|00\rangle = (\sqrt{\alpha}\,|0\rangle) \otimes (\sqrt{\alpha}\,|0\rangle)$, which in general is distinct from $(\alpha\,|0\rangle) \otimes (\alpha\,|0\rangle)$. So applying $clone$ to $|\phi\rangle$ does *not* gives two copies $|\phi\rangle \otimes |\phi\rangle$, but rather two copies of $|\phi\rangle$ with all amplitudes (in the computational basis) replaced by their *square roots*.

Define $clone : b \rightarrowtail b \times b$ by induction on the structure of b. The base cases are $clone_0 = arr(distriboi)$ and $clone_1 = arr(uniti^\times)$. Products are cloned inductively by rearranging:

$$clone_{b\times b'} = (clone_b \mathbin{***} clone'_b) \ggg arr(midswap^\times)$$

Sums are cloned inductively, tagging accordingly, and factoring:

$$clone_{b+b'} = (clone_b +\!\!+\!\!+ clone_{b'}) \ggg ((inl +\!\!+\!\!+ id) \mathbin{***} (inr +\!\!+\!\!+ id)) \ggg arr(distribi).$$

Interestingly, though the languages and semantics are different, cloning is defined *precisely* as for classical information effects [James and Sabry 2012].

3.3.2 Inhabitation. Later, we will need a notion of *inhabited types* in $\mathcal{U}\Pi_a$. By a type b being inhabited in $\mathcal{U}\Pi_a$, we mean that there is a combinator $1 \rightarrowtail b$. For inhabited types b, construct canonical inhabitants as follows. First, $inhab_1 = id$. The inhabitant of a product type $b \times b'$ is the product $inhab_{b\times b'} = uniti^\times \ggg inhab_b \mathbin{***} inhab_{b'}$ of inhabitants. Finally, a sum $b + b'$ is inhabited if either b or b' is: if b is inhabited set $inhab_{b+b'} = inhab_b \ggg inl$, and if b is not inhabited but b' is, $inhab_{b+b'} = inhab_{b'} \ggg inr$.

3.3.3 Expressiveness. We can now extend the expressiveness theorem for $\mathcal{U}\Pi$ to one for $\mathcal{U}\Pi_a$.

Theorem 3. *$\mathcal{U}\Pi_a$ is approximately universal for isometries: For any $2^n \times 2^m$ isometry V and $\delta > 0$ there exists a $\mathcal{U}\Pi_a$ combinator v such that $\left\|V - [\![v]\!]\right\|_{\mathrm{op}} < \delta$.*

3.4 Quantum Combinators with Hiding and Allocation: $\mathcal{U}\Pi_a^\chi$

We finally extend $\mathcal{U}\Pi_a$ with an additional information effect to *hide* information via a combinator $discard : b \rightsquigarrow 1$, giving us the language of $\mathcal{U}\Pi_a^\chi$ (*"yuppie-chi-a"*). Dually to how allocation was introduced in $\mathcal{U}\Pi_a$, discarding is introduced in $\mathcal{U}\Pi_a^\chi$ by letting combinators $b_1 \rightsquigarrow b_2$ be given by $\mathcal{U}\Pi_a$ combinators of type $b_1 \rightarrowtail b_2 \times b_3$, where we think of b_3 as the type of *garbage* produced by the combinator. In order to be able to produce a choice metacombinator, however, we need to

Syntax

$$
\begin{aligned}
b &::= 0 \mid 1 \mid b + b \mid b \times b && \text{(base types)}\\
t &::= b \rightsquigarrow b && \text{(combinator types)}\\
c &::= \mathit{lift}\ v && \text{(primitive combinators)}\\
d &::= \mathit{iso} \mid \mathit{arr}\ v \mid c \ggg c \mid \mathit{first}\ c \mid \mathit{second}\ c \mid \mathit{left}\ c \mid \mathit{right}\ c\\
&\quad \mid c \mathbin{***} c \mid c +\!\!+\!\!+ c \mid \mathit{inhab} \mid \mathit{inl} \mid \mathit{inr} \mid \mathit{alloc} \mid \mathit{clone}\\
&\quad \mid \mathit{discard} \mid \mathit{fst} \mid \mathit{snd} \mid \mathit{merge} \mid \mathit{measure} && \text{(derived combinators)}
\end{aligned}
$$

Typing rules

$$\frac{v : b_1 \rightarrowtail b_2 \times b_3 \quad b_3 \text{ inhabited}}{\mathit{lift}\ v : b_1 \rightsquigarrow b_2} \qquad \frac{v : b_1 \rightarrowtail b_2}{\mathit{arr}\ v : b_1 \rightsquigarrow b_2} \qquad \frac{c_1 : b_1 \rightsquigarrow b_2 \quad c_2 : b_2 \rightsquigarrow b_3}{c_1 \ggg c_2 : b_1 \rightsquigarrow b_3}$$

$$\frac{c : b_1 \rightsquigarrow b_2}{\mathit{first}\ c : b_1 \times b_3 \rightsquigarrow b_2 \times b_3} \qquad \frac{c : b_1 \rightsquigarrow b_2}{\mathit{second}\ c : b_3 \times b_1 \rightsquigarrow b_3 \times b_2}$$

$$\frac{c : b_1 \rightsquigarrow b_2}{\mathit{left}\ c : b_1 + b_3 \rightsquigarrow b_2 + b_3} \qquad \frac{c : b_1 \rightsquigarrow b_2}{\mathit{right}\ c : b_3 + b_1 \rightsquigarrow b_3 + b_2}$$

$$\frac{c_1 : b_1 \rightsquigarrow b_3 \quad c_2 : b_2 \rightsquigarrow b_4}{c_1 +\!\!+\!\!+ c_2 : b_1 + b_2 \rightsquigarrow b_3 + b_4} \qquad \frac{c_1 : b_1 \rightsquigarrow b_3 \quad c_2 : b_2 \rightsquigarrow b_4}{c_1 \mathbin{***} c_2 : b_1 \times b_2 \rightsquigarrow b_3 \times b_4} \qquad \frac{b \text{ inhabited}}{\mathit{inhab} : 1 \rightsquigarrow b}$$

$$\frac{}{\mathit{alloc} : 0 \rightsquigarrow a} \qquad \frac{}{\mathit{inl} : a \rightsquigarrow a + b} \qquad \frac{}{\mathit{inr} : b \rightsquigarrow a + b} \qquad \frac{}{\mathit{clone} : a \rightsquigarrow a \times a}$$

$$\frac{}{\mathit{discard} : a \rightsquigarrow 1} \qquad \frac{}{\mathit{fst} : a \times b \rightsquigarrow a} \qquad \frac{}{\mathit{snd} : a \times b \rightsquigarrow b} \qquad \frac{}{\mathit{merge} : a + a \rightsquigarrow a} \qquad \frac{}{\mathit{measure} : a \rightsquigarrow a}$$

Fig. 6. The syntax and type system of the arrow metalanguage $\mathcal{U}\Pi_a^\chi$ (rules for inhabitation appear in Fig. 5).

make the additional assumption that this garbage is inhabited. This is a very mild assumption, since garbage can always be chosen to be inhabited.

The hiding combinator allows projections $\mathit{fst} : b_1 \times b_2 \rightsquigarrow b_1$ and $\mathit{snd} : b_1 \times b_2 \rightsquigarrow b_2$ to be defined. When combined with the classical cloning combinator inherited from $\mathcal{U}\Pi_a$, we show that a combinator $\mathit{measure} : b \rightsquigarrow b$ for measurement can be derived.

$\mathcal{U}\Pi_a^\chi$ takes its canonical semantics in the category **CPTP** of Hilbert spaces and quantum channels, and as with $\mathcal{U}\Pi_a$, we will show in Section 4 how a model of $\mathcal{U}\Pi_a$ can be extended to one of $\mathcal{U}\Pi_a^\chi$ by a universal construction, connecting isometries to quantum channels (more on this in Section 5). We also extend the approximate universality theorem of $\mathcal{U}\Pi_a$ to one showing approximate universality of $\mathcal{U}\Pi_a^\chi$ combinators with respect to quantum channels.

Like $\mathcal{U}\Pi_a$, $\mathcal{U}\Pi_a^\chi$ is an arrow metalanguage extending $\mathcal{U}\Pi_a$ (see Fig. 6 for an overview). On the surface, $\mathcal{U}\Pi_a^\chi$ may look similar to if we were to apply the arrow construction from the classical case [James and Sabry 2012] to $\mathcal{U}\Pi$, but the result would be quite different. The arrow constructions of $\mathcal{U}\Pi_a$ and $\mathcal{U}\Pi_a^\chi$ are chosen precisely for their semantic properties (which we explore in Section 4), and cannot be replaced without altering semantics. One tangible difference is in the treatment of allocation: ML_Π (of [James and Sabry 2012]) does not have a unit 0 for the sum (as that would lead to an undesirable combinator of type $1 \rightsquigarrow 0$), so allocation has type $1 \rightsquigarrow b$; on the other hand, $\mathcal{U}\Pi_a$ and $\mathcal{U}\Pi_a^\chi$ have a unit 0 for the sum, and allocation has type $0 \rightsquigarrow b$.

