G53NSC and G54NSC
Non-Standard Computation

Dr. Alexander S. Green

2nd of February 2010

Part I

Preliminaries

Course Deadlines

- The module deadlines are now finalised!
- Portfolio
  - Weekly hand-ins for feedback (to me)
  - Final deadline: 12:00 (midday), Thursday the 1st of April 2010
  - Final submission is via the school office
- Research Paper
  - Initial deadline: 12:00 (midday), Friday the 19th of March 2010
  - Presentations on 23rd and 30th of March, 11:00 to 13:00
  - Final deadline: 12:00 (midday), Tuesday the 11th of May 2010
  - Both submissions are via the school office

Recap of the previous lecture

- The Church-Turing thesis
  - What about unfeasible computations?
- The Extended Church-Turing thesis
  - What about Non-Standard models of computation?
  - e.g. Quantum Computation
- Shor's algorithm could factorise large numbers in polynomial time on a suitably sized *Quantum Computer*

Recap of the previous lecture

- Quantum computation is inspired by Quantum Mechanics
- At the *quantum* scale, matter exhibits both wave-like and particle-like behaviour
  - Wave-particle duality
- The Copenhagen interpretation
  - States are described by a wavefunction
  - Amplitudes correspond to probabilities of certain observations
- Dirac (or Bra-Ket) notation is used for describing quantum states

What are we covering today?

- Classical computation
- Universality
- Reversible computation
- Is reversible computation universal?
- A look at Dirac notation for reversible computation
- Reversible computation with the Quantum IO Monad

Part II

Classical computation

What are computations?

- We have bits that can be in the states 0 or 1
- Computations take strings of bits to other such strings
- Abstractly we can treat bits as Boolean values...
- and Computations as logical operations acting on these bits
- Physically, computers must contain physical systems that can represent these abstract "bits"...
- E.g. a system that can exist in two unambiguously distinguishable states
  - Switches that can be "On" or "Off"
  - Magnetic polarisation that can be "Up" or "Down"
  - etc.
- and Logic gates that can manipulate the states accordingly
- We usually call both the abstract values, and the physical systems "bits"

Are all “Computations” possible?

- For Universal computation, we must be able to translate any arbitrary bit string to any other arbitrary bit string
- What logical operations do we require for this?
- How can we prove whether a set of logical operations is universal?
- Logical operations correspond to Boolean functions, and define a universal set if the corresponding Boolean functions are functionally complete

Functional Completeness

Functional Completeness

A set of Boolean functions ($f_i : \{0,1\}^{n_i} \rightarrow \{0,1\}$) is functionally complete, if all other Boolean functions ($f : \{0,1\}^n \rightarrow \{0,1\}$ for all $n \geqslant 1$) can be constructed from this set, along with a set of input variables.

- The set $\{\wedge, \vee, \neg\}$ is a common functionally complete set

Functional Completeness of $\{\wedge, \vee, \neg\}$

- We can sketch a proof that the set $\{\wedge, \vee, \neg\}$ is functionally complete
- First, we can look at the truth tables for these operations

$\wedge : \{0,1\}^2 \to \{0,1\}$

b_0	b_1	$b_0 \wedge b_1$
0	0	0
0	1	0
1	0	0
1	1	1

$\vee : \{0,1\}^2 \to \{0,1\}$

b_0	b_1	$b_0 \vee b_1$
0	0	0
0	1	1
1	0	1
1	1	1

$\neg : \{0,1\} \to \{0,1\}$

b_0	$\neg b_0$
0	1
1	0

Functional Completeness of $\{\wedge, \vee, \neg\}$

- We can now also think of our arbitray Boolean functions in terms of their truth tables.
- E.g. For any $f : \{0,1\}^n \rightarrow \{0,1\}$ we can give the truth table

b_0	b_1	$\dots$	b_n	$f(b_0, b_1, \dots, b_n)$
0	0	$\dots$	0	$f(0,0,\dots,0)$
0	0	$\dots$	1	$f(0,0,\dots,1)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	1	$\dots$	0	$f(1,1,\dots,0)$
1	1	$\dots$	1	$f(1,1,\dots,1)$

