Functional Quantum Programming

Thorsten Altenkirch
University of Nottingham
based on joint work with Jonathan Grattage
supported by EPSRC grant GR/S30818/01
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

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: 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?*
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 $O(\sqrt{n})$
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

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!
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: a functional language for quantum computations on finite types.

Quantum control and quantum data.

Design guided by semantics

Analogy with classical computation

Finite classical computations

Finite quantum computations

Important issue: control of decoherence

Compiler under construction (Jonathan)
QML: a functional language for quantum computations on finite types.
QML

- Quantum control and quantum data.


Compiler under construction (Jonathan).
QML: a functional language for quantum computations on finite types.

Quantum control and quantum data.

Design guided by semantics.
QML

- 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)

Tallinn Feb 06 – p.4/44
QML

- 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

- 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

\[
H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}
\]
Example: Hadamard operation

Matrix

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

QML

\[\text{had} : Q_2 \rightarrow Q_2\]
\[\text{had } x = \text{if}^\circ x\]
\[\text{then } \{ \text{qfalse } | \ (1) \ \text{qtrue}\}\]
\[\text{else } \{ \text{qfalse } | \ \text{qtrue}\}\]
Deutsch algorithm

\[ \text{deutsch} : 2 \rightarrow 2 \rightarrow Q_2 \]

\[ \text{deutsch} \ a \ b = \]

\[ \text{let} \ (x, y) = \text{if}^o \{ \text{qfalse} \mid \text{qtrue} \} \]

\[ \text{then} \ (\text{qtrue}, \text{if} \ a) \]

\[ \text{then} \ \{ \text{qfalse} \mid (-1) \text{qtrue} \} \]

\[ \text{else} \ \{ (-1) \text{qfalse} \mid \text{qtrue} \} \]

\[ \text{else} \ (\text{qfalse}, \text{if} \ b) \]

\[ \text{then} \ \{ (-1) \text{qfalse} \mid \text{qtrue} \} \]

\[ \text{else} \ \{ \text{qfalse} \mid (-1) \text{qtrue} \} \]

\[ \text{in} \ H \ x \]
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):

\[ A \quad \phi \quad B \]

\[ h \quad H \quad G \]
Classical computation (FCC)

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

- a finite set of initial heaps $H$, 

\[
\begin{array}{ccc}
A & \phi & B \\
\downarrow & & \downarrow \\
H & \phi & G \\
\end{array}
\]
Classical computation (FCC)

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

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

\[ A \xrightarrow{\phi} B \]
\[ h \mid H \xrightarrow{\phi} G \]

Tallinn Feb 06 – p.10/44
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

\[ A \xrightarrow{H_\alpha} \phi_\alpha \xrightarrow{B} \phi_\beta \xrightarrow{G_\beta} C \]

\[ H_\beta \]

\[ \phi_{\beta \circ \alpha} \]
Composing classical computations

Theorem:

\[
\phi_{\beta \circ \alpha} = U(\beta \circ \alpha) = (U\beta) \circ (U\alpha)
\]
Extensional equality

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

A classical computation \( \alpha = (H, h, G, \phi) \) induces a function \( \cup \alpha \in A \to B \) by

\[
\begin{array}{ccc}
A \times H & \xrightarrow{\phi} & B \times G \\
\uparrow (\cdot, h) & & \downarrow \pi_1 \\
A & \xrightarrow{\cup \alpha} & B
\end{array}
\]
Extensioinal equality

A classical computation $\alpha = (H, h, G, \phi)$ induces a function $\cup \alpha \in A \rightarrow B$ by

$A \times H \xrightarrow{\phi} B \times G$

$( -, h) \downarrow \quad \pi_1$

$A \xrightarrow{\cup \alpha} B$

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

Theorem:

\[ U(\beta \circ \alpha) = (U\beta) \circ (U\alpha) \]
Extensional equality …

Theorem:

\[ U(\beta \circ \alpha) = (U\beta) \circ (U\alpha) \]

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

- **Theorem:**
  
  \[
  U(\beta \circ \alpha) = (U\beta) \circ (U\alpha)
  \]

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

- **Theorem:** Any function \( f \in A \rightarrow B \) on finite sets \( A, B \) can be realized by a computation.
Extentional equality . . .

