A Functional Quantum Programming Language

Thorsten Altenkirch

University of Nottingham

based on joint work with Jonathan Grattage

and discussions with V.P. Belavkin

supported by EPSRC grant GR/S30818/01

Alternative title

What you always wanted to know about quantum computation but never dared to ask.

Another alternative title

Quantum programming

for the

lazy functional programmer

Background

Background

- Simulation of quantum systems is expensive: Exponential time to simulate polynomial circuits.

Background

- Simulation of quantum systems is expensive: Exponential time to simulate polynomial circuits.
- Feynman: *Can we exploit this fact to perform computations more efficiently?*

Background

- Simulation of quantum systems is expensive: Exponential time to simulate polynomial circuits.
- Feynman: *Can we exploit this fact to perform computations more efficiently?*
- Shor: Factorisation in quantum polynomial time.

Background

- Simulation of quantum systems is expensive: Exponential time to simulate polynomial circuits.
- Feynman: *Can we exploit this fact to perform computations more efficiently?*
- Shor: Factorisation in quantum polynomial time.
- Grover: Blind search in $\Theta(\sqrt{n})$

Background

- Simulation of quantum systems is expensive: Exponential time to simulate polynomial circuits.
- Feynman: *Can we exploit this fact to perform computations more efficiently?*
- Shor: Factorisation in quantum polynomial time.
- Grover: Blind search in $\Theta(\sqrt{n})$
- Can we build a quantum computer?

Background

- Simulation of quantum systems is expensive: Exponential time to simulate polynomial circuits.
- Feynman: *Can we exploit this fact to perform computations more efficiently?*
- Shor: Factorisation in quantum polynomial time.
- Grover: Blind search in $\Theta(\sqrt{n})$
- Can we build a quantum computer?

  yes We can run quantum algorithms.

Background

- Simulation of quantum systems is expensive: Exponential time to simulate polynomial circuits.
- Feynman: *Can we exploit this fact to perform computations more efficiently?*
- Shor: Factorisation in quantum polynomial time.
- Grover: Blind search in $\Theta(\sqrt{n})$
- Can we build a quantum computer?

yes We can run quantum algorithms.

no Nature is classical after all!

Background

- Simulation of quantum systems is expensive: Exponential time to simulate polynomial circuits.
- Feynman: *Can we exploit this fact to perform computations more efficiently?*
- Shor: Factorisation in quantum polynomial time.
- Grover: Blind search in $\Theta(\sqrt{n})$
- Can we build a quantum computer?

yes We can run quantum algorithms.

no Nature is classical after all!

Assumption: Nature is fair. . .

The quantum software crisis

The quantum software crisis

- Quantum algorithms are usually presented using the circuit model.

The quantum software crisis

- Quantum algorithms are usually presented using the circuit model.
- Nielsen and Chuang, p.7, *Coming up with good quantum algorithms is hard.*

The quantum software crisis

- Quantum algorithms are usually presented using the circuit model.
- Nielsen and Chuang, p.7, *Coming up with good quantum algorithms is hard.*
- Richard Josza, QPL 2004: *We need to develop quantum thinking!*

QML

QML

- QML: a first-order functional language for quantum computations on finite types.

QML

- QML: a first-order functional language for quantum computations on finite types.
- Quantum control **and** quantum data.

QML

- QML: a first-order functional language for quantum computations on finite types.
- Quantum control **and** quantum data.
- Design guided by semantics

QML

- QML: a first-order functional language for quantum computations on finite types.
- Quantum control **and** quantum data.
- Design guided by semantics
- Analogy with classical computation

$\mathbf{FCC}$	Finite classical computations
$\mathbf{FQC}$	Finite quantum computations

QML

- QML: a first-order functional language for quantum computations on finite types.
- Quantum control **and** quantum data.
- Design guided by semantics
- Analogy with classical computation

FCC	Finite classical computations
FQC	Finite quantum computations

- Important issue: **control of decoherence**

QML

- QML: a first-order functional language for quantum computations on finite types.
- Quantum control **and** quantum data.
- Design guided by semantics
- Analogy with classical computation

  FCC Finite classical computations

  FQC Finite quantum computations
- Important issue: **control of decoherence**
- Compiler under construction (Jonathan)

Example: Hadamard operation

Example: Hadamard operation

Matrix

$$\mathrm{H} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Example: Hadamard operation

Matrix

$$\mathrm{H} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

QML

$$\begin{aligned} & had : Q_2 \multimap Q_2 \\ & had\ x = \mathbf{if}^{\circ}\ x \\ & \qquad \mathbf{then}\ \{\mathrm{qfalse} \mid (-1)\ \mathrm{qtrue}\} \\ & \qquad \mathbf{else}\ \ \{\mathrm{qfalse} \mid \mathrm{qtrue}\} \end{aligned}$$

Something we know well ...

Something we know well . . .

- Classical computations on finite types.

Something we know well . . .

- Classical computations on finite types.
- Quantum mechanics is time-reversible. . .

Something we know well . . .

- Classical computations on finite types.
- Quantum mechanics is time-reversible. . .
- . . . hence quantum computation is based on reversible operations.

Something we know well . . .

- Classical computations on finite types.
- Quantum mechanics is time-reversible. . .
- . . . hence quantum computation is based on reversible operations.
- **However:** Newtonian mechanics, Maxwellian electrodynamics are also time-reversible. . .