Functional Completeness of $\{\wedge, \vee, \neg\}$

- Lets now look at a subset of all Boolean functions.
- Boolean functions that evaluate to 1 for only one input state, and 0 for all other input states
- We can use $\wedge$ and $\neg$ to explicitly define these Boolean functions.
- We shall call these type of functions minterms
- E.g. The function $f : \{0,1\}^5 \rightarrow \{0,1\}$ which evaluates to 1 only on the input $(b_0, b_1, b_2, b_3, b_4) = (1,0,0,1,1)$ is defined exactly by $b_0 \wedge \neg b_1 \wedge \neg b_2 \wedge b_3 \wedge b_4$

Functional Completeness of $\{\wedge, \vee, \neg\}$

b_0	b_1	b_2	b_3	b_4	$b_0 \wedge \neg b_1 \wedge \neg b_2 \wedge b_3 \wedge b_4$
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	0
0	0	1	0	0	0
0	0	1	0	1	0
0	0	1	1	0	0
0	0	1	1	1	0
0	1	0	0	0	0
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	0	1	0
0	1	1	1	0	0
0	1	1	1	1	0
1	0	0	0	0	0
1	0	0	0	1	0
1	0	0	1	0	0
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	0	1	0
1	0	1	1	0	0
1	0	1	1	1	0
1	1	0	0	0	0
1	1	0	0	1	0
1	1	0	1	0	0
1	1	0	1	1	0
1	1	1	0	0	0
1	1	1	0	1	0
1	1	1	1	0	0
1	1	1	1	1	0

Functional Completeness of $\{\wedge, \vee, \neg\}$

- All Boolean functions (except constant 0) can now be constructed using the type of functions we have just been looking at, along with the $\vee$ operator.
- An arbitrary function is defined by combining the minterms for each input that evaluates to 1.
- E.g. the function $f : \{0,1\}^3 \to \{0,1\}$ with the following truth table:

b_0	b_1	b_2	$f(b_0, b_1, b_2)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

- can be given by
  $f = (\neg b_0 \wedge \neg b_1 \wedge b_2) \vee (\neg b_0 \wedge b_1 \wedge \neg b_2) \vee (b_0 \wedge \neg b_1 \wedge b_2)$

Functional Completeness of $\{\wedge, \vee, \neg\}$

- The special case of the constant 0 function can be defined in its simplest form by the function $0 : \{0,1\} \rightarrow \{0,1\}$ such the $0(b_0) = b_0 \wedge \neg b_0$
- We can use this proof to try and find other universal sets of functions
- Any set of Boolean functions that can be used to define these three functions must also be universal.
- Are there any smaller sets?
- We can use involution and de Morgan's laws to define $\vee$ in terms of $\wedge$ and $\neg$.
- $b_0 \vee b_1 = \neg(\neg b_0 \wedge \neg b_1)$
- So, the set $\{\wedge, \neg\}$ is also universal

What about even smaller sets?

- Can we find a single Boolean function that is functionally complete
- In fact, such functions do exist
- NAND ($\uparrow$) and NOR ($\downarrow$) are both examples of functionally complete Boolean functions

$NAND : \{0,1\}^2 \to \{0,1\}$

b_0	b_1	$b_0 \uparrow b_1$
0	0	1
0	1	1
1	0	1
1	1	0

$NOR : \{0,1\}^2 \to \{0,1\}$

b_0	b_1	$b_0 \downarrow b_1$
0	0	1
0	1	0
1	0	0
1	1	0

- Can we prove that they are universal?

Universality of NAND

- We can define $\neg$ in terms of a NAND gate
- $\neg b_0 = b_0 \uparrow b_0$
- We can now define $\wedge$ in terms of NAND and $\neg$
- $b_0 \wedge b_1 = \neg(b_0 \uparrow b_1)$
- Thus showing NAND is universal
- The proof for NOR is very similar (see this weeks lab exercises)

Towards Quantum Computation

- Can we extend these universal sets to give us quantum computaion?
- Unfortunately it’s not that simple...
- In Quantum Mechanics, all physical processes are by definition, unitary
- That is, every process has an inverse
- Can we define inverses for any of the sets of functions we have seen?