Similarly to $\mathcal{U}\Pi_a$, $\mathcal{U}\Pi_a^\chi$ uses the same base types as $\mathcal{U}\Pi$ and $\mathcal{U}\Pi_a$, but introduces a new combinator type $b \rightsquigarrow b$ to distinguish $\mathcal{U}\Pi_a^\chi$ combinators at the type level. All combinators in $\mathcal{U}\Pi_a^\chi$

are constructed from $\mathcal{U}\Pi_a$ combinators using the single primitive *lift* combinator

$$\frac{v : b_1 \rightarrowtail b_2 \times b_3 \quad b_3 \text{ inhabited}}{lift\ v : b_1 \rightsquigarrow b_2} \ .$$

The definition of the arrow metacombinators *arr*, $\ggg$, and *first* are bound to look very familiar, as they are defined dually to those in $\mathcal{U}\Pi_a$ (indeed, we will see in Section 4 that the two constructions are dual in a formal sense). To turn a $\mathcal{U}\Pi_a$ combinator $b_1 \rightarrowtail b_2$ into a pure $\mathcal{U}\Pi_a^\chi$ combinator $b_1 \rightsquigarrow b_2$ can be done by assigning it the trivial (and trivially inhabited) garbage of 1,

$$arr(v) = lift(v \ggg uniti^\times) \ .$$

Combinators of type $b_1 \rightsquigarrow b_2$ and $b_2 \rightsquigarrow b_3$ with garbage of type b_4 and b_4' respectively can be composed by

$$lift(v_1) \ggg lift(v_2) = lift(v_1 \ggg (v_2 \ast\!\!\ast\!\!\ast\ id) \ggg assoc^\times)$$

resulting in a combinator with garbage $b_4' \times b_4$. For the final arrow combinator *first* allowing parallel execution of arrows, we define $c_1 \ast\!\!\ast\!\!\ast\ c_2$ to simply run the underlying $\mathcal{U}\Pi_a$ combinators in parallel and swap the garbage into the right position as necessary,

$$lift(v_1) \ast\!\!\ast\!\!\ast\ lift(v_2) = lift((v_1 \ast\!\!\ast\!\!\ast\ v_2) \ggg midswap^\times) \ ,$$

such that the garbage of $c_1 \ast\!\!\ast\!\!\ast\ c_2$ is the product of the garbages of c_1 and c_2 respectively. We derive $first(c) = c \ast\!\!\ast\!\!\ast\ id$ and $second(c) = id \ast\!\!\ast\!\!\ast\ c$. All of these definitions are straightforwardly seen to preserve the inhabitation requirement on garbage.

Defining the choice metacombinator $c_1 +\!\!+\!\!+ c_2$ is a bit more tricky, and it turns out to be easier to define *left* and derive $+\!\!+\!\!+$ and *right* from it. The idea is to exploit distributivity and inhabitation of garbage: if $c : b_1 \rightsquigarrow b_2$ produces garbage of type b_4 and the identity produces garbage of type 1, we can use the inhabitation of b_4 to turn the trivial garbage into garbage of type b_4 via $inhab : 1 \rightsquigarrow b_4$, and then distribute out on the right to get something of the required type $(b_2 + b_3) \times b_4$. This gives us the definition

$$left(lift\ v) = lift((v +\!\!+\!\!+ uniti^\times) \ggg (id +\!\!+\!\!+ (id \ast\!\!\ast\ inhab)) \ggg (swap^\times +\!\!+\!\!+ swap^\times) \ggg distribi \ggg swap^\times)$$

from which we derive *right* and $+\!\!+\!\!+$ as usual [Hughes 2005] as

$$right(c) = swap^+ \ggg left(c) \ggg swap^+ \qquad c_1 +\!\!+\!\!+ c_2 = left(c_1) \ggg right(c_2) \ .$$

The combinators *alloc*, *inl*, *inr*, *clone*, and *inhab* related to the allocation effect from $\mathcal{U}\Pi_a$, as well as all of the base combinators of $\mathcal{U}\Pi$ lifted to $\mathcal{U}\Pi_a$, can be further lifted to $\mathcal{U}\Pi_a^\chi$ by applying *arr* to them (these are denoted by *iso* in Fig. 6).

Information hiding is introduced in $\mathcal{U}\Pi_a^\chi$ by means of the effectful $discard : b \rightsquigarrow 1$ combinator. Some finesse is required to manage the inhabitation requirement on garbage, however. On all types aside from 0, *discard* is given by lifting the $\mathcal{U}\Pi_a$ combinator $b \rightarrowtail 1 \times b$ that adds the multiplicative unit on the left,

$$discard = lift(uniti^\times \ggg swap^\times) \ .$$

On 0, we first need to use *alloc* to allocate something of inhabited type, namely 1, before we can discard it:

$$discard = lift(alloc \ggg uniti^\times) \ .$$

Analogously to the injections in $\mathcal{U}\Pi_a$, we can derive *projections* from this discarding effect as

$$fst = id \ast\!\!\ast\!\!\ast\ discard \ggg unit^\times \qquad snd = swap^\times \ggg fst$$

though these can also be defined equivalently as $fst = lift(id)$ and $snd = lift(swap^\times)$.

To allow the choice metacombinator to be used for conditional execution, we need a way to merge branches. This can be defined in $\mathcal{U}\Pi_a^\chi$ as the $merge : b + b \rightsquigarrow b$ combinator, exploiting hiding and the fact that $a + a \cong a \times (1 + 1)$, as in

$$merge = (uniti^\times \mathbin{+\!\!+\!\!+} uniti^\times) \ggg distribi \ggg fst \ .$$

We are finally ready to explore the measurement combinator in $\mathcal{U}\Pi_a^\chi$.

3.4.1 Measurement in the Computational Basis. As we have seen previously, $\mathcal{U}\Pi_a$ permits a notion of classical cloning, and $\mathcal{U}\Pi_a^\chi$ inherits it. When we combine this with the ability to discard information in $\mathcal{U}\Pi_a^\chi$ using the projections, we obtain a surprisingly robust notion of measurement in the computational basis. This measurement combinator $measure : b \rightsquigarrow b$ is defined simply to be

$$measure = clone \ggg fst$$

Classically, this is a complicated way of doing absolutely nothing – the map takes a piece of classical data, copies it, and then immediately throws away the copy. In the quantum case, however, this performs measurement. We illustrate this by an example.

Consider an arbitrary qubit state $|\phi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, and the associated density matrix

$$|\phi\rangle\langle\phi| = (\alpha\,|0\rangle + \beta\,|1\rangle)(\overline{\alpha}\,\langle 0| + \overline{\beta}\,\langle 1|)$$

Conjugating by $[\![clone]\!]$ (noting that this does indeed perform classical cloning in the canonical model of **CPTP**) yields the density matrix

$$\begin{aligned}[\![clone]\!]\,|\phi\rangle\langle\phi|\,[\![clone]\!]^\dagger &= (\alpha\,|00\rangle + \beta\,|11\rangle)(\overline{\alpha}\,\langle 00| + \overline{\beta}\,\langle 11|) \\ &= |\alpha|^2\,|00\rangle\langle 00| + \alpha\overline{\beta}\,|00\rangle\langle 11| + \beta\overline{\alpha}\,|11\rangle\langle 00| + |\beta|^2\,|11\rangle\langle 11| \quad . \end{aligned} \tag{1}$$

We remark that, in **CPTP**, $[\![fst]\!]$ is given by the *partial trace* (see, e.g., [Nielsen and Chuang 2002]) of density matrices. This means that

$$\begin{aligned}&[\![fst]\!](|\alpha|^2\,|00\rangle\langle 00| + \alpha\overline{\beta}\,|00\rangle\langle 11| + \beta\overline{\alpha}\,|11\rangle\langle 00| + |\beta|^2\,|11\rangle\langle 11|) \\ &\quad = |\alpha|^2\,\mathrm{tr}(|0\rangle\langle 0|)\,|0\rangle\langle 0| + \alpha\overline{\beta}\,\mathrm{tr}(|0\rangle\langle 1|)\,|0\rangle\langle 1| + \beta\overline{\alpha}\,\mathrm{tr}(|1\rangle\langle 0|)\,|1\rangle\langle 0| + |\beta|^2\,\mathrm{tr}(|1\rangle\langle 1|)\,|1\rangle\langle 1| \\ &\quad = |\alpha|^2\,|0\rangle\langle 0| + |\beta|^2\,|1\rangle\langle 1| \end{aligned} \tag{2}$$

since for vectors $|a\rangle$ and $|b\rangle$ in the computational basis, $\mathrm{tr}(|a\rangle\langle b|) = 1$ when $|a\rangle = |b\rangle$, and $\mathrm{tr}(|a\rangle\langle b|) = 0$ otherwise. Note that measurement in an arbitrary basis can then be performed by conjugating the measurement combinator with the appropriate change-of-base combinator. For example, measurement in the Hadamard basis $\{|+\rangle, |-\rangle\}$ is performed using $hadamard \ggg measure \ggg hadamard$.

This method of measurement may seem counterintuitive, but it is important to note that (1) and (2) above show that it is physically equivalent to the usual one. Our presentation can be seen as exploiting *purification* of quantum states (see, e.g., [Nielsen and Chuang 2002]) to describe mixed states, as this result implies that every mixed state appears as the partial trace of a pure one.

3.4.2 Expressiveness. Finally we can extend the universality theorems for $\mathcal{U}\Pi$ and $\mathcal{U}\Pi_a$ to one that $\mathcal{U}\Pi_a^\chi$ is approximately universal for arbitrary quantum computations, that is, quantum channels.

