

The Quantum Effect

A recipe for Quantum TI

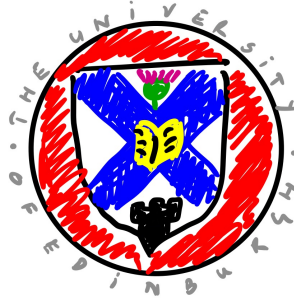
arXiv:2302.01885

Jacques Cavaletto



Chris Heunen

<chris.heunen@ed.ac.uk>



Robin Kaarsgaard



Amr Sabry



What makes Quantum ... Quantum?

Why This Quantum Pioneer Thinks We Need More People Working on Quantum Algorithms

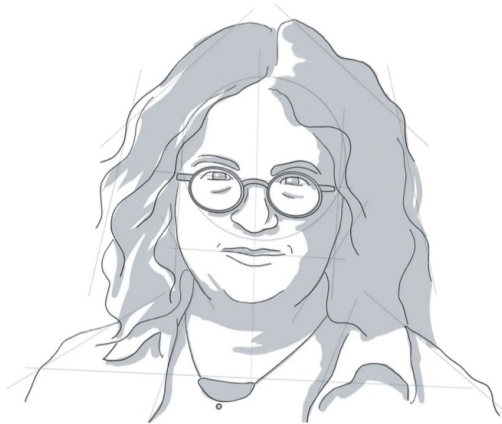


Qiskit · Follow

Published in Qiskit · 8 min read · Mar 15, 2022

94

1






Dorit Aharonov (illustration by J. Russell Huffman)

“One very, very interesting thing about quantum computation is that it touches so many different fields in mathematics,” she said. “It’s not like that in classical computation. It’s really something that is special for quantum computation because it’s somehow ‘complete’ — quantum computation is some kind of completion, mathematically, of classical computation.

Goal: Quantum computing as a completion of classical computing

Computational Effects

make programs do actually useful things:

- receive input, output 
- use randomness 
- provide multiple answers 

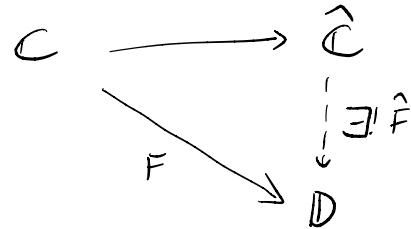
Programs that cannot communicate with the outside world are beautiful, perfect, and absolutely useless

Monads,

Applicatives,
product-preserving
functors

Arrows,
Freyd
categories

Completions:



Quantum Computation

via

Computational Effects

Add quantum features to classical reversible functional programming

The Quantum IO Monad

Thorsten Altenkirch and Alexander S. Green

Math. Struct. in Comp. Science

Structuring Quantum Effects: Superoperators as Arrows

JULIANA VIZZOTTO¹, THORSTEN ALTENKIRCH² and ALEXANDER S. GREEN³

¹ Institute of Informatics, Federal University of Rio Grande do Sul, Porto Alegre, Brazil
² School of Computer Science and IT, The University of Nottingham, UK
³ Department of Computer Science, Indiana University, USA

Received January 2025

The quantum IO monad is implemented as a semantics of quantum (unitary and irreversible) operations (i.e., unitaries and dagger operations) in a classical reversible functional programming language. This is achieved by embedding quantum features into a classical reversible functional programming language.

We present an internalization of quantum program factorization algebraic structure semantics also be understood as Haskell QIO library like to move to a modular functional quantum theory. We discuss the implications of effects (Swierstra 2008). At a hypothetical quantum level, we provide a brief introduction to reversible (i.e., unitaries and dagger operations) in a modular functional programming language.

1. Introduction

A newcomer to the field of quantum computing is immediately overwhelmed by the apparent differences with classical computing that suggest that quantum might require radically new semantic models and programming languages. In this paper we argue that this is not the case. Quantum computing is based on a kind of causality that is different from the classical notion of parallelism, and (2) quantum computation is a notion of observation in which the observed part of the quantum state is entangled with what immediately loses its wave character. It is seen that none of the other differences that are often cited between classical computing are actually relevant semantically. For example, even though it is not often thought of as “reversible,” it is just as reversible computation. Both can be implemented by a set of reversible universal gates (and Chuang 2000), section 1.4.1), but in neither model should the user be concerned about reversibility.

