Compilation to Quantum Circuits for a Language with Quantum Data and Control

Yannis Rouselakis[1,2] Nikolaos S. Papaspyrou[1] Yiannis Tsiouris[1]
Eneia N. Todoran[3]

[1]National Technical University of Athens
School of Electrical and Computer Engineering
{nickie, gtsiour}@softlab.ntua.gr

[2]University of Texas at Austin
Department of Computer Science
jrous@cs.utexas.edu

[3]Technical University of Cluj-Napoca
Computer Science Department
Eneia.Todoran@cs.utcluj.ro

4th Workshop on Advances in Programming Languages
Kraków, Poland, September 8-11, 2013

Outline

Quantum programming languages (i)

- Quantum algorithms: Shor's factoring algorithm, Grover's algorithm for database search, etc.
- Unlike classical algorithms, quantum algorithms are usually studied at a low-level: quantum circuits or their direct mathematical abstractions

Quantum programming languages (i)

- Quantum algorithms: Shor's factoring algorithm, Grover's algorithm for database search, etc.
- Unlike classical algorithms, quantum algorithms are usually studied at a low-level: quantum circuits or their direct mathematical abstractions
- New high-level programming languages are needed
  - They should allow programmers to use the new power of the quantum computational model
  - They should respect the special restrictions of this model
  - but they should not expose the intricacies of the model to the programmers

Quantum programming languages (ii)

- Quantum programming is at the same point that classical programming was in the 1940s
  - Hardware is non existent or faulty
  - Semantics of QPL is understood either at a very low or a very high level of abstraction

Quantum programming languages (ii)

- Quantum programming is at the same point that classical programming was in the 1940s
  - Hardware is non existent or faulty
  - Semantics of QPL is understood either at a very low or a very high level of abstraction
- Many important problems remain, concerning the quantum computational model and its implementation, e.g.
  - quantum error correction
  - approximation of transformations in circuts using a finite set of quantum gates

Quantum programming languages (ii)

- Quantum programming is at the same point that classical programming was in the 1940s
  - Hardware is non existent or faulty
  - Semantics of QPL is understood either at a very low or a very high level of abstraction
- Many important problems remain, concerning the quantum computational model and its implementation, e.g.
  - quantum error correction
  - approximation of transformations in circuts using a finite set of quantum gates
- Similar problems about the classical programming model have been solved
- Such problems should not surface in the context of (high-level) programming languages

Quantum programming languages (iii)

A brief (and certainly incomplete) summary...

- Quantum pseudocode: Knill, 1996
- qGCL: Sanders and Zuliani, 2000
- QCL: Ömer, 2003
- λ-calculus for quantum computation: van Tonder, 2004

- "Quantum data and classical control"
  - QPL: Selinger, 2004
  - λ-calculus extending QPL: Selinger and Valiron, 2005
  - Quipper: Green *et al.*, 2013

- "Quantum data and control"
  - QML: Altenkirch and Grattage, 2005, 2011
  - QIO monad in Haskell: Altenkirch and Green, 2009
  - Arrow calculus for quantum programming: Vizzoto *et al.*, 2009

Quantum computing (i)

- The classical bit

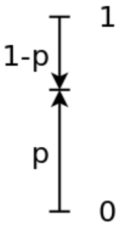
- The classical bit

Quantum computing (i)

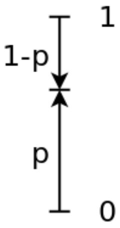
- The classical bit
- The probabilistic bit

Quantum computing (i)

- The classical bit
- The probabilistic bit

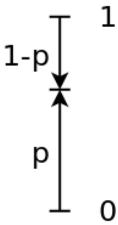


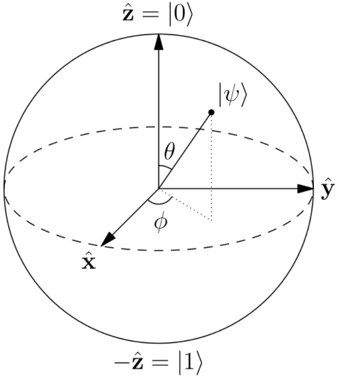
- The classical bit
- The probabilistic bit
- The quantum bit (qubit)



Quantum computing (i)

- The classical bit
- The probabilistic bit
- The quantum bit (qubit)





Quantum computing (ii)

- Superposition
  - $\alpha|0\rangle + \beta|1\rangle$ where $|\alpha|^2 + |\beta|^2 = 1$

Quantum computing (ii)

- Superposition
  - $\alpha|0\rangle + \beta|1\rangle$ where $|\alpha|^2 + |\beta|^2 = 1$
- Measurement
  - will yield $|0\rangle$ with probability $|\alpha|^2$
  - will yield $|1\rangle$ with probability $|\beta|^2$

Quantum computing (ii)

