A Functional Quantum Programming Language

Thorsten Altenkirch
University of Nottingham
based on joint work with Jonathan Grattage
and discussions with V.P. Belavkin
supported by EPSRC grant GR/S30818/01
What you always wanted to know about quantum computation but never dared to ask.
Another alternative title

Quantum programming
for the
lazy functional programmer
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 no Nature is classical after all! Assumption: Nature is fair...
Simulation of quantum systems is expensive:
Exponential time to simulate polynomial circuits.
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?
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.
Simulation of quantum systems is expensive: Exponential time to simulate polynomial circuits.

Feynman: *Can we exploit this fact to perform computations more efficiently?*

Shor: Factorisation in quantum polynomial time.

Grover: Blind search in $\Theta(\sqrt{n})$

Can we build a quantum computer?

**yes** We can run quantum algorithms.

**no** Nature is classical after all!
Background

- Simulation of quantum systems is expensive: Exponential time to simulate polynomial circuits.
- Feynman: *Can we exploit this fact to perform computations more efficiently?*
- Shor: Factorisation in quantum polynomial time.
- Grover: Blind search in $\Theta(\sqrt{n})$
- Can we build a quantum computer?
  - yes  We can run quantum algorithms.
  - no   Nature is classical after all!

*Assumption: Nature is fair...*
The quantum software crisis

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

- Quantum control and quantum data.
QML

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

Quantum control and quantum data.

Design guided by semantics

Analogy with classical computation

FCC Finite classical computations

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

Quantum control and quantum data.

Design guided by semantics

Analogy with classical computation

FCC  Finite classical computations

FQC  Finite quantum computations

Important issue: control of decoherence
QML

- 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

\[ \text{had} : Q_2 \rightarrow Q_2 \]
\[ \text{had } x = \text{if}^\circ x \]
\[ \text{then } \{ \text{qfalse} | (-1) \text{ qtrue} \} \]
\[ \text{else } \{ \text{qfalse} | \text{qtrue} \} \]
Something we know well . . .

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

- Classical computations on finite types.

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

Newtonian mechanics, Maxwellian electrodynamics are also time-reversible . . . hence classical computation should be based on reversible operations.
Something we know well . . .

- Classical computations on finite types.
- Quantum mechanics is time-reversible . . .
Something we know well . . .

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

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

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

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

\[ \begin{array}{c}
A \\ \hline
h \\ \hline
H \\
\end{array} \quad \phi \quad \begin{array}{c}
B \\ \hline
G \\
\end{array} \]
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 finite set of garbage states $G$,
- a bijection $\phi \in A \times H \simeq B \times G$, 

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

\[ h \quad H \quad G \]
Composing computations
Composing computations

\[ \phi_{\beta \circ \alpha} \]
Extensional equality
A classical computation $\alpha = (H, h, G, \phi)$ induces a function $\cup \alpha \in A \rightarrow B$ by

$$
\begin{array}{c}
A \times H \\ \downarrow (\cdot, h) \\
A
\end{array} \quad \xrightarrow{\phi} \quad 
\begin{array}{c}
B \times G \\ \downarrow \pi_1 \\
B
\end{array}
$$
A classical computation $\alpha = (H, h, G, \phi)$ induces a function $\cup \alpha \in A \rightarrow B$ by

$$
\begin{align*}
A \times H & \xrightarrow{\phi} B \times G \\
\downarrow (\cdot, h) & \quad \downarrow \pi_1 \\
A & \xrightarrow{\cup \alpha} B
\end{align*}
$$

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 up to 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 \( \text{FCC} \).

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 upto extensional equality give rise to the category \( \text{FCC} \).

