Compiling a functional quantum programming language

Jonathan Grattage
with Thorsten Altenkirch

www.cs.nott.ac.uk/~jjg/qml
School of Computer Science & IT,
University of Nottingham

Motivation

- 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!*”

- Our Solution:
  A high-level quantum programming language with a structure familiar to functional programmers, which supports reasoning and and algorithm design

Quantum Languages

- P. Zuliani, PhD 2001, *Quantum Programming* (qGCL)
- P. Selinger, MSCS 2003, *Towards a Quantum Programming Language* (QPL)
- A. van Tonder, SIAM 2003, *A Lambda Calculus for Quantum Computation*
- A. Sabry, Haskell 2003, *Modeling quantum computing in Haskell*
- P. Selinger and B. Valiron, TLCA 2005, *A lambda calculus for quantum computation with classical control*
- ...
- All based on "*Quantum* data, *Classical* control"

QML

The University of Nottingham

- A first-order functional language for quantum computations on finite types
- "Quantum Data *and* Control"
- Based on strict linear logic - controlled, explicit, weakening
- Types:

$$\sigma = 1 \mid \sigma \otimes \tau \mid \sigma \oplus \tau$$

- Terms:

$$\begin{aligned} t = x \; & \mid \mathbf{let}\ x = t\ \mathbf{in}\ u \mid x^{\vec{y}} \\ & \mid () \mid \mathbf{let}\ (x, y) = t\ \mathbf{in}\ u \mid (t, u) \\ & \mid \mathbf{if}\ t\ \mathbf{then}\ u\ \mathbf{else}\ u' \\ & \mid \mathbf{if}^{\circ}\ t\ \mathbf{then}\ u\ \mathbf{else}\ u' \\ & \mid \{(\kappa)\ \mathrm{qfalse} \mid (\iota)\ \mathrm{qtrue}\} \end{aligned}$$

$\kappa,\ \iota \in \mathbb{C}$

Deutsch algorithm

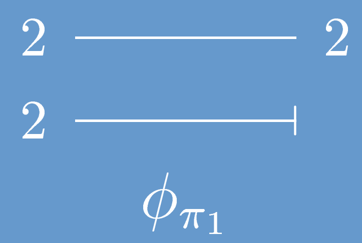
$$deutsch : 2 \multimap 2 \multimap \mathcal{Q}_{\mathbf{2}}$$

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

Control of Decoherence

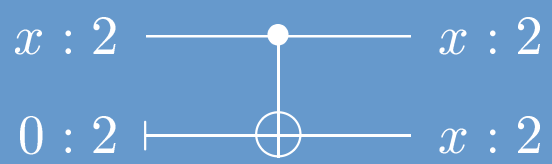
- Projection Function

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



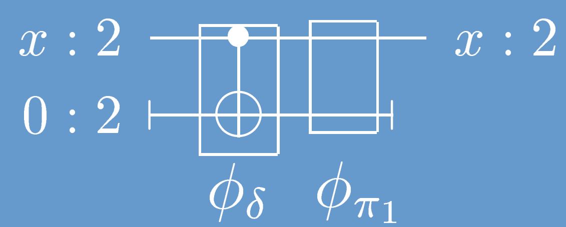
- Diagonal Function

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



Control of Decoherence

- $\pi_1 . \delta : 2 \to 2$



- Classical Case:

- Quantum Case:
  Input $= \{\frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle\}$ (equal superposition)
  Output $= \frac{1}{2}\{|0\rangle\} + \frac{1}{2}\{|1\rangle\}$ (probability distribution)

  Decoherence! Not the identity function

More Decoherence

- *forget* mentions x

$$forget : 2 \multimap 2$$

$$forget\ x = \mathbf{if}\ x\ \mathbf{then}\ \mathrm{qtrue}\ \mathbf{else}\ \mathrm{qtrue}$$

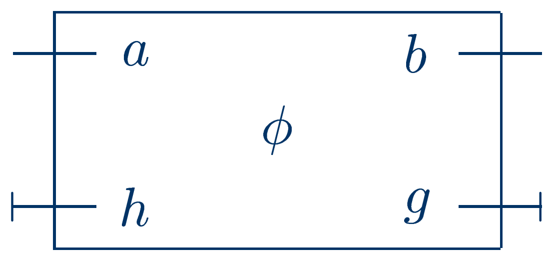
- but doesn't use it.
- Hence, it **has** to measure it!
- **if** always measures the conditional, returning only one branch

$\mathbf{if}^\circ$

- $$forget' : 2 \multimap 2$$
  $$forget'\ x = \mathbf{if}^\circ\ x\ \mathbf{then}\ \mathrm{qtrue}\ \mathbf{else}\ \mathrm{qtrue}$$
- This program has a type error, because `qtrue` $\not\perp$ `qtrue`.
- $$qnot : 2 \multimap 2$$
  $$qnot\ x = \mathbf{if}^\circ\ x\ \mathbf{then}\ \mathrm{qfalse}\ \mathbf{else}\ \mathrm{qtrue}$$
- This program typechecks, because `qfalse` $\perp$ `qtrue`.