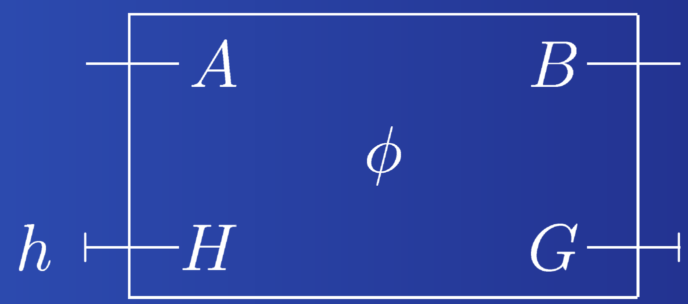
Something we know well . . .

- Classical computations on finite types.
- Quantum mechanics is time-reversible. . .
- . . . hence quantum computation is based on reversible operations.
- **However:** Newtonian mechanics, Maxwellian electrodynamics are also time-reversible. . .
- . . . hence classical computation **should be** based on reversible operations.

Classical computation (FCC)

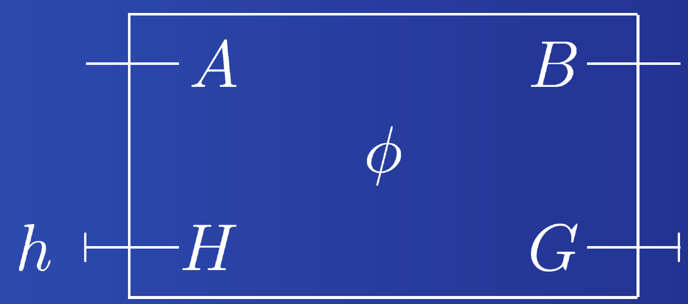
Classical computation (FCC)

Given finite sets A (input) and B (output):



Classical computation (FCC)

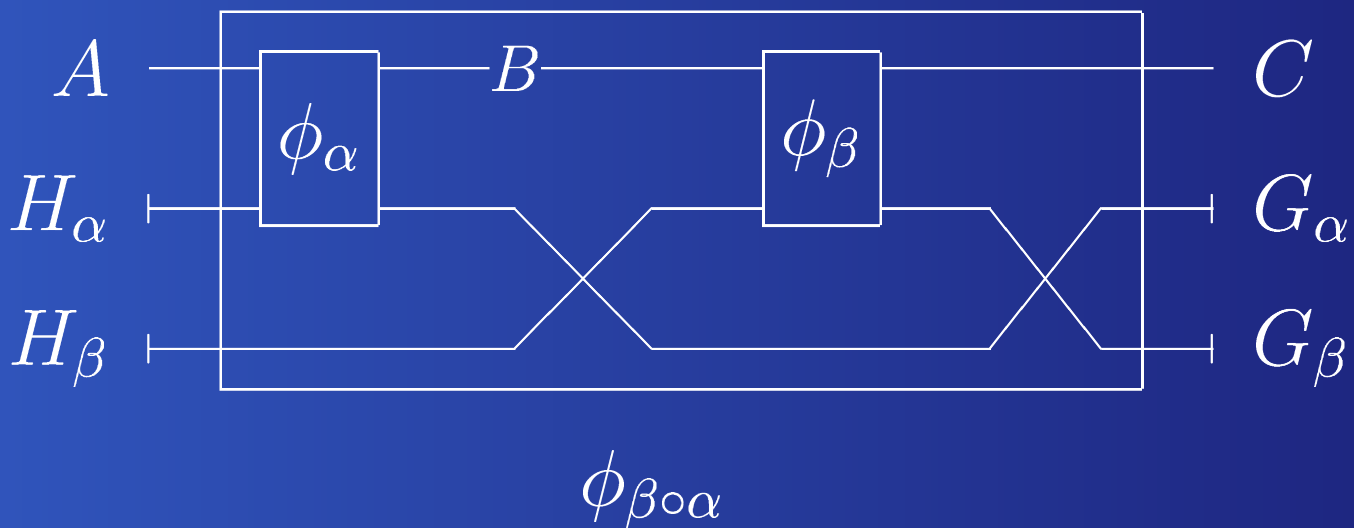
Given finite sets A (input) and B (output):



- a finite set of initial heaps H,
- an initial heap $h \in H$,
- a finite set of garbage states G,
- a bijection $\phi \in A \times H \simeq B \times G$,

Composing computations

Composing computations



Extensional equality

Extensional equality

- A classical computation $\alpha = (H, h, G, \phi)$ induces a function $\mathsf{U}\alpha \in A \to B$ by

$$\begin{tikzcd}
A \times H \arrow[r, "\phi"'] & B \times G \arrow[d, "\pi_1"] \\
A \arrow[u, "(-{,}h)"] \arrow[r, "\mathsf{U}\,\alpha"'] & B
\end{tikzcd}$$

Extensional equality

- A classical computation $\alpha = (H, h, G, \phi)$ induces a function $\mathsf{U}\alpha \in A \to B$ by

$$\begin{tikzcd} A \times H \arrow[r, "\phi"'] & B \times G \arrow[d, "\pi_1"] \\ A \arrow[u, "{(-,h)}"'] \arrow[r, "\mathsf{U}\,\alpha"'] & B \end{tikzcd}$$

- We say that two computations are **extensionally equivalent**, if they give rise to the same function.

Extensional equality ...

- **Theorem:**

$$\mathbf{U}\,(\beta \circ \alpha) = (\mathbf{U}\,\beta) \circ (\mathbf{U}\,\alpha)$$

Extensional equality . . .

- **Theorem:**

$$\mathbf{U}\,(\beta \circ \alpha) = (\mathbf{U}\,\beta) \circ (\mathbf{U}\,\alpha)$$

- **Hence, classical computations upto extensional equality give rise to the category FCC.**

Extensional equality . . .