The two properties of quantum computing discussed above certainly go beyond classical programming and it has been suspected earlier that they might correspond to some notion of computational effect. Following Moggi’s influential paper (Moggi 1989), computational effects like assignments, exceptions, non-determinism, could all be modelled using the categorical construction of a monad. This construction has been internalised in the programming language Haskell as a tool to elegantly express computational

Algebraic Effects, Linearity, and Quantum Programming Languages

Sam Staton
Radboud University Nijmegen

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 hiding and allocation. The resulting type-and-effect system is fully expressive quantum computing, including measurement. We provide universal categorical construction interpret this arrow metalinguage with choice, starting with any rig groupoid interpret base language. Several properties of quantum measurement follow in general, and we treat quantum flow charts into our language. The semantic constructions turn the category of Hilbert spaces into the category of completely positive trace-preserving maps, and they of bijections between finite sets into the category of functions with chosen garbage. This fundamental theorems of classical and quantum reversible computing of Toffoli and Stine

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

ACM Reference Format:

Chris Heunen and Robin Kaarsgaard. 2022. Quantum Information Effects. *Proc. ACM Program. Lang.* 6(1), Article 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 International Postdoc Fellowship No. 0131-00025B. We thank Pablo Andrés-Martínez for their comments, corrections, and suggestions for this paper, and anonymous reviewers for their comments, corrections, and suggestions for this paper.

1 INTRODUCTION

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

Classical computing itself is most often formulated as composed of irreversible operations, yet the seminal works of Toffoli [Toffoli 1980] and Bennett [Bennett 1982] recently by James and Sabry [James and Sabry 2012], we know that it can all be done with reversible operations, as long as we consider systems to be open and environment that is eventually disregarded. This final part is important, as reversible computations (be they classical or quantum) cannot change the amount of information (as measured by an

Abstract

We develop a new framework of algebraic theories of quantum computing, and use it to analyze the equational theory of quantum computing and quantum programming languages.

- we present a new elementary algebraic theory of quantum computing, both from unitary gates and measurement
- we provide a completeness theorem for the algebraic theory by relating it with a model from quantum computing
- we extract an equational theory for a quantum programming language from the algebraic theory
- we compare quantum computation with other computational models by investigating variations on the algebraic theory

1. Introduction

Quantum programming languages took many of the ideas from modern programming language theory: linear use of resources, locality. A good way to understand a program is to understand its underlying algebraic structure. In this paper we develop a general algebraic framework for quantum programming languages. We use it to give a complete equational theory of quantum programs.

What is quantum computing? From a programmer’s perspective, quantum computing involves qubits and operations on them. There is a type theory of qubits. Viewed as an algebra, one can imagine a qubit as having an internal state that can be measured. A good way to understand a program is to understand its underlying algebraic structure. In this paper we develop a general algebraic framework for quantum programming languages. We use it to give a complete equational theory of quantum programs.

• there is a type theory of qubits. Viewed as an algebra, one can imagine a qubit as having an internal state that can be measured. A good way to understand a program is to understand its underlying algebraic structure. In this paper we develop a general algebraic framework for quantum programming languages. We use it to give a complete equational theory of quantum programs.

• there are three access functions: read, write, and measure. The three access functions are: read, write, and measure. The three access functions are: read, write, and measure. The three access functions are: read, write, and measure.

• apply: apply a rotation to the qubit on the surface of a sphere called the Bloch sphere. The three access functions are: read, write, and measure. The three access functions are: read, write, and measure.

• there are three access functions: read, write, and measure. The three access functions are: read, write, and measure. The three access functions are: read, write, and measure.

• apply: apply a rotation to the qubit on the surface of a sphere called the Bloch sphere. The three access functions are: read, write, and measure. The three access functions are: read, write, and measure.

• there are three access functions: read, write, and measure. The three access functions are: read, write, and measure. The three access functions are: read, write, and measure.

• apply: apply a rotation to the qubit on the surface of a sphere called the Bloch sphere. The three access functions are: read, write, and measure. The three access functions are: read, write, and measure.

• there are three access functions: read, write, and measure. The three access functions are: read, write, and measure. The three access functions are: read, write, and measure.

