Shor in Haskell
The Quantum IO Monad

Trends in Functional Programming
May 28th 2008

Alexander S. Green and Thorsten Altenkirch

asg@cs.nott.ac.uk, txa@cs.nott.ac.uk

School of Computer Science,

The University of Nottingham

Introduction

- Quantum Computing is an exciting new area in computer science.

Introduction

- Quantum Computing is an exciting new area in computer science.
- certain Quantum Algorithms can offer an exponential speed up over the best known classically.

Introduction

- Quantum Computing is an exciting new area in computer science.
- certain Quantum Algorithms can offer an exponential speed up over the best known classically.
- Shor's algorithm can factor large numbers in polynomial time O$((logN)^3)$.

Introduction

- Quantum Computing is an exciting new area in computer science.
- certain Quantum Algorithms can offer an exponential speed up over the best known classically.
- Shor's algorithm can factor large numbers in polynomial time $O((log N)^3)$.
- Classically, the best known solution is $O(2^{(log N)^{\frac{1}{3}}})$ which for large numbers is computationally infeasible.

Introduction

- Quantum Computing is an exciting new area in computer science.
- certain Quantum Algorithms can offer an exponential speed up over the best known classically.
- Shor's algorithm can factor large numbers in polynomial time $O((logN)^3)$.
- Classically, the best known solution is $O(2^{(logN)^{\frac{1}{3}}})$ which for large numbers is computationally infeasible.
- The RSA encryption protocol uses this assumption, and hence could be "broken" by a quantum computer.

Introduction

- Other quantum algorithms can offer a speed up over the provably fastest classical solutions.

Introduction

- Other quantum algorithms can offer a speed up over the provably fastest classical solutions.
- Grover's algorithm offers a polynomial speed-up for searching an unsorted database

Introduction

- Other quantum algorithms can offer a speed up over the provably fastest classical solutions.
- Grover's algorithm offers a polynomial speed-up for searching an unsorted database
- Deutsch's algorithm can find out if a boolean function is constant or balanced with only one application of the function.

Introduction

- Other quantum algorithms can offer a speed up over the provably fastest classical solutions.
- Grover's algorithm offers a polynomial speed-up for searching an unsorted database
- Deutsch's algorithm can find out if a boolean function is constant or balanced with only one application of the function.
- Quantum teleportation enables the use of quantum key distribution, allowing provably secure communication.

Introduction

- Other quantum algorithms can offer a speed up over the provably fastest classical solutions.
- Grover's algorithm offers a polynomial speed-up for searching an unsorted database
- Deutsch's algorithm can find out if a boolean function is constant or balanced with only one application of the function.
- Quantum teleportation enables the use of quantum key distribution, allowing provably secure communication.
- There are already commercial companies offering quantum crytography products (BB84)...

Introduction

- However, the state of the art for quantum computer hardware is still only a few qubits.

Introduction

- However, the state of the art for quantum computer hardware is still only a few qubits.
- We would like to look at quantum computing from a Functional Programming point of view.

Introduction

- However, the state of the art for quantum computer hardware is still only a few qubits.
- We would like to look at quantum computing from a Functional Programming point of view.
- We introduce the Quantum IO Monad (QIO), as an interface from Haskell to Quantum Computation

Introduction

- However, the state of the art for quantum computer hardware is still only a few qubits.
- We would like to look at quantum computing from a Functional Programming point of view.
- We introuduce the Quantum IO Monad (QIO), as an interface from Haskell to Quantum Computation
- The Monadic structure is used to deal with the side-effects.

Introduction

The University of Nottingham

- However, the state of the art for quantum computer hardware is still only a few qubits.
- We would like to look at quantum computing from a Functional Programming point of view.
- We introduce the Quantum IO Monad (QIO), as an interface from Haskell to Quantum Computation
- The Monadic structure is used to deal with the side-effects.
- While the design of quantum algorithms can make use of the abstractions available in Haskell.

Introduction

- However, the state of the art for quantum computer hardware is still only a few qubits.
- We would like to look at quantum computing from a Functional Programming point of view.
- We introduce the Quantum IO Monad (QIO), as an interface from Haskell to Quantum Computation
- The Monadic structure is used to deal with the side-effects.
- While the design of quantum algorithms can make use of the abstractions available in Haskell.
- I shall now give a brief introduction to both quantum computing and the Quantum IO Monad .

Qubits

- Qubits have 2 base states ($|0\rangle$ and $|1\rangle$)...