- **Theorem:**

$$\mathsf{U}\,(\beta \circ \alpha) = (\mathsf{U}\,\beta) \circ (\mathsf{U}\,\alpha)$$

- **Hence, classical computations upto extensional equality give rise to the category FCC.**
- **Theorem: Any function $f \in A \to B$ on finite sets A, B can be realized by a computation.**

Extensional equality ...

- **Theorem:**

$$\mathbf{U}\,(\beta \circ \alpha) = (\mathbf{U}\,\beta) \circ (\mathbf{U}\,\alpha)$$

- **Hence, classical computations upto extensional equality give rise to the category FCC.**
- **Theorem: Any function $f \in A \to B$ on finite sets A, B can be realized by a computation.**
- ***Translation for Category Theoreticans:***
  U is full and faithful.

Example π_1 :

function

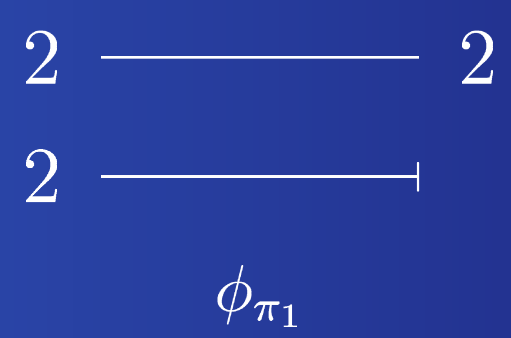
$$\pi_1 \in (2,2) \to 2$$
$$\pi_1\ (x,y) = x$$

Example π_1 :

function

$$\pi_1 \in (2, 2) \to 2$$
$$\pi_1\ (x, y) = x$$

computation



Example δ :

function

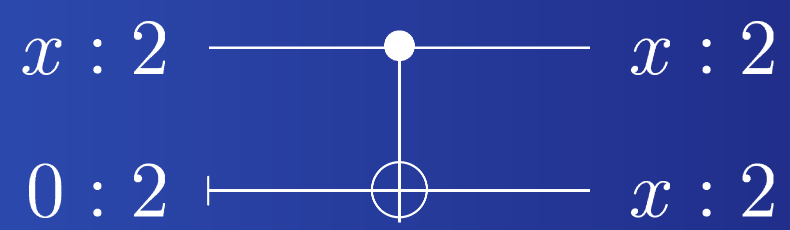
$$\delta \in 2 \to (2, 2)$$
$$\delta\ x = (x, x)$$

Example δ :

function

$$\delta \in 2 \to (2,2)$$
$$\delta\ x = (x, x)$$

computation



ϕ_δ

$$\phi_\delta \in (2,2) \to (2,2)$$
$$\phi_\delta\ (0, x) = (0, x)$$
$$\phi_\delta\ (1, x) = (1, \neg\ x)$$

Classical vs quantum

Classical vs quantum

classical ($\mathbf{FCC}$)	quantum ($\mathbf{FQC}$)

Classical vs quantum

classical ($\mathbf{FCC}$)	quantum ($\mathbf{FQC}$)
finite sets	

Classical vs quantum

classical ($\mathbf{FCC}$)	quantum ($\mathbf{FQC}$)
finite sets	finite dimensional Hilbert spaces

Classical vs quantum

classical ($\mathbf{FCC}$)	quantum ($\mathbf{FQC}$)
finite sets	finite dimensional Hilbert spaces
bijections	

Classical vs quantum

classical ($\mathbf{FCC}$)	quantum ($\mathbf{FQC}$)
finite sets	finite dimensional Hilbert spaces
bijections	unitary operators

Classical vs quantum

classical ($\mathbf{FCC}$)	quantum ($\mathbf{FQC}$)
finite sets	finite dimensional Hilbert spaces
bijections	unitary operators
cartesian product ($\times$)	

Classical vs quantum

classical ($\mathbf{FCC}$)	quantum ($\mathbf{FQC}$)
finite sets	finite dimensional Hilbert spaces
bijections	unitary operators
cartesian product ($\times$)	tensor product ($\otimes$)

Classical vs quantum

classical ($\mathbf{FCC}$)	quantum ($\mathbf{FQC}$)
finite sets	finite dimensional Hilbert spaces
bijections	unitary operators
cartesian product ($\times$)	tensor product ($\otimes$)
functions	

Classical vs quantum

classical ($\mathbf{FCC}$)	quantum ($\mathbf{FQC}$)
finite sets	finite dimensional Hilbert spaces
bijections	unitary operators
cartesian product ($\times$)	tensor product ($\otimes$)
functions	superoperators

Classical vs quantum

classical ($\mathbf{FCC}$)	quantum ($\mathbf{FQC}$)
finite sets	finite dimensional Hilbert spaces
bijections	unitary operators
cartesian product ($\times$)	tensor product ($\otimes$)
functions	superoperators
projections	

Classical vs quantum

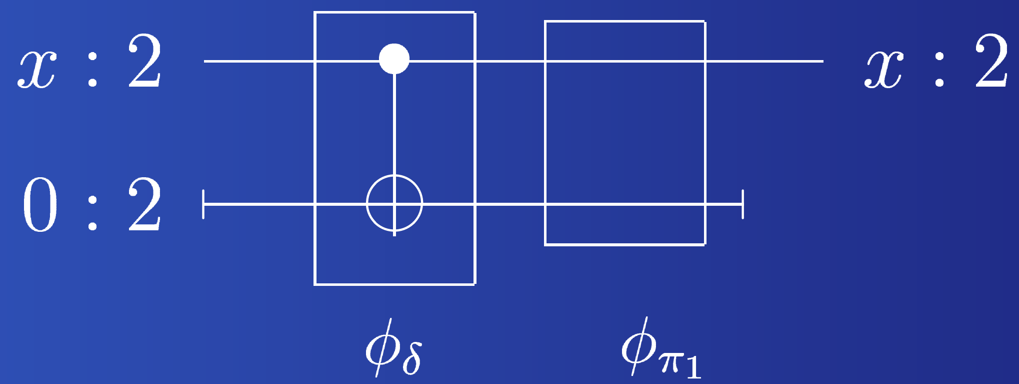
classical (**FCC**)	quantum (**FQC**)
finite sets	finite dimensional Hilbert spaces
bijections	unitary operators
cartesian product ($\times$)	tensor product ($\otimes$)
functions	superoperators
projections	partial trace