Theorem 4. *$\mathcal{U}\Pi_a^\chi$ is approximately universal for quantum channels: For any quantum channel Λ and $\delta > 0$ there exists a $\mathcal{U}\Pi_a^\chi$ combinator c such that $\left\|\Lambda - [\![c]\!]\right\|_{\mathrm{op}} < \delta$.*

4 CATEGORICAL SEMANTICS

In this section we develop denotational semantics for the simple programming languages of the previous section in three stages. First, the base language Π can be interpreted in rig groupoids. This is then extended to $\mathcal{U}\Pi$ by providing interpretations for the phase and Hadamard combinators. Finally, we discuss two categorical constructions, R and L, that model the arrow constructions – i.e., *lift* combinator constructors of $\mathcal{U}\Pi_a$ and $\mathcal{U}\Pi_a^\chi$ respectively – in order to encapsulate information allocation and hiding and account for measurement.

4.1 Rig Groupoids

To interpret the type system and semantics of Π in a category, it needs to have combinators $\oplus$ and $\otimes$ that distribute over each other in the correct way. This is captured in the notion of a *rig groupoid*. Recall that a groupoid is a category in which every morphism is invertible.

Definition 5. A *rig category* is category **C** with two symmetric monoidal structures $(\oplus, 0)$ and $(\otimes, I)$, as well as natural isomorphisms

$$\delta_{A,B,C} \colon A \otimes (B \oplus C) \to (A \otimes B) \oplus (A \otimes C)$$
$$\delta_0 \colon A \otimes 0 \to 0$$

satisfying several coherence laws for which we refer to [Kelly 1974; Laplaza 1972].

Here $\oplus$ need not be a coproduct, and $\otimes$ need not be a product, which is the special case of a *distributive category*.

A rig groupoid suffices to interpret Π. Being a language for classical reversible computing, a canonical such model is the rig groupoid **FinBij** of finite sets and bijective functions. The base types are interpreted as $[\![0]\!] = 0$, $[\![1]\!] = I$, $[\![b + b']\!] = [\![b]\!] \oplus [\![b']\!]$, and $[\![b \times b']\!] = [\![b]\!] \otimes [\![b']\!]$. The combinator type $b \leftrightarrow b'$ becomes the (invertible) morphisms $[\![b]\!] \to [\![b']\!]$. As for the atomic combinators, the swap morphisms for $\oplus$ and $\otimes$ interpret $swap^+$ and $swap^\times$, respectively. The unitor $\rho_A^\oplus \colon A\oplus \to 0$ and its inverses coming from the monoidal structure $(\oplus, 0)$ denote $unit^+$ and $uniti^+$. The associator $\alpha^\oplus_{A,B,C} \colon (A \oplus B) \oplus C \to A \oplus (B \oplus C)$ and its inverse interpret $assoc^+$ and $associ^+$. We use the coherence isomorphisms of $(\otimes, I)$ to interpret $[\![unit^\times]\!] = \rho^\otimes$, $[\![uniti^\times]\!] = (\rho^\otimes)^{-1}$, $[\![assoc^\times]\!] = \alpha^\otimes$, and $[\![associ^\times]\!] = (\alpha^\otimes)^{-1}$. The distributors finish the interpretation: $[\![distrib]\!] = \delta$, $[\![distribi]\!] = \delta^{-1}$, $[\![distribo]\!] = \delta_0$, and $[\![distriboi]\!] = \delta_0^{-1}$. The combinators are simply composition via $\circ$, $\oplus$, and $\otimes$.

Interpreting $\mathcal{U}\Pi$ requires additionally to give semantics to the $phase_\varphi$ and *hadamard* combinators. In principle, they can be given trivial semantics (i.e., as identities) in any rig groupoid, though that would yield mere classical semantics, and thus defeat the purpose of the quantum extension to Π to begin with. A far better approach would be to interpret $\mathcal{U}\Pi$ in a category with quantum capabilities, the canonical choice being the category **Unitary** of finite dimensional Hilbert spaces and unitaries. Here, the semantics of these two combinators are given by

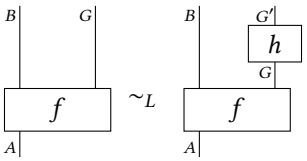
$$[\![phase_\varphi]\!] \quad = \quad x \mapsto \varphi \cdot x \qquad\qquad [\![hadamard]\!] \quad = \quad \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} .$$

Another way of writing $[\![phase_\varphi]\!]$ is as the 1×1 matrix (φ), as phases are exactly unitaries $\mathbb{C} \to \mathbb{C}$.

4.2 Garbage and Heap

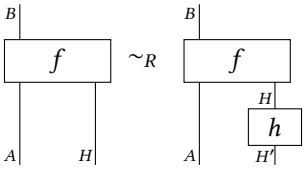
We now extend the categorical semantics of $\mathcal{U}\Pi$ to take quantum information effects into account. Let **C** be a symmetric monoidal category. We think of its objects as types, its morphisms as programs, and its tensor product as parallel composition. Hence a morphism $f \colon A \to B \otimes G$ will denote a program that takes an input of type A and produces an output of type B together with some *garbage* of type G. As we want to disregard the garbage, we will identify this morphism with

$(\mathrm{id}_B \otimes h) \circ f : A \to B \otimes G'$ for any morphism $h : G \to G'$ that postprocesses the garbage, called a *mediator*. That is, we consider the equivalence relation $\sim_L$ generated by:



(3)

Dually, instead of garbage, we can also consider a *heap*. That is, morphisms $f : A \oplus H \to B$ will denote a program that takes input of type A and may use a heap H in producing an output of type B. Again, we only care about access to the heap and not the actual contents of the heap, so we will identify f with $f \circ (\mathrm{id}_A \oplus h) : A \oplus H' \to B$ for any morphisms $h : H' \to H$ that preprocesses the heap. That is, we consider the equivalence relation $\sim_R$ generated by:



(4)

Notice that the diagram in (3) refers to the monoidal product $\otimes$, while the one above in (4) refers to the monoidal sum $\oplus$.

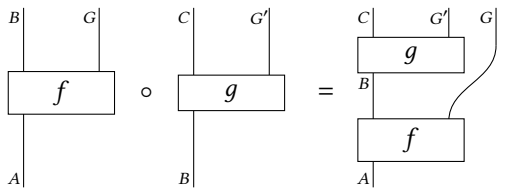
What exactly are $\sim_L$ and $\sim_R$? Equations (3) and (4) define relations that are already reflexive (with the identity as mediator) and transitive (compose mediators), but not always symmetric. The following lemma shows that they are already symmetric in special cases of interest, such as when the base category $\mathbf{C}$ is a groupoid.

Lemma 6. *When every morphism in* $\mathbf{C}$ *is split monic, equation* (3) *defines an equivalence relation. When every morphism in* $\mathbf{C}$ *is split epic, equation* (4) *defines an equivalence relation.*

Proof. It suffices to establish symmetry. If $(\mathrm{id} \otimes h) \circ f = g$, there is k with $k \circ h = \mathrm{id}$, so $(\mathrm{id} \otimes k) \circ g = (\mathrm{id} \otimes k) \circ (\mathrm{id} \otimes h) \circ f = f$. An analogous argument holds for $\sim_R$. □

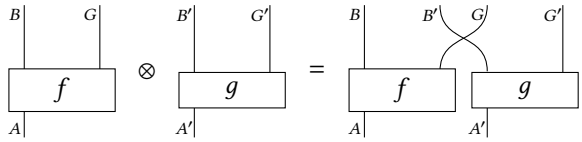
Proposition 7. *If* $(\mathbf{C}, \otimes, I)$ *is a symmetric monoidal category, there is a well-defined symmetric monoidal category* $L[\mathbf{C}]$ *whose:*

- objects *are the same as those of* $\mathbf{C}$*;*
- morphisms $A \to B$ *are equivalence classes of morphisms* $A \to B \otimes G$ *in* $\mathbf{C}$ *under* (3)*;*
- composition *is:*



- identities *are the inverse right unitors:*

- tensor unit I *is as in* $\mathbf{C}$*;*
- tensor product of objects *is as in* $\mathbf{C}$*;*
- tensor product of morphisms *is:*



Dually, if $(\mathbf{C}, \oplus, O)$ *is a symmetric monoidal category, there is a well-defined symmetric monoidal category* $R[\mathbf{C}] = L[\mathbf{C}^{\mathrm{op}}]^{\mathrm{op}}$. *Explicitly:*

- objects *are the same as those of* $\mathbf{C}$*;*
- morphisms $A \to B$ *are equivalence classes of morphisms* $A \oplus H \to B$ *in* $\mathbf{C}$ *under* (4). □

Proof. Well-definedness follows from Lemma 8. It is straightforward to verify that coherence isomorphisms in $L[\mathbf{C}]$ may be taken to be those in $\mathbf{C}$ composed with the inverse right unitor. □

Lemma 8. *Let* $(\mathbf{C}, \oplus)$ *be a monoidal category. If* $f \sim_R f'$ *and* $g \sim_R g'$*, then:*

(i) $g \circ f \sim_R g' \circ f'$ *if* g *and* f *are composable in* $R[\mathbf{C}]$*;*

(ii) $f \oplus g \sim_R f' \oplus g'$*;*

Dually $g \circ f \sim_L g' \circ f'$ *when* g *and* f *composable in* $L[\mathbf{C}]$ *and* $f \otimes g \sim_L f' \otimes g'$ *when* $f \sim_L f'$ *and* $g \sim_L g'$ *in a monoidal category* $(\mathbf{C}, \otimes)$.

