Quantum Programming

Juliana K. Vizzotto[1] **and** **Amr Sabry**[2]

[1] Federal University of Rio Grande do Sul

[2] IU Computer Science Department

8 November 2004

Quantum mechanics

It is a bizarre place . . . normal laws of physics break

> *Real Black Magic Calculus*, Albert Einstein

> *If quantum mechanics hasn't profoundly shocked you, you haven't understood it yet*, Niels Bohr

> *I think I can safely say that nobody today understands quantum mechanics*, Richard Feynman

> *I don't know what I am talking about*, Amr Sabry

The Basics

Bits and qubits

- In a classical computer, a bit is either 0 or 1
- In a quantum computer, a qubit could be both 0 and 1 !

$$q_1 = |0\rangle = 587|0\rangle \qquad \text{same as a bit} = 0$$

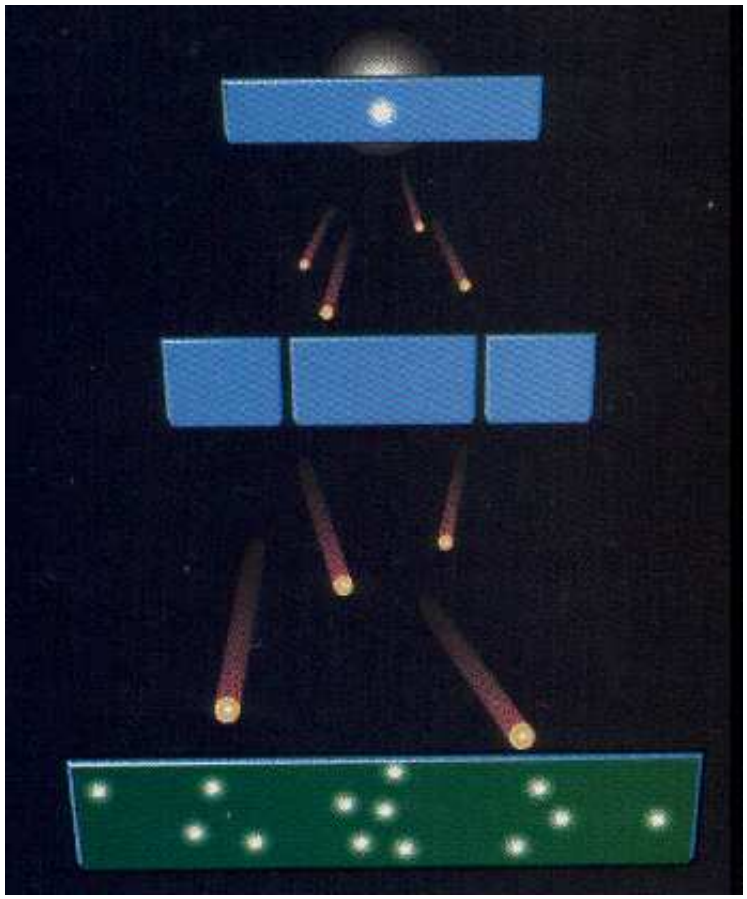
$$q_2 = |1\rangle = 1.2|1\rangle \qquad \text{same as a bit} = 1$$

$$q_2 = \tfrac{1}{\sqrt{2}}|0\rangle + \tfrac{1}{\sqrt{2}}|1\rangle = 8|0\rangle + 8|1\rangle \qquad \text{huh?}$$

$$q_3 = (0.3 + 0.4i)|0\rangle + (0.3 - 0.4i)|1\rangle \qquad \text{HUH?}$$

- As soon as you look at the qubit to find out its value, it becomes either 0 or 1 and never changes after that.

How can a qubit be both true and false?