$\pi_1 \circ \delta$, classically

$$\pi_1 \circ \delta : 2 \to 2$$

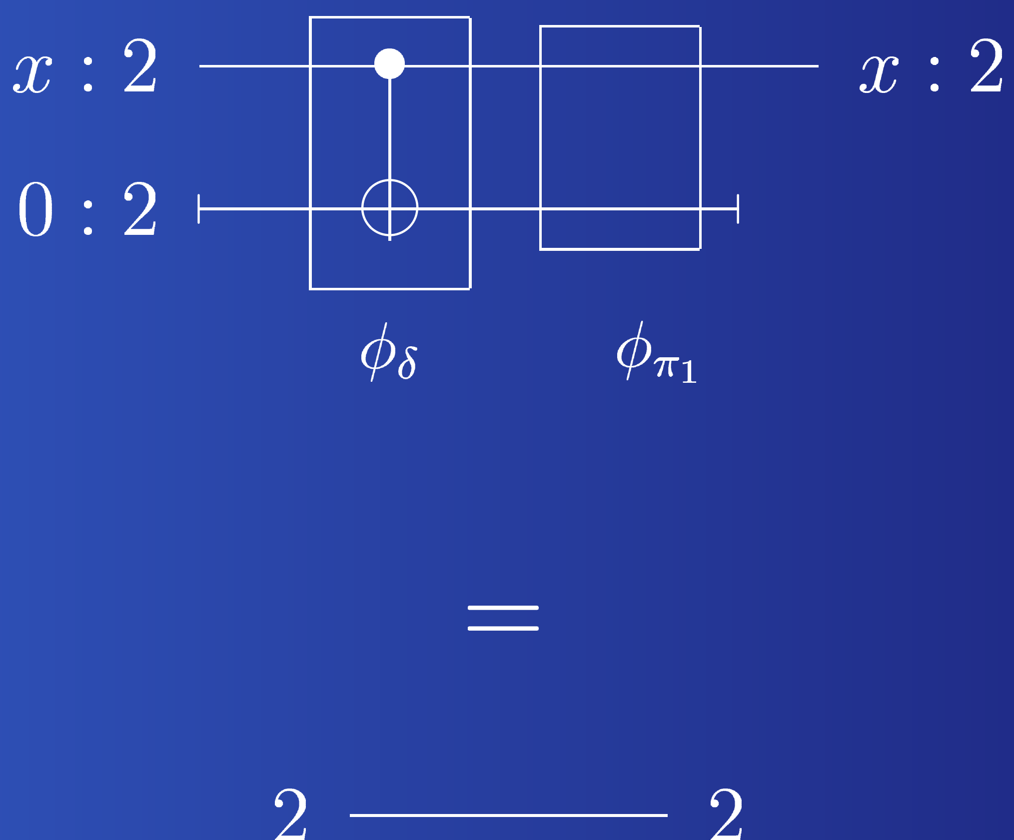
$\pi_1 \circ \delta$, classically

$$\pi_1 \circ \delta : 2 \to 2$$

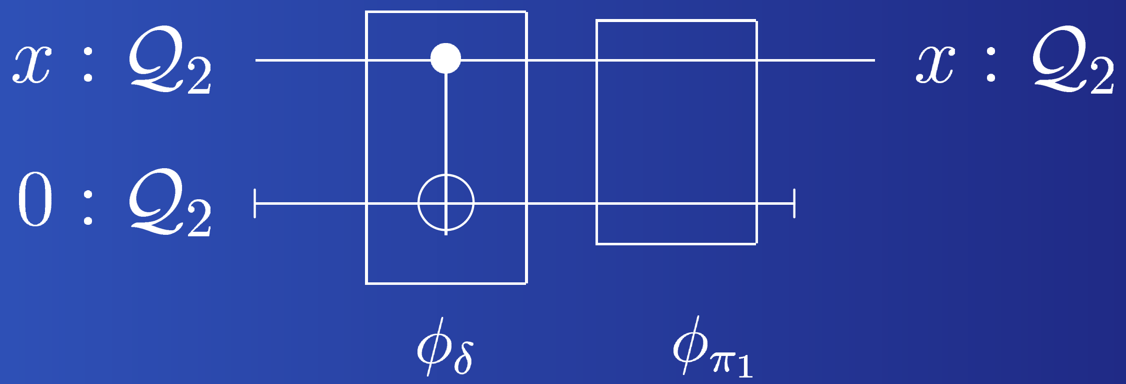


$\pi_1 \circ \delta$, classically

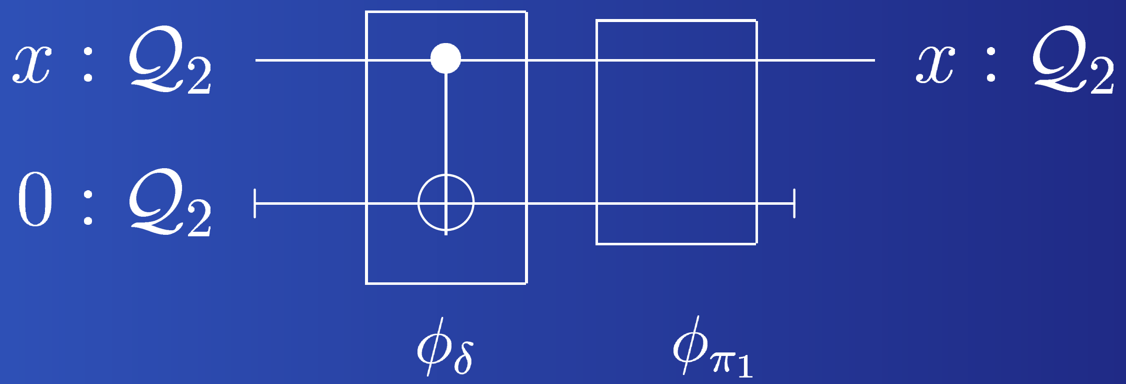
$$\pi_1 \circ \delta : 2 \to 2$$



$\pi_1 \circ \delta$, quantum

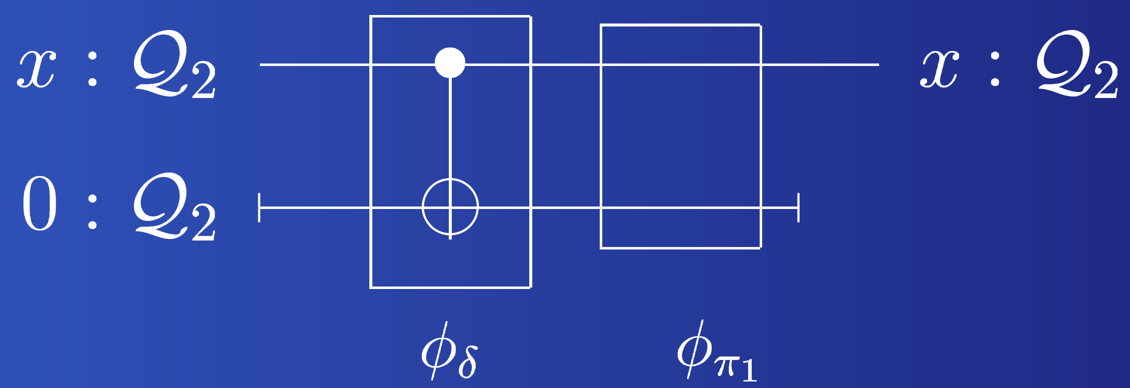


$\pi_1 \circ \delta$, quantum



input: $\left\{\frac{1}{\sqrt{2}}\left|0\right\rangle + \frac{1}{\sqrt{2}}\left|1\right\rangle\right\}$

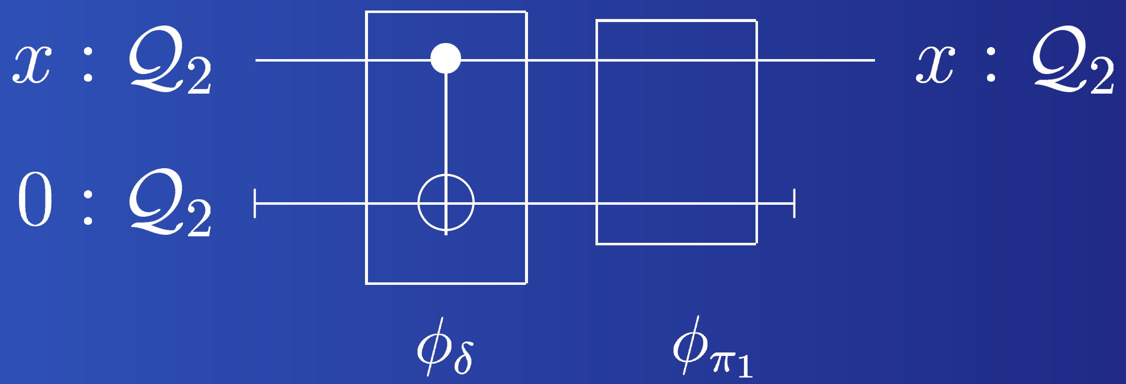
$\pi_1 \circ \delta$, quantum



input: $\{\frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle\}$

output: $\frac{1}{2}\{|0\rangle\} + \frac{1}{2}\{|1\rangle\}$

$\pi_1 \circ \delta$, quantum



input: $\{\frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle\}$

output: $\frac{1}{2}\{|0\rangle\} + \frac{1}{2}\{|1\rangle\}$

Decoherence!

Control of decoherence

Control of decoherence

