Functional Quantum Programming

Thorsten Altenkirch

University of Nottingham

based on joint work with Jonathan Grattage

supported by EPSRC grant GR/S30818/01

Background

Background

- Simulation of quantum systems is expensive: PSPACE complexity for polynomial circuits.

Background

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

Background

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

Background

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

Background

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

yes We can run quantum algorithms.

Background

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

yes We can run quantum algorithms.

no Nature is classical after all!

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 functional language for quantum computations on finite types.

QML

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

QML

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

QML

- QML: a 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 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 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

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

Deutsch algorithm

$$deutsch : 2 \multimap 2 \multimap Q_2$$

$$
\begin{array}{l}
deutsch\ a\ b = \\
\quad \textbf{let}\ (x, y) = \textbf{if}^\circ \{\,\mathrm{qfalse} \mid \mathrm{qtrue}\,\} \\
\qquad\qquad\qquad \textbf{then}\ (\mathrm{qtrue}, \textbf{if}\ a \\
\qquad\qquad\qquad\qquad\qquad \textbf{then}\ \{\,\mathrm{qfalse} \mid (-1)\ \mathrm{qtrue}\,\} \\
\qquad\qquad\qquad\qquad\qquad \textbf{else}\ \ \{\,(-1)\ \mathrm{qfalse} \mid \mathrm{qtrue}\,\} \\
\qquad\qquad\qquad \textbf{else}\ (\mathrm{qfalse},\ \textbf{if}\ b \\
\qquad\qquad\qquad\qquad\qquad \textbf{then}\ \{\,(-1)\ \mathrm{qfalse} \mid \mathrm{qtrue}\,\} \\
\qquad\qquad\qquad\qquad\qquad \textbf{else}\ \ \{\,\mathrm{qfalse} \mid (-1)\ \mathrm{qtrue}\,\} \\
\quad \textbf{in}\ H\ x
\end{array}
$$

Overview

1. Finite classical computation
2. Finite quantum computation
3. QML basics
4. Compiling QML
5. Conclusions and further work

1. Semantics

1. Finite classical computation
2. Finite quantum computation
3. QML basics
4. Compiling QML
5. Conclusions and further work

Something we know well . . .

Something we know well . . .

- Start with classical computations on finite types.

Something we know well . . .

- Start with classical computations on finite types.
- Quantum mechanics is time-reversible. . .

Something we know well . . .

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

Something we know well . . .

- Start with 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 . . .

- Start with 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):

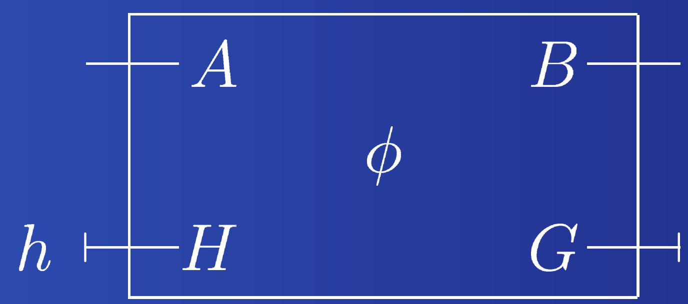
Classical computation (FCC)

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

- a finite set of initial heaps H,

Classical computation (FCC)

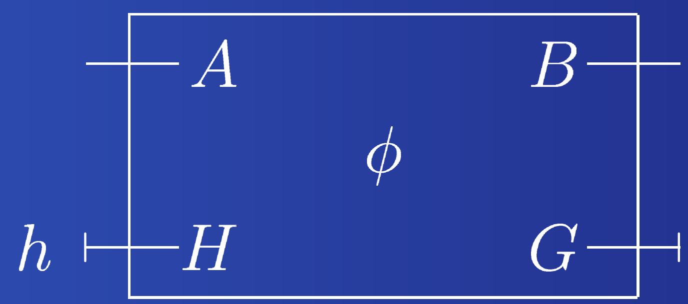
Given finite sets A (input) and B (output):



- a finite set of initial heaps H,
- an initial heap $h \in H$,

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,

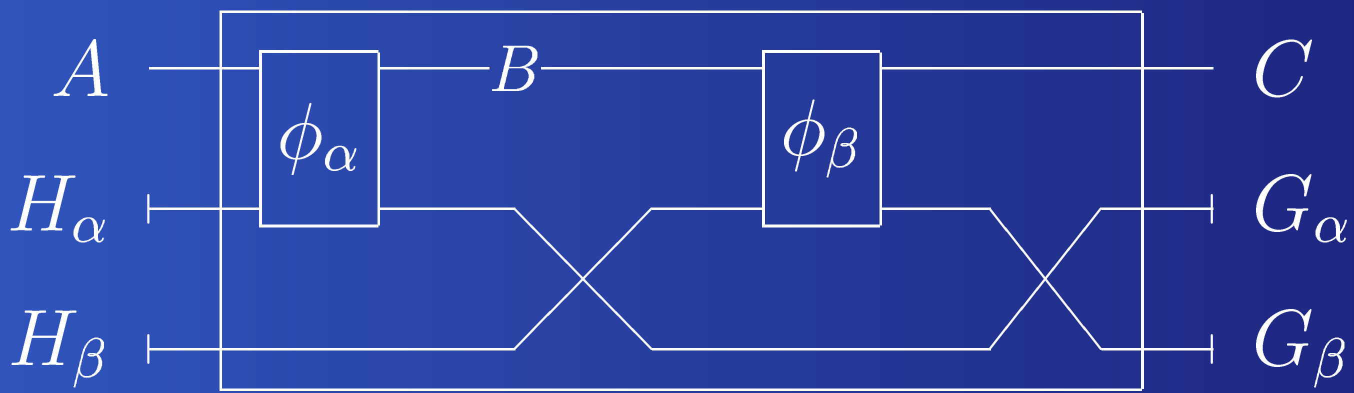
Classical computation (FCC)