Universal Properties of Partial Quantum Maps

Pablo Andrés-Martínez¹, Chris Heunen², Robin Kaarsgaard³
¹University of Edinburgh, ²University of Edinburgh, ³University of Edinburgh

We provide a universal construction of the category of finite-dimensional C^* -algebras and completely positive trace-nonincreasing maps from the rig category of finite-dimensional Hilbert spaces and unitaries. This construction, which can be applied to any dagger rig category, is described in three steps, each associated with their own universal property, and draws on results from dilation theory in finite dimension. In this way, we explicitly construct the category that captures hybrid quantum/classical computation with possible nontermination from the category of its reversible foundations. We discuss how this construction can be used in the design and semantics of quantum programming languages.

1 Introduction

The account of quantum measurement offered by *decoherence* establishes that the irreversible nature of mixed-state evolution occurs when a system is considered in isolation from its environment. When the environment is brought back into view, via mathematical techniques such as *quantum state purification* and *Stinespring dilation*, the reversible underpinnings of mixed-state evolution are exposed.

This perspective has in recent years led to the study of quantum theory through categorical completions of its reversible foundations, the category of finite-dimensional Hilbert spaces and unitaries, demonstrating connections between *universal* constructions and *effective* quantum programming [10]. This article constructs in a universal way the category of finite-dimensional C^* -algebras and *partial quantum channels* (completely positive trace-nonincreasing maps) from the rig category of finite-dimensional Hilbert spaces and unitaries. The construction has three stages, each with a universal property of its own.

- Freely allowing *partiality* respecting the dagger structure (by making the additive unit a zero object) allows contractive maps to be described by unitaries through *Halmos dilation* [7, 24, 19].
- Freely allowing the *hiding of states* in a way that *respects partiality* (by making the multiplicative unit *terminal for total maps*) allows completely positive trace-nonincreasing maps to be described through contractions, using a variant of *Stinespring dilation* [28]. This construction has an interesting universal property as a pushout of monoidal categories.
- Freely splitting measurement maps between finite-dimensional Hilbert spaces yields finite-dimensional C^* -algebras, which describe *hybrid quantum/classical* computation.

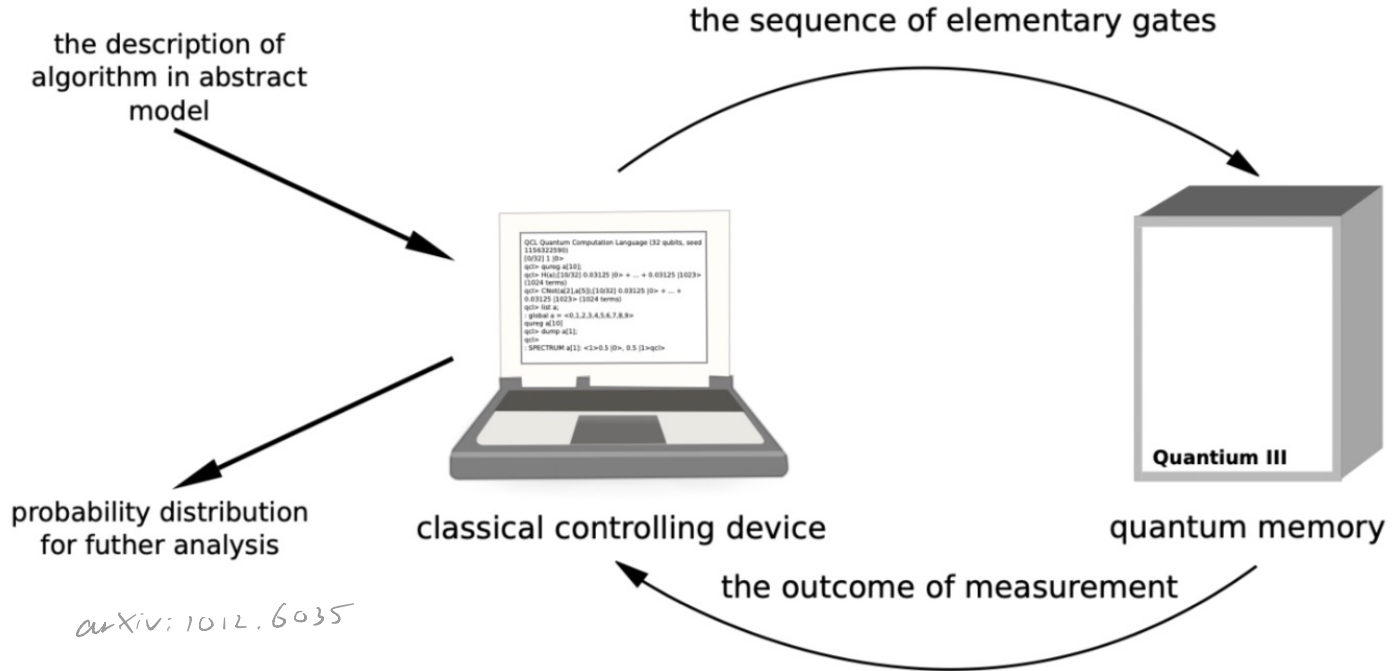
All three universal constructions are abstract and apply to any suitably structured category. They show that the traditional model of C^* -algebras inevitably arises from the mere concepts of quantum circuits, partiality, hiding, and classical communication, without any concept of e.g. norm. Thus they inform the design of quantum programming languages [10], as part of a highly effective broader approach to program semantics from universal properties [29, 15, 26].