- QML is based on strict linear logic

Control of decoherence

- QML is based on strict linear logic
- Contraction is implicit and realized by ϕ_δ.

Control of decoherence

- QML is based on strict linear logic
- Contraction is implicit and realized by ϕ_δ.
- Weakening is explicit and leads to decoherence.

QML overview

QML overview

Types

$$\sigma = 1 \mid \sigma \otimes \tau \mid \sigma \oplus \tau$$

QML overview

Types

$$\sigma = 1 \mid \sigma \otimes \tau \mid \sigma \oplus \tau$$

Terms

$$\begin{aligned} t &= x \mid \mathbf{let}\ x = t\ \mathbf{in}\ u \mid x \uparrow xs \\ &\mid () \mid (t, u) \mid \mathbf{let}\ (x, y) = t\ \mathbf{in}\ u \\ &\mid \mathrm{qinl}\ t \mid \mathrm{qinr}\ u \\ &\mid \mathbf{case}\ t\ \mathbf{of}\ \{\mathrm{qinl}\ x \Rightarrow u \mid \mathrm{qinr}\ y \Rightarrow u'\} \\ &\mid \mathbf{case}^\circ\ t\ \mathbf{of}\ \{\mathrm{qinl}\ x \Rightarrow u \mid \mathrm{qinr}\ y \Rightarrow u'\} \\ &\mid \{(\kappa)\ t \mid (\iota)\ u\} \end{aligned}$$