- Superposition
  - $\alpha|0\rangle + \beta|1\rangle$ where $|\alpha|^2 + |\beta|^2 = 1$
- Measurement
  - will yield $|0\rangle$ with probability $|\alpha|^2$
  - will yield $|1\rangle$ with probability $|\beta|^2$
- Quantum registers
  - $(\alpha_1|0\rangle + \beta_1|1\rangle) \otimes (\alpha_2|0\rangle + \beta_2|1\rangle) =$
    $\alpha_1\alpha_2|00\rangle + \alpha_1\beta_2|01\rangle + \beta_1\alpha_2|10\rangle + \beta_1\beta_2|11\rangle$

Quantum computing (ii)

- Superposition
  - $\alpha|0\rangle + \beta|1\rangle$ where $|\alpha|^2 + |\beta|^2 = 1$
- Measurement
  - will yield $|0\rangle$ with probability $|\alpha|^2$
  - will yield $|1\rangle$ with probability $|\beta|^2$
- Quantum registers
  - $(\alpha_1|0\rangle + \beta_1|1\rangle) \otimes (\alpha_2|0\rangle + \beta_2|1\rangle) =$
    $\alpha_1\alpha_2|00\rangle + \alpha_1\beta_2|01\rangle + \beta_1\alpha_2|10\rangle + \beta_1\beta_2|11\rangle$
- Entanglement
  - e.g. $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$

Quantum computing (iii)

- Quantum gates
  - *not*

$$|0\rangle \mapsto |1\rangle$$
$$|1\rangle \mapsto |0\rangle$$

- Quantum gates
  - *not*

$$|0\rangle \mapsto |1\rangle$$
$$|1\rangle \mapsto |0\rangle$$

$$\alpha|0\rangle + \beta|1\rangle \mapsto$$

Quantum computing (iii)

- Quantum gates
  - *not*

$$|0\rangle \mapsto |1\rangle$$
$$|1\rangle \mapsto |0\rangle$$

$$\alpha|0\rangle + \beta|1\rangle \mapsto \alpha|1\rangle + \beta|0\rangle$$

Quantum computing (iii)

- Quantum gates
  - *not*

$$|0\rangle \mapsto |1\rangle$$
$$|1\rangle \mapsto |0\rangle$$

$$\alpha|0\rangle + \beta|1\rangle \mapsto \alpha|1\rangle + \beta|0\rangle \qquad \textit{not}\ (\textit{not}\ |x\rangle) = |x\rangle$$

- Quantum gates
  - *not*

    $$|0\rangle \mapsto |1\rangle$$
    $$|1\rangle \mapsto |0\rangle$$

    $$\alpha|0\rangle + \beta|1\rangle \mapsto \alpha|1\rangle + \beta|0\rangle \qquad not\ (not\ |x\rangle) = |x\rangle$$

  - *had* — Hadamard

    $$|0\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$
    $$|1\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

- Quantum gates
  - *not*

    $|0\rangle \mapsto |1\rangle$
    $|1\rangle \mapsto |0\rangle$

    $\alpha|0\rangle + \beta|1\rangle \mapsto \alpha|1\rangle + \beta|0\rangle$ $\qquad$ $not\ (not\ |x\rangle) = |x\rangle$

  - *had* $\qquad$ Hadamard

    $|0\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$
    $|1\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ $\qquad$ $had\ (had\ |x\rangle) = |x\rangle$

Quantum computing (iii)

- Quantum gates
  - *not*

    $|0\rangle \mapsto |1\rangle$
    $|1\rangle \mapsto |0\rangle$

    $\alpha|0\rangle + \beta|1\rangle \mapsto \alpha|1\rangle + \beta|0\rangle$ $\qquad$ $not\ (not\ |x\rangle) = |x\rangle$

  - *had* $\qquad$ Hadamard

    $|0\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$
    $|1\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ $\qquad$ $had\ (had\ |x\rangle) = |x\rangle$

- Multi-qubit gates,
  - $had \otimes not$
  - *cnot* $\qquad$ $|x, y\rangle \mapsto |x, x \oplus y\rangle$

Quantum computing (iii)

- Quantum gates
  - *not*

    $|0\rangle \mapsto |1\rangle$
    $|1\rangle \mapsto |0\rangle$

    $\alpha|0\rangle + \beta|1\rangle \mapsto \alpha|1\rangle + \beta|0\rangle$ $\quad$ $not\ (not\ |x\rangle) = |x\rangle$

  - *had* $\quad$ Hadamard

    $|0\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$
    $|1\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ $\quad$ $had\ (had\ |x\rangle) = |x\rangle$

- Multi-qubit gates,
  - $had \otimes not$
  - *cnot* $\quad$ $|x, y\rangle \mapsto |x, x \oplus y\rangle$

- Reversibility!