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 $\phi \in A \times H \simeq B \times G$,

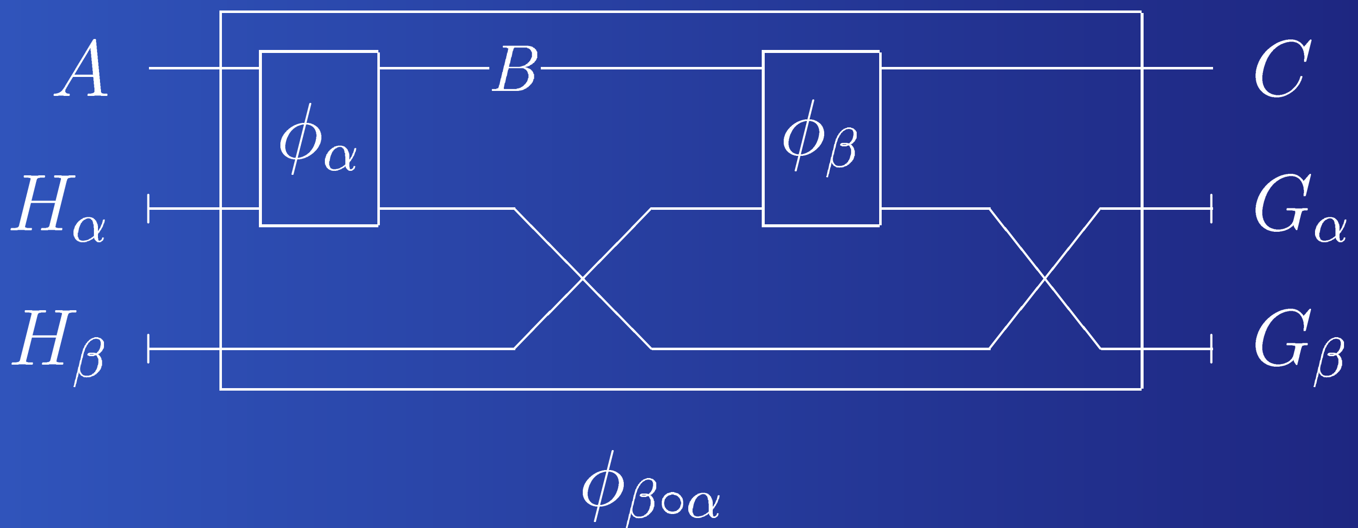
Composing classical computations

Composing classical computations



$\phi_{\beta\circ\alpha}$

Composing classical computations



Theorem:

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

Extensional equality

Extensional equality

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

$$\begin{array}{ccc} A \times H & \xrightarrow[\phi]{} & B \times G \\ \uparrow{\scriptstyle (-,h)} & & \downarrow{\scriptstyle \pi_1} \\ A & \xrightarrow[\mathsf{U}\,\alpha]{} & B \end{array}$$

Extensional equality

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

$$\begin{array}{ccc}
A \times H & \xrightarrow[\phi]{} & B \times G \\
\uparrow{\scriptstyle (-,h)} & & \downarrow{\scriptstyle \pi_1} \\
A & \xrightarrow[\mathsf{U}\,\alpha]{} & B
\end{array}$$

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

Extensional equality . . .

- **Theorem:**

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

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 $\mathbf{FCC}$.

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 $\mathbf{FCC}$.
- **Theorem:** Any function $f \in A \to B$ on finite sets A, B can be realized by a computation.

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 $\mathbf{FCC}$.
- **Theorem:** Any function $f \in A \to B$ on finite sets A, B can be realized by a computation.
- *Translation for Category Theoreticians:* U is full and faithful.

Example π_1 :

function

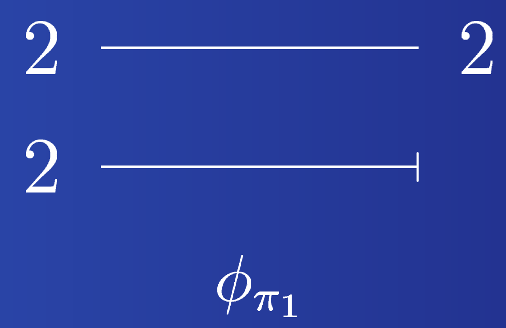
$$\pi_1 \in (2, 2) \to 2$$
$$\pi_1 \; (x, y) = x$$