Qubits

- Qubits have 2 base states ($|0\rangle$ and $|1\rangle$)...
- In QIO we define the type

$$Qbit :: *$$

Qubits

- Qubits have 2 base states ($|0\rangle$ and $|1\rangle$)...
- In QIO we define the type

  $Qbit :: *$
- along with the initialisation function

  $mkQbit :: Bool \rightarrow QIO\ Qbit$

Qubits

- Qubits have 2 base states ($|0\rangle$ and $|1\rangle$)...
- In QIO we define the type

  $$Qbit :: *$$

- along with the initialisation function

  $$mkQbit :: Bool \rightarrow QIO\ Qbit$$

- $$|0\rangle\,, |1\rangle :: QIO\ Qbit$$
  $$|0\rangle = mkQbit\ False$$
  $$|1\rangle = mkQbit\ True$$

Qubits

- Qubits have 2 base states ($|0\rangle$ and $|1\rangle$)...
- In QIO we define the type

  $$Qbit :: *$$

- along with the initialisation function

  $$mkQbit :: Bool \rightarrow QIO\ Qbit$$

- $$|0\rangle\ , |1\rangle :: QIO\ Qbit$$
  $$|0\rangle = mkQbit\ False$$
  $$|1\rangle = mkQbit\ True$$

- Qubits can exist in a super-position of both states simultaneously.

Qubits

- An arbitrary state of a single qubit system can be given by $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$

Qubits

- An arbitrary state of a single qubit system can be given by $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$
- where $\alpha, \beta \in \mathbf{C}$ are the complex amplitudes of each base state.

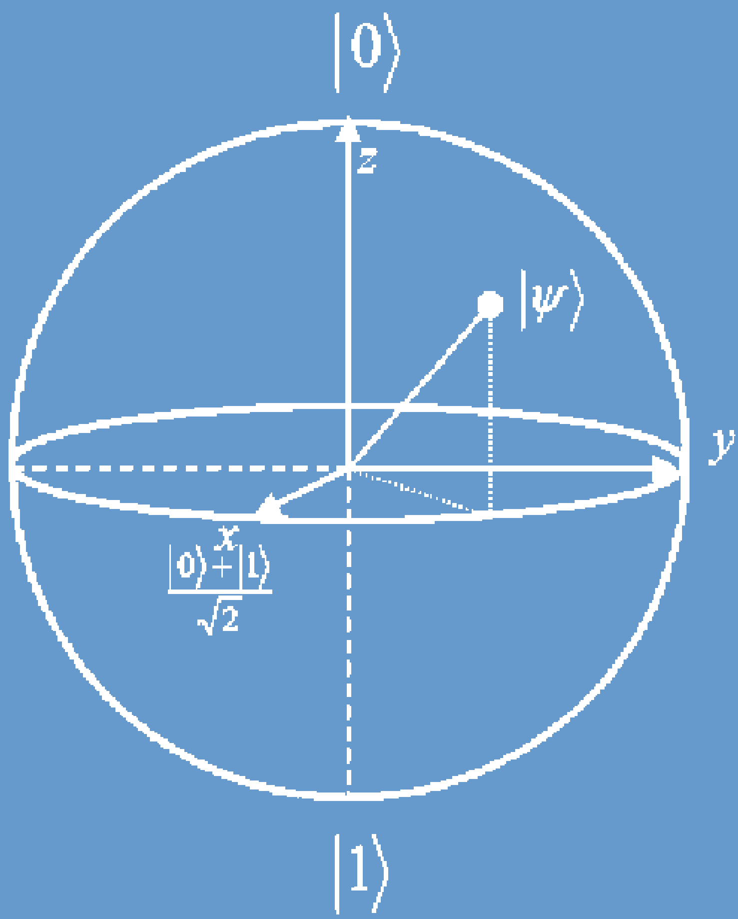
Qubits

- An arbitrary state of a single qubit system can be given by $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$
- where $\alpha, \beta \in \mathbf{C}$ are the complex amplitudes of each base state.
- and with the side condition that $|\alpha|^2 + |\beta|^2 = 1$.

Qubits

- An arbitrary state of a single qubit system can be given by $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$
- where $\alpha, \beta \in \mathbf{C}$ are the complex amplitudes of each base state.
- and with the side condition that $|\alpha|^2 + |\beta|^2 = 1$.
- The Bloch sphere can be used to visualise this...

Bloch Sphere



An arbitrary (single qubit) state can be thought of as any point on the surface of the sphere.