Irreversibility of $\wedge$, NAND, and NOR

- Lets look again at the truth tables for $\wedge$, NAND, and NOR

b_0	b_1	$b_0 \wedge b_1$
0	0	0
0	1	0
1	0	0
1	1	1

b_0	b_1	$b_0 \uparrow b_1$
0	0	1
0	1	1
1	0	1
1	1	0

b_0	b_1	$b_0 \downarrow b_1$
0	0	1
0	1	0
1	0	0
1	1	0

- None of them are reversible

Part III

Reversible Computation

Reversible Computation

- How can we define Reversible Computation?
- A reversible computation is one whose transition functions are of a one-to-one nature (or injective)
- Reversible computation is interesting to us, as the operations in classical reversible computation form a subset of the operations available in quantum computation
- Reversible computation has been well studied for various other reasons too!
- We'll look now at a brief history of classical reversible computation

A History of Reversible Computation

Rolf Landauer
Landauer's principle

Landauer's principle

"any logically irreversible manipulation of information, such as the erasure of a bit or the merging of two computation paths, must be accompanied by a corresponding entropy increase in non-information bearing degrees of freedom of the information processing apparatus or its environment"

Landauer's principle

- Landauer's principle follows from the second law of thermodynamics
  - The entropy of a closed system cannot decrease
- Entropy can be thought of as the number of ways in which a system can be arranged
- A logically irreversible computation defines a process that decreases entropy, so this must be accounted for by an increase of entropy in the rest of the system
- For a change in entropy (S), this will result in energy ($E = ST$) being released (where T is the temperature of the system).

A History of Reversible Computation

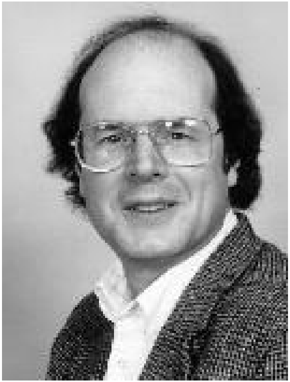
John von Neumann

- John von Neumann suggested that the minimum energy dissipation from a logically reversible binary operation is $kTlog_e 2$, where k is the Boltzmann constant, and T is the temperature of the environment
- Landauer justified this limit
- Giving us what is now called the von Neumann-Landauer limit

The von Neumann-Landauer limit

- The von Neumann-Landauer limit of $kTlog_e 2$ per bit of *lost* information gives us a fundamental limit for the energy efficiency of irreversible computation
- Reversible computation allows us to overcome this limit.
- Rolf Landauer also concluded that for any computational process to be reversible, it must be logically reversible
- This means we can look at reversible computation in a similar manner as we have been irreversible classical computation, in terms of reversible logic gates
- What can we actually do in computational terms?

Logical reversibility of computation



Charles H. Bennett

- Bennett wrote what is now thought of as the seminal paper on reversible computation
- "Logical reversibility of computation"
- published in 1973 in the IBM journal of Research and Development

Reversible logic gates

- How can we think of reversible computation today?
- What logic gates can we define that are reversible?
- Do these reversible logic gates give us a universal set?
- Have we seen any reversible logic gates already?
- The $\neg$ operator is logically reversible

b_0	$\neg b_0$
0	1
1	0

- What is its inverse?
  - It is its own inverse
  - E.g. $\neg(\neg b_0) = b_0$

Reversible logic gates

- The only other 1-bit reversible logic gate is the identity
- These two reversible logic gates are not universal, so we need to look at 2-bit reversible logic gates
- There are 24 2-bit reversible logic gates
- In fact, for n bits, there are $2^n!$ reversible logic gates
- It is useful to look at 2-bit reversible logic gates to see what is required to construct them (above and beyond the 1-bit reversible logic gates)