Goal: Given a function $f : \mathbf{bit} \to \mathbf{bit}$, determine whether $f(0) = f(1)$ by calling f only once

Goal: Given a function $f : \mathbf{bit} \to \mathbf{bit}$, determine whether $f(0) = f(1)$ by calling f only once

1. Start with $i = |0\rangle$ and $j = |1\rangle$

Goal: Given a function $f : \mathbf{bit} \to \mathbf{bit}$, determine whether $f(0) = f(1)$ by calling f only once

1. Start with $i = |0\rangle$ and $j = |1\rangle$
2. Apply *had* to both: $i = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $j = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$

Goal: Given a function $f : \mathbf{bit} \rightarrow \mathbf{bit}$, determine whether $f(0) = f(1)$ by calling f only once

1. Start with $i = |0\rangle$ and $j = |1\rangle$
2. Apply had to both: $i = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $j = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$
   $|i, j\rangle = \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle)$

Goal: Given a function $f : \mathbf{bit} \rightarrow \mathbf{bit}$, determine whether $f(0) = f(1)$ by calling f only once

1. Start with $i = |0\rangle$ and $j = |1\rangle$
2. Apply had to both: $i = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $j = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$
   $|i, j\rangle = \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle)$
3. Apply $|i, j\rangle \mapsto |i, f(i) \oplus j\rangle$

Goal: Given a function $f : \mathbf{bit} \to \mathbf{bit}$, determine whether $f(0) = f(1)$ by calling f only once

1. Start with $i = |0\rangle$ and $j = |1\rangle$
2. Apply had to both: $i = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $j = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$
   $|i, j\rangle = \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle)$
3. Apply $|i, j\rangle \mapsto |i, f(i) \oplus j\rangle$
   $|i, j\rangle = \frac{1}{2}(|0, f(0) \oplus 0\rangle - |0, f(0) \oplus 1\rangle + |1, f(1) \oplus 0\rangle - |1, f(1) \oplus 1\rangle)$

Goal: Given a function $f : \mathbf{bit} \to \mathbf{bit}$, determine whether $f(0) = f(1)$ by calling f only once

1. Start with $i = |0\rangle$ and $j = |1\rangle$
2. Apply *had* to both: $i = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $j = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$
   $|i,j\rangle = \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle)$
3. Apply $|i,j\rangle \mapsto |i, f(i) \oplus j\rangle$
   $$\begin{aligned} |i,j\rangle &= \tfrac{1}{2}(|0,f(0)\oplus 0\rangle - |0,f(0)\oplus 1\rangle + |1,f(1)\oplus 0\rangle - |1,f(1)\oplus 1\rangle) \\ &= \tfrac{1}{2}(|0,f(0)\rangle - |0,\neg f(0)\rangle + |1,f(1)\rangle - |1,\neg f(1)\rangle) \end{aligned}$$

Goal: Given a function $f : \mathbf{bit} \to \mathbf{bit}$, determine whether $f(0) = f(1)$ by calling f only once

1. Start with $i = |0\rangle$ and $j = |1\rangle$
2. Apply *had* to both: $i = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $j = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$
   $|i, j\rangle = \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle)$
3. Apply $|i, j\rangle \mapsto |i, f(i) \oplus j\rangle$
   $$\begin{aligned} |i, j\rangle &= \tfrac{1}{2}(|0, f(0) \oplus 0\rangle - |0, f(0) \oplus 1\rangle + |1, f(1) \oplus 0\rangle - |1, f(1) \oplus 1\rangle) \\ &= \tfrac{1}{2}(|0, f(0)\rangle - |0, \neg f(0)\rangle + |1, f(1)\rangle - |1, \neg f(1)\rangle) \end{aligned}$$
4. Notice that if $f(0) = f(1)$ then

Goal: Given a function $f : \mathbf{bit} \to \mathbf{bit}$, determine whether $f(0) = f(1)$ by calling f only once

1. Start with $i = |0\rangle$ and $j = |1\rangle$
2. Apply *had* to both: $i = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $j = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$
   $|i, j\rangle = \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle)$
3. Apply $|i, j\rangle \mapsto |i, f(i) \oplus j\rangle$
   $$\begin{aligned} |i, j\rangle &= \tfrac{1}{2}(|0, f(0) \oplus 0\rangle - |0, f(0) \oplus 1\rangle + |1, f(1) \oplus 0\rangle - |1, f(1) \oplus 1\rangle) \\ &= \tfrac{1}{2}(|0, f(0)\rangle - |0, \neg f(0)\rangle + |1, f(1)\rangle - |1, \neg f(1)\rangle) \end{aligned}$$
4. Notice that if $f(0) = f(1)$ then
   $|i, j\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|f(0)\rangle + |\neg f(0)\rangle)$

Goal: Given a function $f : \mathbf{bit} \to \mathbf{bit}$, determine whether $f(0) = f(1)$ by calling f only once

1. Start with $i = |0\rangle$ and $j = |1\rangle$
2. Apply *had* to both: $i = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $j = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$
   $|i, j\rangle = \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle)$
3. Apply $|i, j\rangle \mapsto |i, f(i) \oplus j\rangle$
   $$\begin{aligned} |i, j\rangle &= \tfrac{1}{2}(|0, f(0) \oplus 0\rangle - |0, f(0) \oplus 1\rangle + |1, f(1) \oplus 0\rangle - |1, f(1) \oplus 1\rangle) \\ &= \tfrac{1}{2}(|0, f(0)\rangle - |0, \neg f(0)\rangle + |1, f(1)\rangle - |1, \neg f(1)\rangle) \end{aligned}$$
4. Notice that if $f(0) = f(1)$ then
   $|i, j\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|f(0)\rangle + |\neg f(0)\rangle) \quad \Rightarrow \quad i = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$