The double-slit experiment

- Even if we send one photon per second, there is still interference.
- At this rate, a photon cannot be interacting with other photons to produce the interference pattern.
- Each photon must be interacting with itself.
- The math says that the photon must somehow go through both slits at the same time:

$$\frac{1}{\sqrt{2}}|\textit{go to left slit}\rangle + \frac{1}{\sqrt{2}}|\textit{go to right slit}\rangle$$

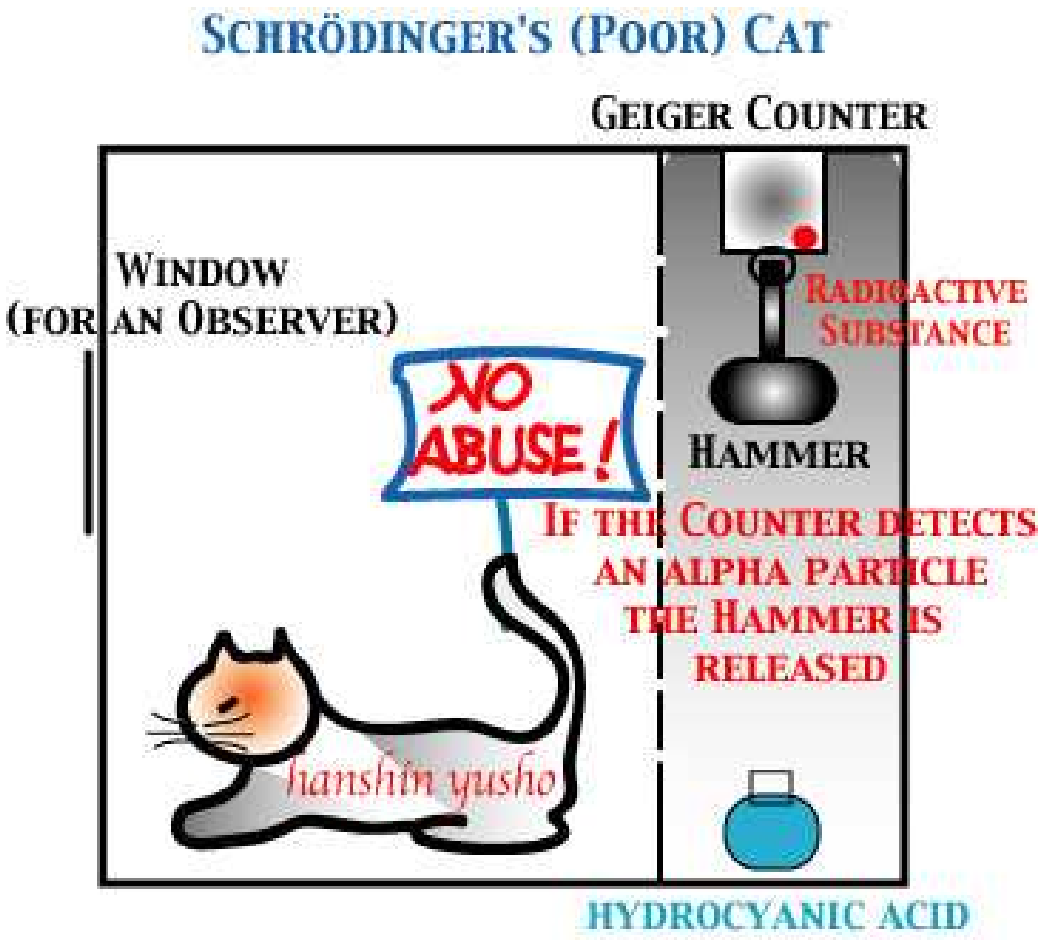
Can we try to find out which way the photon goes?

- We can try ...
- Let's put any measuring device of your choice by one of the slits.
- The interference disappears!
- The measurement changes the behavior of the photons.
- If you look at the quantum value:

$$\frac{1}{\sqrt{2}}|\textit{go to left slit}\rangle + \frac{1}{\sqrt{2}}|\textit{go to right slit}\rangle$$

you either get $|\textit{go to left slit}\rangle$ or $|\textit{go to right slit}\rangle$ and it never changes after that.

Maybe photons are weird. Who cares?



Schrödinger's cat

- After some time the state of one of the radioactive atoms is:

$$\frac{1}{\sqrt{2}}|\text{no decay}\rangle + \frac{1}{\sqrt{2}}|\text{decay}\rangle$$

- Because we are in the state $\frac{1}{\sqrt{2}}|\text{decay}\rangle$, the counter tube discharges, turns the relay on, releases a hammer, shatters the flask of acid, and kills the cat.
- Because we are in the state $\frac{1}{\sqrt{2}}|\text{no decay}\rangle$ the counter tube does not discharge, does not turn the relay on, does not release a hammer, does not shatter the flask of acid, and does not kill the cat.
- Because we are in a superposition of the two states, the state of the cat is:

$$\frac{1}{\sqrt{2}}|\text{alive}\rangle + \frac{1}{\sqrt{2}}|\text{dead}\rangle$$