¹Supported by EPSRC grant EP/L01503X/1 via the CDT in Pervasive Parallelism
²Supported by EPSRC Fellowship EP/R044759/1
³Supported by DFF / Natural Sciences International Postdoctoral Fellowship 0131-00025B

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

measurement, decoherence, nontermination

Is Quantum capability a computational effect?

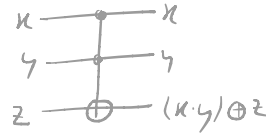


What makes universal (quantum) computing?

classical computing: NAND gate

	0	1
0	1	1
1	1	0

classical reversible computing: TOFFOLI gate



quantum (reversible) computing (Aharonov-Shi):

Toffoli + Hadamard is computationally universal

Π : a reversible combinator language

Syntax

$b ::= 0 \mid 1 \mid b + b \mid b \times b$ (base types)

$t ::= b \leftrightarrow b$ (combinator types)

$a ::= id \mid swap^+ \mid unit^+ \mid uniti^+ \mid assoc^+ \mid associ^+$
 $\mid swap^\times \mid unit^\times \mid uniti^\times \mid assoc^\times \mid associ^\times$

$\mid distrib \mid distribi \mid distribo \mid distriboi$ (primitive combinators)

$c ::= a \mid c \circ c \mid c + c \mid c \times c$ (combinators)

Typing rules

id	:	$b \leftrightarrow b$:	id
$swap^+$:	$b_1 + b_2 \leftrightarrow b_2 + b_1$:	$swap^+$
$unit^+$:	$b + 0 \leftrightarrow b$:	$uniti^+$
$assoc^+$:	$(b_1 + b_2) + b_3 \leftrightarrow b_1 + (b_2 + b_3)$:	$associ^+$
$swap^\times$:	$b_1 \times b_2 \leftrightarrow b_2 \times b_1$:	$swap^\times$
$unit^\times$:	$b \times 1 \leftrightarrow b$:	$uniti^\times$
$assoc^\times$:	$(b_1 \times b_2) \times b_3 \leftrightarrow b_1 \times (b_2 \times b_3)$:	$associ^\times$
$distrib$:	$b_1 \times (b_2 + b_3) \leftrightarrow (b_1 \times b_2) + (b_1 \times b_3)$:	$distribi$
$distribo$:	$b \times 0 \leftrightarrow 0$:	$distriboi$

$$\frac{c_1 : b_1 \leftrightarrow b_2 \quad c_2 : b_2 \leftrightarrow b_3}{c_1 \circ c_2 : b_1 \leftrightarrow b_3}$$

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

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

Π is universal for classical reversible computing;