So what is the point of doing these constructions? This is made clear in the following proposition. Indeed, in Section 5, we will see that $L[\mathbf{C}]$ and $R[\mathbf{C}]$ are, in a certain sense, the smallest categories containing $\mathbf{C}$ with these properties.

Proposition 9. *The monoidal unit* I *is terminal in* $L[\mathbf{C}]$*, and the monoidal unit* O *is initial in* $R[\mathbf{C}]$.

As a consequence of this fact, there are canonical projections $X \otimes Y \xrightarrow{\pi_1} X$ and $Y \otimes Y \xrightarrow{\pi_2} X$ in $L[\mathbf{C}]$ given by $X \otimes Y \xrightarrow{\mathrm{id} \otimes !} X \otimes I \xrightarrow{\rho^{\otimes}} X$ (and symmetrically for π_2). Likewise, there are canonical injections $X \xrightarrow{\amalg_1} X \oplus Y$ and $Y \xrightarrow{\amalg_2} X \oplus Y$ (defined dually to the above) in $R[\mathbf{C}]$.

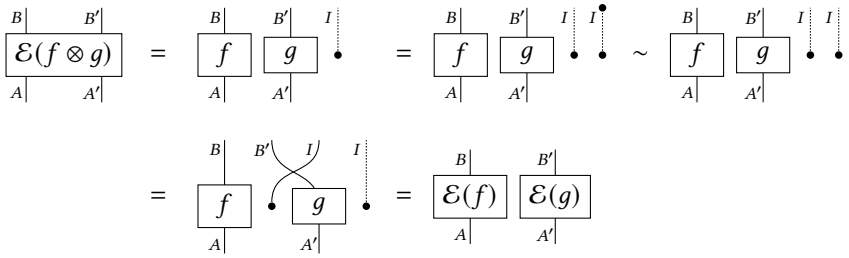
Proposition 10. *If* $(\mathbf{C}, \otimes)$ *is a monoidal category, there is a strict monoidal functor* $\mathcal{E} \colon \mathbf{C} \to L[\mathbf{C}]$ *given by* $\mathcal{E}(A) = A$ *on objects, and on morphisms as:*

$$\mathcal{E}(A \xrightarrow{f} B) = A \xrightarrow{f} B \xrightarrow{\rho^{\otimes -1}} B \otimes I$$

Dually, if $(\mathbf{C}, \oplus)$ *is a monoidal category, there is a strict monoidal functor* $\mathcal{D} \colon \mathbf{C} \to R[\mathbf{C}]$ *given by* $\mathcal{D}(A) = A$ *on objects, and on morphisms as:*

$$\mathcal{D}(A \xrightarrow{f} B) = A \oplus 0 \xrightarrow{\rho^{\oplus}} A \xrightarrow{f} B$$

Proof. By definition $\mathcal{E}(A \otimes B) = \mathcal{E}(A) \otimes \mathcal{E}(B)$ on objects. On morphisms:



Coherence isomorphisms in $L[\mathbf{C}]$ are precisely the image under $\mathcal{E}$ of those in $\mathbf{C}$. □

4.3 Lifting Tensor Products

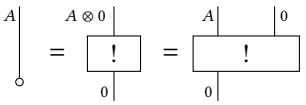
We will compose the L and R constructions, applying them to the base rig groupoid consecutively. In this section we show that if $\mathbf{C}$ is a rig category then so is $R[\mathbf{C}]$, while $L[R[\mathbf{C}]]$ loses its direct sum and becomes merely a monoidal category. However, we will see later that the direct sum in $L[R[\mathbf{C}]]$ is *binoidal* and satisfies the laws associated with an arrow with choice.

The overall idea with these constructions is that if $\mathbf{C}$ interprets the base language $\mathcal{U}\Pi$, then $R[\mathbf{C}]$ interprets the arrow metalanguage $\mathcal{U}\Pi_a$ of $\mathcal{U}\Pi$ extended with allocation. More precisely, if $[\![u]\!] : [\![b_1]\!] \oplus [\![b_3]\!] \to [\![b_2]\!]$ is the interpretation of some $\mathcal{U}\Pi$ combinator u in a rig groupoid $\mathbf{C}$, the interpretation $[\![lift(u)]\!]$ in $\mathcal{U}\Pi_a$ is given by the equivalence class $\left[[\![u]\!]\right]_{\sim_R} : [\![b_1]\!] \to [\![b_2]\!]$ in $R[\mathbf{C}]$. In turn, $L[R[\mathbf{C}]]$ interprets the arrow metalanguage $\mathcal{U}\Pi_a^X$ extending $\mathcal{U}\Pi$ with allocation and hiding by interpreting $[\![lift(v)]\!] : [\![b_1]\!] \to [\![b_2]\!]$ as the equivalence class of $[\![v]\!] : [\![b_1]\!] \to [\![b_2]\!] \otimes [\![b_3]\!]$ in $R[\mathbf{C}]$. Later, in Section 5.1 we will exhibit universal properties of $R[\mathbf{C}]$ and $L[\mathbf{C}]$, and argue that they justify these constructions in the canonical semantics of $\mathcal{U}\Pi$: $R[\mathbf{Unitary}]$ is equivalent to the category **Isometry** of finite dimensional Hilbert spaces and isometries, while $L[R[\mathbf{Unitary}]]$ is equivalent to the category **CPTP** of finite dimensional Hilbert spaces and quantum channels.

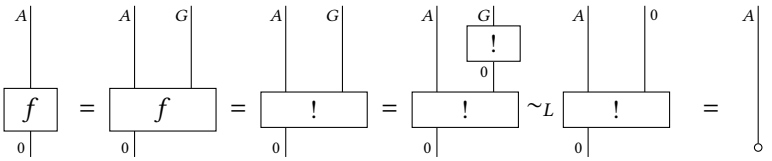
Although L and R are dual constructions, the order in which they transform $\mathbf{C}$ is important: we first add heaps and then garbage. The reason for this asymmetry is the following lemma.

Lemma 11. *Let $\mathbf{C}$ be a symmetric monoidal category. If $\mathbf{C}$ has an initial object, then so does $L[\mathbf{C}]$.*

Proof. Let 0 be an initial object in $\mathbf{C}$. For each object A there is then a morphism $0 \to A$ in $L[\mathbf{C}]$ given by the morphism



in $\mathbf{C}$. If $f : 0 \to A$ is a morphism in $L[\mathbf{C}]$, represented by a morphism $f : 0 \to A \otimes G$ in $\mathbf{C}$, then:



Thus 0 is indeed initial in $L[\mathbf{C}]$ as well. □

We start by endowing $\otimes$ and $\oplus$ with the capability to allow heaps.

Lemma 12. *If $\mathbf{C}$ is a rig category, then so is $R[\mathbf{C}]$.*

Proof sketch. The monoidal structure $(\oplus, 0)$ is inherited from $\mathbf{C}$ straightforwardly. More intricately, $R[\mathbf{C}]$ is monoidal with $(\otimes, I)$ inherited from $\mathbf{C}$:

- the tensor product of objects is $A \otimes B$;
- the tensor unit is I;
- the tensor product of morphisms $f : A \oplus H \to B$ and $f' : A' \oplus H' \to B'$ is

$$(A \otimes A') \oplus H'' \xrightarrow{\delta^{-1}} (A \oplus H) \otimes (A' \oplus H') \xrightarrow{f \otimes f'} B \otimes B'$$

  where $H'' = (H \otimes A') \oplus (A \otimes H') \oplus (H \otimes H')$;

□

The dual result does not hold: if $\mathbf{C}$ is a rig category, then $L[\mathbf{C}]$ need not be (though it is always monoidal by Proposition 7). This asymmetry is caused by the fact that $\otimes$ distributes over $\oplus$, but not the other way around. Lemma 11 is the nullary case of this fact. For a special case where this does hold, regard a Boolean algebra as a posetal category $\mathbf{C}$; then $\wedge$ and $\vee$ do distribute over each other both ways, so in that case $L[\mathbf{C}]$ is again a rig category.

Lemma 13. *If $\mathbf{C}$ is a rig category, then $\mathcal{D}\colon \mathbf{C} \to R[\mathbf{C}]$ is a strict rig functor.*

Proof. By definition $\mathcal{D}(A \otimes B) = \mathcal{D}(A) \otimes \mathcal{D}(B)$ on objects. On morphisms, $\mathcal{D}(f) \otimes \mathcal{D}(g)$ is

$$(A \otimes A') \oplus H'' \xrightarrow{\delta} (A \oplus 0) \otimes (A' \oplus 0) \xrightarrow{\rho^{\oplus} \otimes \rho^{\oplus}} A \otimes A' \xrightarrow{f \otimes g} B \otimes B'$$

with $H'' = (0 \otimes A') \oplus (A \otimes 0) \oplus (0 \otimes 0)$. Now $H'' \simeq 0$ by successive applications of the annihilators and unitors for $\oplus$, so by coherence $\mathcal{D}(f) \otimes \mathcal{D}(g)$ is the composition of $\rho^{\oplus}\colon (A \otimes A') \oplus 0 \to A \otimes A'$ and $f \otimes g$, which is precisely $\mathcal{D}(f \otimes g)$. □

4.4 Arrows with Choice

To complete the categorical semantics, we will show that $L[R[\mathbf{C}]]$ supports *arrows with choice* [Hughes 2005]. We saw in Proposition 7 and Lemma 12 that if $\mathbf{C}$ is a rig category, then $L[R[\mathbf{C}]]$ is monoidal under $\otimes$. What happens to $\oplus$? It is no longer necessarily a monoidal structure on $L[R[\mathbf{C}]]$, but we will show that it is *binoidal* – roughly, it is monoidal except that $\oplus$ is only a functor in each variable separately rather than in both variables jointly [Power and Robinson 1997] (see also [Jacobs et al. 2009]).