Qubit Rotations

- Computations that act on qubits are often referred to as Unitary Operators .

Qubit Rotations

- Computations that act on qubits are often referred to as Unitary Operators .
- This follows from the fact that they must keep the sum of the squares of the amplitudes equal to 1.

Qubit Rotations

- Computations that act on qubits are often referred to as Unitary Operators .
- This follows from the fact that they must keep the sum of the squares of the amplitudes equal to 1.
- We like to refer to single qubit unitary operators as Rotations (think of them as rotating a point around the surface of the Bloch sphere).

Qubit Rotations

- Computations that act on qubits are often referred to as Unitary Operators .
- This follows from the fact that they must keep the sum of the squares of the amplitudes equal to 1.
- We like to refer to single qubit unitary operators as Rotations (think of them as rotating a point around the surface of the Bloch sphere).
- In QIO, unitary operators occupy the type

$$U :: *$$

Qubit Rotations

- Computations that act on qubits are often referred to as Unitary Operators .
- This follows from the fact that they must keep the sum of the squares of the amplitudes equal to 1.
- We like to refer to single qubit unitary operators as Rotations (think of them as rotating a point around the surface of the Bloch sphere).
- In QIO, unitary operators occupy the type

$$U :: *$$

- Rotations are used in QIO to create the single qubit super-positions.

Qubit Rotations

- Rotations are defined by unitary 2 by 2 complex valued matrices, e.g.

$$unot = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \ uhad = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

and $uphase\ \phi = \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i \phi} \end{bmatrix}$.

Qubit Rotations

- Rotations are defined by unitary 2 by 2 complex valued matrices, e.g.

  $$unot = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \; uhad = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

  and $uphase\ \phi = \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i \phi} \end{bmatrix}$.

- using the type

  $$\textbf{type}\ Rotation = ((Bool, Bool) \rightarrow \mathbb{C})$$

Qubit Rotations

- Rotations are defined by unitary 2 by 2 complex valued matrices, e.g.

$$unot = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \; uhad = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

and $uphase\ \phi = \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i \phi} \end{bmatrix}$.

- using the type

$$\mathbf{type}\ Rotation = ((Bool, Bool) \rightarrow \mathbb{C})$$

- which is extended to a member of the U type by

$$rot :: Qbit \rightarrow Rotation \rightarrow U$$

Qubit Rotations

- In QIO, a unitary operator can be applied to the current state using

$$applyU :: U \rightarrow QIO\ ()$$

Qubit Rotations

- In QIO, a unitary operator can be applied to the current state using
$$applyU :: U \rightarrow QIO\ ()$$
- So we could now create the state
$|+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$
which is an equal super-position of the single qubit base states.

Qubit Rotations

- In QIO, a unitary operator can be applied to the current state using
  $$applyU :: U \rightarrow QIO\ ()$$
- So we could now create the state
  $|+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$
  which is an equal super-position of the single qubit base states.
- $$\begin{aligned} &|+\rangle :: QIO\ Qbit \\ &|+\rangle = \mathbf{do}\ q \leftarrow |0\rangle \\ &\qquad\qquad applyU\ (uhad\ q) \\ &\qquad\qquad return\ q \end{aligned}$$

Measurement

- Quantum Mechanics tells us that we can't tell which arbitrary super-position a quantum system is in .

Measurement

- Quantum Mechanics tells us that we can't tell which arbitrary super-position a quantum system is in .
- Upon measurement, the system will collapse into one of the base states.

Measurement

- Quantum Mechanics tells us that we can't tell which arbitrary super-position a quantum system is in .
- Upon measurement, the system will collapse into one of the base states.
- The probability of each base state being measured is equal to the square of the amplitude of that base state within the super-position.

Measurement

- Quantum Mechanics tells us that we can't tell which arbitrary super-position a quantum system is in .
- Upon measurement, the system will collapse into one of the base states.
- The probability of each base state being measured is equal to the square of the amplitude of that base state within the super-position.
- This leads to the fact that measurements can cause side-effects in the rest of the system.

Measurement

- Quantum Mechanics tells us that we can't tell which arbitrary super-position a quantum system is in .
- Upon measurement, the system will collapse into one of the base states.
- The probability of each base state being measured is equal to the square of the amplitude of that base state within the super-position.
- This leads to the fact that measurements can cause side-effects in the rest of the system.

Measurement

- Measurement in QIO is by use of

$$measQbit :: Qbit \rightarrow QIO\ Bool$$

Measurement

- Measurement in QIO is by use of

$$measQbit :: Qbit \rightarrow QIO\ Bool$$

- We could now use the $|+\rangle$ state to create a random boolean.