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

**Translation for Category Theoreticans:**

\( U \) 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

\[
\begin{array}{ccc}
2 & \rightarrow & 2 \\
\downarrow & & \\
2 & \rightarrow & 1
\end{array}
\]

$\phi_{\pi_1}$
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**

$\phi_\delta$

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

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

$\phi_\delta \ (1, x) = (1, \bar{x})$
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>
<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>
</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></td>
</tr>
</tbody>
</table>

A Functional Quantum Programming Language – p.15/?
## 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>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></td>
</tr>
</tbody>
</table>
### Classical vs quantum

<table>
<thead>
<tr>
<th>classical (<strong>FCC</strong>)</th>
<th>quantum (<strong>FQC</strong>)</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>
</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 (×)</td>
<td>tensor product (⊗)</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 (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>partial trace</td>
</tr>
</tbody>
</table>
\[ \pi_1 \circ \delta, \text{ classically} \]

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

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

$x : 2$ \hspace{1cm} $x : 2$

$0 : 2$

$\phi_\delta$ \hspace{1cm} $\phi_{\pi_1}$
\[ \pi_1 \circ \delta, \text{ classically} \]

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

\[ x : 2 \quad \phi_\delta \quad \phi_{\pi_1} \]

\[ 0 : 2 \]

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

\[
\begin{array}{c}
x : Q_2 \\
\phi_{\delta} \\
0 : Q_2 \\
\phi_{\pi_1}
\end{array}
\]
\[ \pi_1 \circ \delta, \text{ quantum} \]

\[ x : Q_2 \]

\[ 0 : Q_2 \]

\[ \phi_\delta \]

\[ \phi_{\pi_1} \]

input: \( \left\{ \frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle \right\} \)
\( \pi_1 \circ \delta, \text{ quantum} \)

\[
x : Q_2 \\
0 : Q_2
\]

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

**input:** \[ \left\{ \frac{1}{\sqrt{2}} \left| 0 \right\rangle + \frac{1}{\sqrt{2}} \left| 1 \right\rangle \right\} \]

**output:** \[ \frac{1}{2} \left\{ \left| 0 \right\rangle \right\} + \frac{1}{2} \left\{ \left| 1 \right\rangle \right\} \]
$\pi_1 \circ \delta$, quantum

\[
\begin{array}{c}
x : Q_2 \\
0 : Q_2
\end{array}
\quad \phi_\delta 
\quad \phi_{\pi_1}
\quad \begin{array}{c}
x : Q_2 \\
0 : Q_2
\end{array}
\]

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

output: \( \frac{1}{2} \left\{ |0\rangle \right\} + \frac{1}{2} \left\{ |1\rangle \right\} \)

Decoherence!
Control of decoherence
Control of decoherence

- QML is based on strict linear logic
Control of decoherence

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

\[ t = x \mid \text{let } x = t \text{ in } u \mid x \uparrow xs \]
\[ \mid () \mid (t, u) \mid \text{let } (x, y) = t \text{ in } u \]
\[ \mid \text{qinl } t \mid \text{qinr } u \]
\[ \mid \text{case } t \text{ of } \{ \text{qinl } x \Rightarrow u \mid \text{qinr } y \Rightarrow u' \} \]
\[ \mid \text{case}^\circ t \text{ of } \{ \text{qinl } x \Rightarrow u \mid \text{qinr } y \Rightarrow u' \} \]
\[ \mid \{ (\kappa) t \mid (\iota) u \} \]
Q_2 = 1 \oplus 1
qtrue = qinl ()
qfalse = qinr ()
if \ t \ then \ u \ else \ u'
 \quad = \text{case} \{ \text{qinl} \_ \Rightarrow u \mid \text{qinr} \_ \Rightarrow u' \}
if^\circ \ t \ then \ u \ else \ u'
 \quad = \text{case}^\circ \{ \text{qinl} \_ \Rightarrow u \mid \text{qinr} \_ \Rightarrow u' \}
QML overview ...
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
\[ \Gamma \vdash t : \sigma \quad \Rightarrow \quad [t] \in \text{FQC}[\Gamma][\sigma] \]
\[ \Gamma \vdash^\circ t : \sigma \quad \Rightarrow \quad [t] \in \text{FQC}^\circ[\Gamma][\sigma] \]
The let-rule

$$\Gamma \vdash t : \sigma$$

$$\Delta, x : \sigma \vdash u : \tau$$

$$\Gamma \otimes \Delta \vdash \text{let } x = t \ \text{in} \ u : \tau$$
The let-rule

\[
\begin{align*}
\Gamma &\vdash t : \sigma \\
\Delta, \; x : \sigma &\vdash u : \tau \\
\frac{}{\Gamma \otimes \Delta \vdash \text{let } x = t \text{ in } u : \tau}
\end{align*}
\]
on contexts
on contexts