2-bit reversible logic gates

- We shall start giving logic gates a notation in the form of circuits, along with their corresponding truth tables
- The simplest circuit is just a wire and represents the identity gate

b_0 ——— b_0

b_0	b_0
0	0
1	1

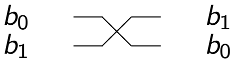
- The $\neg$ operation can also be represented as a circuit

b_0 —$\boxed{X}$— $\neg b_0$

b_0	$\neg b_0$
0	1
1	0

2-bit reversible logic gates

- For more than 1-bit, we only need to introduce two new constructs
- Firstly, the Swap operation

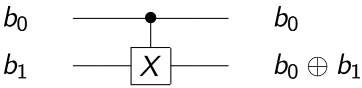


b_{0in}	b_{1in}	b_{0out}	b_{1out}
0	0	0	0
0	1	1	0
1	0	0	1
1	1	1	1

- Swaps are used to *wire up* circuits
- The other new construct is the control structure
- Depending on the value of a control wire, a logic gate is applied

2-bit reversible logic gates

- For example, a controlled-X operation



b_{0in}	b_{1in}	b_{0out}	b_{1out}
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

- Any other logic gate can be controlled, such as a controlled-Swap
- ...or even another control structure

Are 2-bit reversible logic gates universal?

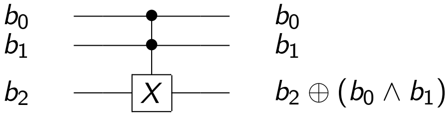
- We could now construct any of the 24 2-bit reversible logic gates
- Does this give us a universal set of reversible logic gates?
- Unfortunately it doesn't...
- The only new logical operation we can achieve is the XOR gate (or permutations thereof)
- The set $\{XOR, \neg\}$ is not universal
- So, is reversible computation universal?
- Can we find a 3-bit reversible circuit that is universal?

Tommaso Toffoli
The Toffoli gate

Edward Fredkin
The Fredkin gate

They are both examples of universal 3-bit reversible logic gates

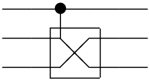
The Toffoli gate



b_{0in}	b_{1in}	b_{2in}	b_{0out}	b_{1out}	b_{2out}
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

The Fredkin gate

b_0
b_1
b_2



b_0
$b_1 \oplus ((b_1 \oplus b_2) \wedge b_0)$
$b_2 \oplus ((b_1 \oplus b_2) \wedge b_0)$

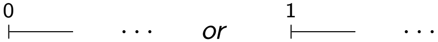
b_{0in}	b_{1in}	b_{2in}	b_{0out}	b_{1out}	b_{2out}
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	0	1
1	1	1	1	1	1

Universality of the Toffoli gate

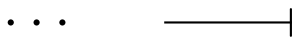
- There are other universal 3-bit reversible logic gates
- We shall look at the Toffoli gate
- In order to prove universality, we must introduce the idea of defining irreversible computations embedded within reversible computations
- This is possible if we have both a set of heap inputs, and garbage outputs
- The heap consists of any extra bits initialised to 0 or 1 that are required by the reversible computation
- The garbage consists of output bits that aren't part of the result of the irreversible computation, but are required as part of the reversible computation

Universality of the Toffoli gate

- Heap inputs are denoted:

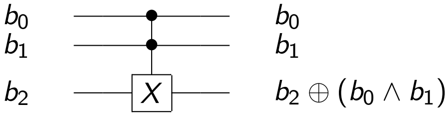


- Garbage outputs are denoted:



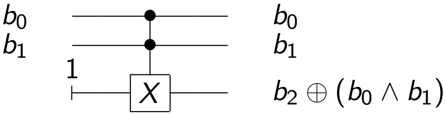
- What can we do with the Toffoli gate, along with heap and garbage?