Measurement

- Measurement in QIO is by use of

$$measQbit :: Qbit \rightarrow QIO\ Bool$$

- We could now use the $|+\rangle$ state to create a random boolean.

- $$\begin{aligned} &randomBool :: QIO\ Bool \\ &randomBool = \textbf{do}\ q \leftarrow |+\rangle \\ &\qquad\qquad\qquad\qquad c \leftarrow measQbit \\ &\qquad\qquad\qquad\qquad return\ c \end{aligned}$$

Multiple Qubits...

- A 2-qubit system can be in a super-postion of the base states $|00\rangle$, $|01\rangle$, $|10\rangle$, and $|11\rangle$.

Multiple Qubits...

- A 2-qubit system can be in a super-postion of the base states $|00\rangle$, $|01\rangle$, $|10\rangle$, and $|11\rangle$.
- For an n-qubit system, the state can occupy a super-position of any of the 2^n bit strings of length n.

Multiple Qubits...

- A 2-qubit system can be in a super-postion of the base states $|00\rangle$, $|01\rangle$, $|10\rangle$, and $|11\rangle$.
- For an n-qubit system, the state can occupy a super-position of any of the 2^n bit strings of length n.
- We still have the side-condition that the sum of the squared amplitudes must equal 1 .

Multiple Qubits...

- In QIO we can create multiple qubit states by initialising each qubit and then applying some of our unitary operators to them.

Multiple Qubits...

- In QIO we can create multiple qubit states by initialising each qubit and then applying some of our unitary operators to them.
- Our U data-type consists of two; two-qubit unitary operators, which (along with rotations) form a Monoid that is complete for quantum computation.

Multiple Qubits...

- In QIO we can create multiple qubit states by initialising each qubit and then applying some of our unitary operators to them.
- Our U data-type consists of two; two-qubit unitary operators, which (along with rotations) form a Monoid that is complete for quantum computation.
- We use $\bullet$ as the identity operator, and $\blacktriangleright$ for the append operation.

Multiple Qubits...

- In QIO we can create multiple qubit states by initialising each qubit and then applying some of our unitary operators to them.
- Our U data-type consists of two; two-qubit unitary operators, which (along with rotations) form a Monoid that is complete for quantum computation.
- We use • as the identity operator, and ▶ for the append operation.
- It is possible to swap the position of 2 qubits

$$swap :: Qbit \rightarrow Qbit \rightarrow U$$

Multiple Qubits...

- In QIO we can create multiple qubit states by initialising each qubit and then applying some of our unitary operators to them.
- Our U data-type consists of two; two-qubit unitary operators, which (along with rotations) form a Monoid that is complete for quantum computation.
- We use $\bullet$ as the identity operator, and $\blacktriangleright$ for the append operation.
- It is possible to swap the position of 2 qubits

$$swap :: Qbit \rightarrow Qbit \rightarrow U$$

- and create conditional operations

$$cond :: Qbit \rightarrow (Bool \rightarrow U) \rightarrow U$$

Multiple Qubits...

- An example conditional statement would be

$$ifQ :: Qbit \rightarrow U \rightarrow U$$
$$ifQ\ q\ u = cond\ q\ (\lambda x \rightarrow \textbf{if}\ x\ \textbf{then}\ u\ \textbf{else}\ \bullet)$$

Multiple Qubits...

- An example conditional statement would be

$$ifQ :: Qbit \rightarrow U \rightarrow U$$

$$ifQ\ q\ u = cond\ q\ (\lambda x \rightarrow \textbf{if}\ x\ \textbf{then}\ u\ \textbf{else}\ \bullet)$$

- The Unitary aspect of U defines exactly that they are reversible, and we provide the function $urev :: U \rightarrow U$ which constructs the inverse.

Multiple Qubits...

- An example conditional statement would be

$$ifQ :: Qbit \rightarrow U \rightarrow U$$
$$ifQ\ q\ u = cond\ q\ (\lambda x \rightarrow \textbf{if}\ x\ \textbf{then}\ u\ \textbf{else}\ \bullet)$$

- The Unitary aspect of U defines exactly that they are reversible, and we provide the function $urev :: U \rightarrow U$ which constructs the inverse.
- The No Cloning theorem tells us that we cannot create a copy of an arbitrary quantum state.

Multiple Qubits...