Example π_1 :

function

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

computation



Example δ :

function

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

$$\delta\ x = (x, x)$$

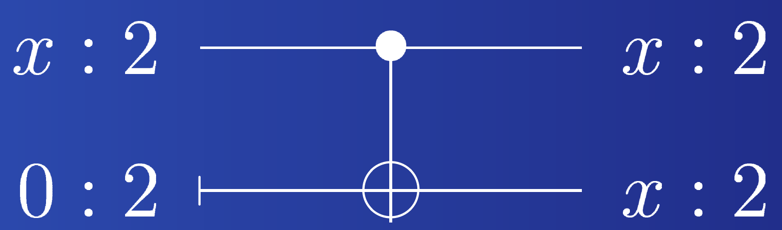
Example δ :

function

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

$$\delta\ x = (x, x)$$

computation



ϕ_δ

$$\phi_\delta \in (2, 2) \to (2, 2)$$

$$\phi_\delta\ (0, x) = (0, x)$$

$$\phi_\delta\ (1, x) = (1, \neg\ x)$$

2. Finite quantum computation

1. Finite classical computation
2. Finite quantum computation
3. QML basics
4. Compiling QML
5. Conclusions and further work

Linear algebra revision

Linear algebra revision

Given a finite set A (the base)
$\mathbb{C}^A = A \to \mathbb{C}$ is a **Hilbert space**.

Linear algebra revision

Given a finite set A (the base)
$\mathbb{C}^A = A \to \mathbb{C}$ is a **Hilbert space**.
Linear operators:
$f \in A \to B \to \mathbb{C}$ induces $\hat{f} \in \mathbb{C}^A \to \mathbb{C}^B$.
we write $f \in A \multimap B$

Linear algebra revision

Given a finite set A (the base)
$\mathbb{C}^A = A \to \mathbb{C}$ is a **Hilbert space**.
Linear operators:
$f \in A \to B \to \mathbb{C}$ induces $\hat{f} \in \mathbb{C}^A \to \mathbb{C}^B$.
we write $f \in A \multimap B$
Norm of a vector:
$\|v\| = \Sigma_{a \in A}(va)^*(va) \in \mathbb{R}^+$,

Linear algebra revision

Given a finite set A (the base)
$\mathbb{C}^A = A \to \mathbb{C}$ is a **Hilbert space**.
Linear operators:

$f \in A \to B \to \mathbb{C}$ induces $\hat{f} \in \mathbb{C}^A \to \mathbb{C}^B$.
we write $f \in A \multimap B$
Norm of a vector:
$\|v\| = \Sigma_{a \in A}(va)^*(va) \in \mathbb{R}^+$,
Unitary operators:

A unitary operator $\phi \in A \multimap_{\text{unitary}} B$ is a linear isomorphism that preserves the norm.

Basics of quantum computation

Basics of quantum computation

- A **pure state** over A is a vector $v \in \mathbb{C}^A$ with unit norm $\|v\| = 1$.

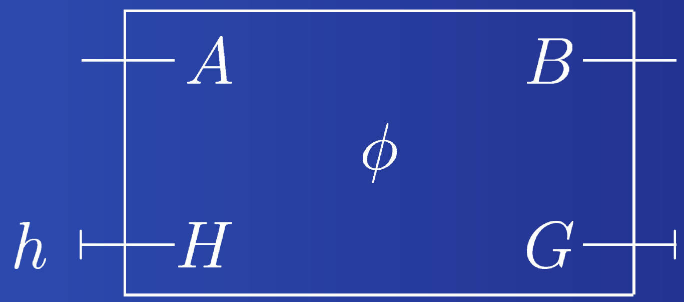
Basics of quantum computation

- A **pure state** over A is a vector $v \in \mathbb{C}^A$ with unit norm $\|v\| = 1$.
- A **reversible computation** is given by a unitary operator $\phi \in A \multimap_{\text{unitary}} B$.

Quantum computations (FQC)

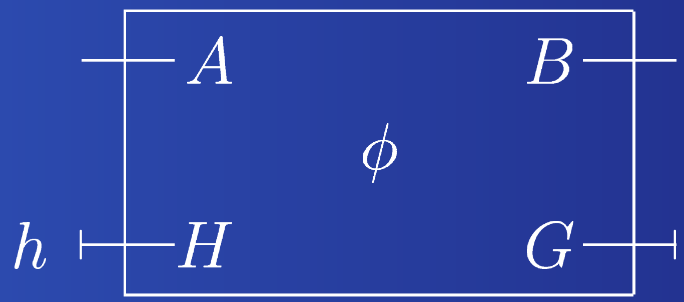
Quantum computations (FQC)

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



Quantum computations (FQC)

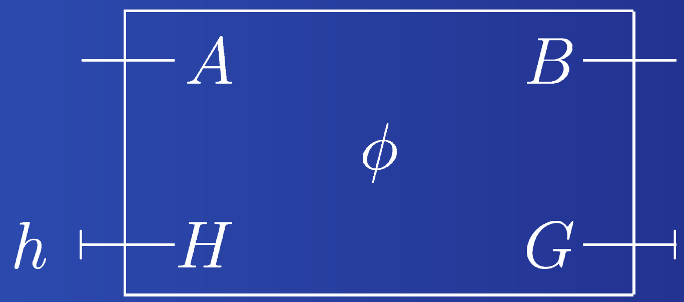
Given finite sets A (input) and B (output):



- a finite set H, the base of the space of initial heaps,

Quantum computations (FQC)