Definition 14. A category $\mathbf{C}$ is *binoidal* when it is equipped with functors $A \oplus -\colon \mathbf{C} \to \mathbf{C}$ and $- \oplus B\colon \mathbf{C} \to \mathbf{C}$ for each choice of objects $A, B \in \mathbf{C}$ such that applying the first functor to B results in the same object $A \oplus B$ as applying the second functor to A.

An *arrow with choice* is a functor F from a monoidal category $(\mathbf{C}, \oplus, 0)$ where 0 is initial to a binoidal category $(\mathbf{D}, \oplus)$ such that for any $g\colon B \to B'$:

$$F(f \oplus B) = F(f) \oplus B \tag{5}$$

$$F(\amalg_1) \circ f = (f \oplus B) \circ F(\amalg_1) \tag{6}$$

$$(f \oplus B') \circ F(A \oplus g) = F(A \oplus g) \circ (f \oplus B) \tag{7}$$

$$\alpha_{\oplus} \circ ((f \oplus B) \oplus C) = (f \oplus (B \oplus C)) \circ \alpha_{\oplus} \tag{8}$$

To prove that $L[R[\mathbf{C}]]$ is binoidal under $\oplus$, we will require that it has *inhabited garbage*: for any equivalence class of morphisms $A \to B \otimes G$ in $R[\mathbf{C}]$, there is a morphism $I \to G$ in $R[\mathbf{C}]$. Because 0 is initial in $R[\mathbf{C}]$, the problematic case where there is an isomorphism $\vartheta\colon G \to 0$ is handily avoided as $f\colon A \to B \otimes G$ is then equivalent to $A \xrightarrow{f} B \otimes G \xrightarrow{\mathrm{id} \otimes \vartheta} B \otimes 0 \xrightarrow{\mathrm{id} \otimes !} B \otimes I$.

In the remaining cases, inhabited garbage can be constructed when $\mathbf{C}$ is *semisimple*, meaning that any object is isomorphic to one built out of copies of the tensor unit I using $\oplus$ and $\otimes$. This is the case for **Unitary**. In this case, canonical inhabitants, i.e., morphisms $I \to G$ for each inhabited object G, can be constructed as the interpretation of the $\mathcal{U}\Pi_a$ combinators given in Section 3.3.2.

Lemma 15. *If $\mathbf{C}$ is a semisimple rig category, then $L[R[\mathbf{C}]]$ is a binoidal category under $\oplus$.*

Proposition 16. *If $\mathbf{C}$ is a semisimple rig category, $\mathcal{E}\colon R[\mathbf{C}] \to L[R[\mathbf{C}]]$ is an arrow with choice.*

Combining the previous Proposition with Lemma 13 shows that $\mathcal{E} \circ \mathcal{D}\colon \mathbf{C} \to L[R[\mathbf{C}]]$ is an arrow with choice. The categorical semantics of $\mathcal{U}\Pi$, $\mathcal{U}\Pi_a$, and $\mathcal{U}\Pi_a^{\chi}$, including certain derived combinators with structural or otherwise significant interpretations, are summarised in Fig. 7.

Base types

$$[\![0]\!] = O \quad [\![1]\!] = I \quad [\![b + b']\!] = [\![b]\!] \oplus [\![b']\!] \quad [\![b \times b']\!] = [\![b]\!] \otimes [\![b']\!]$$

Semantics of $\mathcal{U}\Pi$ $\quad [\![b_1 \leftrightarrow b_2]\!] = [\![b_1]\!] \to [\![b_2]\!]$ in $\mathbf{C}$

$$\begin{array}{llll} [\![swap^+]\!] = \sigma^\oplus & [\![unit^+]\!] = \rho^\oplus & [\![uniti^+]\!] = (\rho^\oplus)^{-1} & [\![assoc^+]\!] = \alpha^\oplus \quad [\![associ^+]\!] = (\alpha^\oplus)^{-1} \\ [\![swap^\times]\!] = \sigma^\otimes & [\![unit^\times]\!] = \rho^\otimes & [\![uniti^+]\!] = (\rho^\otimes)^{-1} & [\![assoc^\times]\!] = \alpha^\otimes \quad [\![associ^\times]\!] = (\alpha^\otimes)^{-1} \\ [\![id]\!] = \mathrm{id} & [\![distrib]\!] = \delta & [\![distribi]\!] = \delta^{-1} & [\![distribo]\!] = \delta_0 \quad [\![distribo i]\!] = \delta_0^{-1} \end{array}$$

$$[\![c_1 \fatsemi c_2]\!] = [\![c_2]\!] \circ [\![c_1]\!] \quad [\![c_1 + c_2]\!] = [\![c_1]\!] \oplus [\![c_2]\!] \quad [\![c_1 \times c_2]\!] = [\![c_1]\!] \otimes [\![c_2]\!] \quad [\![inv(c)]\!] = [\![c]\!]^{-1}$$

In **Unitary**:

$$[\![phase_\varphi]\!] = (\varphi) \quad [\![hadamard]\!] = \tfrac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Semantics of $\mathcal{U}\Pi_a$ $\quad [\![b_1 \rightarrowtail b_2]\!] = [\![b_1]\!] \to [\![b_2]\!]$ in $R[\mathbf{C}]$

$$\begin{array}{lll} [\![lift\ u]\!] = \big[\![[\![u]\!]\big]_{\sim_R} & [\![arr\ v]\!] = \mathcal{D}([\![v]\!]) & [\![c_1 \ggg c_2]\!] = [\![c_2]\!] \circ [\![c_1]\!] \\ [\![c_1 \mathbin{***} c_2]\!] = [\![c_1]\!] \otimes [\![c_2]\!] & [\![c_1 \mathbin{+++} c_2]\!] = [\![c_1]\!] \oplus [\![c_2]\!] & [\![first\ c]\!] = [\![c]\!] \otimes \mathrm{id} \\ [\![second\ c]\!] = \mathrm{id} \otimes [\![c]\!] & [\![left\ c]\!] = [\![c]\!] \oplus \mathrm{id} & [\![right\ c]\!] = \mathrm{id} \oplus [\![c]\!] \\ [\![alloc]\!] = O \xrightarrow{!} [\![b_2]\!] & [\![inl]\!] = \amalg_1 & [\![inr]\!] = \amalg_2 \end{array}$$

Semantics of $\mathcal{U}\Pi_a^\chi$ $\quad [\![b_1 \rightsquigarrow b_2]\!] = [\![b_1]\!] \to [\![b_2]\!]$ in $L[R[\mathbf{C}]]$

$$\begin{array}{lll} [\![lift\ v]\!] = \big[\![[\![v]\!]\big]_{\sim_L} & [\![arr\ v]\!] = \mathcal{E}([\![v]\!]) & [\![c_1 \ggg c_2]\!] = [\![c_2]\!] \circ [\![c_1]\!] \\ [\![c_1 \mathbin{***} c_2]\!] = [\![c_1]\!] \otimes [\![c_2]\!] & [\![c_1 \mathbin{+++} c_2]\!] = [\![right\ c_2]\!] \circ [\![left\ c_1]\!] & [\![first\ c]\!] = [\![c]\!] \otimes \mathrm{id} \\ [\![second\ c]\!] = \mathrm{id} \otimes [\![c]\!] & [\![left\ c]\!] = [\![c]\!] \oplus [\![b_1']\!] & [\![right\ c]\!] = [\![b_1']\!] \oplus [\![c]\!] \\ [\![discard]\!] = [\![b_1]\!] \xrightarrow{!} I & [\![fst]\!] = \pi_1 & [\![snd]\!] = \pi_2 \end{array}$$

Fig. 7. The categorical semantics of $\mathcal{U}\Pi$, $\mathcal{U}\Pi_a$, and $\mathcal{U}\Pi_a^\chi$ in summary, including the semantics of various derived combinators (marked yellow). The semantics of $phase_\varphi$ and $hadamard$ in $\mathcal{U}\Pi$ refer to their canonical semantics in **Unitary**. In $\mathcal{U}\Pi_a^\chi$, b_1' in the semantics of $left\ c$ and $right\ c$ refer to the type of the alternate choice; e.g., when c has type $b_1 \rightsquigarrow b_2$, $left\ c$ has type $b_1 + b_1' \rightsquigarrow b_2 + b_1'$.

5 PROPERTIES OF THE CATEGORICAL CONSTRUCTIONS

The previous section developed enough properties of our categorical semantics to interpret $\mathcal{U}\Pi_a$. But the L- and R-constructions have more properties, that give them the status of a very useful generic construction. From these universal properties it follows that two fundamental embeddings in reversible computing and quantum computing are both captured by our categorical semantics.