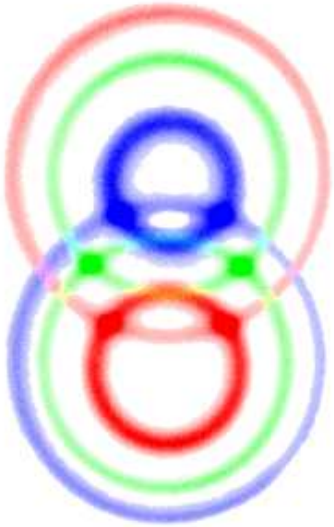
So . . .

if we accept that a small particle can be both true and false, we must also accept that:

- a cat can be both dead and alive;
- our dinner is both here and not here;
- this door is both open and closed;
- etc;

And that's the easy part of quantum mechanics!

Entanglement



- Two particles A and B
- A is both true and false
- B is both true and false
- But they are entangled to always have opposite values:

$$\frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|10\rangle$$

The EPR experiment

- Put particle A on a spaceship going to the Δ-quadrant. Keep particle B in Lindley Hall.
- When particle A reaches the Enterprise in 100 light-years, they look at it and observe that it is, say `false`.
- Instantaneously, particle B becomes `true`.
- Einstein was not happy! This appears to violate special relativity where nothing can propagate faster than light.
- Einstein called this "spooky action at a distance"

Interpretations

All of what we discussed has been verified experimentally. But what does it mean? How does it happen?

- Copenhagen: don't ask; we can observe the world but it is beyond comprehension
- Collapse: Particles act sometimes as waves but when you observe them the wave collapses instantaneously
- Multiple universes: the universe splits all the time; the cat is alive in one universe and dead in another
- Transactional: Messages send back from the future to the past; if you see the cat is dead the message is sent back to the particle to decay
- Many more ideas but just as strange or unhelpful

Programming with this Crazy Model?

Why?

- Understanding Nature

- Hardware is getting smaller and smaller that quantum effects will become visible

- Efficiency: quantum computing can use massive parallelism and entanglement to solve problems faster than any classical computing model.

  The two most famous examples are Shor's algorithm to factor numbers quickly (breaks RSA) and Grover's search algorithm

Deutsch's algorithm

- There are four functions from booleans to booleans:

$$f_1\ F = F \qquad f_2\ F = T$$
$$f_1\ T = F \qquad f_2\ T = T$$

$$f_3\ F = F \qquad f_4\ F = T$$
$$f_3\ T = T \qquad f_4\ T = F$$

- Functions f_1 and f_2 are constant but f_3 and f_4 are balanced.
- Problem: given one of f_1, f_2, f_3, or f_4, is it constant or balanced?
- Classical solution $\lambda f.f\ T == f\ F$ requires two calls to f.
- Quantum solution requires one call to f with an argument that is both true and false: $\frac{1}{\sqrt{2}}|F\rangle + \frac{1}{\sqrt{2}}|T\rangle$
- Doing such tricks in loops changes the complexity of programs

A Low-Level Programming Model

Classical circuits vs. quantum circuits

- wires carry bits
  wires carry qubits
- gates are boolean functions
  gates are reversible boolean functions
- registers (memory) are used to build sequential functions
  impossible to copy a qubit
- observing the value of a bit does not change anything
  observing a qubit collapses its wave function and might affect every other qubit
- it is always possible not to use a bit that we have access to
  not using a qubit is the same as erasing it which is the same as measuring it

Examples of reversible gates

- *Not* (obviously reversible)

$$\begin{aligned} Not(T) &= F \\ Not(F) &= T \end{aligned} \qquad\qquad \begin{aligned} Not(|T\rangle) &= |F\rangle \\ Not(|F\rangle) &= |T\rangle \\ Not(\alpha|F\rangle + \beta|T\rangle) &= \beta|F\rangle + \alpha|T\rangle) \end{aligned}$$

- Hadamard H:

$$H(\alpha|F\rangle + \beta|T\rangle) = (\alpha+\beta)|F\rangle + (\alpha-\beta)|T\rangle$$

  For example: $H(|F\rangle + |T\rangle) = |F\rangle$

- If we look at the coefficients of α and β in the output, we can specify *Not* and H using the matrices:

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad\qquad \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

How about non-reversible classical gates?