The Toffoli gate



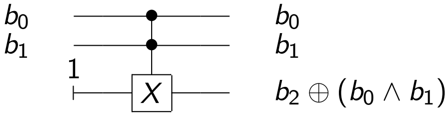
b_{0in}	b_{1in}	b_{2in}	b_{0out}	b_{1out}	b_{2out}
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

The Toffoli gate with Heap



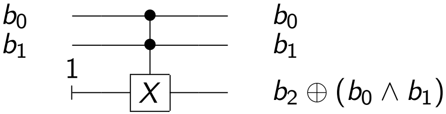
b_{0in}	b_{1in}	b_{2in}	b_{0out}	b_{1out}	b_{2out}
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

The Toffoli gate with Heap



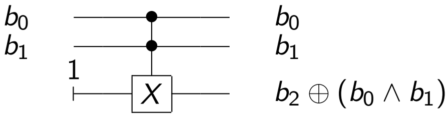
b_{0in}	b_{1in}	b_{2in}	b_{0out}	b_{1out}	b_{2out}
0	0	1	0	0	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

The Toffoli gate with Heap



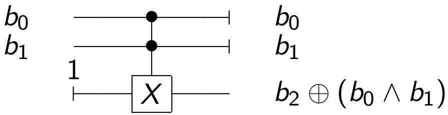
b_{0in}	b_{1in}	b_{2in}	b_{0out}	b_{1out}	b_{2out}
0	0	1	0	0	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

The Toffoli gate with Heap



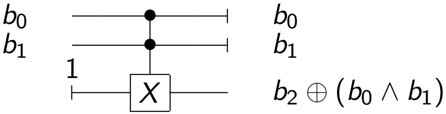
b_{0in}	b_{1in}	b_{0out}	b_{1out}	b_{2out}
0	0	0	0	1
0	1	0	1	1
1	0	1	0	1
1	1	1	1	0

The Toffoli gate with Heap and Garbage



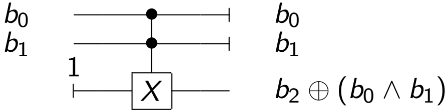
b_{0in}	b_{1in}	b_{0out}	b_{1out}	b_{2out}
0	0	0	0	1
0	1	0	1	1
1	0	1	0	1
1	1	1	1	0

The Toffoli gate with Heap and Garbage



b_{0in}	b_{1in}	b_{0out}	b_{1out}	b_{2out}
0	0			1
0	1			1
1	0			1
1	1			0

The Toffoli gate with Heap and Garbage



b_{0in}	b_{1in}	b_{2out}
0	0	1
0	1	1
1	0	1
1	1	0

Universality of the Toffoli gate

b_{0in}	b_{1in}	b_{2out}
0	0	1
0	1	1
1	0	1
1	1	0

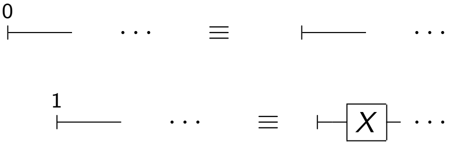
- Using Heaps and Garbage we can embed the NAND operation into a Toffoli gate
- This means that the Toffoli gate is universal
- Hence, reversible computation is universal
- However, to keep the computations reversible we must keep track of the Garbage outputs

Generalised Reversible Computation

- In general, we can use extra Heap inputs as ancilliary bits to *copy out* the results, allowing us to run the reverse computation over the inputs so we don't need to keep track of garbage
- It is this generalised type of reversible computation that was first suggested in Bennett's paper
- We can re-implement our NAND construction following this practice
- First, we need the inverse of the toffoli gate
- Fortunately for us, it is self inverse
- We make use of a controlled-X to *copy out* the result onto a zeroed ancilliary bit

Generalised Reversible Computation

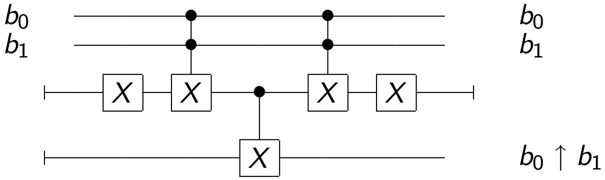
- We can simplify things even further by restricting heap inputs to only be 0
- Adding a X gate to the circuit on heaps that need to be 1
- Using this we can simplify our notation for heaps



- All outputs are now either the inputs, zeroes, or the result of the computation

Generalised Reversible Computation

- We can give our NAND construct using this generalisation



- The lab exercises this week will look at more complicated reversible computations.