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



- a finite set H, the base of the space of initial heaps,
- a heap initialisation vector $h \in \mathbb{C}^H$,

Quantum computations (FQC)

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

- a finite set H, the base of the space of initial heaps,
- a heap initialisation vector $h \in \mathbb{C}^H$,
- a finite set G, the base of the space of garbage states,

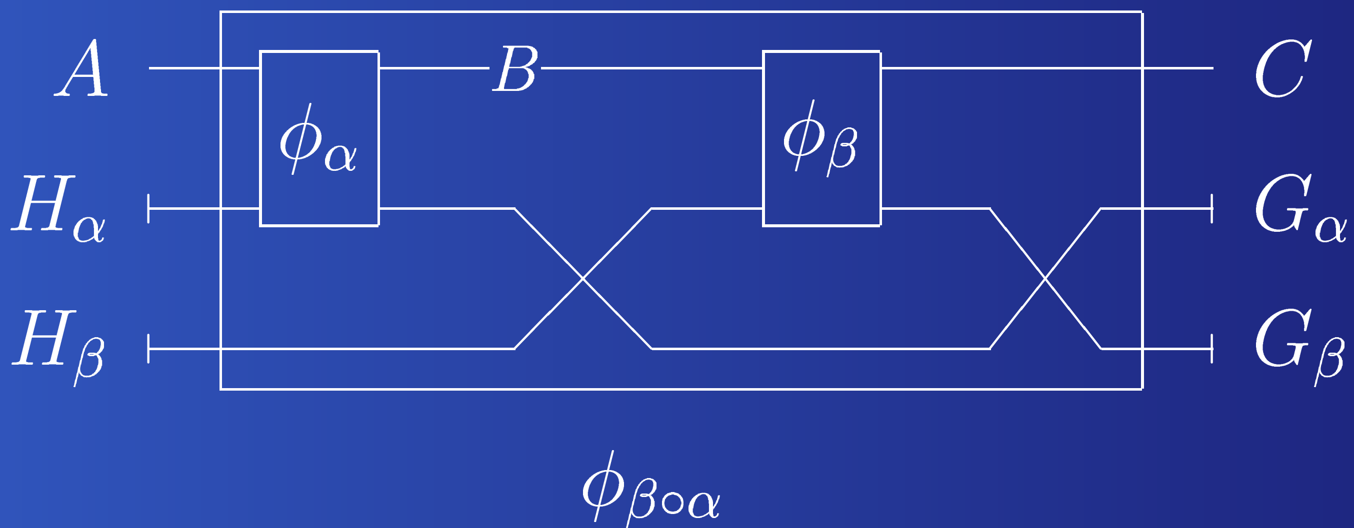
Quantum computations (FQC)

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

- a finite set H, the base of the space of initial heaps,
- a heap initialisation vector $h \in \mathbb{C}^H$,
- a finite set G, the base of the space of garbage states,
- a unitary operator $\phi \in A \otimes H \multimap_{\text{unitary}} B \otimes G$.

Composing quantum computations

Composing quantum computations



Semantics of quantum computations. . .

Semantics of quantum computations. . .

- . . . is a bit more subtle.

Semantics of quantum computations. . .

- . . . is a bit more subtle.
- There is no (sensible) operator on vector spaces, replacing $\pi_1 \in B \times G \to B$.

Semantics of quantum computations. . .

- . . . is a bit more subtle.
- There is no (sensible) operator on vector spaces, replacing $\pi_1 \in B \times G \to B$.
- **Indeed:** Forgetting part of a **pure state** results in a **mixed state**.

Density matrizes

Density matrizes

- Mixed states can be represented by *density matrizes* $\rho \in A \multimap A$.

Density matrizes

- Mixed states can be represented by *density matrizes* $\rho \in A \multimap A$.
- Eigenvalues represent probabilities

$$\rho \vec{v} = \lambda \vec{v}$$

System is in state $\vec{v}$ *with prob.* λ

Density matrizes

- Mixed states can be represented by *density matrizes* $\rho \in A \multimap A$.
- Eigenvalues represent probabilities

$$\rho\vec{v} = \lambda\vec{v}$$

  System is in state $\vec{v}$ *with prob.* λ
- Eigenvalues have to be positive and their sum (the trace) is 1.

Example: forgetting a qbit

Example: forgetting a qbit

- EPR is represented by $\rho \in \mathcal{Q}_2 \otimes \mathcal{Q}_2 \multimap \mathcal{Q}_2 \otimes \mathcal{Q}_2$:

$$\begin{pmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \end{pmatrix}$$

Example: forgetting a qbit

- EPR is represented by $\rho \in \mathcal{Q}_2 \otimes \mathcal{Q}_2 \multimap \mathcal{Q}_2 \otimes \mathcal{Q}_2$:

$$\begin{pmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \end{pmatrix}$$

- $\rho\left(\frac{1}{\sqrt{2}}\left|00\right\rangle + \frac{1}{\sqrt{2}}\left|11\right\rangle\right) = \frac{1}{\sqrt{2}}\left|00\right\rangle + \frac{1}{\sqrt{2}}\left|11\right\rangle$

Example: forgetting a qbit ...

