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 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*”
- 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:
  \[ t = x \mid \text{let } x = t \text{ in } u \mid x \bar{y} \]
  \[ \mid () \mid \text{let } (x, y) = t \text{ in } u \mid (t, u) \]
  \[ \mid \text{if } t \text{ then } u \text{ else } u' \]
  \[ \mid \text{if}^\circ t \text{ then } u \text{ else } u' \]
  \[ \mid \{(\kappa) \text{ qfalse} \mid (\iota) \text{ qtrue}\} \]
  \[ \kappa, \iota \in \mathbb{C} \]
Deutsch algorithm

deutsch : 2 → 2 → Q_2

deutsch a b =

let (x, y) = if° \{ qfalse | qtrue \}

then (qtrue, if a

then \{ qfalse | (−1) qtrue \}

else \{ (−1) qfalse | qtrue \}

else (qfalse, if b

then \{ (−1) qfalse | qtrue \}

else \{ qfalse | (−1) qtrue \}

in \ H \ x
• Projection Function

\[ \pi_1 \in (2,2) \rightarrow 2 \]
\[ \pi_1 (x,y) = x \]

• Diagonal Function

\[ \delta \in 2 \rightarrow (2,2) \]
\[ \delta x = (x,x) \]
Control of Decoherence

- $\pi_1 \cdot \delta : 2 \rightarrow 2$

- 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

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

- but doesn’t use it.
- Hence, it \textbf{has} to measure it!
- \textbf{if} always measures the conditional, returning only one branch
- \[ \text{forget'} : 2 \rightarrow 2 \]

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

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

- \[ \text{qnot} : 2 \rightarrow 2 \]

\[ \text{qnot} x = \text{if}^{\circ} x \text{ then } \text{qfalse} \text{ else } \text{qtrue} \]

- This program typechecks, because \( \text{qfalse} \perp \text{qtrue} \).
Compiler Design

- Takes in QML expressions
- Compiled into FQC (Finite Quantum Computation) objects
- \( \phi = \text{quantum circuit} \)
- Circuit represented as simple combinators
- Can be directly simulated, or passed to any standard simulator
- \ldots\ or a real quantum computer
Quantum Machine Code

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

$$a \xrightarrow{\psi} \phi$$

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

$$a \xrightarrow{\phi}$$

$b \xrightarrow{\psi}$

- Permutations (rewiring)

$$a \xleftrightarrow{} b$$
Quantum Machine Code

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

- Rotation (rot $u$, where $u \in \mathbb{C}$ is a unitary matrix)
Compiling the let-rule

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

- QML is a first-order functional language for quantum computations on finite types, with quantum control structures (if°)
- 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 coincides with denotational semantics
  - 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)