- An example conditional statement would be
  $$ifQ :: Qbit \to U \to U$$
  $$ifQ\ q\ u = cond\ q\ (\lambda x \to \mathbf{if}\ x\ \mathbf{then}\ u\ \mathbf{else}\ \bullet)$$
- The Unitary aspect of U defines exactly that they are reversible, and we provide the function $urev :: U \to U$ which constructs the inverse.
- The No Cloning theorem tells us that we cannot create a copy of an arbitrary quantum state.
- We can however "share" the state of one qubit with another.

Multiple Qubits...

- An example conditional statement would be

$$ifQ :: Qbit \rightarrow U \rightarrow U$$
$$ifQ\ q\ u = cond\ q\ (\lambda x \rightarrow \mathbf{if}\ x\ \mathbf{then}\ u\ \mathbf{else}\ \bullet)$$

- The Unitary aspect of U defines exactly that they are reversible, and we provide the function $urev :: U \rightarrow U$ which constructs the inverse.
- The No Cloning theorem tells us that we cannot create a copy of an arbitrary quantum state.
- We can however "share" the state of one qubit with another.
- e.g. from a state $|\phi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ we can create the state $|\psi\rangle = \alpha\,|00\rangle + \beta\,|11\rangle$.

Multiple Qubits...

- $$
\begin{aligned}
&share :: Qbit \rightarrow QIO\ Qbit \\
&share\ qa = \mathbf{do}\ qb \leftarrow |0\rangle \\
&\qquad\qquad\qquad applyU\ (ifQ\ qa\ (unot\ qb)) \\
&\qquad\qquad\qquad return\ qb
\end{aligned}
$$

Multiple Qubits...

- $$\begin{aligned} share &:: Qbit \rightarrow QIO\ Qbit \\ share\ qa &= \mathbf{do}\ qb \leftarrow |0\rangle \\ &\qquad applyU\ (ifQ\ qa\ (unot\ qb)) \\ &\qquad return\ qb \end{aligned}$$
- We can now use this to create the bell state $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$ from the $|+\rangle$ state.

Multiple Qubits...

- $$\begin{aligned} &share :: Qbit \rightarrow QIO\ Qbit \\ &share\ qa = \mathbf{do}\ qb \leftarrow |0\rangle \\ &\qquad\qquad\qquad applyU\ (ifQ\ qa\ (unot\ qb)) \\ &\qquad\qquad\qquad return\ qb \end{aligned}$$
- We can now use this to create the bell state $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$ from the $|+\rangle$ state.
- $$\begin{aligned} &bell :: QIO\ (Qbit, Qbit) \\ &bell = \mathbf{do}\ qa \leftarrow |+\rangle \\ &\qquad\qquad qb \leftarrow share\ qa \\ &\qquad\qquad return\ (qa, qb) \end{aligned}$$

Measurement Side Effects

- lets now consider this 2-qubit bell state
  $|\psi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$

Measurement Side Effects

- lets now consider this 2-qubit bell state
  $|\psi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$
- What happens if we measure one of the qubits?

Measurement Side Effects

- lets now consider this 2-qubit bell state
  $|\psi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$
- What happens if we measure one of the qubits?
- With equal probability we'll measure $|0\rangle$ or $|1\rangle$.

Measurement Side Effects

- lets now consider this 2-qubit bell state $|\psi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$
- What happens if we measure one of the qubits?
- With equal probability we'll measure $|0\rangle$ or $|1\rangle$.
- Meaning that the measurement has collapsed the entire state into either $|00\rangle$ or $|11\rangle$.

Measurement Side Effects

- lets now consider this 2-qubit bell state $|\psi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$
- What happens if we measure one of the qubits?
- With equal probability we'll measure $|0\rangle$ or $|1\rangle$.
- Meaning that the measurement has collapsed the entire state into either $|00\rangle$ or $|11\rangle$.
- Measuring the first qubit, has the side-effect of also collapsing the second qubit into one of its base states.

Measurement Side Effects

The University of Nottingham

- lets now consider this 2-qubit bell state $|\psi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$
- What happens if we measure one of the qubits?
- With equal probability we'll measure $|0\rangle$ or $|1\rangle$.
- Meaning that the measurement has collapsed the entire state into either $|00\rangle$ or $|11\rangle$.
- Measuring the first qubit, has the side-effect of also collapsing the second qubit into one of its base states.
- In this example, the 2 qubits are Entangled . The state of one depends on the state of the other.

