Towards a High Level 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
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

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

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

Design based on semantic analogy:

- **FCC**  Finite classical computations
- **FQC**  Finite quantum computations
QML: a first-order functional language for quantum computations on finite types.

Design based on semantic analogy:

- **FCC** Finite classical computations
- **FQC** Finite quantum computations

Quantum control **and** quantum data.
QML:

- Design based on semantic analogy:
  - FCC Finite classical computations
  - FQC Finite quantum computations
- Quantum control **and** quantum data.
- Contraction is interpreted as sharing not cloning.
QML: a first-order functional language for quantum computations on finite types.

Design based on semantic analogy:

- **FCC**  Finite classical computations
- **FQC**  Finite quantum computations

Quantum control and quantum data.

Contraction is interpreted as sharing not cloning.

Control of decoherence,

hence no implicit weakening.
QML: a first-order functional language for quantum computations on finite types.

Design based on semantic analogy:
- FCC  Finite classical computations
- FQC  Finite quantum computations

Quantum control and quantum data.

Contraction is interpreted as sharing not cloning.

Control of decoherence,
hence no implicit weakening.

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

\[ eq : Q_2 \rightarrow Q_2 \rightarrow Q_2 \]

\[ eq \ a \ b = \]

\[ \text{let} \ (x, y, (a', b')) = \]

\[ \text{if}^\circ \ \{qfalse \mid qtrue\} \]

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

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

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

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

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

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

\[ \text{in} \ \text{had} \ x \]
Overview

1. Finite classical computation
2. Finite quantum computation
3. QML
4. Conclusions and further work
1. Finite classical computation
2. Finite quantum computation
3. QML
4. Conclusions and further work
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 computations on finite types

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

- Quantum mechanics is time-reversible...
- ...hence quantum computation is based on reversible operations.
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...
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):

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

\[ \begin{array}{c}
\text{A} \\
\phi \\
\text{B} \\
\hline
\text{h} \\
\text{H} \\
\phi \\
\text{G} \\
\end{array} \]
Composing computations
Composing computations

\[ \phi_{\beta \circ \alpha} \]
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 $\cup \alpha \in A \rightarrow B$ by

$$
\begin{array}{ccc}
A \times H & \xrightarrow{\phi} & B \times G \\
\downarrow (-, h) & & \downarrow \pi_1 \\
A & \xrightarrow{\cup \alpha} & B
\end{array}
$$
Extensional equality

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

\[
\begin{align*}
A \times H & \xrightarrow{\phi} B \times G \\
A & \xrightarrow{\cup \alpha} B
\end{align*}
\]

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

Theorem:

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

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

Hence, classical computations upto extensional equality give rise to the category \( \text{FCC} \).
Extensional 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 \( f \in A \rightarrow B \) on finite sets \( A, B \) can be realized by a computation.
**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 \( f \in A \rightarrow B \) 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**

$\phi_{\pi_1}$
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
Pure quantum values
A pure quantum value over a finite set $A$ is given by $\vec{v} \in A \rightarrow \mathbb{C}$ with unit norm:

$$||\vec{v}|| = \sum_{a \in A} |\vec{v}a|^2 = 1$$
A pure quantum value over a finite set $A$ is given by $\vec{v} \in A \rightarrow \mathbb{C}$ with unit norm:

$$||\vec{v}|| = \sum a \in A. |\vec{v}a|^2 = 1$$

$A \rightarrow \mathbb{C}$ is monadic, giving rise to the category of (finite dimensional) vector spaces.
Vector spaces as a monad

\[
\text{type } \text{Vec } a = a \to \mathbb{C}
\]

\[
\text{return } \in \text{Eq } a \Rightarrow a \to \text{Vec } a
\]

\[
\text{return } a \ b = \text{if } a \equiv b \text{ then } 1 \text{ else } 0
\]

\[
(\gg=) \in \text{Finite } a \Rightarrow
\text{Vec } a \to (a \to \text{Vec } b) \to \text{Vec } b
\]

\[
\text{as } \gg= f = \lambda b \to \text{sum } [(\text{as } a) \ast (f \ a \ b) \\
| a \leftarrow \text{enumerate}]
\]
Reversible quantum operations