- **Theorem:**
  \[ U(\beta \circ \alpha) = (U\beta) \circ (U\alpha) \]

Hence, classical computations up to 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 Theoreticians:* \( U \) is full and faithful.
Example $\pi_1$:

function

\[ \pi_1 \in (2, 2) \rightarrow 2 \]

\[ \pi_1 (x, y) = x \]
Example $\pi_1$:

**function**

$$\pi_1 \in (2, 2) \rightarrow 2$$

$$\pi_1 (x, y) = x$$

**computation**

$$\begin{array}{c}
2 & \overline{\phantom{2}} & 2 \\
2 & \overline{\phantom{2}} & 2
\end{array}$$

$$\phi_{\pi_1}$$
Example $\delta$:

function

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

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

**function**

\[ \delta \in 2 \rightarrow (2, 2) \]
\[ \delta \; x = (x, x) \]

**computation**

\[ \phi_\delta \]
\[ \phi_\delta \in (2, 2) \rightarrow (2, 2) \]
\[ \phi_\delta (0, x) = (0, x) \]
\[ \phi_\delta (1, x) = (1, \neg x) \]
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

A finite set is a Hilbert space.

Linear operators induce

We write

The norm of a vector:

Unitary operators:

A unitary operator is a linear isomorphism that preserves the norm.

Tallinn Feb 06 – p.17/44
Given a finite set $A$ (the base) $\mathbb{C}^A = A \rightarrow \mathbb{C}$ is a Hilbert space.
Given a finite set $A$ (the base) $\mathbb{C}^A = A \rightarrow \mathbb{C}$ is a **Hilbert space**.

**Linear operators:**

$f \in A \rightarrow B \rightarrow \mathbb{C}$ induces $\hat{f} \in \mathbb{C}^A \rightarrow \mathbb{C}^B$. We write $f \in A \rightsquigarrow B$.
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 $f \in A \rightarrow B$

**Norm of a vector:**

$\|v\| = \sum_{a \in A} (va)^* (va) \in \mathbb{R}^+$,
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 $f \in A \to B$

**Norm of a vector:**

$\|v\| = \sum_{a \in A} (va)^*(va) \in \mathbb{R}^+$,

**Unitary operators:**

A unitary operator $\phi \in A \overset{\text{unitary}}{\to} B$ is a linear isomorphism that preserves the norm.
Basics of quantum computation

A pure state over is a vector with unit norm.

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

A **reversible computation** is given by a unitary operator $\phi \in A \xrightarrow{\text{unitary}} B$. 
Quantum computations (FQC)
Quantum computations (FQC)

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

\[ \begin{array}{c}
    A & \phi & B \\
    h & \phantom{\phi} & H \\
    & \phantom{\phi} & G
\end{array} \]
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 $\phi \in A \otimes H \to \text{unitary } B \otimes G$. 
Composing quantum computations
Composing quantum computations

\[
\begin{array}{c}
A \\
H_\alpha \\
H_\beta \\
\phi_\alpha \\
B \\
\phi_\beta \\
C \\
G_\alpha \\
G_\beta \\
\phi_{\beta \circ \alpha}
\end{array}
\]
Semantics of quantum computations... is a bit more subtle. There is no (sensible) operator on vector spaces, replacing Forgetting part of a pure state results in a mixed state.
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 $\pi_1 \in B \times G \rightarrow B$. 
... is a bit more subtle.

There is no (sensible) operator on vector spaces, replacing $\pi_1 \in B \times G \rightarrow B$.

Indeed: Forgetting part of a pure state results in a mixed state.
Density matrices

Mixed states can be represented by density matrices. Eigenvalues represent probabilities. A system is in state with probability. Eigenvalues have to be positive and their sum (the trace) is a real number.