Measurement Side Effects

- lets now consider this 2-qubit bell state $|\psi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$
- What happens if we measure one of the qubits?
- With equal probability we'll measure $|0\rangle$ or $|1\rangle$.
- Meaning that the measurement has collapsed the entire state into either $|00\rangle$ or $|11\rangle$.
- Measuring the first qubit, has the side-effect of also collapsing the second qubit into one of its base states.
- In this example, the 2 qubits are Entangled . The state of one depends on the state of the other.
- In QIO, conditional statements are used to introduce entanglement into a computation.

Deutsch's Algorithm

- We now have everything we need to start defining quantum computations in Haskell. E.g. Deutsch's Algorithm

Deutsch's Algorithm

- We now have everything we need to start defining quantum computations in Haskell. E.g. Deutsch's Algorithm
- $$\begin{aligned}
&\mathit{deutsch} :: (\mathit{Bool} \rightarrow \mathit{Bool}) \rightarrow \mathit{QIO}\ \mathit{Bool} \\
&\mathit{deutsch}\ f = \mathbf{do}\ x \leftarrow |+\rangle \\
&\qquad\qquad\qquad\quad y \leftarrow |-\rangle \\
&\qquad\qquad\qquad\quad \mathit{applyU}\ (\mathit{cond}\ x\ (\lambda b \rightarrow \\
&\qquad\qquad\qquad\qquad\qquad\qquad \mathbf{if}\ f\ b\ \mathbf{then}\ \mathit{unot}\ y \\
&\qquad\qquad\qquad\qquad\qquad\qquad \mathbf{else}\ \bullet)) \\
&\qquad\qquad\qquad\quad \mathit{applyU}\ (\mathit{uhad}\ x) \\
&\qquad\qquad\qquad\quad \mathit{measQbit}\ x
\end{aligned}$$

Running Quantum Computations

- These QIO programs are Quantum Computations of the given embedded type, but can we actually run or evaluate these quantum computations?

Running Quantum Computations

- These QIO programs are Quantum Computations of the given embedded type, but can we actually run or evaluate these quantum computations?
- We could get our handy USB quantum register, and just get it to evaluate the program for us...

Running Quantum Computations

- These QIO programs are Quantum Computations of the given embedded type, but can we actually run or evaluate these quantum computations?
- We could get our handy USB quantum register, and just get it to evaluate the program for us...
- but it doesn't exist yet!

Running Quantum Computations

- These QIO programs are Quantum Computations of the given embedded type, but can we actually run or evaluate these quantum computations?
- We could get our handy USB quantum register, and just get it to evaluate the program for us...
- but it doesn't exist yet!
- So, we also provide three evaluation functions for QIO Programs

Running Quantum Computations

- These QIO programs are Quantum Computations of the given embedded type, but can we actually run or evaluate these quantum computations?
- We could get our handy USB quantum register, and just get it to evaluate the program for us...
- but it doesn't exist yet!
- So, we also provide three evaluation functions for QIO Programs
- $$run :: QIO\ a \rightarrow IO\ a$$
  $$sim :: QIO\ a \rightarrow Prob\ a$$
  $$runC :: QIO\ a \rightarrow a$$

Running Quantum Computations

- e.g.

$$> run\ (deutsch\ \neg)$$

$$True$$

$$> sim\ (deutsch\ id)$$

$$[(True, 1.0)]$$

$$> run\ (deutsch\ (\lambda x \rightarrow True))$$

$$False$$

$$> sim\ (deutsch\ (\lambda x \rightarrow False))$$

$$[(False, 1.0)]$$

$$> sim\ randomBool$$

$$[(True, 0.5), (False, 0.5)]$$

Quantum Data-types

- We don't always want to think of quantum computations simply as acting on qubits, for example, it would be natural to think of Shor's algorithm as having the type $shor :: Int \rightarrow QIO\ Int$.

Quantum Data-types

- We don't always want to think of quantum computations simply as acting on qubits, for example, it would be natural to think of Shor's algorithm as having the type $shor :: Int \rightarrow QIO\ Int$.
- We decided to implement a class of Quantum Data-types , that defines a relation between a classical type, and a corresponding quantum version of that type.