Qbits

$$Q_2 = 1 \oplus 1$$

$$\mathrm{qtrue} = \mathrm{qinl}\ ()$$

$$\mathrm{qfalse} = \mathrm{qinr}\ ()$$

$$\mathbf{if}\ t\ \mathbf{then}\ u\ \mathbf{else}\ u'$$

$$= \mathbf{case}\ \{\mathrm{qinl}\ \_ \Rightarrow u \mid \mathrm{qinr}\ \_ \Rightarrow u'\}$$

$$\mathbf{if}^{\circ}\ t\ \mathbf{then}\ u\ \mathbf{else}\ u'$$

$$= \mathbf{case}^{\circ}\{\mathrm{qinl}\ \_ \Rightarrow u \mid \mathrm{qinr}\ \_ \Rightarrow u'\}$$

QML overview . . .

QML overview . . .

Typing judgements

$\Gamma \vdash t : \sigma$	programs
$\Gamma \vdash^{\circ} t : \sigma$	strict programs

QML overview . . .

Typing judgements

$\Gamma \vdash t : \sigma$ programs

$\Gamma \vdash^\circ t : \sigma$ strict programs

Semantics

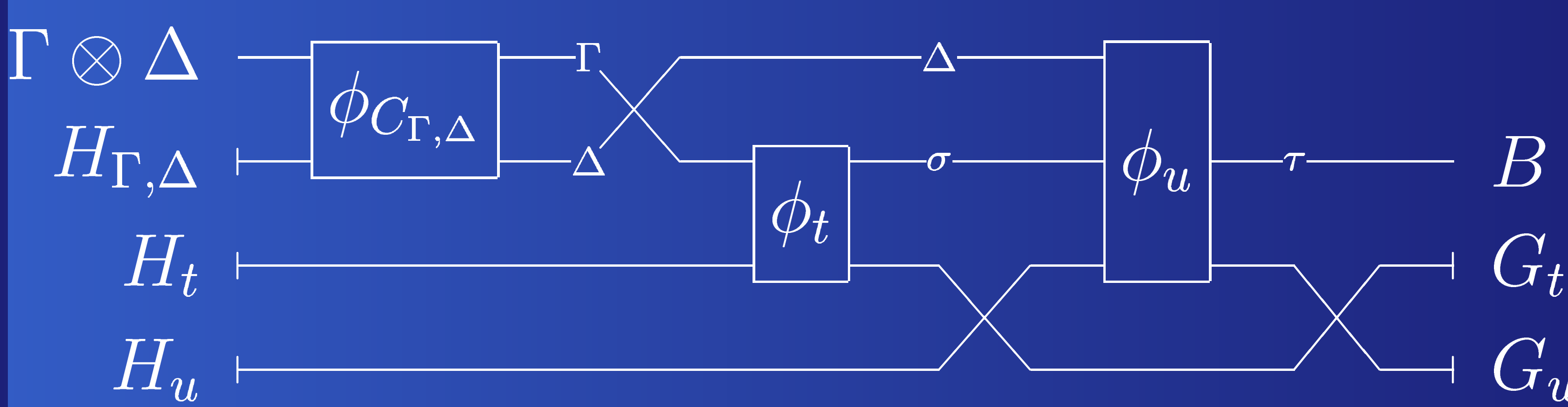
$$\frac{\Gamma \vdash t : \sigma}{[\![t]\!] \in \mathbf{FQC}[\![\Gamma]\!][\![\sigma]\!]} \qquad \frac{\Gamma \vdash^\circ t : \sigma}{[\![t]\!] \in \mathbf{FQC}^\circ[\![\Gamma]\!][\![\sigma]\!]}$$

The let-rule

$$\frac{\Gamma \vdash t : \sigma \qquad \Delta, x : \sigma \vdash u : \tau}{\Gamma \otimes \Delta \vdash \mathtt{let}\ x = t\ \mathtt{in}\ u : \tau}\ \text{let}$$

The let-rule

$$\dfrac{\begin{array}{c}\Gamma \vdash t : \sigma \\ \Delta, x : \sigma \vdash u : \tau\end{array}}{\Gamma \otimes \Delta \vdash \texttt{let}\ x = t\ \texttt{in}\ u : \tau}\ \text{let}$$



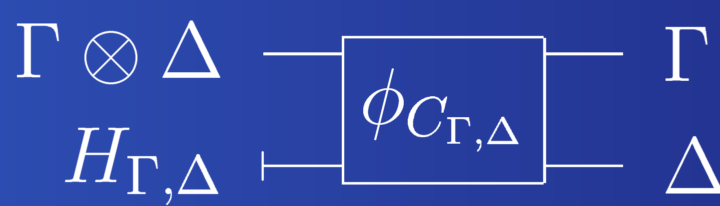
$\otimes$ on contexts

$\otimes$ on contexts

$$
\begin{array}{lcll}
\Gamma, x:\sigma \otimes \Delta, x:\sigma & = & (\Gamma \otimes \Delta), x:\sigma & \\
\Gamma, x:\sigma \otimes \Delta & = & (\Gamma \otimes \Delta), x:\sigma & \text{if } x \notin \operatorname{dom} \Delta \\
\bullet \otimes \Delta & = & \Delta &
\end{array}
$$

$\otimes$ on contexts

$$
\begin{aligned}
\Gamma, x:\sigma \otimes \Delta, x:\sigma &= (\Gamma \otimes \Delta), x:\sigma \\
\Gamma, x:\sigma \otimes \Delta &= (\Gamma \otimes \Delta), x:\sigma \quad \text{if } x \notin \mathrm{dom}\,\Delta \\
\bullet \otimes \Delta &= \Delta
\end{aligned}
$$



Another source of decoherence

Another source of decoherence

- *forget* mentions x

$$forget : 2 \multimap 2$$

$$forget\ x = \mathbf{if}\ x\ \mathbf{then}\ \mathrm{qtrue}\ \mathbf{else}\ \mathrm{qtrue}$$

Another source of decoherence

- $forget$ mentions x

$$forget : 2 \multimap 2$$

$$forget\ x = \mathbf{if}\ x\ \mathbf{then}\ \mathrm{qtrue}\ \mathbf{else}\ \mathrm{qtrue}$$

- but doesn't use it.