Tallinn Feb 06 – p.22/44
Density matrices

Mixed states can be represented by *density matrices* $\rho \in \mathcal{A}$.
Density matrices

- Mixed states can be represented by density matrices $\rho \in A \rightarrow A$.
- Eigenvalues represent probabilities

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

*System is in state $\vec{v}$ with prob. $\lambda$*
Density matrices

- Mixed states can be represented by density matrices $\rho \in A \rightarrow A$.
- Eigenvalues represent probabilities $\rho \vec{v} = \lambda \vec{v}$

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

- Eigenvalues have to be positive and their sum (the trace) is 1.
Example: forgetting a qbit
Example: forgetting a qbit

EPR is represented by

\[ \rho \in \mathbb{Q}_2 \otimes \mathbb{Q}_2 \rightarrow \mathbb{Q}_2 \otimes \mathbb{Q}_2 : \]

\[
\begin{pmatrix}
\frac{1}{2} & 0 & 0 & \frac{1}{2} \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\frac{1}{2} & 0 & 0 & \frac{1}{2}
\end{pmatrix}
\]
Example: forgetting a qbit

EPR is represented by
\[ \rho \in \mathbb{Q}_2 \otimes \mathbb{Q}_2 \longrightarrow \mathbb{Q}_2 \otimes \mathbb{Q}_2 : \]
\[
\begin{pmatrix}
\frac{1}{2} & 0 & 0 & \frac{1}{2} \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\frac{1}{2} & 0 & 0 & \frac{1}{2}
\end{pmatrix}
\]

\[ \rho \left( \frac{1}{\sqrt{2}} \left| 00 \right\rangle + \frac{1}{\sqrt{2}} \left| 11 \right\rangle \right) = \frac{1}{\sqrt{2}} \left| 00 \right\rangle + \frac{1}{\sqrt{2}} \left| 11 \right\rangle \]
Example: forgetting a qbit . . .

After measuring one qbit we obtain

\[ \rho' \in Q_2 \rightarrow Q_2: \]

\[
\begin{pmatrix}
\frac{1}{2} & 0 \\
0 & \frac{1}{2}
\end{pmatrix}
\]
Example: forgetting a qbit . . .

After measuring one qbit we obtain

$$\rho' \in Q_2 \rightarrow Q_2 :$$

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

$$\rho' \ket{0} = \frac{1}{2} \ket{0}$$

$$\rho' \ket{1} = \frac{1}{2} \ket{1}$$
Superoperators

Morphisms on density matrices are called superoperators, these are linear maps, which are completely positive, and trace preserving. Every unitary operator gives rise to a superoperator.
Morphisms on density matrices are called **superoperators**, these are linear maps, which are
- completely positive, and
- trace preserving
Morphisms on density matrices are called *superoperators*, these are linear maps, which are

- completely positive, and
- trace preserving

Every unitary operator $\phi$ gives rise to a superoperator $\hat{\phi}$. 
Superoperators…

There is an operator

$$\text{tr}_{B,G} \in B \otimes G \overset{\text{super}}{\longrightarrow} B$$

called *partial trace*. 
Superoperators...

There is an operator $\text{tr}_{B,G} \in B \otimes G \rightarrow_{\text{super}} B$
called *partial trace*.

E.g. $\text{tr}_{Q_2,Q_2} \in Q_2 \otimes Q_2 \rightarrow_{\text{super}} Q_2$ is represented by a $16 \times 4$ matrix.
Semantics

Theorem: Every superoperator (on finite Hilbert spaces) comes from a quantum computation.
Every quantum computation $\alpha$ gives rise to a superoperator $U\alpha \in A \xrightarrow{\text{super}} B$.
Semantics

Every quantum computation $\alpha$ gives rise to a superoperator $U \alpha \in A \rightarrow_{\text{super}} B$

\[
\begin{array}{c}
A \otimes H \xrightarrow{\hat{\phi}} B \otimes G \\
\downarrow \otimes \widetilde{h} \quad \downarrow \text{tr}_G \quad \uparrow \cup \alpha \\
A \xrightarrow{\cup \alpha} B
\end{array}
\]