Quantum Data-types

- We don't always want to think of quantum computations simply as acting on qubits, for example, it would be natural to think of Shor's algorithm as having the type $shor :: Int \rightarrow QIO\ Int$.
- We decided to implement a class of Quantum Data-types , that defines a relation between a classical type, and a corresponding quantum version of that type.
- $$\begin{aligned} &\textbf{class}\ Qdata\ a\ qa \mid a \rightarrow qa, qa \rightarrow a\ \textbf{where} \\ &\quad mkQ :: a \rightarrow QIO\ qa \\ &\quad measQ :: qa \rightarrow QIO\ a \\ &\quad condQ :: qa \rightarrow (a \rightarrow U) \rightarrow U \end{aligned}$$

Quantum Data-types

- The simplest instance of this class would be the correspondance between boolean values and qubits, which just uses the QIO constructors we have already seen.

Quantum Data-types

- The simplest instance of this class would be the correspondance between boolean values and qubits, which just uses the QIO constructors we have already seen.
- We can also define pairs, lists...

Quantum Data-types

- The simplest instance of this class would be the correspondance between boolean values and qubits, which just uses the QIO constructors we have already seen.
- We can also define pairs, lists...
-

$$
\begin{array}{l}
\textbf{instance } Qdata\ a\ qa \Rightarrow Qdata\ [a]\ [qa]\ \textbf{where} \\
\quad mkQ\ n = sequence\ (map\ mkQ\ n) \\
\quad measQ\ qs = sequence\ (map\ measQ\ qs) \\
\quad condQ\ qs\ qsu = condQ'\ qs\ [\,] \\
\qquad \textbf{where } condQ'\ [\,] \qquad xs = qsu\ xs \\
\qquad\qquad condQ'\ (a : as)\ xs = condQ\ a\ (\lambda x \rightarrow condQ'\ as\ (xs +\!\!+ [x]))
\end{array}
$$

Quantum Data-types

- The simplest instance of this class would be the correspondance between boolean values and qubits, which just uses the QIO constructors we have already seen.
- We can also define pairs, lists...
- $$\begin{aligned}
&\textbf{instance } Qdata\ a\ qa \Rightarrow Qdata\ [a]\ [qa]\ \textbf{where}\\
&\quad mkQ\ n = sequence\ (map\ mkQ\ n)\\
&\quad measQ\ qs = sequence\ (map\ measQ\ qs)\\
&\quad condQ\ qs\ qsu = condQ'\ qs\ [\,]\\
&\qquad \textbf{where } condQ'\ [\,] \qquad xs = qsu\ xs\\
&\qquad\qquad\quad condQ'\ (a : as)\ xs = condQ\ a\ (\lambda x \rightarrow condQ'\ as\ (xs +\!\!+ [x]))
\end{aligned}$$
- and a Quantum Integer , which converts an Int to a (fixed-length) list of booleans, and defines a $QInt$ as a synonym for a list of qubits.

More QIO Programs

- We have implemented other Quantum Algorithms using QIO, including Quantum Teleportation, and Shor's Algorithm.

More QIO Programs

- We have implemented other Quantum Algorithms using QIO, including Quantum Teleportation, and Shor's Algorithm.
- The paper goes into much more detail about the implementation of Shor's algorithm and the QIO evaluator.

More QIO Programs

- We have implemented other Quantum Algorithms using QIO, including Quantum Teleportation, and Shor's Algorithm.
- The paper goes into much more detail about the implementation of Shor's algorithm and the QIO evaluator.
- Shor's algorithm required that we have a set of unitary operators that can perform reversible arithmetic , specifically modular exponentiation.

More QIO Programs

- We have implemented other Quantum Algorithms using QIO, including Quantum Teleportation, and Shor's Algorithm.
- The paper goes into much more detail about the implementation of Shor's algorithm and the QIO evaluator.
- Shor's algorithm required that we have a set of unitary operators that can perform reversible arithmetic , specifically modular exponentiation.
- Many of the arithmetic functions require auxilliary qubits, so we have also added a unitary-let operation $ulet :: Bool \to (Qbit \to U) \to U$ that enables the system to keep track of them.

More QIO Programs

- Shor's algorithm also requires a unitary operator that performs the Quantum Fourier Transform , and details of the implementation can also be found in the paper.

More QIO Programs

- Shor's algorithm also requires a unitary operator that performs the Quantum Fourier Transform , and details of the implementation can also be found in the paper.
- The Quantum Fourier Transform is essentially an implementation of the discrete Fourier transform, that can act on a quantum state, and is used in many quantum algorithms.

More QIO Programs

- Shor's algorithm also requires a unitary operator that performs the Quantum Fourier Transform , and details of the implementation can also be found in the paper.
- The Quantum Fourier Transform is essentially an implementation of the discrete Fourier transform, that can act on a quantum state, and is used in many quantum algorithms.
- It's use in Shor's algorithm is to find the order of a modular exponentiation function that's constructed depending on the input.