Reversible operations on pure quantum values are given by unitary operators. On finite dimensional vector spaces, unitary = norm preserving linear isomorphism. The inverse is given by the adjoint:
Reversible quantum operations

Reversible operations on pure quantum values are given by *unitary operators*. 
Reversible quantum operations

- Reversible operations on pure quantum values are given by *unitary operators*.
- On finite dimensional vector spaces: unitary = norm preserving linear iso.
Reversible quantum operations

Reversible operations on pure quantum values are given by *unitary operators*.

On finite dimensional vector spaces: unitary = norm preserving linear iso.

The inverse is given by the adjoint:
\[
adj \in (a \rightarrow \text{Vec} \ b) \rightarrow b \rightarrow \text{Vec} \ a
\]
\[
adj \ f \ b \ a = \text{conjugate} \ (f \ a \ b)
\]
Quantum computations (FQC)
Quantum computations (FQC)

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

Diagram:

```
A ------------ B
|   \phi    |
|          |
h  H ----------- G
```
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 $\vec{h} \in H \rightarrow \mathbb{C}$,
- a finite set $G$, the base of the space of garbage states,
- a unitary operator $\phi \in A \otimes H \rightarrow^{\text{unitary}} B \otimes G$. 
Composing quantum computations
Composing quantum computations
Extensional equality?

...is a bit more subtle. There is no (sensible) operator on vector spaces replacing.

Indeed: Forgetting part of a pure state results in a mixed state.
Extensional equality?

... is a bit more subtle.
Extensional equality?

... is a bit more subtle.

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

... 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 operators

Mixed states are represented by density operators (positive operators with unit trace).
Density operators

Mixed states are represented by density operators $\rho \in A \rightarrow A$ (positive operators with unit trace).
Density operators

- Mixed states are represented by *density operators* $\rho \in A \rightarrow A$ (positive operators with unit trace).

- $\rho \overline{\psi} = \lambda \overline{\psi}$ is interpreted as the system is in the pure state $\overline{\psi}$ with probability $\lambda$. 
Superoperators

Morphisms on mixed states are completely positive linear operators on the space of density operators, called superoperators. Every unitary operator gives rise to a superoperator.
Morphisms on mixed states are completely positive linear operators on the space of density operators, called superoperators.
Superoperators

- Morphisms on mixed states are completely positive linear operators on the space of density operators, called superoperators.
- Every unitary operator $\phi$ gives rise to a superoperator $\hat{\phi}$. 
Superoperators

- Morphisms on mixed states are completely positive linear operators on the space of density operators, called superoperators.
- Every unitary operator $\phi$ gives rise to a superoperator $\hat{\phi}$.
- There is an operator $\text{tr}_{B,G} \in B \otimes G \to_{\text{super}} B$

called partial trace.
Extensiojnal equality
A quantum computation $\alpha \in \text{FQC} AB$ gives rise to a superoperator $U\alpha \in A \rightarrow_{\text{super}} B$

\[
\begin{array}{ccc}
A \otimes H & \xrightarrow{\phi} & B \otimes G \\
\downarrow \otimes \tilde{h} & & \downarrow \text{tr}_G \\
A & \xleftarrow{U\alpha} & B
\end{array}
\]
Extensional equality

A quantum computation $\alpha \in \text{FQC} \ A \ B$ gives rise to a superoperator $\cup \alpha \in A \rightarrow \text{super} \ B$

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

Theorem:

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

Hence, quantum computations upto extensional equality give rise to the category FQC.
Extensional equality ...

Theorem:

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

Hence, quantum computations upto extensional equality give rise to the category \( \mathbf{FQC} \).

Theorem: Every superoperator \( F \in \mathcal{A} \xrightarrow{\text{super}} \mathcal{B} \) (on finite Hilbert spaces) comes from a quantum computation.
Extensional equality …

- Theorem:

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

- Hence, quantum computations upto extensional equality give rise to the category $\mathbf{FQC}$.