**Theorem:** Every superoperator $F \in A \rightarrow_{\text{super}} B$ (on finite Hilbert spaces) comes from a quantum computation.
Classical vs quantum
## Classical vs Quantum

<table>
<thead>
<tr>
<th>Classical (FCC)</th>
<th>Quantum (FQC)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Classical vs quantum

<table>
<thead>
<tr>
<th>classical (FCC)</th>
<th>quantum (FQC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>finite sets</td>
<td></td>
</tr>
</tbody>
</table>
## Classical vs quantum

<table>
<thead>
<tr>
<th>classical (FCC)</th>
<th>quantum (FQC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>finite sets</td>
<td>finite dimensional Hilbert space</td>
</tr>
</tbody>
</table>
## Classical vs quantum

<table>
<thead>
<tr>
<th>classical (FCC)</th>
<th>quantum (FQC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>finite sets</td>
<td>finite dimensional Hilbert space</td>
</tr>
<tr>
<td>cartesian product ($\times$)</td>
<td></td>
</tr>
</tbody>
</table>
## Classical vs quantum

<table>
<thead>
<tr>
<th>Classical (FCC)</th>
<th>Quantum (FQC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>finite sets</td>
<td>finite dimensional Hilbert space</td>
</tr>
<tr>
<td>cartesian product ((\times))</td>
<td>tensor product ((\otimes))</td>
</tr>
<tr>
<td>Classical (FCC)</td>
<td>Quantum (FQC)</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>finite sets</td>
<td>finite dimensional Hilbert space</td>
</tr>
<tr>
<td>cartesian product ($\times$)</td>
<td>tensor product ($\otimes$)</td>
</tr>
<tr>
<td>bijections</td>
<td></td>
</tr>
</tbody>
</table>
### Classical vs quantum

<table>
<thead>
<tr>
<th>classical ($\text{FCC}$)</th>
<th>quantum ($\text{FQC}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>finite sets</td>
<td>finite dimensional Hilbert space</td>
</tr>
<tr>
<td>cartesian product ($\times$)</td>
<td>tensor product ($\otimes$)</td>
</tr>
<tr>
<td>bijections</td>
<td>unitary operators</td>
</tr>
</tbody>
</table>
## Classical vs quantum

<table>
<thead>
<tr>
<th>classical (FCC)</th>
<th>quantum (FQC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>finite sets</td>
<td>finite dimensional Hilbert space</td>
</tr>
<tr>
<td>cartesian product ($\times$)</td>
<td>tensor product ($\otimes$)</td>
</tr>
<tr>
<td>bijections</td>
<td>unitary operators</td>
</tr>
<tr>
<td>functions</td>
<td></td>
</tr>
</tbody>
</table>
# Classical vs quantum

<table>
<thead>
<tr>
<th>classical (FCC)</th>
<th>quantum (FQC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>finite sets</td>
<td>finite dimensional Hilbert space</td>
</tr>
<tr>
<td>cartesian product ($\times$)</td>
<td>tensor product ($\otimes$)</td>
</tr>
<tr>
<td>bijections</td>
<td>unitary operators</td>
</tr>
<tr>
<td>functions</td>
<td>superoperators</td>
</tr>
</tbody>
</table>
## Classical vs quantum

<table>
<thead>
<tr>
<th>classical (FCC)</th>
<th>quantum (FQC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>finite sets</td>
<td>finite dimensional Hilbert spaces</td>
</tr>
<tr>
<td>cartesian product ($\times$)</td>
<td>tensor product ($\otimes$)</td>
</tr>
<tr>
<td>bijections</td>
<td>unitary operators</td>
</tr>
<tr>
<td>functions</td>
<td>superoperators</td>
</tr>
<tr>
<td>injective functions (FCC°)</td>
<td></td>
</tr>
</tbody>
</table>
## Classical vs Quantum