- After measuring one qbit we obtain $\rho' \in \mathcal{Q}_2 \multimap \mathcal{Q}_2$:

$$\begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix}$$

Example: forgetting a qbit . . .

- After measuring one qbit we obtain $\rho' \in \mathcal{Q}_2 \multimap \mathcal{Q}_2$:

$$\begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix}$$

-

$$\begin{aligned} \rho' \, |0\rangle &= \frac{1}{2} \, |0\rangle \\ \rho' \, |1\rangle &= \frac{1}{2} \, |1\rangle \end{aligned}$$

Superoperators

Superoperators

- Morphisms on density matrizes are called *superoperators*, these are linear maps, which are
  - completely positive, and
  - trace preserving

Superoperators

- Morphisms on density matrizes are called *superoperators*, these are linear maps, which are
  - completely positive, and
  - trace preserving
- Every unitary operator ϕ gives rise to a superoperator $\widehat{\phi}$.

Superoperators. . .

- There is an operator

$$\mathrm{tr}_{B,G} \in B \otimes G \multimap_{\text{super}} B$$

called *partial trace*.

Superoperators...

- There is an operator

$$\mathrm{tr}_{B,G} \in B \otimes G \multimap_{\text{super}} B$$

called *partial trace*.

- E.g. $\mathrm{tr}_{\mathcal{Q}_2,\mathcal{Q}_2} \in \mathcal{Q}_2 \otimes \mathcal{Q}_2 \multimap_{\text{super}} \mathcal{Q}_2$ is represented by a 16×4 matrix.

Semantics

Semantics

Every quantum computation α gives rise to a superoperator $\mathsf{U}\,\alpha \in A \multimap_{\mathsf{super}} B$

$$
\begin{array}{ccc}
A \otimes H & \xrightarrow[\widehat{\phi}]{} & B \otimes G \\
\uparrow {\scriptstyle -\otimes\widetilde{h}} & & \downarrow {\scriptstyle \mathrm{tr}_G} \\
A & \xrightarrow[\mathsf{U}\,\alpha]{} & B
\end{array}
$$

Semantics

Every quantum computation α gives rise to a superoperator $\mathsf{U}\,\alpha \in A \multimap_{\mathsf{super}} B$

$$\begin{array}{ccc} A \otimes H & \xrightarrow[\widehat{\phi}]{} & B \otimes G \\ \uparrow {\scriptstyle -\otimes \widetilde{h}} & & \downarrow {\scriptstyle \mathrm{tr}_G} \\ A & \xrightarrow[\mathsf{U}\,\alpha]{} & B \end{array}$$

Theorem: Every superoperator $F \in A \multimap_{\mathsf{super}} B$ (on finite Hilbert spaces) comes from a quantum computation.

Classical vs quantum

Classical vs quantum

classical (FCC)	quantum (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 space

Classical vs quantum

classical ($\mathbf{FCC}$)	quantum ($\mathbf{FQC}$)
finite sets	finite dimensional Hilbert space
cartesian product ($\times$)	

Classical vs quantum

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

Classical vs quantum

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

Classical vs quantum

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

Classical vs quantum

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

Classical vs quantum

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

Classical vs quantum

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

Classical vs quantum

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

Classical vs quantum

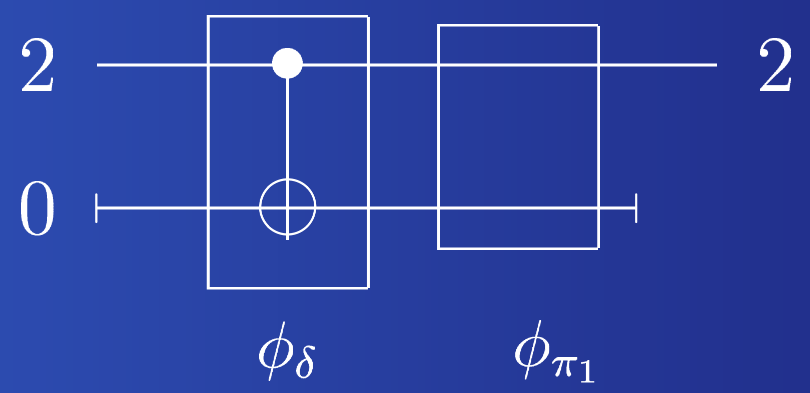
classical ($\mathbf{FCC}$)	quantum ($\mathbf{FQC}$)
finite sets	finite dimensional Hilbert spaces
cartesian product ($\times$)	tensor product ($\otimes$)
bijections	unitary operators
functions	superoperators
injective functions ($\mathbf{FCC}^\circ$)	isometries ($\mathbf{FQC}^\circ$)
projections	