NOT :: $2 \leftrightarrow 2$

NOT = $swap^+$

$ctrl$:: $b \leftrightarrow b \rightarrow 2 \times b \leftrightarrow 2 \times b$

$ctrl\ f = swap^x \ggg distrib \ggg (unit^x + unit^x) \ggg$

$(id + f) \ggg (uniti^x + uniti^x) \ggg distribi \ggg swap^x$

CNOT :: $2 \times 2 \leftrightarrow 2 \times 2$

CNOT = $ctrl$ NOT

TOFFOLI :: $2 \times (2 \times 2) \leftrightarrow 2 \times (2 \times 2)$

TOFFOLI = $ctrl$ CNOT

Semantics of Π : rig category $(C, \otimes, I, \oplus, 0)$

$$A \otimes (B \oplus C) \simeq (A \otimes B) \oplus (A \otimes C)$$

$$(A \oplus B) \otimes C \simeq (A \otimes C) \oplus (B \otimes C)$$

$$0 \otimes A \simeq 0$$

$$A \otimes 0 \simeq 0$$



e.g. FinBij, Top, Hilb, Unitary

Thm (Elgueta): FinBij is bi-initial in RigCat:

every rig cat C has a rig functor $\text{FinBij} \rightarrow C$

unique up to natural iso

Cor: Π is fully abstract wrt its FinBij-semantics:

$$\llbracket c \rrbracket = \llbracket c' \rrbracket \text{ in FinBij} \iff \llbracket c \rrbracket = \llbracket c' \rrbracket \text{ in any rig cat}$$

So Π is the programming language of rig cats; has TOFFOLI but not Hadamard.

Classical semantics

$$\llbracket \text{NOT} \rrbracket = \text{NOT}$$

$$\llbracket \text{Toffoli} \rrbracket = \text{Toffoli}$$

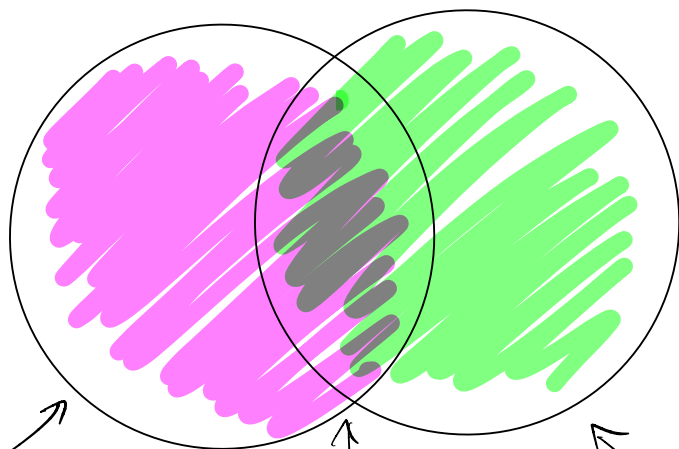
Twisted semantics

for 2×2 unitary M

$$\llbracket C \rrbracket_M = M^{-1} \circ \llbracket C \rrbracket \circ M$$

if $M = R_y(\pi/4)$

then $\llbracket \text{NOT} \rrbracket_M = H$



✓ Teffoli

✗ Hadamard

✗ Toffoli

✓ Hadamard

Need both
for universal quantum computation!

What if we had two languages?

How to bake a Quantum Π

A simple programming language for combining programs written in two other languages

Syntax

$b ::= 0 \mid 1 \mid b + b \mid b \times b$ (base types)

$t ::= b \leftrightarrow b$ (combinator types)

$a ::= id \mid swap^+ \mid unit^+ \mid uniti^+ \mid assoc^+ \mid associ^+$
 $\mid swap^\times \mid unit^\times \mid uniti^\times \mid assoc^\times \mid associ^\times$
 $\mid distrib \mid distribi \mid distribo \mid distriboi$ (primitive combinators)

$c ::= a \mid c \% c \mid c + c \mid c \times c$ (combinators)

Typing rules

id	:	$b \leftrightarrow b$:	id
$swap^+$:	$b_1 + b_2 \leftrightarrow b_2 + b_1$:	$swap^+$
$unit^+$:	$b + 0 \leftrightarrow b$:	$unit^+$
$assoc^+$:	$(b_1 + b_2) + b_3 \leftrightarrow b_1 + (b_2 + b_3)$:	$assoc^+$
$swap^\times$:	$b_1 \times b_2 \leftrightarrow b_2 \times b_1$:	$swap^\times$
$unit^\times$:	$b \times 1 \leftrightarrow b$:	$unit^\times$
$assoc^\times$:	$(b_1 \times b_2) \times b_3 \leftrightarrow b_1 \times (b_2 \times b_3)$:	$assoc^\times$
$distrib$:	$b_1 \times (b_2 + b_3) \leftrightarrow (b_1 \times b_2) + (b_1 \times b_3)$:	$distrib$
$distribo$:	$b \times 0 \leftrightarrow 0$:	$distribo$