Another source of decoherence

- *forget* mentions x

$$forget : 2 \multimap 2$$

$$forget\ x = \mathbf{if}\ x\ \mathbf{then}\ \mathrm{qtrue}\ \mathbf{else}\ \mathrm{qtrue}$$

- but doesn't use it.
- Hence, it **has** to measure it!

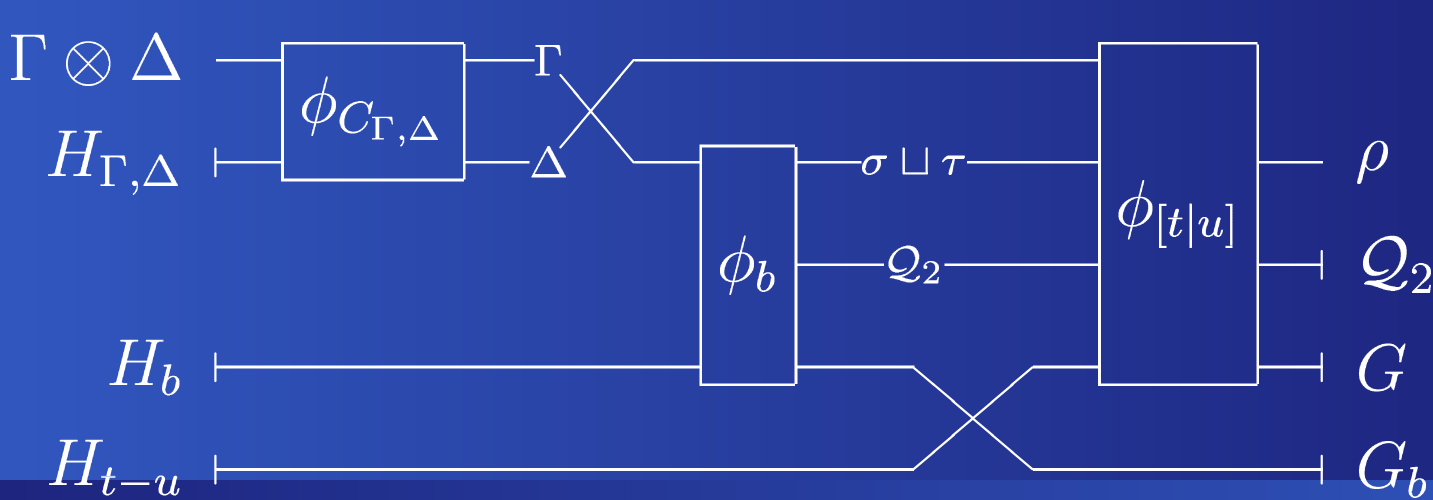
$\oplus$-elim

$\oplus$-elim

$$\frac{\Gamma \vdash c : \sigma \oplus \tau \qquad \Delta, x : \sigma \vdash t : \rho \qquad \Delta, y : \tau \vdash u : \rho}{\Gamma \otimes \Delta \vdash \texttt{case}\ c\ \texttt{of}\ \{\texttt{inl}\ x \Rightarrow t \,|\, \texttt{inr}\ y \Rightarrow u\} : \rho} + \mathrm{elim}$$

$\bigoplus$-elim

$$\frac{\Gamma \vdash c : \sigma \bigoplus \tau \qquad \Delta, x : \sigma \vdash t : \rho \qquad \Delta, y : \tau \vdash u : \rho}{\Gamma \otimes \Delta \vdash \texttt{case}\ c\ \texttt{of}\ \{\texttt{inl}\ x \Rightarrow t \,|\, \texttt{inr}\ y \Rightarrow u\} : \rho} + \mathrm{elim}$$