5.1 Universal Properties

The first insight is the following factorisation lemma, that brings morphisms in $L[\mathbf{C}]$ and $R[\mathbf{C}]$ in a normal form in terms of pure morphisms from $\mathbf{C}$.

Lemma 17. *If $\mathbf{C}$ is a rig category, then:*

(i) any map $A \to B$ in $R[\mathbf{C}]$ is represented by $A \xrightarrow{\amalg_1} A \oplus H \xrightarrow{\mathcal{D}(f)} B$ for some $f : A \oplus H \to B$ in $\mathbf{C}$;

(ii) any map $A \to B$ in $L[\mathbf{C}]$ is represented by $A \xrightarrow{\mathcal{E}(f)} B \otimes G \xrightarrow{\pi_1} B$ for some $f : A \to B \otimes G$ in $\mathbf{C}$.

These factorisations are unique.

Proof. Point (ii) follows from [Hermida and Tennent 2012, Lemma 8], see [Heunen and Kaarsgaard 2021, Lemma 15]. Point (i) then follows from (ii) by Proposition 7. □

For any object A there is a unique morphism $A \to I$ in $L[\mathbf{C}]$, represented by the unitor $A \to I \otimes A$ in $\mathbf{C}$, that simply considers the input as garbage and does nothing else. This makes $L[\mathbf{C}]$ affine, and in fact it is the universal affine category including $\mathbf{C}$.

Theorem 18. *$L[\mathbf{C}]$ is the affine completion of a monoidal category $\mathbf{C}$: for an affine monoidal category $\mathbf{D}$ and a monoidal functor $F\colon \mathbf{C} \to \mathbf{D}$ there is a unique monoidal functor $\hat{F}\colon L[\mathbf{C}] \to \mathbf{D}$ with $F = \hat{F} \circ \mathcal{E}$.*

Proof. This is an easy generalisation of [Heunen and Kaarsgaard 2021, Theorem 16]. □

Theorem 19. *$R[\mathbf{C}]$ is the coaffine completion of a rig category $\mathbf{C}$: for any coaffine rig category $\mathbf{D}$ and rig functor $F\colon \mathbf{C} \to \mathbf{D}$ there is a unique rig functor $\hat{F}\colon R[\mathbf{C}] \to \mathbf{D}$ such that $F = \hat{F} \circ \mathcal{D}$.*

Proof. By Proposition 7 and Theorem 18, there is a unique functor $\hat{F}$ making the triangle commute that is monoidal with respect to $\oplus$. Lemmas 13 and 17 show that it is also monoidal with respect to $\otimes$, that is, a rig functor. □

It follows immediately that the L- and R-constructions are functorial: a monoidal functor $\mathbf{C} \to \mathbf{D}$ induces a monoidal functor $L[\mathbf{C}] \to L[\mathbf{D}]$, and a rig functor $\mathbf{C} \to \mathbf{D}$ induces a rig functor $R[\mathbf{C}] \to R[\mathbf{D}]$. The combination $L[R[\mathbf{C}]]$ is more than the sum of Theorems 18 and 19. The following lemma shows that it is also universal for arrows with choice as in Section 4.4.

Lemma 20. *Let $\mathbf{C}$ be a category with an initial object.*

(i) The category $L[\mathbf{C}]$ has an initial object, and $\mathcal{E}\colon \mathbf{C} \to L[\mathbf{C}]$ preserves it.
(ii) If $F\colon \mathbf{C} \to \mathbf{D}$ preserves the initial object then so does $\hat{F}\colon L[\mathbf{C}] \to \mathbf{D}$.
(iii) If $F\colon \mathbf{C} \to \mathbf{D}$ preserves injections then so does $\hat{F}\colon L[\mathbf{C}] \to \mathbf{D}$.

Proof. For (i), observe that any morphism $0 \to A$ in $L[\mathbf{C}]$ must be represented by the unique morphism $0 \to A \otimes G$ in $\mathbf{C}$ for some G, and these are all equivalent under $\sim_L$. By construction $\mathcal{E}(0) = 0$, see Proposition 10. Points (ii) and (iii) now follow immediately. □

5.2 Toffoli and Stinespring

A central question of foundational importance for reversible modes of computing concerns that of *reversible expressivity*. Any irreversible computing machine is trivially able to simulate a reversible one – after all, reversible operations are just ordinary operations which happen to be invertible – but what is lost by considering *only* the reversible ones? Fortunately for the viability of reversible modes of computing, the answer turns out to be "nothing at all," if one is willing to accept some *ancillary inputs* and *garbage outputs* to occur.

5.2.1 The Fundamental Theorem of Classical Reversible Computing. In the case of classical reversible circuit logic, Toffoli found this question to be of so supreme importance to his theory that he dubbed the expressivity theorem the *fundamental theorem* [Toffoli 1980] of classical reversible computing. Toffoli showed this by demonstrating that any finite function ϕ can be simulated by a network consisting of an encoder, a finite bijective function f, and a decoder. This encoder and decoder should be *"essentially independent of ϕ and contain as little "computing power" as possible,"* [Toffoli 1980] and though this characterisation is (perhaps intentionally) vague, Toffoli goes on to show that one can always choose a "trivial encoder" (an injection) and a "trivial decoder" (a projection).

It is perhaps surprising that this highly specific statement (and its proof) on the nature of finite functions, down to decomposition of a finite function as an injection, a bijective function, and a projection, can be formulated as the following purely categorical statement:

Theorem 21. *The monoidal functor* $L[R[\mathbf{FinBij}]] \to \mathbf{FinSet}$ *induced by the (monoidal) inclusion functor* $\mathbf{FinBij} \to \mathbf{FinSet}$ *is full.*

To clarify, the inclusion functor $\mathbf{FinBij} \xrightarrow{I} \mathbf{FinSet}$ acts as the identity on both objects and morphisms, noting that any finite bijection is trivially also a finite function. The monoidal functor $L[R[\mathbf{FinBij}]] \to \mathbf{FinSet}$ then arises by successively applying Theorems 19 and 18, as in

$$\begin{array}{ccccc} \mathbf{FinBij} & \xrightarrow{\mathcal{D}} & R[\mathbf{FinBij}] & \xrightarrow{\mathcal{E}} & L[R[\mathbf{FinBij}]] \\ & \searrow^{I} & \downarrow{\scriptstyle \check{I}} & & \downarrow{\scriptstyle \hat{\check{I}}} \\ & & \mathbf{FinSet} & \xrightarrow[\mathrm{id}]{} & \mathbf{FinSet} \end{array}$$

Notice that it follows from Lemma 17 that any morphism of $L[R[\mathbf{FinBij}]]$ factors (essentially uniquely) as $A \xrightarrow{\mathcal{E}(\amalg_1)} A \oplus H \xrightarrow{\mathcal{E}(\mathcal{D}(f))} B \otimes G \xrightarrow{\pi_1} B$ for some bijection $A \oplus H \xrightarrow{f} B \otimes G$. Recall that all morphisms in **FinBij** are isomorphisms, and that all functors preserve isomorphisms. Using Lemma 20 and noting that $R[\mathbf{FinBij}] \xrightarrow{\check{I}} \mathbf{FinSet}$ preserves injections, it follows that any function in the image of $\hat{\check{I}}$ factors as

$$A \xrightarrow[\textit{injection}]{\amalg_1} A \oplus H \xrightarrow[\textit{bijection}]{\hat{\check{I}}(\mathcal{E}(\mathcal{D}(f)))} B \otimes G \xrightarrow[\textit{projection}]{\pi_1} B$$

What the fundamental theorem of reversible computing then states is that *all* functions are, in fact, in the image of $\hat{\check{I}}$, and so permit such a factorisation. That is exactly to say that $\hat{\check{I}}$ is full.

5.2.2 Stinespring's Dilation Theorem. Similarly, in the setting of quantum theory, the L- and R-constructions capture Stinespring's dilation theorem. As discussed in Section 2.1.2, this theorem shows that any mixed quantum operation can be modeled by a pure quantum operation, if one is willing to enlarge the situation with an auxiliary system. Write **Isometry** for the category of Hilbert spaces and isometric linear maps, and **Unitary** for the subcategory of unitary linear maps.

Theorem 22 (Huot & Staton). *There is an equivalence* $R[\mathbf{Unitary}] \simeq \mathbf{Isometry}$ *of rig categories, and an equivalence* $L[R[\mathbf{Unitary}]] \simeq \mathbf{CPTP}$ *of monoidal categories.*

Proof. See [Huot and Staton 2018, 2019] and [Heunen and Kaarsgaard 2021]. □

Notice that the previous theorem holds for Hilbert space that are not necessarily finite-dimensional.

6 APPLICATIONS

We will now consider some areas where our results can be put to work. First, we argue that our categorical model can be used to prove useful, nontrivial properties about measurement entirely algebraically, without ever needing to consider the gritty details of quantum channels. Second, we illustrate the use of $\mathcal{U}\Pi_a^\chi$ as a metalanguage by providing a translation from Selinger's quantum flowcharts [Selinger 2004] to $\mathcal{U}\Pi_a^\chi$.

6.1 Properties of Measurement

In this section, we will use type subscripts on combinators whenever it is necessary to disambiguate between definitions, or to make the presentation clearer. For example, we will write $measure_{b+b'}$ to mean the measurement combinator on the type $b + b' \rightsquigarrow b + b'$.