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

Syntax

$b ::= 0 \mid 1 \mid b + b \mid b \times b$ (base types)

$t ::= b \leftrightarrow b$ (combinator types)

$a ::= id \mid swap^+ \mid unit^+ \mid uniti^+ \mid assoc^+ \mid associ^+$
 $\mid swap^\times \mid unit^\times \mid uniti^\times \mid assoc^\times \mid associ^\times$
 $\mid distrib \mid distribi \mid distribo \mid distriboi$ (primitive combinators)

$c ::= a \mid c \% c \mid c + c \mid c \times c$ (combinators)

Typing rules

id	:	$b \leftrightarrow b$:	id
$swap^+$:	$b_1 + b_2 \leftrightarrow b_2 + b_1$:	$swap^+$
$unit^+$:	$b + 0 \leftrightarrow b$:	$unit^+$
$assoc^+$:	$(b_1 + b_2) + b_3 \leftrightarrow b_1 + (b_2 + b_3)$:	$assoc^+$
$swap^\times$:	$b_1 \times b_2 \leftrightarrow b_2 \times b_1$:	$swap^\times$
$unit^\times$:	$b \times 1 \leftrightarrow b$:	$unit^\times$
$assoc^\times$:	$(b_1 \times b_2) \times b_3 \leftrightarrow b_1 \times (b_2 \times b_3)$:	$assoc^\times$
$distrib$:	$b_1 \times (b_2 + b_3) \leftrightarrow (b_1 \times b_2) + (b_1 \times b_3)$:	$distrib$
$distribo$:	$b \times 0 \leftrightarrow 0$:	$distribo$

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


Types: same as Π

Terms: $[c_1, c_2, c_3, c_4, \dots]$

Composition: list concatenation

Identities: empty list

Computationally
universal



Semantics of Quantum Π

① Automorphism construction

Let \mathcal{C} be semi-simple rig cat,
and $R: I \oplus I \rightarrow I \oplus I$.

Make $\text{Aut}_R(\mathcal{C})$ with same objects as \mathcal{C}
and conjugated morphisms

Then $\text{Aut}_R(\mathcal{C})$ again rig cat ($\cong \mathcal{C}$)

Unitary gives semantics for Π

$\text{Aut}_{R_4(\pi/4)}(\text{Unitary})$ for Π

② Amalgamation of sym. mon. cats

Let \mathcal{C}, \mathcal{D} be sym. mon. cats w same objs.

Make $\text{Amalg}(\mathcal{C}, \mathcal{D})$ with same objects
and morphisms $[f_1, f_2, f_3, f_4, f_5, f_6, \dots]$
with $\text{cod}(f_i) = \text{dom}(f_{i+1})$ subject to

$$[f_1, \dots, f_n, \text{id}, f_{n+2}, \dots, f_n] \sim [f_1, \dots, f_n, f_{n+2}, \dots, f_n]$$

$$[f_1, \dots, f_n, f_{n+1}, \dots, f_n] \sim [f_1, \dots, f_n \circ f_{n+1}, \dots, f_n]$$

Further identify \otimes in \mathcal{C} and \mathcal{D} .

Then $\text{Amalg}(\mathcal{C}, \mathcal{D})$ is again sym. mon. cat.

$\text{Amalg}(\text{Unitary}, \text{Aut}_{R_4(\pi/4)}(\text{Unitary}))$
gives semantics for Quantum Π

Carefully chosen semantics \rightsquigarrow computationally universal
Can equation about Π and $\overline{\Pi}$ guarantee this?

Prop: Π is unique real, unitary, involutive transformation
between computational basis and mutually unbiased one
up to correction by X and -1

- Add states & effects (as further computational effects), define copy (and $\overline{\text{copy}}$)
- Demand Frobenius: $\Psi = \overline{\Psi}$ $\delta = \overline{\delta}$ $\phi = 1$ $\eta = \overline{\eta}$
- Demand complementarity: $\overline{\delta} \eta = 1$
- Then NOT is involutive transformation between mutually unbiased bases
- One more equation makes NOT real.