Classical vs quantum

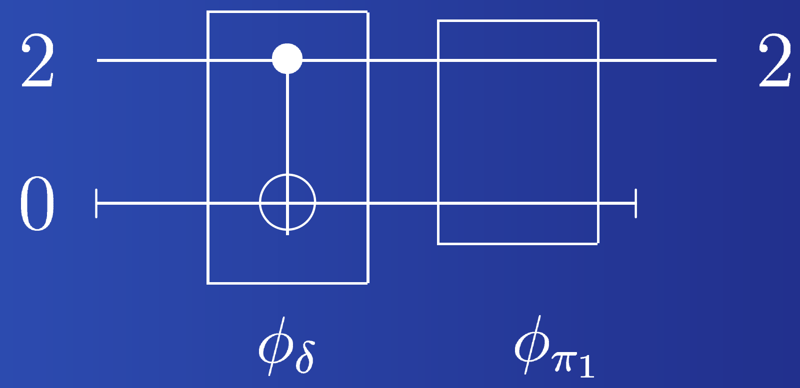
classical ($\mathbf{FCC}$)	quantum ($\mathbf{FQC}$)
finite sets	finite dimensional Hilbert spaces
cartesian product ($\times$)	tensor product ($\otimes$)
bijections	unitary operators
functions	superoperators
injective functions ($\mathbf{FCC}^{\circ}$)	isometries ($\mathbf{FQC}^{\circ}$)
projections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

Decoherence

Decoherence



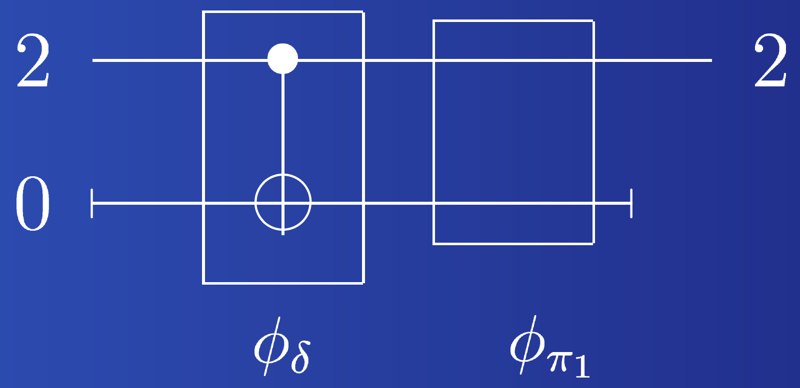
Decoherence



Classically

$$\pi_1 \circ \delta = \mathrm{I}$$

Decoherence

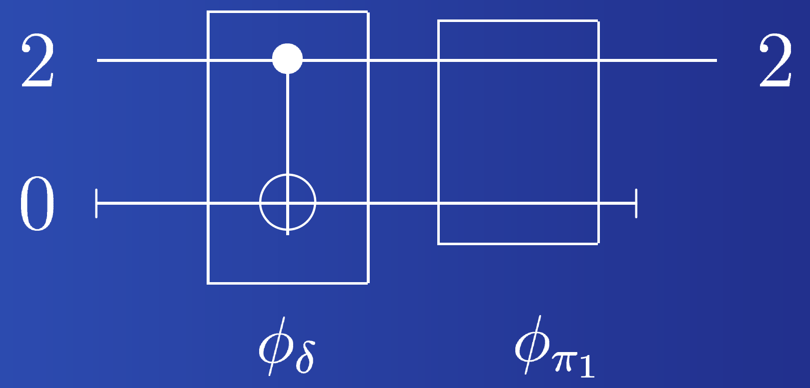


Classically

$$\pi_1 \circ \delta = \mathrm{I}$$

Quantum

Decoherence



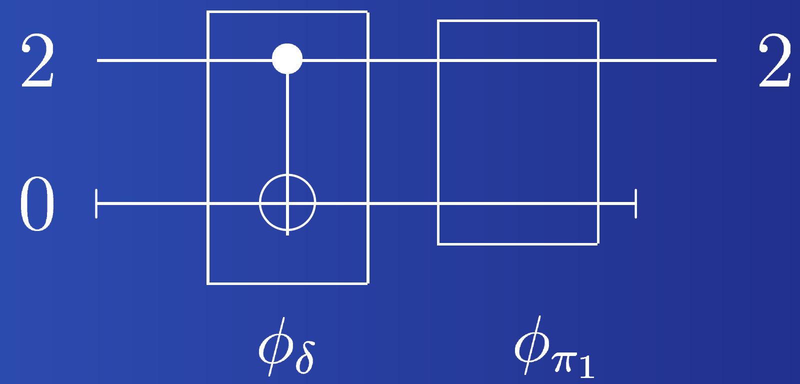
Classically

$$\pi_1 \circ \delta = I$$

Quantum

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

Decoherence



Classically

$$\pi_1 \circ \delta = \mathrm{I}$$

Quantum

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

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

3. QML basics

1. Finite classical computation
2. Finite quantum computation
3. QML basics
4. Compiling QML
5. Conclusions and further work

QML basics

QML basics

- QML is a first order functional languages, i.e. programs are well-typed expressions.

QML basics

- QML is a first order functional languages, i.e. programs are well-typed expressions.
- QML types are $1, \sigma \otimes \tau, \mathcal{Q}_2$

QML basics

- QML is a first order functional languages, i.e. programs are well-typed expressions.
- QML types are $1, \sigma \otimes \tau, \mathcal{Q}_2$
- Qbytes
  $\mathcal{Q}_2^8 = \mathcal{Q}_2 \otimes \mathcal{Q}_2 \otimes \mathcal{Q}_2 \otimes \mathcal{Q}_2 \otimes \mathcal{Q}_2 \otimes \mathcal{Q}_2 \otimes \mathcal{Q}_2 \otimes \mathcal{Q}_2$.

QML basics . . .