Our first property is bit technical, concerning the behaviour of injections with measurement. We will see shortly how it can be used to prove far more interesting things.

Proposition 23. *Measurement commutes with injections:* $[\![measure_b \ggg inl]\!] = [\![inl \ggg measure_{b+b'}]\!]$, *and likewise for inr.*

Any complex, finite dimensional Hilbert space is isomorphic to one of the form $\mathbb{C}^n$, where n is its dimension. From this it follows that each canonical injection $\amalg_i : \mathbb{C} \to \mathbb{C}^n$ is associated with a distinct vector $|i\rangle$ linearly independent from all other $|j\rangle$ for $1 \leq j \leq n$, $j \neq i$. Together, these are precisely the classical states forming the computational basis of $\mathbb{C}^n$.

Abstracting, we say that a classical state is nothing but an injection, i.e., composition of *inl* and *inr*. This intuition is correct, in that we can show that measurement does nothing to classical states:

Proposition 24. *If s is a classical state then* $[\![s \ggg measure]\!] = [\![s]\!]$.

Proof. We first see that measurement on 1 is the identity: $measure_1 = clone_1 \ggg fst = uniti^\times \ggg (id \times discard) \ggg unit^\times$. Since $[\![discard]\!]$ when applied to $[\![1]\!] = I$ is nothing but the identity by I terminal, it follows that

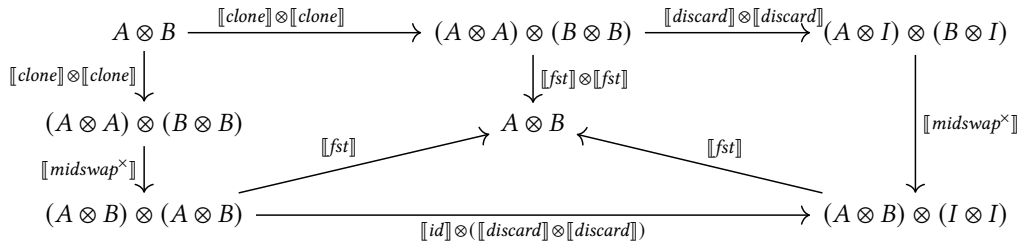
$$\begin{aligned}[\![measure_1]\!] &= [\![uniti^\times \ggg (id \times discard) \ggg unit^\times]\!] = [\![unit^\times]\!] \circ [\![id \times discard]\!] \circ [\![uniti^\times]\!] \\ &= [\![unit^\times]\!] \circ [\![uniti^\times]\!] = [\![id]\!] \ .\end{aligned}$$

Now, since a classical state $s : 1 \rightsquigarrow b$ is precisely an injection, it follows by Proposition 23 that $[\![s \ggg measure_b]\!] = [\![measure_1 \ggg s]\!] = [\![s]\!] \circ [\![measure_1]\!] = [\![s]\!]$. □

A very useful property of measurement is that the result of measuring a joint system is nothing but the product of measurements on each constituent system individually. This is shown as follows:

Proposition 25. *Measurement of products is the product of measurements:* $[\![measure_{b\times b'}]\!] = [\![measure_b \times measure_{b'}]\!]$.

Proof. Using the fact that $measure = clone \ggg fst$ and $clone_{b\times b'} = clone_b \times clone_{b'} \ggg midswap^\times$, the property follows by naturality of $[\![midswap^\times]\!]$:



Again, A and B range over interpretations of arbitrary $\mathcal{U}\Pi_a^\chi$ types b and b'. □

An immediate consequence of this property is that measurements also commute with projections:

Proposition 26. *Measurement on a product type commutes with projections:* $[\![measure_{b\times b'} \ggg fst]\!] = [\![fst \ggg measure_b]\!]$ *and likewise for snd.*

Proof. Since $[\![measure_{b\times b'}]\!] = [\![measure_b \times measure_{b'}]\!] = [\![measure_b]\!] \otimes [\![measure_{b'}]\!]$ by Proposition 25, it follows by naturality of $\pi_1 = [\![fst]\!]$ that

$$[\![measure_{b\times b'} \ggg fst]\!] = \pi_1 \circ [\![measure_b]\!] \otimes [\![measure_{b'}]\!] = [\![measure_b]\!] \circ \pi_1 = [\![fst \ggg measure_b]\!] \ .$$

which was what we wanted. □

The final property we want to show is that measurement is idempotent. Conceptually, this can be seen as an extension to the property that measurement does nothing to classical states. This is because the result of measuring a quantum state will always be a mixed classical state, so further measuring this has no effect. To do this, we remark that cloning can be shown to be associative.

Proposition 27. *Cloning is associative:* $[\![clone \ggg (clone \times id) \ggg assoc^\times]\!] = [\![clone \ggg (id \times clone)]\!]$.

We can then show idempotence:

Proposition 28. *Measurement is idempotent:* $[\![measure \ggg measure]\!] = [\![measure]\!]$.

Proof. Since $[\![fst]\!] = [\![lift\ id]\!]$, it can be shown that $[\![measure]\!] = [\![lift\ clone]\!]$, so by the definition of $\ggg$ in $\mathcal{U}\Pi_a^\chi$ we have that

$$[\![measure \ggg measure]\!] = [\![lift(clone \ggg (clone \times id) \ggg assoc^\times)]\!] = [\![lift(clone \ggg (id \times clone))]\!],$$

the final equality following from coassociativity of cloning (Proposition 27). But then $[\![id \times clone]\!] = \mathrm{id} \otimes [\![clone]\!]$ mediates between $[\![clone]\!]$ and $[\![clone \ggg (id \times clone)]\!]$ in $R[\mathbf{C}]$, so $[\![measure]\!] = [\![lift\ clone]\!] = [\![lift\ (clone \ggg (id \times clone))]\!] = [\![measure \ggg measure]\!]$ in $L[R[\mathbf{C}]]$. □

6.2 Quantum Flow Charts

In this section, we demonstrate the translation of (noniterative) quantum flow charts into $\mathcal{U}\Pi_a^\chi$. As the name suggests, quantum flow charts are the quantum extension of the classical imperative flow chart languages, used extensively in areas such as program compilation and partial evaluation [Hatcliff 1998; Jones et al. 1993]. They were first considered by Selinger in several variations [Selinger 2004]: here, we consider the purely quantum variant, which has only quantum data, in the form of qubit ensembles. The only type of data supported is the type *Qbit* of qubits, so typing contexts Γ are simply given by lists of active *Qbit* variables. In its textual form, a quantum flowchart is simply a list of commands. The supported commands are as follows (with q ranging over variables):

$$c ::= new\ qbit\ q := 0 \mid discard\ q \mid E \mathrel{*}= U \mid permute\ \varphi \mid initial \mid measure\ q \mid merge \mid c; c \mid c \oplus c$$

Commands take sums of typing contexts to sums of typing contexts, each summand denoting a program branch. Briefly, *new qbit* and *discard* allocate and discard variables respectively, $E \mathrel{*}= U$ applies a unitary U to a non-empty list of variables $E = p, q, r, \ldots$, *permute* φ changes the variable order by applying an arbitrary permutation φ, *initial* initialises an empty typing context, and *merge* merges two program branches (with the same typing context) into one. The most novel command is arguably *measure* q, which measures the qubit q and branches on the measurement result: this style of measurement-based flow control goes by the motto of "quantum data, classical control." Flow charts are composed in sequence and in parallel using ; and $\oplus$ respectively.

We begin with the translation of types and contexts, given simply by $\mathcal{T}[\![q : Qbit]\!] = 1 + 1$ and $\mathcal{T}[\![\Gamma, \Gamma']\!] = \mathcal{T}[\![\Gamma]\!] \times \mathcal{T}[\![\Gamma']\!]$. Before we proceed with the translation of commands, we will use the abuse of notation $|0\rangle$ and $|1\rangle$ to refer to the injections $\mathrm{inl} : 1 \rightsquigarrow 1+1$ and $\mathrm{inr} : 1 \rightsquigarrow 1+1$ respectively, to indicate that these serve as allocation of the constant classical values of $|0\rangle$ and $|1\rangle$. Commands

are translated as follows:

$$\begin{array}{rclrcl}
\mathcal{T}[\![new\ qbit\ q := 0]\!] & = & uniti^{\times} \ggg id \times |0\rangle & \mathcal{T}[\![discard\ q]\!] & = & fst \\
\mathcal{T}[\![E \mathrel{*}= U]\!] & = & id \times arr(arr(\hat{U})) & \mathcal{T}[\![permute\ \varphi]\!] & = & arr(arr(\hat{\varphi})) \\
\mathcal{T}[\![initial]\!] & = & alloc & \mathcal{T}[\![merge]\!] & = & merge \\
\mathcal{T}[\![c; c']\!] & = & \mathcal{T}[\![c]\!] \ggg \mathcal{T}[\![c']\!] & \mathcal{T}[\![c \oplus c']\!] & = & \mathcal{T}[\![c]\!] +\!\!+\!\!+ \mathcal{T}[\![c']\!]
\end{array}$$

$$\mathcal{T}[\![measure\ q]\!] = id \times measure \ggg distrib \ggg ((id \times |0\rangle) +\!\!+\!\!+ (id \times |1\rangle))$$

In the above, the terms $\hat{U}$ and $\hat{\varphi}$ denote the approximations of the unitary U and permutation φ (a particularly simple kind of unitary) respectively as $\mathcal{U}\Pi$ terms, as given by Theorem 2.