- Theorem: Every superoperator $F \in A \xrightarrow{\text{super}} B$ (on finite Hilbert spaces) comes from a quantum computation. (U is full and faithful).
Classical vs quantum

- Classical
- Quantum
- Finite sets
- Finite-dimensional Hilbert spaces
- Bijections
- Unitary operators
- Cartesian product
- Tensor product
- Functions
- Superoperators
- Projections
- Partial trace

...
## 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>

- Finite sets
- Cartesian product
- Tensor product
- Functions
- Linear operators
- Unitary operators
- Bijections
- Hilbert spaces
- Projections
- Partial trace
- Superoperators
### 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>Classical (FCC)</td>
<td>Quantum (FQC)</td>
</tr>
<tr>
<td>----------------</td>
<td>--------------</td>
</tr>
<tr>
<td>finite sets</td>
<td>finite dimensional Hilbert spaces</td>
</tr>
<tr>
<td>bijections</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>bijections</td>
<td>unitary operators</td>
</tr>
</tbody>
</table>
## Classical vs Quantum

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

<table>
<thead>
<tr>
<th>Classical ($\textbf{FCC}$)</th>
<th>Quantum ($\textbf{FQC}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>finite sets</td>
<td>finite dimensional Hilbert spaces</td>
</tr>
<tr>
<td>bijections</td>
<td>unitary operators</td>
</tr>
<tr>
<td>cartesian product ($\times$)</td>
<td>tensor product ($\otimes$)</td>
</tr>
<tr>
<td>functions</td>
<td>superoperators</td>
</tr>
<tr>
<td>projections</td>
<td>partial trace</td>
</tr>
</tbody>
</table>
\( \pi_1 \circ \delta, \text{ classically} \)

\[ \pi_1 \circ \delta : 2 \rightarrow 2 \]
$\pi_1 \circ \delta$, classically

$\pi_1 \circ \delta : 2 \rightarrow 2$

$x : 2$

$0 : 2$

$\phi_\delta$

$\phi_{\pi_1}$

$x : 2$
\[ \pi_1 \circ \delta, \text{ classically} \]

\[ \pi_1 \circ \delta : 2 \rightarrow 2 \]

\[ x : 2 \]

\[ 0 : 2 \]

\[ \phi_\delta \]

\[ \phi_{\pi_1} \]

\[ = \]

\[ 2 \quad 2 \]
\( \pi_1 \circ \delta, \text{ quantum} \)
\[\pi_1 \circ \delta, \text{ quantum}\]

\[\begin{array}{c}
\text{input: } \left\{ \frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle \right\}
\end{array}\]
$\pi_1 \circ \delta$, quantum

\[
x : \mathcal{Q}_2 \quad \phi_\delta \quad \phi_{\pi_1} \quad x : \mathcal{Q}_2
\]

input: \( \left\{ \frac{1}{\sqrt{2}} \ket{0} + \frac{1}{\sqrt{2}} \ket{1} \right\} \)

output: \( \frac{1}{2}\left\{ \ket{0} \right\} + \frac{1}{2}\left\{ \ket{1} \right\} \)
\[ \pi_1 \circ \delta, \text{ quantum} \]

\[
\begin{align*}
\text{input: } & \left\{ \frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle \right\} \\
\text{output: } & \frac{1}{2} \left\{ |0\rangle \right\} + \frac{1}{2} \left\{ |1\rangle \right\}
\end{align*}
\]

\text{Decoherence!}
Control of decoherence

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

- QML is based on strict linear logic

Weakening is explicit and leads to decoherence.
Control of decoherence

- QML is based on strict linear logic
- Contraction is implicit and realized by $\phi_\delta$. 
Control of decoherence

- QML is based on strict linear logic
- Contraction is implicit and realized by $\phi_\delta$.
- Weakening is explicit and leads to decoherence.
3. QML