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

\[ \begin{align*}
\Gamma, x : \sigma \otimes \Delta, x : \sigma & = (\Gamma \otimes \Delta), x : \sigma \\
\Gamma, x : \sigma \otimes \Delta & = (\Gamma \otimes \Delta), x : \sigma \quad \text{if } x \notin \text{dom } \Delta \\
\bullet \otimes \Delta & = \Delta
\end{align*} \]

\[ \begin{align*}
\Gamma \otimes \Delta & \quad \phi C_{\Gamma, \Delta} \quad \Gamma \\
H_{\Gamma, \Delta} & \quad \Delta
\end{align*} \]
Another source of decoherence
Another source of decoherence

- \texttt{forget} mentions \( x \)

- \texttt{forget} : 2 \rightarrow 2

- \texttt{forget} \( x = \text{if } x \text{ then qtrue else qtrue} \)
Another source of decoherence

- `forget` mentions $x$
  - `forget : 2 → 2`
  - `forget x = if x then qtrue else qtrue`
- but doesn’t use it.
Another source of decoherence

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

- but doesn’t use it.

- Hence, it **has** to measure it!
\[\Gamma \vdash c : \sigma \oplus \tau \]
\[\Delta, x : \sigma \vdash t : \rho \]
\[\Delta, y : \tau \vdash u : \rho \]
\[\Gamma \otimes \Delta \vdash \text{case } c \text{ of } \{ \text{inl } x \Rightarrow t \mid \text{inr } y \Rightarrow u \} : \rho \] + elim
\[ \Gamma \vdash c : \sigma \oplus \tau \]
\[ \Delta, x : \sigma \vdash t : \rho \]
\[ \Delta, y : \tau \vdash u : \rho \]
\[ \Gamma \otimes \Delta \vdash \text{case } c \text{ of } \{ \text{inl } x \Rightarrow t \mid \text{inr } y \Rightarrow u \} : \rho \]
-elim decoherence-free
\( \oplus \)-elim decoherence-free

\[
\Gamma \vdash^a c : \sigma \oplus \tau \\
\Delta, x : \sigma \vdash^o t : \rho \\
\Delta, y : \tau \vdash^o u : \rho \quad t \perp u \\
\Gamma \otimes \Delta \vdash^a \text{case}^o c \text{ of } \{ \text{inl } x \Rightarrow t \mid \text{inr } y \Rightarrow u \} : \rho \quad \oplus - \text{elim}^o
\]
⊕-elim decoherence-free

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

\[ H_{\Gamma,\Delta} \]
\[ \phi C_{\Gamma,\Delta} \]
\[ \phi_c \]
\[ \phi[f|g] \]
\[ \phi_{t \perp u} \]
\[ H_c \]
\[ H_{f-g} \]
\[ \Delta \]
\[ \sigma \sqcup \tau \]
\[ H_{\Gamma,\Delta} \]
\[ \rho \]
\[ G \]
\[ G_c \]
This program has a type error, because
This program type checks because

A Functional Quantum Programming Language – p.27?
\( \text{if}^\circ \)

\[
\text{forget'} : 2 \rightarrow 2 \\
\text{forget'} \ x = \text{if}^\circ \ x \ \text{then} \ q\text{true} \ \text{else} \ q\text{true}
\]
This program has a type error, because qtrue \not\equiv qtrue.
This program has a type error, because `qtrue` ∉ `qtrue`.

```latex
\text{if}^\circ
```

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

\[
\text{forget'} x = \text{if}^\circ x \text{ then } \text{qtrue} \text{ else } \text{qtrue}
\]
\( \text{if}^\circ \)

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

This program has a type error, because \( \text{qtrue} \neq \text{qtrue} \).

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

This program typechecks, because \( \text{qfalse} \downarrow \text{qtrue} \).
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?
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.