- The classic function:

$$\begin{aligned} And(F,F) &= F \\ And(F,T) &= F \\ And(T,F) &= F \\ And(T,T) &= T \end{aligned}$$

  is not reversible

- Trick. Add garbage outputs to make it reversible:

$$\begin{aligned} And\text{-}r(F,F) &= (F,(F,F)) \\ And\text{-}r(F,T) &= (F,(F,T)) \\ And\text{-}r(T,F) &= (F,(T,F)) \\ And\text{-}r(T,T) &= (T,(T,T)) \end{aligned}$$

Controlled gates

For any function f, a controlled-f gate (U_f):

- takes two inputs: a control qubit c and a value qubit v
- if the control qubit is false, the output is the same as the input (c, v)
- if the control qubit is true, the output is $(c, f\ v)$
- Something like:

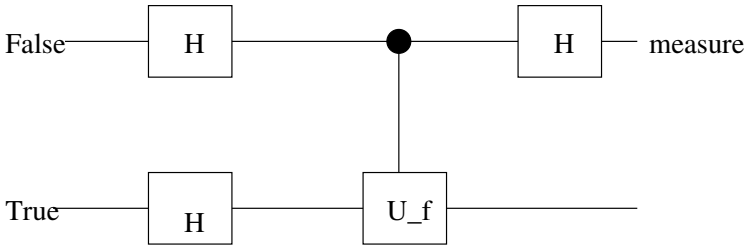
$$U_f(c, v) = \textbf{if } c \textbf{ then } (c, f\ v) \textbf{ else } (c, v)$$

- But the control qubit can be both false and true:

$$U_{Not}(|F\rangle + |T\rangle, |F\rangle) = |FF\rangle + |TT\rangle$$

  In this case, we get an entangled pair of particles.

Deutsch's algorithm



Tracing with the four possible functions, we find:

- If we are given a constant function the top output is $|F\rangle$
- If we are given a balanced function the top output is $|T\rangle$

with one application of f to a superposition of true and false.

Is this a "good" programming model

- It is accurate and correct but it is low-level (at the level of gates) and exposes physical properties of information.
- We usually think of information as abstract but it is in reality a form of energy
- When erasing information (like formatting your hard disk), the energy representing information must dissipate somehow (cf. First law of thermodynamics which is a statement about the conservation of energy)
- If the energy dissipates as entropy (heat) then the process is irreversible
- Reversible functions preserve energy

We don't want to think about information as a form of energy. We want to have an abstract view of computers.

A Compiler

Hide some of the physical properties

The general problem is hard. Let's focus on something simple:

- Quantum circuits are built from reversible functions

- High-level languages should not be restricted to reversible functions

Reversible functions in Scheme

```
(define f1-f
  (lambda (x)
    x))
```

```
(define f1-r
  (lambda (x)
    x))
```

```
(define f2-f
  (lambda (x)
    (+ x 4)))
```

```
(define f2-r
  (lambda (y)
    (- y 4)))
```

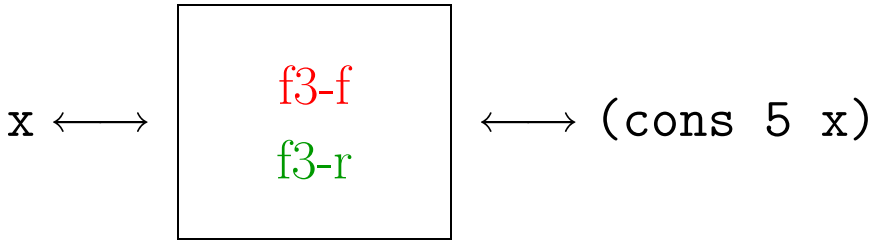
```
(define f3-f
  (lambda (x)
    5))
```

```
(define f3-r
  (lambda (y)
    ???))
```

Adding garbage output

```
(define f3-f
  (lambda (x)
    (cons 5 x)))
```

```
(define f3-r
  (lambda (y)
    (cdr y)))
```



- The compiler can do that. We can write any function we want and the compiler can make it reversible by adding garbage outputs.
- You can run your program forward for a little, then rewind, then continue, etc.