- A QML program is an expression in a context of typed variables, e.g.

$$
\begin{aligned}
qnot &: Q_2 \multimap Q_2 \\
qnot\ x &= \mathbf{if}^{\circ}\ x \\
&\quad\ \mathbf{then}\ \mathrm{qfalse} \\
&\quad\ \mathbf{else}\ \ \mathrm{qtrue}
\end{aligned}
$$

QML basics . . .

- A QML program is an expression in a context of typed variables, e.g.

$$
\begin{aligned}
&qnot : Q_2 \multimap Q_2 \\
&qnot\ x = \mathbf{if}^\circ\ x \\
&\qquad\qquad \mathbf{then}\ \mathrm{qfalse} \\
&\qquad\qquad \mathbf{else}\ \ \mathrm{qtrue}
\end{aligned}
$$

- We can compile QML programs into quantum computations (i.e. quantum circuits).

QML basics . . .

- Forgetting variables has to be explicit.

QML basics . . .

- Forgetting variables has to be explicit. E.g.

$$qfst : Q_2 \otimes Q_2 \multimap Q_2$$
$$qfst\ (x, y) = x$$

is illegal,

QML basics . . .

- Forgetting variables has to be explicit.
  E.g.

  $$qfst : Q_2 \otimes Q_2 \multimap Q_2$$
  $$qfst\ (x, y) = x$$

  is illegal,
  but

  $$qfst : Q_2 \otimes Q_2 \multimap Q_2$$
  $$qfst\ (x, y) = x \uparrow \{\, y \,\}$$

  is ok.

QML basics . . .

- There are two different if-then-else constructs.

QML basics . . .

- There are two different if-then-else constructs.

$$id : Q_2 \multimap Q_2$$
$$id\ x = \mathbf{if}^\circ\ x$$
$$\quad \mathbf{then}\ \mathrm{qtrue}$$
$$\quad \mathbf{else}\ \ \mathrm{qfalse}$$

  is just the identity,

QML basics . . .

- There are two different if-then-else constructs.

$$
\begin{aligned}
&id : Q_2 \multimap Q_2\\
&id\ x = \mathbf{if}^{\circ}\ x\\
&\qquad\qquad \mathbf{then}\ \mathrm{qtrue}\\
&\qquad\qquad \mathbf{else}\ \ \mathrm{qfalse}
\end{aligned}
$$

is just the identity, but

$$
\begin{aligned}
&meas : Q_2 \multimap Q_2\\
&meas\ x = \mathbf{if}\ x\\
&\qquad\qquad \mathbf{then}\ \mathrm{qtrue}\\
&\qquad\qquad \mathbf{else}\ \ \mathrm{qfalse}
\end{aligned}
$$

introduces a measurement (end hence decoherence).

QML basics . . .

- Using $\mathbf{if}^{\circ}$ is only allowed, if the branches are orthogonal, i.e. observable different.

QML basics ...

- Using $\mathbf{if}^{\circ}$ is only allowed, if the branches are orthogonal, i.e. observable different.

$$
\begin{aligned}
&cswap : Q_2 \otimes Q_2 \multimap Q_2 \multimap Q_2 \otimes Q_2 \\
&cswap\ (x, y)\ c = \mathbf{if}^{\circ}\ c \\
&\quad \mathbf{then}\ (y, x) \\
&\quad \mathbf{else}\ \ (x, y)
\end{aligned}
$$

is illegal,

QML basics . . .

- Using $\mathbf{if}^\circ$ is only allowed, if the branches are orthogonal, i.e. observable different.

$$cswap : Q_2 \otimes Q_2 \multimap Q_2 \multimap Q_2 \otimes Q_2$$
$$\begin{aligned} cswap\ (x, y)\ c &= \mathbf{if}^\circ\ c \\ &\quad \mathbf{then}\ (y, x) \\ &\quad \mathbf{else}\ \ (x, y) \end{aligned}$$

is illegal, but

$$cswap : Q_2 \otimes Q_2 \multimap Q_2 \multimap Q_2 \otimes (Q_2 \otimes Q_2)$$
$$\begin{aligned} cswap\ (x, y)\ c &= \mathbf{if}^\circ\ c \\ &\quad \mathbf{then}\ (\mathrm{qtrue}, (y, x)) \\ &\quad \mathbf{else}\ \ (\mathrm{qfalse}, (x, y)) \end{aligned}$$

is ok.

QML basics . . .

- We can introduce superpositions, e.g.

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

QML basics . . .

- We can introduce superpositions, e.g.

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

  However, the terms in the superposition have to be orthogonal.