Quantum computing (iv) — Deutsch's algorithm

Goal: Given a function $f : \mathbf{bit} \to \mathbf{bit}$, determine whether $f(0) = f(1)$ by calling f only once

1. Start with $i = |0\rangle$ and $j = |1\rangle$
2. Apply *had* to both: $i = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $j = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$
   $|i, j\rangle = \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle)$
3. Apply $|i, j\rangle \mapsto |i, f(i) \oplus j\rangle$
   $$\begin{aligned} |i, j\rangle &= \tfrac{1}{2}(|0, f(0) \oplus 0\rangle - |0, f(0) \oplus 1\rangle + |1, f(1) \oplus 0\rangle - |1, f(1) \oplus 1\rangle) \\ &= \tfrac{1}{2}(|0, f(0)\rangle - |0, \neg f(0)\rangle + |1, f(1)\rangle - |1, \neg f(1)\rangle) \end{aligned}$$
4. Notice that if $f(0) = f(1)$ then
   $|i, j\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|f(0)\rangle + |\neg f(0)\rangle) \quad \Rightarrow \quad i = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$
5. Notice that if $f(0) \neq f(1)$ then

Goal: Given a function $f : \mathbf{bit} \to \mathbf{bit}$, determine whether $f(0) = f(1)$ by calling f only once

1. Start with $i = |0\rangle$ and $j = |1\rangle$
2. Apply *had* to both: $i = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $j = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$
   $|i, j\rangle = \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle)$
3. Apply $|i, j\rangle \mapsto |i, f(i) \oplus j\rangle$
   $$|i, j\rangle = \frac{1}{2}(|0, f(0) \oplus 0\rangle - |0, f(0) \oplus 1\rangle + |1, f(1) \oplus 0\rangle - |1, f(1) \oplus 1\rangle)$$
   $$= \frac{1}{2}(|0, f(0)\rangle - |0, \neg f(0)\rangle + |1, f(1)\rangle - |1, \neg f(1)\rangle)$$
4. Notice that if $f(0) = f(1)$ then
   $|i, j\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|f(0)\rangle + |\neg f(0)\rangle) \quad \Rightarrow \quad i = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$
5. Notice that if $f(0) \neq f(1)$ then
   $|i, j\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \otimes \frac{1}{\sqrt{2}}(|f(0)\rangle - |\neg f(0)\rangle)$

Goal: Given a function $f : \mathbf{bit} \to \mathbf{bit}$, determine whether $f(0) = f(1)$ by calling f only once

1. Start with $i = |0\rangle$ and $j = |1\rangle$
2. Apply *had* to both: $i = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $j = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$
   $|i, j\rangle = \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle)$
3. Apply $|i, j\rangle \mapsto |i, f(i) \oplus j\rangle$
   $$|i, j\rangle = \frac{1}{2}(|0, f(0) \oplus 0\rangle - |0, f(0) \oplus 1\rangle + |1, f(1) \oplus 0\rangle - |1, f(1) \oplus 1\rangle)$$
   $$= \frac{1}{2}(|0, f(0)\rangle - |0, \neg f(0)\rangle + |1, f(1)\rangle - |1, \neg f(1)\rangle)$$
4. Notice that if $f(0) = f(1)$ then
   $$|i, j\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|f(0)\rangle + |\neg f(0)\rangle) \quad \Rightarrow \quad i = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$
5. Notice that if $f(0) \neq f(1)$ then
   $$|i, j\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \otimes \frac{1}{\sqrt{2}}(|f(0)\rangle - |\neg f(0)\rangle) \quad \Rightarrow \quad i = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

Goal: Given a function $f : \mathbf{bit} \to \mathbf{bit}$, determine whether $f(0) = f(1)$ by calling f only once

1. Start with $i = |0\rangle$ and $j = |1\rangle$
2. Apply *had* to both: $i = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $j = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$
   $|i, j\rangle = \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle)$
3. Apply $|i, j\rangle \mapsto |i, f(i) \oplus j\rangle$
   $$\begin{aligned} |i, j\rangle &= \tfrac{1}{2}(|0, f(0) \oplus 0\rangle - |0, f(0) \oplus 1\rangle + |1, f(1) \oplus 0\rangle - |1, f(1) \oplus 1\rangle) \\ &= \tfrac{1}{2}(|0, f(0)\rangle - |0, \neg f(0)\rangle + |1, f(1)\rangle - |1, \neg f(1)\rangle) \end{aligned}$$
4. Notice that if $f(0) = f(1)$ then
   $|i, j\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|f(0)\rangle + |\neg f(0)\rangle) \quad \Rightarrow \quad i = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$
5. Notice that if $f(0) \neq f(1)$ then
   $|i, j\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \otimes \frac{1}{\sqrt{2}}(|f(0)\rangle - |\neg f(0)\rangle) \quad \Rightarrow \quad i = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$
6. Apply *had* to i:

Quantum computing (iv) — Deutsch's algorithm

Goal: Given a function $f : \mathbf{bit} \to \mathbf{bit}$, determine whether $f(0) = f(1)$ by calling f only once

1. Start with $i = |0\rangle$ and $j = |1\rangle$
2. Apply *had* to both: $i = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $j = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$
   $|i,j\rangle = \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle)$
3. Apply $|i,j\rangle \mapsto |i, f(i) \oplus j\rangle$
   $$|i,j\rangle = \tfrac{1}{2}(|0,f(0)\oplus 0\rangle - |0,f(0)\oplus 1\rangle + |1,f(1)\oplus 0\rangle - |1,f(1)\oplus 1\rangle)$$
   $$= \tfrac{1}{2}(|0,f(0)\rangle - |0,\neg f(0)\rangle + |1,f(1)\rangle - |1,\neg f(1)\rangle)$$
4. Notice that if $f(0) = f(1)$ then
   $|i,j\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|f(0)\rangle + |\neg f(0)\rangle) \Rightarrow i = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$
5. Notice that if $f(0) \neq f(1)$ then
   $|i,j\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \otimes \frac{1}{\sqrt{2}}(|f(0)\rangle - |\neg f(0)\rangle) \Rightarrow i = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$
6. Apply *had* to i: it yields $|0\rangle$ iff $f(0) = f(1)$!

nQML — Yet another QPL? (i)

- nQML: a quantum programming language with "quantum data and control" (Lampis *et al.*, 2006, 2008)
- Based on QML (Altenkirch and Grattage, 2005)
- Its design goals:
  - to give programmers sufficient expressive power to implement quantum algorithms easily
  - while preventing them from breaking the rules of quantum computation

nQML — Yet another QPL? (ii)

- Simple type system and denotational semantics
  - Both use structures and techniques typical in the study of classical programming languages
  - The type system does not use linear types
  - The denotational semantics uses density matrices to describe quantum states

⇒ Lampis *et al.*, 2006, 2008

- Simple type system and denotational semantics
  - Both use structures and techniques typical in the study of classical programming languages
  - The type system does not use linear types
  - The denotational semantics uses density matrices to describe quantum states

  $\Rightarrow$ Lampis *et al.*, 2006, 2008

- Compilation to quantum circuits

  $\Rightarrow$ This work

nQML — Yet another QPL? (ii)

- Simple type system and denotational semantics
  - Both use structures and techniques typical in the study of classical programming languages
  - The type system does not use linear types
  - The denotational semantics uses density matrices to describe quantum states

  ⇒ Lampis *et al.*, 2006, 2008

- Compilation to quantum circuits
  ⇒ This work

- Straightforward implementation in Haskell
  http://www.softlab.ntua.gr/~nickie/Research/nqml/

Overview of nQML (i)

- Variables: $x, y, \ldots$
  - References to quantum information that is stored in a global quantum state

Overview of nQML (i)

- Variables: $x, y, \ldots$
  - References to quantum information that is stored in a global quantum state
- Two types of information: qubits and products

Overview of nQML (i)

- Variables: $x, y, \ldots$
  - References to quantum information that is stored in a global quantum state
- Two types of information: qubits and products
- Quantum superposition: $\{\,(\lambda)\,\mathbf{qfalse} + (\lambda')\,\mathbf{qtrue}\,\}$
  - It allocates a new qubit in the same way that new objects are allocated on the heap when a constructor is used in a functional language

Overview of nQML (i)

- Variables: $x, y, \dots$
  - References to quantum information that is stored in a global quantum state
- Two types of information: qubits and products
- Quantum superposition: $\{(\lambda)\,\mathbf{qfalse} + (\lambda')\,\mathbf{qtrue}\}$
  - It allocates a new qubit in the same way that new objects are allocated on the heap when a constructor is used in a functional language
- Binding construct: $\mathbf{let}\ x = e_1\ \mathbf{in}\ e_2$
- Products: (e_1, e_2) $\mathbf{let}\ (x_1, x_2) = e_1\ \mathbf{in}\ e_2$

Overview of nQML (ii)

Three control constructs:

- Quantum measurement: **ifm** e **then** e_1 **else** e_2
- Quantum branching: **if** e **then** e_1 **else** e_2

Three control constructs:

- Quantum measurement: $\textbf{ifm } e \textbf{ then } e_1 \textbf{ else } e_2$
- Quantum branching: $\textbf{if } e \textbf{ then } e_1 \textbf{ else } e_2$
- Quantum unitary transformation: $|e\rangle \to x, y.\, c$

$$|x\rangle \to \sum_{y=0}^{2^n-1} f(x,y)\,|y\rangle$$

Three control constructs:

- Quantum measurement: $\textbf{ifm}\ e\ \textbf{then}\ e_1\ \textbf{else}\ e_2$
- Quantum branching: $\textbf{if}\ e\ \textbf{then}\ e_1\ \textbf{else}\ e_2$
- Quantum unitary transformation: $|e\rangle \rightarrow x, y.\, c$

$$|x\rangle \rightarrow \sum_{y=0}^{2^n-1} f(x,y)\,|y\rangle$$

e.g.

$$\begin{aligned} not(q) &\equiv |q\rangle \rightarrow x,y.\ \textbf{if}\ y = x\ \textbf{then}\ 0\ \textbf{else}\ 1 \\ had(q) &\equiv |q\rangle \rightarrow x,y.\ \textbf{if}\ x\ \textbf{then}\ (\textbf{if}\ y\ \textbf{then}\ -\tfrac{1}{\sqrt{2}}\ \textbf{else}\ \tfrac{1}{\sqrt{2}})\ \textbf{else}\ \tfrac{1}{\sqrt{2}} \end{aligned}$$

Three control constructs:

- Quantum measurement: $\textbf{ifm } e \textbf{ then } e_1 \textbf{ else } e_2$
- Quantum branching: $\textbf{if } e \textbf{ then } e_1 \textbf{ else } e_2$
- Quantum unitary transformation: $|e\rangle \rightarrow x, y. c$

$$|x\rangle \rightarrow \sum_{y=0}^{2^n-1} f(x,y)\,|y\rangle$$

e.g.

$$\begin{aligned}
not(q) &\equiv |q\rangle \rightarrow x,y.\ \textbf{if } y = x \textbf{ then } 0 \textbf{ else } 1 \\
had(q) &\equiv |q\rangle \rightarrow x,y.\ \textbf{if } x \textbf{ then } (\textbf{if } y \textbf{ then } -\tfrac{1}{\sqrt{2}} \textbf{ else } \tfrac{1}{\sqrt{2}}) \textbf{ else } \tfrac{1}{\sqrt{2}} \\
add(r,c) &\equiv |r\rangle \rightarrow x,y.\ \textbf{if int } y = \textbf{int } x + c \textbf{ then } 1 \textbf{ else } 0
\end{aligned}$$

Type system of nQML

- A type τ keeps track of the exact qubits of the state in which the value of an expression is stored

$$\tau \ ::= \ \mathbf{qbit}[n] \ | \ \tau_1 \otimes \tau_2$$

e.g. an expression has type $\mathbf{qbit}[5]$ if its value is stored in the 5th qubit of the state.

Type system of nQML

- A type τ keeps track of the exact qubits of the state in which the value of an expression is stored

  $$\tau ::= \mathbf{qbit}[n] \mid \tau_1 \otimes \tau_2$$

  e.g. an expression has type $\mathbf{qbit}[5]$ if its value is stored in the 5th qubit of the state.

- Typing relation
  - For pure quantum expressions $\Gamma; n \vdash^{\circ} e : \tau; m$
  - For impure quantum expressions $\Gamma; n \vdash e : \tau; m$

Type system of nQML

- A type τ keeps track of the exact qubits of the state in which the value of an expression is stored

  $$\tau \ ::= \ \mathbf{qbit}[n] \ | \ \tau_1 \otimes \tau_2$$

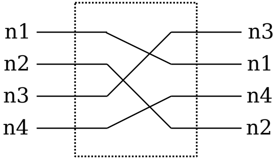
  e.g. an expression has type $\mathbf{qbit}[5]$ if its value is stored in the 5th qubit of the state.

- Typing relation
  - For pure quantum expressions $\Gamma; n \vdash^{\circ} e : \tau; m$
  - For impure quantum expressions $\Gamma; n \vdash e : \tau; m$
- n is the number of qubits of the original quantum state, before e starts evaluating
- m is the number of new qubits that are allocated during the evaluation of e

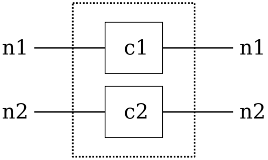
- Rotation: introduces unitary transformation where $\lambda_0^* \kappa_0 + \lambda_1 \kappa_1^* = 0$

$$\begin{pmatrix} \lambda_0 & \lambda_1 \\ \kappa_0 & \kappa_1 \end{pmatrix}$$

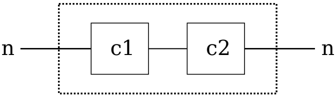
- Wire reordering



- Parallel composition

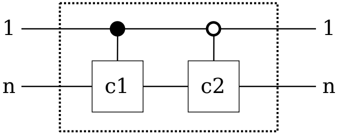


- Sequential composition

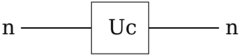


Quantum circuits (iii)

- Conditional



- Arbitrary unitary matrix



Quantum circuits (iv)

Three categories of circuits: $\mathsf{FQC}^{\approx} \subset \mathsf{FQC}^{\circ} \subset \mathsf{FQC}$
(Altenkirch and Grattage)