Problems?

- As it stands, conditionals can be created that aren't unitary.

Problems?

- As it stands, conditionals can be created that aren't unitary.
- $$notUnitary :: U$$
  $$notUnitary = cond\ x\ (\lambda x \rightarrow \textbf{if}\ x\ \textbf{then}\ unot\ x\ \textbf{else}\ \bullet)$$

Problems?

- As it stands, conditionals can be created that aren't unitary.
- $$notUnitary :: U$$
  $$notUnitary = cond\ x\ (\lambda x \rightarrow \textbf{if}\ x\ \textbf{then}\ unot\ x\ \textbf{else}\ \bullet)$$
- The given function always leaves the qubit x in the state $|0\rangle$.

Problems?

- As it stands, conditionals can be created that aren't unitary.
- $$notUnitary :: U$$
  $$notUnitary = cond\ x\ (\lambda x \rightarrow \mathbf{if}\ x\ \mathbf{then}\ unot\ x\ \mathbf{else}\ \bullet)$$
- The given function always leaves the qubit x in the state $|0\rangle$.
- A side condition for conditionals must be introduced, that the branches of the conditional must not reference the control qubit.

Problems?

- As it stands, conditionals can be created that aren't unitary.
- $$notUnitary :: U$$
  $$notUnitary = cond\ x\ (\lambda x \rightarrow \textbf{if}\ x\ \textbf{then}\ unot\ x\ \textbf{else}\ \bullet)$$
- The given function always leaves the qubit x in the state $|0\rangle$.
- A side condition for conditionals must be introduced, that the branches of the conditional must not reference the control qubit.
- Trying to run the $notUnitary$ function will result in a run-time error.

Side Conditions...

- The *ulet* constructor could easily give rise to non-unitary behaviour...

Side Conditions...

- The *ulet* constructor could easily give rise to non-unitary behaviour...
- e.g. the temporary qubit could be left entangled with the rest of the state.

Side Conditions...

- The $ulet$ constructor could easily give rise to non-unitary behaviour...
- e.g. the temporary qubit could be left entangled with the rest of the state.
- The side-condition imposed for $ulet$ is that the temporary qubit must be returned to its original state.

Side Conditions...

- The *ulet* constructor could easily give rise to non-unitary behaviour...
- e.g. the temporary qubit could be left entangled with the rest of the state.
- The side-condition imposed for *ulet* is that the temporary qubit must be returned to its original state.
- It would also be possible to create a non-unitary single qubit rotation.

Side Conditions...

- The *ulet* constructor could easily give rise to non-unitary behaviour...
- e.g. the temporary qubit could be left entangled with the rest of the state.
- The side-condition imposed for *ulet* is that the temporary qubit must be returned to its original state.
- It would also be possible to create a non-unitary single qubit rotation.
- The side-condition for rotations is that they must be unitary!

Side Conditions...

- The *ulet* constructor could easily give rise to non-unitary behaviour...
- e.g. the temporary qubit could be left entangled with the rest of the state.
- The side-condition imposed for *ulet* is that the temporary qubit must be returned to its original state.
- It would also be possible to create a non-unitary single qubit rotation.
- The side-condition for rotations is that they must be unitary!
- Again, in both cases, failure to comply will result in a run-time error.

Conclusions

- A Dependently-Typed implementation of QIO could move the side conditions to the Type level...

Conclusions

- A Dependently-Typed implementation of QIO could move the side conditions to the Type level...
- leaving a sound implementation that could be used to start reasoning about quantum computations.

Conclusions

- A Dependently-Typed implementation of QIO could move the side conditions to the Type level...
- leaving a sound implementation that could be used to start reasoning about quantum computations.
- We are also looking at possibly extending QIO to be a full language...

Conclusions

- A Dependently-Typed implementation of QIO could move the side conditions to the Type level...
- leaving a sound implementation that could be used to start reasoning about quantum computations.
- We are also looking at possibly extending QIO to be a full language...
- The current implementation of QIO is available online: http://www.cs.nott.ac.uk/~asg/QIO/

Conclusions

- A Dependently-Typed implementation of QIO could move the side conditions to the Type level...
- leaving a sound implementation that could be used to start reasoning about quantum computations.
- We are also looking at possibly extending QIO to be a full language...
- The current implementation of QIO is available online: http://www.cs.nott.ac.uk/˜asg/QIO/
- Thank you!