Reversible Computing

- Reversible computing makes sense even in classical (non-quantum) computing. (In principle you can use it to control how much energy your computer consumes)
- Similar to computation done by enzymes. For example, copying the genetic information in DNA genes to an RNA transcript proceeds forwards and backwards. At equilibrium nothing gets copied and no energy is spent. To drive the computation in one way, the concentration of some phosphate reactants is varied.
- DNA computing, biocomputing, etc
- Fascinating subject but not for today ...

A High-Level Programming Model

Many proposals

- qGCL [J. W. Sanders and P. Zuliani, 2000] Extended version of Dijkstra's guarded-command language including three high-level quantum commands: initialization, evolution, and finalization

- QCL [B. Omer, 2001] Procedural quantum programming language

- QPL [P. Selinger, 2003] Functional quantum programming language based on the slogan "quantum data, classical control" control state is always classical

- QML [T. Altenkirch and J. Grattage, 2004] Functional quantum language based on the slogan "quantum data, quantum control"

QML Classical Core

- Basic expressions:

$$\begin{aligned} e \ ::= \ & x \mid \textbf{let } x{=}e_1 \textbf{ in } e_2 \\ & \mid (e_1, e_2) \mid \textbf{let } (x_1, x_2){=}e \textbf{ in } e' \\ & \mid \mathit{qTrue} \mid \mathit{qFalse} \mid \textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3 \end{aligned}$$

- Compiler can take care of making things reversible

- No dynamic data structures (no higher-order procedures, lists, etc): the model of quantum computing usually assumes data structures of fixed size; relaxing this constraint is possible but quite a challenge

QML extension I: superpositions

- Need a syntax for values that are both true and false
- $\{(\alpha)e_1 \mid (\beta)e_2\}$ are α and β are any expressions that evaluate to complex numbers
- But e_1 and e_2 cannot be arbitrary expressions. They must be orthogonal
- Examples:

$$\{qTrue \mid qFalse\}$$
$$\{(qTrue, qTrue) \mid (qFalse, qFalse)\}$$
$$\{\{qFalse \mid (-1)qTrue\} \mid \{qFalse \mid qTrue\}\}$$

- Non-example $\{qTrue \mid qTrue\}$

QML extension II: quantum control operators

- Given a value v which corresponds to a superposition like: $\{qTrue \mid qFalse\}$
- The expression **qif** v **then** e_1 **else** e_2 produces $\{e_1 \mid e_2\}$ as follows: v is partly $qTrue$ so the true-branch is evaluated; v is partly $qFalse$ so the false-branch is evaluated; the results of both branches are combined using the appropriate probability amplitudes.
- **if** v **then** e_1 **else** e_2 is very different from **qif** v **then** e_1 **else** e_2
- **if** v **then** e_1 **else** e_2 is a classical control operator which operates on classical values: it first measures v to produces either true or false, and then selects the appropriate branch.

Examples

- $Not(b) = \textbf{qif}\ b\ \textbf{then}\ qFalse\ \textbf{else}\ qTrue$
- $Z(b) = \textbf{qif}\ b\ \textbf{then}\ (-1)qTrue\ \textbf{else}\ qFalse$
- Hadamard:

  $H(x) = \textbf{qif}\ x\ \textbf{then}\ \{qFalse \mid (-1)qTrue\}\ \textbf{else}\ \{qFalse \mid qTrue\}$

- Controlled not:

  $U_{Not}(p) = \textbf{let}\ (x, y){=}p\ \textbf{in}\ \textbf{qif}\ x\ \textbf{then}\ (x, Not(y))\ \textbf{else}\ (x, y)$

Ignoring values

- What if we had written:

$$F(p) = \textbf{let } (x, y) = p \textbf{ in } x$$

- What happens to y ? In a classical language, y becomes garbage. In a quantum language, this is a measurement.