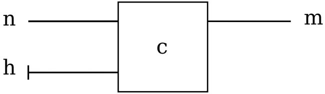
Three categories of circuits: $\mathsf{FQC}^{\approx} \subset \mathsf{FQC}^{\circ} \subset \mathsf{FQC}$
(Altenkirch and Grattage)

- $\mathsf{FQC}^{\approx}$: reversible finite quantum circuits of n qubits

Quantum circuits (iv)

Three categories of circuits: $\mathsf{FQC}^{\approx} \subset \mathsf{FQC}^{\circ} \subset \mathsf{FQC}$
(Altenkirch and Grattage)

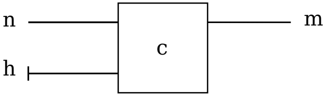
- $\mathsf{FQC}^{\approx}$: reversible finite quantum circuits of n qubits
- FQC° : circuits with a heap $\quad n + h = m$



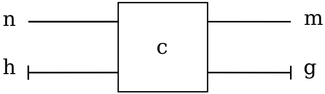
Quantum circuits (iv)

Three categories of circuits: $\text{FQC}^{\approx} \subset \text{FQC}^{\circ} \subset \text{FQC}$
(Altenkirch and Grattage)

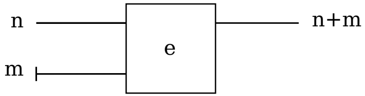
- $\text{FQC}^{\approx}$: reversible finite quantum circuits of n qubits
- FQC° : circuits with a heap $n+h=m$



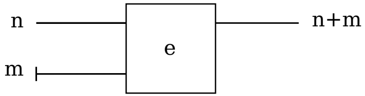
- FQC : circuits with a heap and garbage $n+h=m+g$



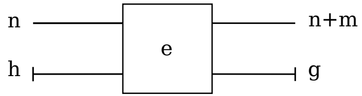
- Pure expressions compile to FQC$^\circ$ $\Gamma; n \vdash^\circ e : \tau; m$



- Pure expressions compile to FQC$^\circ$ $\Gamma; n \vdash^\circ e : \tau; m$

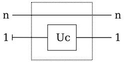


- Impure expressions compile to FQC $\Gamma; n \vdash e : \tau; m,$

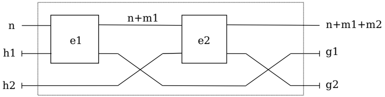


where $h = m + g$

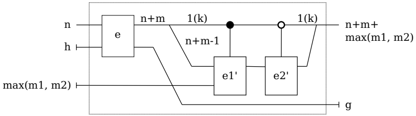
Superposition:
$\Gamma;n \vdash^{\circ} \{ (\lambda)\,\textbf{qfalse} + (\lambda')\,\textbf{qtrue} \} : \textbf{qbit}[n];1$



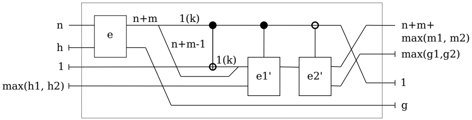
Let and products:
$\Gamma;n \vdash^{\alpha}$ **let** $x = e_1$ **in** $e_2 : \tau;m_1+m_2$
$\Gamma;n \vdash^{\alpha} (e_1,e_2) : \tau;m_1+m_2$
$\Gamma;n \vdash^{\alpha}$ **let** $(x_1,x_2) = e_1$ **in** $e_2 : \tau;m_1+m_2$
where:
$\Gamma_1;n \vdash^{\alpha} e_1 : \tau_1;m_1$
$\Gamma_2;n+m_1 \vdash^{\alpha} e_2 : \tau_2;m_2$



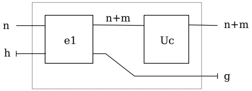
Quantum conditional:
$\Gamma;n \vdash^{\alpha}$ **if** e **then** e_1 **else** $e_2 : \tau;m+\max(m_1,m_2)$
where:
$\Gamma;n \vdash^{\alpha} e : \textbf{qbit}[k];m$
$\Gamma|_k;n+m \vdash^{\circ} e_1 : \tau;m_1$
$\Gamma|_k;n+m \vdash^{\circ} e_2 : \tau;m_2$



Measurement:
$\Gamma;n \vdash$ **ifm** e **then** e_1 **else** $e_2 : \tau;m+\max(m_1,m_2)$
where:
$\Gamma;n \vdash e : \textbf{qbit}[k];m$
$\Gamma;n+m \vdash e_1 : \tau;m_1$
$\Gamma;n+m \vdash e_2 : \tau;m_2$



Unitary transformation:
$\Gamma;n \vdash^{\alpha} |e\rangle \rightarrow x,x'.c : \tau;m$
where:
$\Gamma;n \vdash^{\alpha} e : \tau;m$
$c(x,x')$ defines a unitary transformation on $n+m$ qubits



