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

*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!
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: 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.
Draft paper available (Google: Thorsten, functional, quantum)
Compiler under construction (Jonathan)

Towards a High Level Quantum Programming Language – p.4/36
QML: a functional language for quantum computations on finite types.
QML: a functional language for quantum computations on finite types.

Quantum control and quantum data.
QML

- 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
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: 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**

Draft paper available
(Google: Thorsten, functional, quantum)
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
- Draft paper available
  (Google:Thorsten,functional,quantum)

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

had : \( Q_2 \rightarrow Q_2 \)

had \( x = \text{if}^\circ x \)

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

\[ \text{else} \{ \text{qfalse} | \text{qtrue} \} \]
Overview

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

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

\[
\begin{array}{c}
A & B \\
\hline
\phi & \\
\hline
h & H & G
\end{array}
\]
Classical computation (FCC)

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

\[ A \xrightarrow{\phi} B \]

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

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

\[ \cup \alpha \ a = \pi_1 \ \phi (h, a) \]
Semantics

- A classical computation
  \[ \alpha = (A, B, H, h \in H, G, \phi \in A \times H \simeq B \times G) \]
  induces a function \( \alpha \in A \rightarrow B \) by
  \[ \Upsilon \alpha a = \pi_1 \phi(h, a) \]

- **Theorem** Any function \( f \in A \rightarrow B \) (on finite sets \( A, B \)) can be realized by a quantum computation.
Composing classical computations
Composing classical computations

\[
\phi_{\beta \circ \alpha}
\]

Towards a High Level Quantum Programming Language – p.11/36
Composing classical computations

Theorem:

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

$$U(\beta \circ \alpha) = (U\beta) \circ (U\alpha)$$
Coming next: Quantum computations

Develop FQC analogously to FCC...
Linear algebra revision
Linear algebra revision

Given a finite set $A$ (the base) $\mathbb{C}A = A \rightarrow \mathbb{C}$ is a **Hilbert space**.
Linear algebra revision

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 $f \in A \rightarrow B$
Linear algebra revision

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 \( f \in A \rightarrow B \)

**Norm of a vector:**

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

Given a finite set $A$ (the base)
$\mathbb{C} A = A \rightarrow \mathbb{C}$ 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 $f \in A \rightarrow_{\text{linear}} B$

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

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

A pure state 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 $\|\nu\| = 1$. 
Basics of quantum computation

- 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):
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

\[ \Phi_\beta \circ \Phi_\alpha \]
Semantics of quantum computations...

...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.
... is a bit more subtle.
... is a bit more subtle.

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

Mixed states are represented by density matrices. Operations on mixed states (i.e. density matrices) are represented by superoperators. Every unitary operator gives rise to a superoperator.
Density matrices and superoperators

- Mixed states are represented by density matrices.
Density matrices and superoperators

- Mixed states are represented by *density matrices*.
- Operations on mixed states (i.e. density matrices) are represented by superoperators.
Mixed states are represented by *density matrices*.

Operations on mixed states (i.e. density matrices) are represented by superoperators.

Every unitary operator $\phi$ gives rise to a superoperator $\hat{\phi}$. 
Density matrices and superoperators

- Mixed states are represented by density matrices.
- Operations on mixed states (i.e. density matrices) are represented by superoperators.
- Every unitary operator $\phi$ gives rise to a superoperator $\hat{\phi}$.
- There is an operator $\text{tr}_{B,G} \in B \otimes G \overset{\text{super}}{\rightarrow} B$
  
  called partial trace.
Semantics
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 $U \alpha \in A \xrightarrow{\text{super}} B$

\[ A \otimes H \xrightarrow{\tilde{\phi}} B \otimes G \]
\[ A \xrightarrow{\cup \alpha} B \]
\[ A \xrightarrow{\otimes \tilde{h}} B \]
\[ B \xrightarrow{\text{tr}_G} \]

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

-Finite sets
-Finite dimensional Hilbert spaces
-Bijections
-Unitary operators
-Cartesian product (x)
-Tensor product (⊗)
-Functions
-Superoperators
-Projections
-Partial trace
## Classical vs quantum

<table>
<thead>
<tr>
<th>classical</th>
<th>quantum</th>
</tr>
</thead>
</table>

## Classical vs Quantum

<table>
<thead>
<tr>
<th>Classical</th>
<th>Quantum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finite sets</td>
<td></td>
</tr>
<tr>
<td>classical</td>
<td>quantum</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</th>
<th>Quantum</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>
## Classical vs quantum

<table>
<thead>
<tr>
<th>classical</th>
<th>quantum</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</th>
<th>quantum</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></td>
</tr>
<tr>
<td>classical</td>
<td>quantum</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</th>
<th>Quantum</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</th>
<th>quantum</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>Classical</td>
<td>Quantum</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 (×)</td>
<td>tensor product (⊗)</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</th>
<th>Quantum</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>
Decoherence
Decoherence

\[
\begin{array}{c}
2 \\
\phi_\delta \\
0 \\
\phi_\pi_1 \\
2
\end{array}
\]
Decoherence