4. Compiling QML

1. Finite classical computation
2. Finite quantum computation
3. QML basics
4. Compiling QML
5. Conclusions and further work

Compilation

Compilation

- Correct QML programs are defined by typing rules, e.g.

$$\frac{\Gamma \vdash t : \sigma \otimes \tau \qquad \Delta, x : \sigma, y : \tau \vdash u : C}{\Gamma \otimes \Delta \vdash \texttt{let}\ (x, y) = t\ \texttt{in}\ u : C} \otimes \text{elim}$$

Compilation

- Correct QML programs are defined by typing rules, e.g.

$$\frac{\Gamma \vdash t : \sigma \otimes \tau \qquad \Delta, x : \sigma, y : \tau \vdash u : C}{\Gamma \otimes \Delta \vdash \texttt{let}\ (x, y) = t\ \texttt{in}\ u : C} \otimes \mathrm{elim}$$

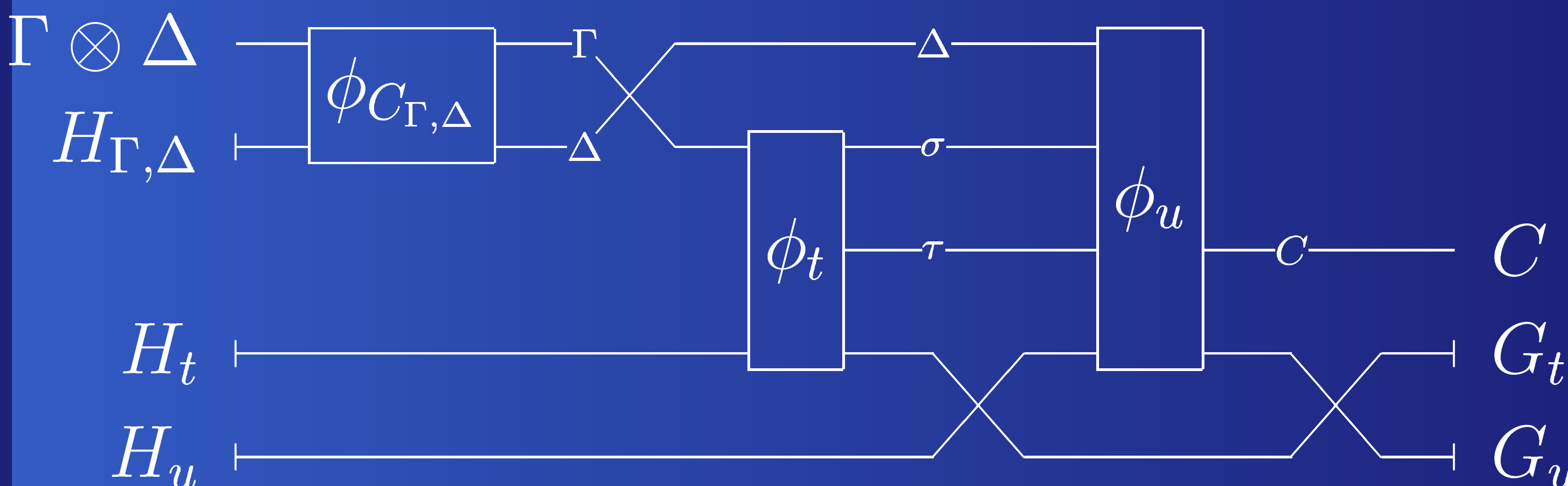
For each rule we can construct a quantum computation, i.e. a circuit.

$\otimes$-elim

$$\frac{\Gamma \vdash t : \sigma \otimes \tau \qquad \Delta, x : \sigma, y : \tau \vdash u : C}{\Gamma \otimes \Delta \vdash \texttt{let } (x, y) = t \texttt{ in } u : C} \otimes \text{elim}$$

$\otimes$-elim

$$\frac{\Gamma \vdash t : \sigma \otimes \tau \qquad \Delta, x : \sigma, y : \tau \vdash u : C}{\Gamma \otimes \Delta \vdash \texttt{let } (x, y) = t \texttt{ in } u : C} \otimes \text{elim}$$



Compiler

Compiler

- A compiler is currently being implemented by my student Jonathan Grattage (in Haskell).

Compiler

- A compiler is currently being implemented by my student Jonathan Grattage (in Haskell).
- The output of the compiler are quantum circuits which can be simulated by a quantum circuit simulator.

Compiler

- A compiler is currently being implemented by my student Jonathan Grattage (in Haskell).
- The output of the compiler are quantum circuits which can be simulated by a quantum circuit simulator.
- Amr Sabry and Juliana Vizotti (Indiana University) embarked on an independent implementation of QML based on our paper.

5. Conclusions

1. Semantics of finite classical and quantum computation
2. QML basics
3. Compiling QML
4. Conclusions and further work

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.
- We have developed an *algebra of quantum programs* which for pure programs is complete wrt the semantics and a normalisation algorithm.

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.
- We have developed an *algebra of quantum programs* which for pure programs is complete wrt the semantics and a normalisation algorithm.
- 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.
- We should be able to extend our algebra and normalisation to the full language (including measurements).

Further work

- We have to analyze more quantum programs to evaluate the practical usefulness of our approach.
- We should be able to extend our algebra and normalisation to the full language (including measurements).
- 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.
- We should be able to extend our algebra and normalisation to the full language (including measurements).
- 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.
- We should be able to extend our algebra and normalisation to the full language (including measurements).
- 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.
- We should be able to extend our algebra and normalisation to the full language (including measurements).
- 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.

Papers, available from

`//www.cs.nott.ac.uk/˜txa/publ/`

A functional quantum programming language LICS 2005

with J.Grattage

Structuring Quantum Effects: Superoperators as Arrows

MFCS 2006

with J.Vizzotto and A.Sabry

An Algebra of Pure Quantum Programming QPL 2005

with J.Grattage, J.Vizzotto and A.Sabry