Canonicity by complementarity

THEOREM 28 (CANONICITY). *If a categorical semantics $\llbracket - \rrbracket$ for $\langle \Pi \diamond \rangle$ in Contraction satisfies the classical structure laws and the execution laws (defined in Prop. 24) and the complementarity law (Def. 26), then it is computationally universal. Specifically, it must be the semantics of Sec. 7.3 with the semantics of x_ϕ being the Hadamard gate (up to conjugation by X and Z) and:*

$$\llbracket \text{copy}_Z \rrbracket : |i\rangle \mapsto |ii\rangle$$

$$\llbracket \text{zero} \rrbracket = |0\rangle$$

$$\llbracket \text{copy}_X \rrbracket : |\pm\rangle \mapsto |\pm\pm\rangle$$

$$\llbracket \text{assertZero} \rrbracket = \langle 0|$$

up to a global unitary.

Q: What's the effect?

A: $\mathbb{C} \rightarrow \text{Amalg}(\mathbb{C}, \text{Aut}_{\mathbb{R}}(\mathbb{C}))$ is a Freyd category
i.e. a computational effect over the programming language (TT) of \mathbb{C}

c.f. SILQ:

qfree

ad hoc

We use the annotation `qfree` to indicate that evaluating functions or expressions **neither introduces nor destroys superpositions**. Annotation `qfree (i)` ensures that evaluating `qfree` functions on classical arguments yields classical results and (ii) enables automatic uncomputation.

Example 1 (not `qfree`): `H` is not `qfree` as it introduces superpositions: It maps $|0\rangle$ to $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.

Example 2: `X` is `qfree` as it neither introduces nor destroys superpositions: It maps $\sum_{b=0}^1 \gamma_b |b\rangle$ to $\sum_{b=0}^1 \gamma_b |1-b\rangle$.

Example 3: Logical disjunction (as in `x||y`) is of type `const B x const B -> qfree B`, since ORing two values neither introduces nor destroys superpositions.

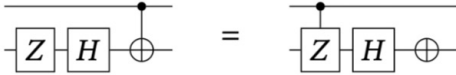
Example 4: Function `myEval` (below) takes a `qfree` function `f` and evaluates it on `false`. Thus, `myEval` itself is also `qfree`.

```
1 def myEval(f: B -> qfree B) qfree {  
2   return f(false); // ^ myEval is qfree  
3 }
```

Q: Can effect system
give more principled
solution?

Reasoning in Quantum TT

Formalised in Agda



```
zhcX : (id<= ** Z) >>> (id<= ** H) >>> cx ≡ cz >>> (id<= ** H) >>> (id<= ** X)
zhcX = begin
  (id<= ** Z) >>> (id<= ** H) >>> cx
  ≡⟨ id≡ ⟩
  (id<= ** (H >>> X >>> H)) >>> (id<= ** H) >>> cx
  ≡⟨ assoc>>>l ∘ (homL*** ∘ (idl>>>l) ∘ id) ⟩§⟨id ⟩
  (id<= ** ((H >>> X >>> H) >>> H)) >>> cx
  ≡⟨ id⟩∗⟨ pullr (cancelr hadInv) ⟩§⟨id ⟩
  id<= ** (H >>> X) >>> cx
  ≡⟨ (idl>>>r) ∘ id ∘ homR*** ⟩§⟨id ∘ assoc>>>r ⟩
  (id<= ** H) >>> (id<= ** X) >>> cx
  ≡⟨ id⟩§⟨ xcxA ⟩
  (id<= ** H) >>> cx >>> (id<= ** X)
  ≡⟨ id⟩§⟨ id⟩∗⟨ insertt l*HInv ⟩
  (id<= ** H) >>> cx >>> (id<= ** H) >>> (id<= ** H) >>> (id<= ** X)
  ≡⟨ assoc>>>l ∘ assoc>>>l ∘ assoc>>>r ⟩§⟨id ⟩
  (id<= ** H >>> cx >>> id<= ** H) >>> (id<= ** H) >>> (id<= ** X)
  ≡⟨ id≡ ⟩
  cz >>> (id<= ** H) >>> (id<= ** X) ■
```


"If quantum mechanics hasn't profoundly shocked you, you haven't understood it yet"



Two classical programming languages and a couple of equations