<table>
<thead>
<tr>
<th>Classical (FCC)</th>
<th>Quantum (FQC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>finite sets</td>
<td>finite dimensional Hilbert spaces</td>
</tr>
<tr>
<td>cartesian product ($\times$)</td>
<td>tensor product ($\otimes$)</td>
</tr>
<tr>
<td>bijections</td>
<td>unitary operators</td>
</tr>
<tr>
<td>functions</td>
<td>superoperators</td>
</tr>
<tr>
<td>injective functions (FCC°)</td>
<td>isometries (FQC°)</td>
</tr>
</tbody>
</table>
## Classical vs quantum

<table>
<thead>
<tr>
<th>classical (FCC)</th>
<th>quantum (FQC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>finite sets</td>
<td>finite dimensional Hilbert spaces</td>
</tr>
<tr>
<td>cartesian product ($\times$)</td>
<td>tensor product ($\otimes$)</td>
</tr>
<tr>
<td>bijections</td>
<td>unitary operators</td>
</tr>
<tr>
<td>functions</td>
<td>superoperators</td>
</tr>
<tr>
<td>injective functions (FCC°)</td>
<td>isometries (FQC°)</td>
</tr>
<tr>
<td>projections</td>
<td></td>
</tr>
</tbody>
</table>
### Classical vs Quantum

<table>
<thead>
<tr>
<th>Classical (FCC)</th>
<th>Quantum (FQC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>finite sets</td>
<td>finite dimensional Hilbert spaces</td>
</tr>
<tr>
<td>cartesian product ((\times))</td>
<td>tensor product ((\otimes))</td>
</tr>
<tr>
<td>bijections</td>
<td>unitary operators</td>
</tr>
<tr>
<td>functions</td>
<td>superoperators</td>
</tr>
<tr>
<td>injective functions (FCC°)</td>
<td>isometries (FQC°)</td>
</tr>
<tr>
<td>projections</td>
<td>partial trace</td>
</tr>
</tbody>
</table>
Decoherence
Decoherence

\[ \phi_\delta \quad \phi_{\pi_1} \]
Decoherence

Classically

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

Classically

Quantum

\( \pi_1 \circ \delta = I \)
Decoherence

Classically

Quantum

input: \( \left\{ \frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |0\rangle \right\} \)
Decoherence

Classically

Classically

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 $1, \sigma \otimes \tau, Q_2$
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

$\mathcal{Q}^8_2 = \mathcal{Q}_2 \otimes \mathcal{Q}_2 \otimes \mathcal{Q}_2 \otimes \mathcal{Q}_2 \otimes \mathcal{Q}_2 \otimes \mathcal{Q}_2 \otimes \mathcal{Q}_2 \otimes \mathcal{Q}_2$. 
A QML program is an expression in a context of typed variables, e.g.

\[ \text{qnot} : \mathbb{Q}_2 \rightarrow \mathbb{Q}_2 \]

\[ \text{qnot } x = \text{if}^\circ x \]

\[ \text{then } q\text{false} \]

\[ \text{else } q\text{true} \]
A QML program is an expression in a context of typed variables, e.g.

\[ qnot : Q_2 \rightarrow Q_2 \]

\[ qnot \ x = \text{if}^\circ \ x \]

\[ \text{then } qfalse \]

\[ \text{else } qtrue \]

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.

\[ qfst : Q_2 \times Q_2 \rightarrow Q_2 \]
\[ qfst (x, y) = x \]

is illegal,
QML basics . . .

Forgetting variables has to be explicit. E.g.

\[ q_{fst} : Q_2 \otimes Q_2 \to Q_2 \]
\[ q_{fst} (x, y) = x \]

is illegal,

but

\[ q_{fst} : Q_2 \otimes Q_2 \to Q_2 \]
\[ q_{fst} (x, y) = x \uparrow \{ y \} \]

is ok.
QML basics ...

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

There are two different if-then-else constructs.

\[ id : Q_2 \rightarrow Q_2 \]

\[ id \ x = \text{if}^\delta \ x \]

then \ q_{\text{true}}

else \ q_{\text{false}}

is just the identity,
QML basics ... 