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

\[ t = x \mid \text{let } x = t \text{ in } u \mid x \uparrow \vec{y} \]

\[ \mid () \mid (t, u) \mid \text{let } (x, y) = t \text{ in } u \]

\[ \mid \text{qinl } t \mid \text{qinr } u \]

\[ \mid \text{case } t \text{ of } \{ \text{qinl } x \Rightarrow u \mid \text{qinr } y \Rightarrow u' \} \]

\[ \mid \text{case}^{\circ} t \text{ of } \{ \text{qinl } x \Rightarrow u \mid \text{qinr } y \Rightarrow u' \} \]

\[ \mid \{ (\kappa) t \mid (\iota) u \} \]
\[ Q_2 = 1 \oplus 1 \]

\[ q\text{true} = q\text{inl}() \]

\[ q\text{false} = q\text{inr}() \]

\[ \text{if } t \text{ then } u \text{ else } u' \]

\[ = \text{case } \{ q\text{inl }_\rightarrow u \mid q\text{inr }_\rightarrow u' \} \]

\[ \text{if}^\circ t \text{ then } u \text{ else } u' \]

\[ = \text{case}^\circ \{ q\text{inl }_\rightarrow u \mid q\text{inr }_\rightarrow u' \} \]
QML overview ...
Typing judgements

\[ \Gamma \vdash t : \sigma \quad \text{programs} \]

\[ \Gamma \vdash^\circ t : \sigma \quad \text{strict programs} \]
QML overview ...

Typing judgements

\[ \Gamma \vdash t : \sigma \quad \text{programs} \]
\[ \Gamma \vdash^\circ t : \sigma \quad \text{strict programs} \]

Semantics

\[ \Gamma \vdash t : \sigma \]
\[ [t] \in \mathbf{FQC}[\Gamma][\sigma] \]

\[ \Gamma \vdash^\circ t : \sigma \]
\[ [t] \in \mathbf{FQC}^\circ[\Gamma][\sigma] \]
The let-rule

\[
\frac{\Gamma \vdash t : \sigma \quad \Delta, x : \sigma \vdash u : \tau}{\Gamma \otimes \Delta \vdash \text{let } x = t \text{ in } u : \tau} \text{ let}
\]
The let-rule

\[ \Gamma \vdash t : \sigma \]

\[ \Delta, \ x : \sigma \vdash u : \tau \]

\[ \Gamma \otimes \Delta \vdash \text{let } x = t \text{ in } u : \tau \]
on contexts
on contexts

\[ \Gamma, x : \sigma \otimes \Delta, x : \sigma = (\Gamma \otimes \Delta), x : \sigma \]
\[ \Gamma, x : \sigma \otimes \Delta = (\Gamma \otimes \Delta), x : \sigma \quad \text{if } x \notin \text{dom } \Delta \]
\[ \bullet \otimes \Delta = \Delta \]
on contexts

\[ \Gamma, x : \sigma \otimes \Delta, x : \sigma = (\Gamma \otimes \Delta), x : \sigma \]
\[ \Gamma, x : \sigma \otimes \Delta = (\Gamma \otimes \Delta), x : \sigma \quad \text{if} \ x \notin \text{dom} \Delta \]

\[ \bullet \otimes \Delta = \Delta \]

\[ \begin{array}{ccc}
\Gamma \otimes \Delta & \phi C_{\Gamma,\Delta} & \Gamma \\
H_{\Gamma,\Delta} & & \Delta
\end{array} \]
Another source of decoherence
Another source of decoherence

\[ \text{forget mentions } x \]
\[ \text{forget : } 2 \rightarrow 2 \]
\[ \text{forget } x = \text{if } x \text{ then } \text{qtrue} \text{ else } \text{qtrue} \]
Another source of decoherence

- *forget* mentions \( x \)
  - \( \text{forget} : 2 \rightarrow 2 \)
  - \( \text{forget} \ x = \text{if} \ x \ \text{then} \ qtrue \ \text{else} \ qtrue \)

  but doesn’t use it.
Another source of decoherence

- *forget* mentions $x$
  - *forget* : 2 $\rightarrow$ 2
    - *forget* $x = \text{if } x \text{ then } \text{qtrue} \text{ else } \text{qtrue}$

- but doesn’t use it.