- Preliminaries

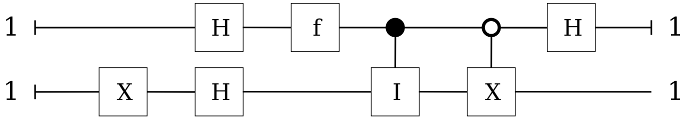
```
def not q = |q> -> x, x'.
  if x' = x then 0 else 1;

def had q = |q> -> x, x'.
  (if x then (if x' then -1 else 1) else 1)
    / sqrt(2);
```

- Deutsch's algorithm

```
def Deutsch f =
  let (i, j) = (had qfalse, had qtrue) in
  let r = if f i then j else not j in
  ifm had i then qtrue else qfalse;
```

- Deutsch's algorithm

```
def Deutsch f =
  let (i, j) = (had qfalse, had qtrue) in
  let r = if f i then j else not j in
  ifm had i then qtrue else qfalse;
```

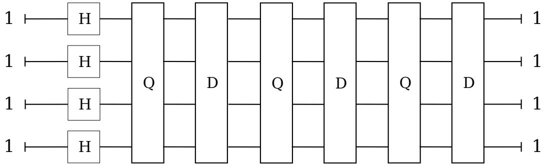


- Grover's algorithm

```
def query q = |q> -> x, x'.
  if x = x' then
    if int x = correct then -1 else 1
  else
    0;

def diffusion q = |q> -> x, x'.
  if x = x' then 2 / 2^n - 1 else 2 / 2^n;

def grover4 =
  let qs = (had qfalse, had qfalse,
            had qfalse, had qfalse) in
  let step1 = diffusion (query qs) in
  let step2 = diffusion (query qs) in
  let step3 = diffusion (query qs) in
  qs
```

Examples (iv)

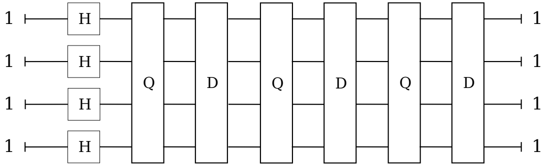
- Grover's algorithm




```
$ ntua-qml-circ < examples/grover4
...
STATE VECTOR
 False False False False: (-5.078124999999997e-2) :+ 0.0
 False False False True : (-5.078124999999997e-2) :+ 0.0
 False False True  False: (-5.078124999999997e-2) :+ 0.0
 False False True True :  0.9804687499999996 :+ 0.0
 False True  False False: (-5.078125e-2) :+ 0.0
 False True  False True : (-5.078125e-2) :+ 0.0
 False True  True  False: (-5.078125e-2) :+ 0.0
 False True  True  True : (-5.078125e-2) :+ 0.0
 True  False False False: (-5.078124999999999e-2) :+ 0.0
 True  False False True : (-5.078124999999999e-2) :+ 0.0
 True  False True  False: (-5.078124999999999e-2) :+ 0.0
 True  False True  True : (-5.078124999999999e-2) :+ 0.0
 True  True  False False: (-5.078124999999999e-2) :+ 0.0
 True  True  False True : (-5.078124999999999e-2) :+ 0.0
 True  True  True  False: (-5.078124999999999e-2) :+ 0.0
 True  True  True  True : (-5.078124999999999e-2) :+ 0.0
...
```

Examples (iv)

- Grover's algorithm




```
$ ntua-qml-circ < examples/grover4
...
STATE VECTOR
 False False False False: (-5.078124999999997e-2) :+ 0.0
 False False False True : (-5.078124999999997e-2) :+ 0.0
 False False True  False: (-5.078124999999997e-2) :+ 0.0
 False False True  True :  0.9804687499999996 :+ 0.0
 False True  False False: (-5.078125e-2) :+ 0.0
 False True  False True : (-5.078125e-2) :+ 0.0
 False True  True  False: (-5.078125e-2) :+ 0.0
 False True  True  True : (-5.078125e-2) :+ 0.0
 True  False False False: (-5.078124999999999e-2) :+ 0.0
 True  False False True : (-5.078124999999999e-2) :+ 0.0
 True  False True  False: (-5.078124999999999e-2) :+ 0.0
 True  False True  True : (-5.078124999999999e-2) :+ 0.0
 True  True  False False: (-5.078124999999999e-2) :+ 0.0
 True  True  False True : (-5.078124999999999e-2) :+ 0.0
 True  True  True  False: (-5.078124999999999e-2) :+ 0.0
 True  True  True  True : (-5.078124999999999e-2) :+ 0.0
...
```

Conclusion

- nQML: a language following "quantum data and control"
- Non linear type system; types carry qubit information
- Denotational semantics based on density matrices
- Compilation into quantum circuits in the category FQC
- Complete implementation (type-checker, interpreter and compiler) in Haskell

Conclusion

- nQML: a language following "quantum data and control"
- Non linear type system; types carry qubit information
- Denotational semantics based on density matrices
- Compilation into quantum circuits in the category FQC
- Complete implementation (type-checker, interpreter and compiler) in Haskell

Challenges for the future:

- Integration of high-level programming features
- Make quantum programming as easy as classical programming