Part IV

Dirac notation for classical reversible computation

Dirac notation

- We can represent the states of a bit as either $|0\rangle$ or $|1\rangle$
- For more than one bit we can extend this notation
- For 2 bits, there are four possible states

$$|0\rangle\,|0\rangle,\ |0\rangle\,|1\rangle,\ |1\rangle\,|0\rangle,\ |1\rangle\,|1\rangle$$

- For 3 bits, there are eight possible states

$$\begin{array}{c} |0\rangle\,|0\rangle\,|0\rangle,\ |0\rangle\,|0\rangle\,|1\rangle,\ |0\rangle\,|1\rangle\,|0\rangle,\ |0\rangle\,|1\rangle\,|1\rangle \\ |1\rangle\,|0\rangle\,|0\rangle,\ |1\rangle\,|0\rangle\,|1\rangle,\ |1\rangle\,|1\rangle\,|0\rangle,\ |1\rangle\,|1\rangle\,|1\rangle \end{array}$$

- For n bits, there are 2^n possible states

Dirac notation

- To ease notation, we can write each possible state in its own ket construct
- The possible 2-bit states now become

$$|00\rangle,\ |01\rangle,\ |10\rangle,\ |11\rangle$$

- The possible 3-bit states become

$$|000\rangle,\ |001\rangle,\ |010\rangle,\ |011\rangle,\ |100\rangle,\ |101\rangle,\ |110\rangle,\ |111\rangle$$

- But why do we use Dirac notation?
- It is useful to start thinking of states in terms of vectors, Dirac notation is used to represent these vectors

States as vectors

- We can think of the states of a bit in terms of two orthogonal unit vectors in a two-dimensional space
- What do all these terms mean?
- For classical reversible computation we can restrict ourselves to real valued vectors, which correspond nicely to Euclidean geometry
- A vector in an *n*-dimensional space can be given in terms of *n* (real valued) components
- A unit vector is a vector whose norm is 1
- The norm of a vector can be thought of (geometrically) as its length, or Euclidean norm
- Two vectors are orthogonal if their inner product equals zero
- In a Euclidean space, inner product is simply dot product

Definitions

For vectors $x = \begin{pmatrix} x_o \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix}$ and $y = \begin{pmatrix} y_o \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix}$ in an n-dimensional vector space (with $x_i, y_i \in \mathbb{R}$ for all $0 \leqslant i < n$)

the Euclidean norm of $x = \sqrt{x_0^2 + x_1^2 + \ldots + x_{n-1}^2}$

the innner product of x and $y = x^T y =$

$$(x_0, x_1, \ldots, x_{n-1}) \begin{pmatrix} y_o \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix} = x_0 y_0 + x_1 y_1 + \ldots + x_{n-1} y_{n-1}$$

Dirac notation

- So we can think of a two dimensional vector space over the real numbers as a plane
- All unit vectors form a circle of radius 1 about the origin
- For classical computation, we need two orthogonal vectors to represent the states of our bit
- Orthogonality in our geometric interpretation corresponds to vectors seperated by an angle of 90^o
- To keep things simple we choose the following two vectors:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

- Operations on bits must only map between these two states

Single-bit operations

- How can we define these operations?
- Well these operations (or computations) are thought of in terms of matrices
- The reversibility is enforced by restricting ourselves to unitary matrices
- Having only the states $|0\rangle$ and $|1\rangle$ also means we are restricted to matrices that only contain 1s and 0s
- So, what operations can we define on a single bit?
- How can we apply these operations to our bit?