- Ok so we measure y and throw away the result

- But x and y may be entangled; measuring y might change the value of x

Explicit control of variables

- $F_1(y) =$ **let** $x{=}y$ **in** x is the identity function
- $F_2(y) =$ **let** $x{=}y$ **in let** $z{=}y$ **in** x measures z (and hence y)
- For example $F_1(\{qFalse \mid qTrue\}) = (\{qFalse \mid qTrue\})$
- But
  $F_2(\{qFalse \mid qTrue\}) = qFalse$ or
  $F_2(\{qFalse \mid qTrue\}) = qTrue$ with equal probability.
- Can I measure y without having to introduce a new name z and then ignore it?
- New syntax for explicit measurement:

$$F_2(y) = \textbf{let } x{=}y \textbf{ in } x^{\{y\}}$$

Examples

- Swapping two (possibly entangled) quantum values:

$$S(p) = \textbf{let } (x, y) = p \textbf{ in } (y^{\{\}}, x^{\{\}})$$

- Measure the two qubits, and then swap the resulting classical values:

$$MS(p) = \textbf{let } (x, y) = p \textbf{ in } (y^{\{p\}}, x^{\{p\}})$$

- Need a type system to keep track of all that.

Type system

- Keep track of all variables in scope
- If variables x, y, and z are in scope and we want to type (e_1, e_2), we can pass all three variables to both e_1 and e_2. If e_1 only uses x then y and z will be measured.
- Alternatively we could pass only x to e_1 and pass y and z to e_2 and perhaps avoid measurements not explicitly given by the programmer.

Summary of programmer's model

- Classical functional language (with no dynamic data-structures)
- Superpositions
- Quantum control operators
- Explicit measurement and specialized type system

Example: Teleportation of qubit q

let epr=$\{(qFalse, qFalse) \mid (qTrue, qTrue)\}$ **in**
let (x, y)=epr **in**
let msg=$alice(q, x)$ **in**
$bob(msg, y)$

- Alice and Bob each have one of two entangled qubits
- Alice has a qubit q that she wants to "teleport" to Bob
- Alice cannot measure q as this will destroy it
- Alice does a small computation using q and her half of the entangled pair and then sends a message with classical bits to Bob
- Bob uses the message and his half of the entangled pair to reconstitute q which "apparates" at his site.

Example: Teleportating qubit q

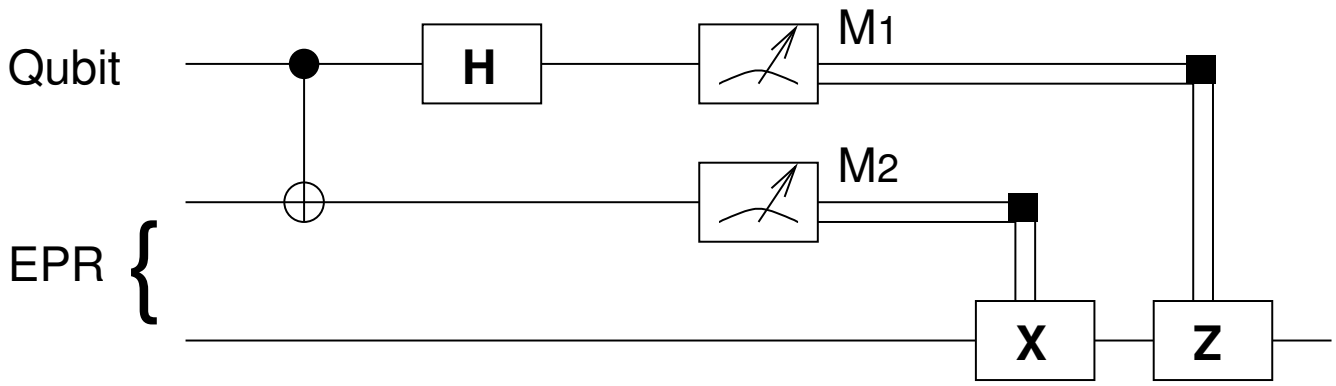
Code for Alice:

$alice(q, x) =$
  let $(a, b) = U_{Not}(x, q)$ **in**
  let $c = H(a)$ **in**
  $(c^{\{b,c\}}, b^{\{b,c\}})$

Code for Bob:

$bob((m_1, m_2), y) =$
  if m_1
  then if m_2 **then** $Z(Not(y))$ **else** $Z(y)$
  else if m_2 **then** $Not(y)$ **else** y

or using circuits



Conclusion

We have a programming model.

Where is the computer?

- Quantum computers exist today!
- They can find the prime factors of a given number
- But only if the input number is less than fifteen
- The λ-calculus (the foundation of functional languages) was discovered at least twenty years before electronic computers.