Classically

\[ \pi_1 \circ \delta = I \]
Decoherence

\[ \phi_\delta \circ \phi_{\pi_1} = I \]
Decoherence

Classically

Quantum

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

Classically

\[ \pi_1 \circ \delta = I \]

Quantum

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

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

1. Semantics of finite classical and quantum computation
2. QML basics
3. Compiling QML
4. 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 $1, \sigma \otimes \tau, \sigma \oplus \tau$
QML basics

- QML is a first order functional languages, i.e. programs are well-typed expressions.
- QML types are $1, \sigma \otimes \tau, \sigma \oplus \tau$
- Qbits $Q_2 = 1 \oplus 1$
QML basics

- QML is a first order functional languages, i.e. programs are well-typed expressions.
- QML types are $1, \sigma \otimes \tau, \sigma \oplus \tau$
- Qbits $Q_2 = 1 \oplus 1$
- Qbytes
  $$Q_2^8 = Q_2 \otimes Q_2 \otimes Q_2 \otimes Q_2 \otimes Q_2 \otimes Q_2 \otimes Q_2 \otimes Q_2.$$
A QML program is an expression in a context of typed variables, e.g.

\[
qnot : Q_2 \rightarrow Q_2 \\
qnot x = \text{if}^\circ x \\
\text{then } qfalse \\
\text{else } qtrue
\]
QML basics . . .

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

$$ qnot : Q_2 \rightarrow Q_2 $$

$$ qnot \ x = \text{if}^\circ \ x $$

then $ qfalse $

derse \ 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.

\[ qftst : Q_2 \otimes Q_2 \rightarrow Q_2 \]

\[ qftst (x, y) = x \]

is illegal,
QML basics …

Forgetting variables has to be explicit. E.g.

\[ qfst : Q_2 \otimes Q_2 \to Q_2 \]
\[ qfst (x, y) = x \]

is illegal, but

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

is ok.
QML basics . . .

There are two different if-then-else (or more generally case) constructs.
QML basics . . .

There are two different if-then-else (or more generally case) constructs.

\[
id : Q_2 \to Q_2
\]

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

\[
\text{then } qtrue
\]

\[
\text{else } qfalse
\]

is just the identity,
QML basics . . .

There are two different if-then-else (or more generally case) constructs.

\[ id : Q_2 \rightarrow Q_2 \]
\[ id \ x = \operatorname{if}^\circ \ x \]

then \( qtrue \)

else \( qfalse \)

is just the identity, but

\[ \text{meas} : Q_2 \rightarrow Q_2 \]
\[ \text{meas} \ x = \operatorname{if} \ x \]

then \( qtrue \)

else \( qfalse \)

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

Using if° is only allowed, if the branches are orthogonal, i.e. observable different.
QML basics . . .

Using `if` is only allowed, if the branches are orthogonal, i.e. observable different.

\[
\text{cswap} : \mathcal{Q}_2 \otimes \mathcal{Q}_2 \rightarrow \mathcal{Q}_2 \rightarrow \mathcal{Q}_2 \otimes \mathcal{Q}_2
\]

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

\[
\text{then} \ (y, x)
\]

\[
\text{else} \ (x, y)
\]

is illegal,
QML basics . . .

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

\begin{align*}
\texttt{cswap} : Q_2 \otimes Q_2 &\rightarrow Q_2 \rightarrow Q_2 \otimes Q_2 \\
\texttt{cswap} \ (x, y) \ c &\ = \ \texttt{if}^\circ \ c \\
\texttt{then} \ (y, x) \\
\texttt{else} \ (x, y)
\end{align*}

is illegal, but

\begin{align*}
\texttt{cswap} : Q_2 \otimes Q_2 &\rightarrow Q_2 \rightarrow Q_2 \otimes (Q_2 \otimes Q_2) \\
\texttt{cswap} \ (x, y) \ c &\ = \ \texttt{if}^\circ \ c \\
\texttt{then} \ (q\texttt{true}, (y, x)) \\
\texttt{else} \ (q\texttt{false}, (x, y))
\end{align*}

is ok.
We can introduce superpositions, e.g.

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

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

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

\[
\text{else } \{ \text{qfalse} | \text{qtrue} \}
\]
QML basics ...

We can introduce superpositions, e.g.

$had: Q_2 \rightarrow Q_2$

$had \ x = \ if^\circ \ x$

then \ \{ qfalse \mid (−1) \ qtrue \} \\
else \ \{ qfalse \mid qtrue \}$

However, the terms in the superposition have to be orthogonal.
3. Compiling QML

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

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

Foreach rule we can construct a quantum computation, i.e. a circuit.
Compilation

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.

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

For each rule we can construct a quantum computation, i.e. a circuit.
\( \otimes\text{-elim} \)

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

\[
\Gamma \vdash t : \sigma \otimes \tau \\
\Delta, x : \sigma, y : \tau \vdash u : C \\
\Gamma \otimes \Delta \vdash \text{let } (x, y) = t \text{ in } u : C
\]

\(\otimes\)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.
4. 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. 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.