There are two different if-then-else constructs.

\[
id : Q_2 \rightarrow Q_2
\]

\[
id \ x = \text{if}^\circ \ x
\]

then \text{qtrue}

else \text{qfalse}

is just the identity, but

\[
\text{meas} : Q_2 \rightarrow Q_2
\]

\[
\text{meas} \ x = \text{if} \ x
\]

then \text{qtrue}

else \text{qfalse}

introduces a measurement (end hence decoherence).
QML basics . . .

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

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

$$c\text{swap} : Q_2 \otimes Q_2 \rightarrow Q_2 \rightarrow Q_2 \otimes Q_2$$

$$c\text{swap} (x, y) \ c = \text{if}^\circ \ c$$

then $(y, x)$

else $(x, y)$

is illegal,
QML basics ... 

Using \( \text{if}^\circ \) is only allowed, if the branches are orthogonal, i.e. observable different.

\[
\text{cswap} : Q_2 \otimes Q_2 \rightarrow Q_2 \rightarrow Q_2 \otimes Q_2
\]

\[
\text{cswap} (x, y) c = \text{if}^\circ c \quad \text{then} \ (y, x) \\
\text{else} \ (x, y)
\]

is illegal, but

\[
\text{cswap} : Q_2 \otimes Q_2 \rightarrow Q_2 \rightarrow Q_2 \otimes (Q_2 \otimes Q_2)
\]

\[
\text{cswap} (x, y) c = \text{if}^\circ c \\
\text{then} \ (\text{qtrue}, (y, x)) \\
\text{else} \ (\text{qfalse}, (x, y))
\]

is ok.
QML basics . . .

We can introduce superpositions, e.g.

\[
\text{had} : Q_2 \rightarrow Q_2 \\
\text{had } x = \text{if}^\circ x
\]

\[
\text{then } \{ \text{qfalse} \mid (\!-\!1) \text{ qtrue} \} \\
\text{else } \{ \text{qfalse} \mid \text{qtrue} \}
\]
We can introduce superpositions, e.g.
\[
\text{had} : Q_2 \rightarrow Q_2
\]
\[
\text{had } x = \text{if}^\circ x
\]
\[
\text{then } \{ \text{qfalse} \mid (-1) \text{qtrue} \}
\]
\[
\text{else } \{ \text{qfalse} \mid \text{qtrue} \}
\]
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

Correct QML programs are dened by typing rules, e.g.

Foreach rule we can construct a quantum computation, i.e. a circuit.
Correct QML programs are defined by typing rules, e.g.

\[
\Gamma \vdash t : \sigma \otimes \tau \\
\Delta, x : \sigma, y : \tau \vdash u : C \\
\Gamma \otimes \Delta \vdash \text{let } (x, y) = t \text{ in } u : C \otimes \text{elim}
\]
Correct QML programs are defined by typing rules, e.g.

\[
\begin{align*}
\Gamma & \vdash t : \sigma \otimes \tau \\
\Delta, x : \sigma, y : \tau & \vdash u : C \\
\Gamma \otimes \Delta & \vdash \text{let } (x, y) = t \text{ in } u : C \quad \otimes\text{elim}
\end{align*}
\]

For each rule we can construct a quantum computation, i.e. a circuit.
\[ \quad \quad \Gamma 
\vdash t : \sigma \otimes \tau \\
\Delta, x : \sigma, y : \tau \vdash u : C \\
\quad \vdash \text{let } (x, y) = t \text{ in } u : C \quad \otimes \text{elim} \]
\[ 
\Gamma \vdash t : \sigma \otimes \tau \\
\Delta, x : \sigma, y : \tau \vdash u : C \\
\Gamma \otimes \Delta \vdash \text{let } (x, y) = t \text{ in } u : C \quad \otimes\text{elim}
\]
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.
A compiler is currently being implemented by my student Jonathan Grattage (in Haskell).
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.
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

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.
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

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.
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.
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.
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