7 CONCLUSION AND FUTURE WORK

We have shown how quantum measurement, often presented in somewhat mysterious terms, arises through two suprisingly simple arrow constructions, associated with the information effects of *allocation* and *hiding*, on a reversible quantum combinator language. We have provided categorical semantics for all of these languages through elementary universal constructions on *rig categories*. These let us prove useful nontrivial properties of measurement as semantic equivalences, recast the fundamental theorem of reversible computation as an elementary categorical statement, and interpret (noniterative) quantum flow charts. There are several avenues for further research:

Classical and Quantum Data. In the current formulation, $\mathcal{U}\Pi$ only allows forming quantum data types. Hence qubit measurement, for example, can only be given the type $Qbit \longrightarrow Qbit$, rather than $Qbit \longrightarrow Bit$. Extending $\mathcal{U}\Pi_a^\chi$ with classical data would address this shortcoming. Semantically, this would need a (sufficiently nice) construction of the category of C*-algebras and quantum channels from the category of Hilbert spaces and quantum channels.

Categorical Semantics of SILQ. The quantum programming language SILQ [Bichsel et al. 2020] has several original features: *measurement-free* and *quantum-free* functions, a linear type system, and the automatic uncomputation of garbage. $\mathcal{U}\Pi_a^\chi$ enables a type-level interpretation of *measurement-free* functions (i.e., *pure* combinators), and access to a canonical reversibilisation of combinators. It would be interesting to extend the $\mathcal{U}\Pi$ family with the remaining features to provide combinator semantics for SILQ, much like Π^0 does in the classical case for Theseus [James and Sabry 2014].

Is There a Quantum Effect? We have shown that there are elementary arrow constructions connecting reversible and irreversible quantum computations. Is there a similar arrow construction connecting *classical* reversible computation and *quantum* reversible computation? Such a construction would likely involve several steps, such as adjoining the circle group to introduce phases, and considering a variation of convex combinations of morphisms respecting direct sums, to give morphisms that introduce and eliminate superpositions.

Recursion and Subnormalised Channels. The notion of quantum channel considered in this paper is too rigid to enable recursion or iteration, which is why we were only able to give semantics to the noniterative fragment of quantum flow charts. To enable recursion requires relaxing the notion of quantum channel, from (completely positive) trace-preserving maps to trace-nonincreasing ones. Can recursion be added as an effect, i.e., an arrow construction from **CPTP** to **CPTN**?

Measurement-Based Quantum Computation. Measurement-based quantum computation is in a sense opposite to the quantum circuit model. In the latter all operations are reversible except the very last one. In the former all operations are irreversible up to reversible corrections being fed forward. What is the relationship to the model given in this article? Is there a translation from the language of measurement patterns used in measurement-based quantum computation to $\mathcal{U}\Pi_a^\chi$?

REFERENCES

T. Altenkirch and A. S. Green. 2010. The Quantum IO Monad. *Semantic Techniques in Quantum Computation* (2010), 173–205. https://doi.org/10.1017/CBO9781139193313.006

C. H. Bennett. 1973. Logical Reversibility of Computation. *IBM Journal of Research and Development* 17, 6 (1973), 525–532. https://doi.org/10.1147/rd.176.0525

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 (PLDI 2020) (PLDI 2020)*. ACM, 286–300. https://doi.org/10.1145/3385412.3386007

W. J. Bowman, R. P. James, and A. Sabry. 2011. Dagger Traced Symmetric Monoidal Categories and Reversible Programming. (2011). Work-in-progress report presented at the 3rd International Workshop on Reversible Computation.

J. Carette and A. Sabry. 2016. Computing with Semirings and Weak Rig Groupoids. In *Proceedings of the 25th European Symposium on Programming (ESOP 2016)*. Springer, 123–148. https://doi.org/10.1007/978-3-662-49498-1_6

C.-H. Chen, V. Choudhury, J. Carette, and A. Sabry. 2020. Fractional Types: Expressive and Safe Space Management for Ancilla Bits. In *Proceedings of the 12th International Conference on Reversible Computation (RC 2020)*. Springer, 169–186. https://doi.org/10.1007/978-3-030-52482-1_10

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 (jan 2021), 29 pages. https://doi.org/10.1145/3434290

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

G. S. H. Cruttwell, B. Gavranović, N. Ghani, P. Wilson, and F. Zanasi. 2021. Categorical Foundations of Gradient-Based Learning. (2021). arXiv:2103.01931 [cs.LG]

B. Fong, D. Spivak, and R. Tuyéras. 2019. Backprop as Functor: A Compositional Perspective on Supervised Learning. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2019)*. IEEE, 1–13. https://doi.org/10.1109/LICS.2019.8785665

A. S. Green and T. Altenkirch. 2008. From Reversible to Irreversible Computations. In *Proceedings of the 4th International Workshop on Quantum Programming Languages (QPL 2006) (Electronic Notes in Theoretical Computer Science)*, Vol. 210. Elsevier, 65–74. https://doi.org/10.1016/j.entcs.2008.04.018

A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, and B. Valiron. 2013. Quipper: A Scalable Quantum Programming Language. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2013)*. ACM, 333–342. https://doi.org/10.1145/2491956.2462177

J. Hatcliff. 1998. An Introduction to Online and Offline Partial Evaluation Using a Simple Flowchart Language. In *Partial Evaluation: Practice and Theory (Lecture Notes in Computer Science)*, Vol. 1706. Springer, 20–82. https://doi.org/10.1007/3-540-47018-2_2

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

C. Heunen and R. Kaarsgaard. 2021. Bennett and Stinespring, Together at Last. In *Proceedings 18th International Conference on Quantum Physics and Logic (QPL 2021) (Electronic Proceedings in Theoretical Computer Science)*, Vol. 343. OPA, 102–118. https://doi.org/10.4204/EPTCS.343.5

C. Heunen and J. Vicary. 2019. *Categories for Quantum Theory*. Oxford University Press. https://doi.org/10.1093/oso/9780198739623.001.0001

N. G. Houghton-Larsen. 2021. *A Mathematical Framework for Causally Structured Dilations and its Relation to Quantum Self-Testing*. Ph.D. Dissertation. Department of Mathematical Sciences, University of Copenhagen.

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

M. Huot and S. Staton. 2018. Universal Properties in Quantum Theory. In *Proceedings of the 15th International Conference on Quantum Physics and Logic (QPL 2018) (Electronic Proceedings in Theoretical Computer Science)*, Vol. 287. OPA, 213–224. https://doi.org/10.4204/EPTCS.287.12

M. Huot and S. Staton. 2019. Quantum Channels as a Categorical Completion. In *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2019)*. IEEE, 1–13. https://doi.org/10.1109/LICS.2019.8785700

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

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

R. P. James and A. Sabry. 2014. Theseus: A High Level Language for Reversible Computing. (2014).

N. D. Jones, C. K. Gomard, and P. Sestoft. 1993. *Partial Evaluation and Automatic Program Generation*. Prentice Hall International.

R. Kaarsgaard and N. Veltri. 2019. En Garde! Unguarded Iteration for Reversible Computation in the Delay Monad. In *Proceedings of the 13th International Conference on Mathematics of Program Construction (MPC 2019)*. Springer, 366–384. https://doi.org/10.1007/978-3-030-33636-3_13

G. Kelly. 1974. Coherence Theorems for Lax Algebras and Distributive Laws. *Lecture Notes in Mathematics* 420 (1974), 281–375. https://doi.org/10.1007/BFb0063106

R. Landauer. 1961. Irreversibility and Heat Generation in the Computing Process. *IBM Journal of Research and Development* 5, 3 (1961), 261–269. https://doi.org/10.1147/rd.53.0183

M. Laplaza. 1972. Coherence for Distributivity. *Lecture Notes in Mathematics* 281 (1972), 29–72. https://doi.org/10.1007/BFb0059555

T. Leinster. 2016. *Basic Category Theory*. Cambridge University Press.

M. A. Nielsen and I. Chuang. 2002. *Quantum Computation and Quantum Information*. American Association of Physics Teachers.

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

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

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

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

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. https://doi.org/10.1007/978-3-319-89366-2_19

P. Selinger. 2004. Towards a Quantum Programming Language. *Mathematical Structures in Computer Science* 14, 4 (2004), 527–586. https://doi.org/10.1017/S0960129504004256

T. Toffoli. 1980. Reversible Computing. In *Proceedings of the 7th Colloquium on Automata, Languages, and Programming (ICALP 1980)*. Springer, 632–644.

J. Vizzotto, T. Altenkirch, and A. Sabry. 2006. Structuring Quantum Effects: Superoperators as Arrows. *Mathematical Structures in Computer Science* 16, 3 (2006), 453–468. https://doi.org/10.1017/S0960129506005287

J. K. Vizzotto, A. R. Du Bois, and A. Sabry. 2009. The Arrow Calculus as a Quantum Programming Language. In *International Workshop on Logic, Language, Information, and Computation (WoLLIC 2009)*. Springer, 379–393. https://doi.org/10.1007/978-3-642-02261-6_30

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

N. Yanofsky and M. A. Mannucci. 2008. *Quantum Computing for Computer Scientists*. Cambridge University Press.