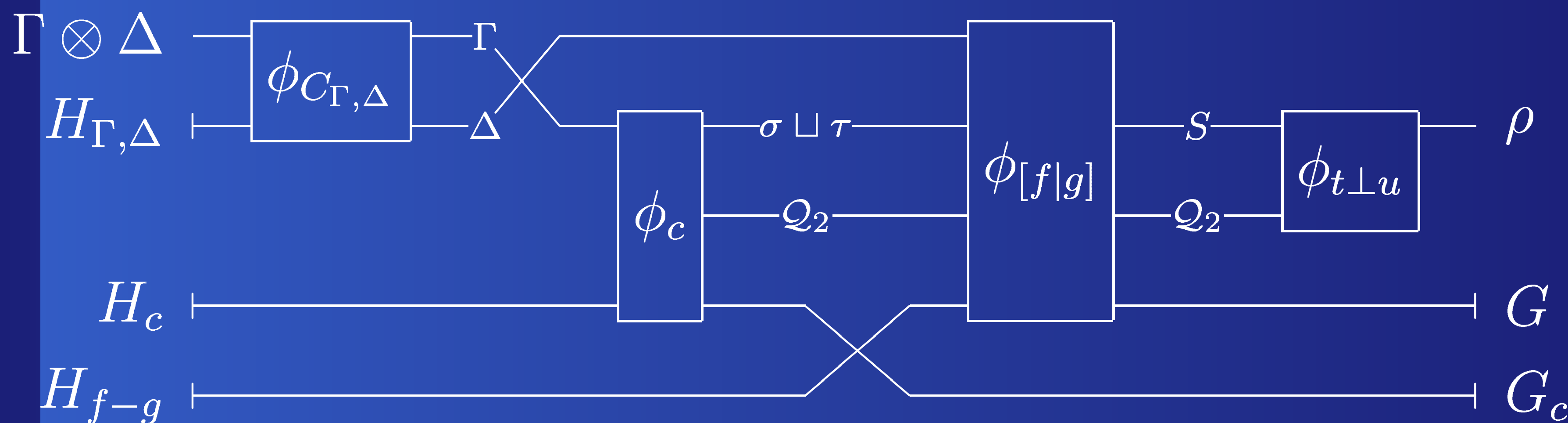
$\oplus$-elim decoherence-free

$\bigoplus$-elim decoherence-free

$$\frac{\Gamma \vdash^{a} c : \sigma \oplus \tau \qquad \Delta,\ x : \sigma \vdash^{\circ} t : \rho \qquad \Delta,\ y : \tau \vdash^{\circ} u : \rho \quad t \perp u}{\Gamma \otimes \Delta \vdash^{a} \texttt{case}^{\circ}\ \ c\ \texttt{of}\ \ \{\texttt{inl}\ x \Rightarrow t \mid \texttt{inr}\ y \Rightarrow u\} : \rho} \bigoplus - \mathrm{elim}^{\circ}$$

$\bigoplus$-elim decoherence-free

$$\frac{\Gamma \vdash^{a} c : \sigma \bigoplus \tau \qquad \Delta,\ x : \sigma \vdash^{\circ} t : \rho \qquad \Delta,\ y : \tau \vdash^{\circ} u : \rho \quad t \perp u}{\Gamma \otimes \Delta \vdash^{a} \texttt{case}^{\circ}\ c\ \texttt{of}\ \{\texttt{inl}\ x \Rightarrow t \mid \texttt{inr}\ y \Rightarrow u\} : \rho} \bigoplus - \mathrm{elim}^{\circ}$$



if°

$\mathbf{if}^\circ$

- $$forget' : 2 \multimap 2$$
  $$forget'\ x = \mathbf{if}^\circ\ x\ \mathbf{then}\ \mathrm{qtrue}\ \mathbf{else}\ \mathrm{qtrue}$$

$\mathbf{if}^\circ$

- $$\begin{aligned} &\mathit{forget}' : 2 \multimap 2 \\ &\mathit{forget}'\ x = \mathbf{if}^\circ\ x\ \mathbf{then}\ \mathrm{qtrue}\ \mathbf{else}\ \mathrm{qtrue} \end{aligned}$$
- This program has a type error, because `qtrue` $\not\perp$ `qtrue`.

if$^\circ$

- $$forget' : 2 \multimap 2$$
  $$forget'\ x = \mathbf{if}^\circ\ x\ \mathbf{then}\ \mathrm{qtrue}\ \mathbf{else}\ \mathrm{qtrue}$$
- This program has a type error, because `qtrue` $\not\perp$ `qtrue`.
- $$qnot : 2 \multimap 2$$
  $$qnot\ x = \mathbf{if}\ x\ \mathbf{then}\ \mathrm{qfalse}\ \mathbf{else}\ \mathrm{qtrue}$$

$\mathbf{if}^\circ$

- $$forget' : 2 \multimap 2$$
  $$forget'\ x = \mathbf{if}^\circ\ x\ \mathbf{then}\ \mathrm{qtrue}\ \mathbf{else}\ \mathrm{qtrue}$$
- This program has a type error, because `qtrue` $\not\perp$ `qtrue`.
- $$qnot : 2 \multimap 2$$
  $$qnot\ x = \mathbf{if}\ x\ \mathbf{then}\ \mathrm{qfalse}\ \mathbf{else}\ \mathrm{qtrue}$$
- This program typechecks, because `qfalse` $\perp$ `qtrue`.

Conclusions

Conclusions

- Our semantic ideas proved useful when designing a quantum programming language, analogous concepts are modelled by the same syntactic constructs.

Conclusions

- Our semantic ideas proved useful when designing a quantum programming language, analogous concepts are modelled by the same syntactic constructs.
- Our analysis also highlights the differences between classical and quantum programming.

Conclusions

- Our semantic ideas proved useful when designing a quantum programming language, analogous concepts are modelled by the same syntactic constructs.
- Our analysis also highlights the differences between classical and quantum programming.
- Quantum programming introduces the problem of *control of decoherence*, which we address by making forgetting variables explicit and by having different if-then-else constructs.

Further work

Further work

- We have to analyze more quantum programs to evaluate the practical usefulness of our approach.

Further work

- We have to analyze more quantum programs to evaluate the practical usefulness of our approach.
- Are we able to come up with completely new algorithms using QML?

Further work

- We have to analyze more quantum programs to evaluate the practical usefulness of our approach.
- Are we able to come up with completely new algorithms using QML?
- How to deal with higher order programs?

Further work

- We have to analyze more quantum programs to evaluate the practical usefulness of our approach.
- Are we able to come up with completely new algorithms using QML?
- How to deal with higher order programs?
- How to deal with infinite datatypes?

Further work

- We have to analyze more quantum programs to evaluate the practical usefulness of our approach.
- Are we able to come up with completely new algorithms using QML?
- How to deal with higher order programs?
- How to deal with infinite datatypes?
- Investigate the similarities/differences between FCC and FQC from a categorical point of view.

The end

Thank you for your attention.