Single-bit operations

- There are only two, two-dimensional unitary matrices that contain only 0s and 1s

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

- We shall now call these operations, unitary operators
- How do we apply these unitary operators?
- The application of a unitary operator corresponds to matrix (pre) multiplication
- Lets see what these two single bit unitary operators correspond to

Single-bit operations

- This kind of matrix (pre) multiplication is defined by:

$$\begin{bmatrix} a_{00} & a_{01} & \cdots & a_{0(n-1)} \\ a_{10} & a_{11} & \cdots & a_{1(n-1)} \\ \vdots & \vdots & \cdots & \vdots \\ a_{(n-1)0} & a_{(n-1)1} & \cdots & a_{(n-1)(n-1)} \end{bmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{pmatrix}$$

$$= \begin{pmatrix} a_{00}b_0 + a_{01}b_1 + \ldots + a_{0(n-1)}b_{n-1} \\ a_{10}b_0 + a_{11}b_1 + \ldots + a_{1(n-1)}b_{n-1} \\ \vdots \\ a_{(n-1)0}b_0 + a_{(n-1)1}b_1 + \ldots + a_{(n-1)(n-1)}b_{n-1} \end{pmatrix}$$

Single-bit operations

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} |0\rangle = |0\rangle \quad \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} |1\rangle = |1\rangle$$

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} |0\rangle = |1\rangle \quad \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} |1\rangle = |0\rangle$$

- These unitary operators correspond exactly to the single-bit operations defined previously
- In fact we can keep the circuit diargram notation
- But what about more than one bit?

Mutiple bits

- We've already seen how we can write multiple bit states using Dirac notation
- But what does this mean in terms of our vectors?
- Within Dirac notation we implicitly use the tensor product to create higher-dimensional vector spaces
- The tensor of two single bit states is an element of a 4-dimensional vector space
- The tensor of n single bit states is an element of a 2^n-dimensional vector space
- For example, when we write $|101\rangle$ we actually mean $|1\rangle \otimes |0\rangle \otimes |1\rangle$

Tensor product

$$\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} \otimes \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{m-1} \end{pmatrix} = \begin{pmatrix} a_0 b_0 \\ a_0 b_1 \\ \vdots \\ a_0 b_{m-1} \\ a_1 b_0 \\ a_1 b_1 \\ \vdots \\ a_1 b_{m-1} \\ \vdots \\ a_{n-1} b_0 \\ a_{n-1} b_1 \\ \vdots \\ a_{n-1} b_{m-1} \end{pmatrix}$$

two-bit states

- We can give the four, two-bit states

$$|00\rangle = |0\rangle \otimes |0\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} |01\rangle = |0\rangle \otimes |1\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

$$|10\rangle = |1\rangle \otimes |0\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} |11\rangle = |1\rangle \otimes |1\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

- In general, the state of n bits is defined by a vector of length 2^n, such that all elements are 0 except a single 1 exactly the number of rows down that corresponds to the decimal expansion of the corresponding bit string

Multiple-bit operations

- To define universal reversible computation, we must be able to define the constructs used previouly
- Namely the swap operation and the control structure
- These can easily be converted into unitary matrices
- We must also be able to compose operations to form larger circuits
- These compositions correspond exactly to operations on the matrices
- Sequential composition is just matrix (pre) multiplication
- Parallel composition is just matrix tensor product
- So, what are the matrices that represent our constructs?
- If we know the truth table for a reversible operation, then it is easy to create a unitary matrix that represents it

From truth table to matrix

- Each row of the matrix corresponds to an input state, and each column corresponds to an output state
- For each input state, we must put a single 1 in the column that represents its output state
- Lets try it for Swap and Controlled-X

$$Swap = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad Controlled - X = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Why model reversible computation like this?

- This may seem like a complicated way of defining reversible computation
- However, this approach extends nicely to a model of quantum computation
- Quantum computation is less restrictive on the states a quantum bit can be in
- We will start looking next week at quantum computation
- It is useful to start thinking of reversible computation in two ways:
  - Syntactically in terms of the circuits we have described
  - Semantically in terms of the underlying vector-space model

Labs and the Quantum IO Monad

- Remember to come to the lab on Thursday!
- We shall be looking at using the Quantum IO Monad to define reversible computations
- The exercise sheet will be online from around 2pm on Thursday, and will contain a detailed introduction to the classical subset of QIO
- I hope to see you there, and will be willing to accept hand-ins of last weeks exercises in order to give you feedback next week
- There is now a link to the module forum on the course webpage