- Hence, it **has** to measure it!
⊕-elim

\[
\Gamma \vdash c : \sigma \oplus \tau \\
\Delta, x : \sigma \vdash t : \rho \\
\Delta, y : \tau \vdash u : \rho \\
\Gamma \otimes \Delta \vdash \text{case } c \text{ of } \{ \text{inl } x \Rightarrow t \mid \text{inr } y \Rightarrow u \} : \rho + \text{elim}
\]
$$
\Gamma \vdash c : \sigma \oplus \tau \\
\Delta, x : \sigma \vdash t : \rho \\
\Delta, y : \tau \vdash u : \rho
$$

$$
\Gamma \otimes \Delta \vdash \text{case } c \text{ of } \{ \text{inl } x \Rightarrow t \mid \text{inr } y \Rightarrow u \} : \rho
$$

Diagram: 

- $\Gamma \otimes \Delta$
- $H_{\Gamma,\Delta}$
- $\phi_{C_{\Gamma,\Delta}}$
- $\phi_b$
- $\phi_{[t|u]}$
- $\sigma \sqcup \tau$
- $Q_2$
- $G$
- $G_b$
- $H_{b}$
- $H_{t-u}$
-elim decoherence-free
$$\begin{align*}
\Gamma \vdash^a c : \sigma \oplus \tau \\
\Delta, x : \sigma \vdash^o t : \rho \\
\Delta, y : \tau \vdash^o u : \rho \quad t \perp u \\
\Gamma \otimes \Delta \vdash^a \text{case}^o_c \text{ of } \{ \text{inl } x \Rightarrow t \mid \text{inr } y \Rightarrow u \} : \rho
\end{align*}$$
⊕-elim decoherence-free

\[ \Gamma \vdash^a c : \sigma \oplus \tau \]
\[ \Delta, x : \sigma \vdash^\circ t : \rho \]
\[ \Delta, y : \tau \vdash^\circ u : \rho \quad t \perp u \]
\[ \Gamma \otimes \Delta \vdash^a \text{case}^\circ \ c \ 	ext{of} \ \{ \text{inl } x \Rightarrow t \mid \text{inr } y \Rightarrow u \} : \rho \]

\[\phi_{C,\Delta}(\phi_c, \phi_{f\mid g}) \]

\[\phi_{t \perp u}\]

\[\phi_c\]

\[\phi_{C,\Delta}\]

\[\phi_{f\mid g}\]

\[H_{\Gamma,\Delta}\]

\[H_c\]

\[H_{f\mid g}\]

\[G_c\]
This program has a type error, because
This program type checks, because

MGS Xmas 04 – p.40?
\[ forget' : 2 \rightarrow 2 \]
\[ forget' \ x = \text{if}^o \ x \ \text{then qtrue else qtrue} \]
This program has a type error, because `qtrue` is not a valid type.
This program has a type error, because `qtrue \neq qtrue`.

\[
\begin{align*}
\text{forget'} : 2 \rightarrow 2 \\
\text{forget'} x &= \text{if}^\circ x \text{ then } qtrue \text{ else } qtrue \\
\text{qnot} : 2 \rightarrow 2 \\
\text{qnot } x &= \text{if}^\circ x \text{ then } qfalse \text{ else } qtrue
\end{align*}
\]
\[
\text{if}^\circ
\]

\[
\text{forget'} : 2 \to 2
\]

\[
\text{forget'}\ x = \text{if}^\circ\ x \text{ then } q\text{true} \text{ else } q\text{true}
\]

This program has a type error, because \( q\text{true} \not\rightarrow q\text{true}. \)

\[
\text{qnot} : 2 \to 2
\]

\[
\text{qnot}\ x = \text{if}^\circ\ x \text{ then } q\text{false} \text{ else } q\text{true}
\]

This program typechecks, because \( q\text{false} \downarrow q\text{true}. \)
4. QML

1. Finite classical computation
2. Finite quantum computation
3. 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. 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.

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. 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.
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.
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.