Compiler Design

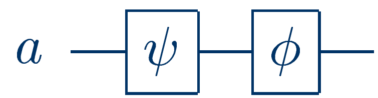
- Takes in QML expressions
- Compiled into **FQC** (Finite Quantum Computation) objects
-



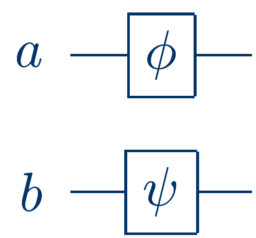
- ϕ = quantum circuit
- Circuit represented as simple combinators
- Can be directly simulated, or passed to any standard simulator
- . . . or a real quantum computer

Quantum Machine Code

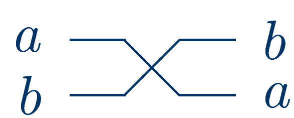
- *Quantum circuits* of size $a \in \mathbb{N}$, defined inductively
- Sequential Composition $(\phi \circ \psi)$



- Parallel Composition $(\phi \otimes \psi)$



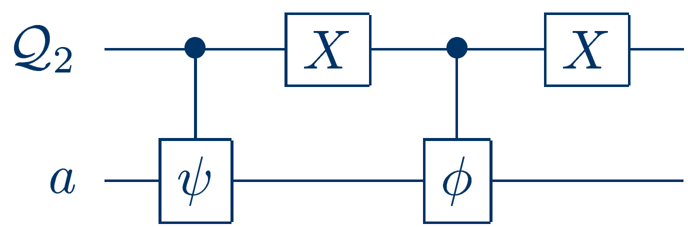
- Permutations (rewiring)



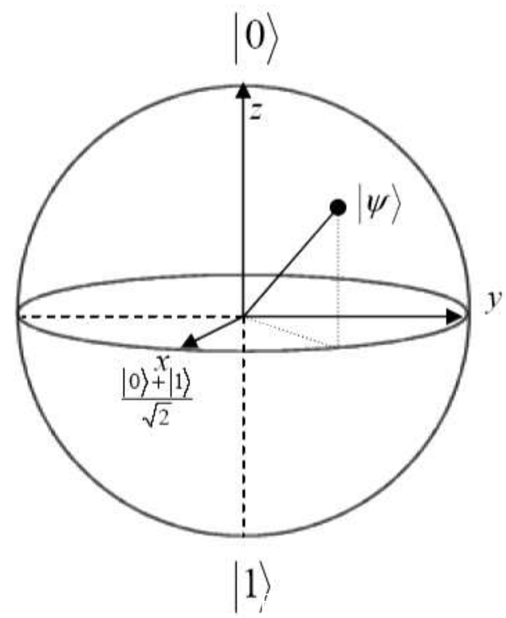
Quantum Machine Code

The University of Nottingham

- Conditional Application $(\phi \mid \psi)$

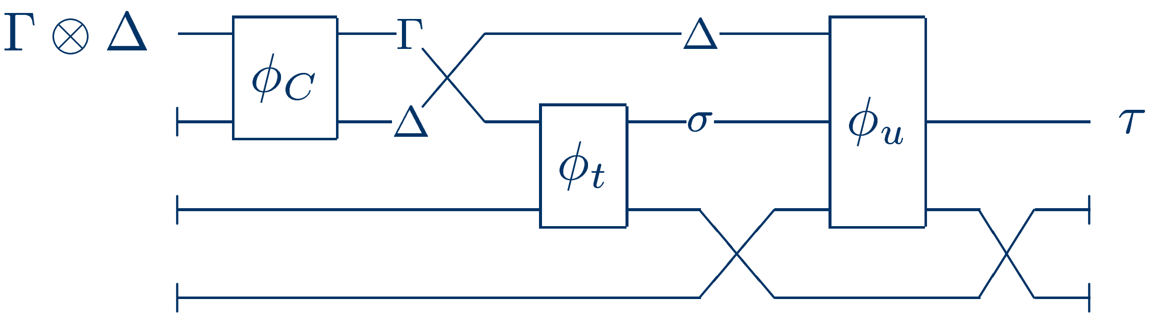


- Rotation (rot u, where $u \in 2 \to 2 \to \mathbb{C}$ is a unitary matrix)



Compiling the let-rule

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



Summary

- QML is a first-order functional language for quantum computations on finite types, with quantum control structures ($\mathbf{if}^{\circ}$)
- Compiler into quantum circuits gives the operational semantics
- Denotational semantics is given as ‘density matrices’ and ‘super operators’
- Future work:
- Define equational theory, and show this conicides with denotational sematics
- Only small programs currently; we need bigger and better examples
- . . .

Thanks

- Papers on QML can be found at:
- www.cs.nott.ac.uk/~jjg/qml

- Jonathan Grattage (jjg@cs.nott.ac.uk)