Quantum Effects

Juliana K. Vizzotto\textsuperscript{1}  Thorsten Altenkirch\textsuperscript{2}

Amr Sabry\textsuperscript{3}

\textsuperscript{1} Federal University of Rio Grande do Sul
\textsuperscript{2} The University of Nottingham
\textsuperscript{3} Indiana University

20 January 2005
Quantum mechanics

*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 can’t possibly know what I am talking about* — Amr Sabry
Models of (quantum) computation

Abstract (Compositional) — Values and functions

Circuits — Vectors and matrices

Physics — Particle spins and electromagnetic fields
Abstract models of (quantum) computation

Semantic foundation for functional quantum programming language:

- Category theory — categorical models of quantum computation [Abramsky, Selinger, van Tonder]
- $\lambda$-calculus — quantum $\lambda$-calculus [Van Tonder]
- Domain theory, logic, etc — [Birkhoff, von Neumann]
- Haskell (not perfect but rich executable language)
Plan

- Quantum computation (I): unitary operations on state vectors
- Embedding in Haskell using monads
- What to do with measurement?

- Quantum computation (II): superoperators on density matrices
- Arrows
- Embedding in Haskell using arrows

- QML [Altenkirch and Grattage]
- Open problems; related work; conclusions
Quantum Computing (I)
Example: Toffoli circuit

- In this example, input is $|TTF\rangle$
- After first step, state vector is a superposition $|TTF\rangle + |TTT\rangle$
- Result: Negate the last bit
Entanglement

- The state vector of multiple qubits can sometimes be teased into the product of simpler state vectors:

\[ |FF\rangle + |FT\rangle = |F\rangle * (|F\rangle + |T\rangle) \]

- If the qubits are entangled, this is impossible:

\[ |FF\rangle + |TT\rangle \neq (|F\rangle + |T\rangle) * (|F\rangle + |T\rangle) \]

  (or any other product we might try)

- Must basically manipulate the global state at all times

  even if we want to apply an operation to only one qubit
QCL [Knill]

- A global state with $n$ qubits
- Registers are realized using pointers to the global state
- Apply operation $U$ to register $r$ using $\Pi^\dagger.(U \times I_{(n-m)})\Pi$
  where $\Pi^\dagger$ is the inverse of $\Pi$ and $I$ is the identity
Flowchart notation [Selinger]

- A global state with \( n \) variables that can be assigned once

- To apply operation \( U \) to part of the state, use the same idea that is used in QCL:
  
  - re-order the variables to bring relevant variables to the front
  
  - compose \( U \) with the identity and apply it to the entire state
Lambda-calculus extension [Valiron and Selinger]

Idea: the lambda term gives classical control over the quantum data which is accessed via pointers to a global data structure.

- The state is a triple $[Q, Q_f, M]$
- $Q$ is the state vector
- $M$ is a lambda term with free variables
- $Q_f$ is a linking function which maps every free variable of $M$ to a qubit in $Q$
Virtual values and adaptors [Sabry]

- Also uses a global state and pointers mediated using adaptors
- Hides the management of pointers using virtual values
- Adaptors can almost be derived from the types but their actual generation is tedious and ugly

```
toffoli state =
  let b  = virt state adaptor_0
  mb = virt state adaptor_1
  tm = virt state adaptor_2
  tb = virt state adaptor_3
  in do app hadamard b
       app cv mb
       app cnot tm
       app cvt mb
       app cnot tm
       app cv tb
       app hadamard b
```
Can we do better than pointers to a global state?

Common theme so far:

- A global state vector accessed via pointers
- Each operation transforms the global state to a new state
- **Monads** are often used to structure and reason about computational effects.

```
class Monad m where
    return :: \( a \rightarrow m a \)

    (\_\_\_\_\_\_) :: \( a \rightarrow m a \rightarrow (a \rightarrow m b) \rightarrow m b \)
```

- Is there a nice **monad** here?
Embedding in Haskell using monads
Finite sets

- We only consider computations over finite bases. For a type \( a \) to be a type of observables, it needs to represent a finite set:

\[
\text{class } Eq \; a \Rightarrow \; Basis \; a \quad \text{where} \quad \text{basis} :: [a]
\]

\[
\text{instance } Basis \; \text{Bool} \quad \text{where} \quad \text{basis} = [\text{False}, \text{True}]
\]

- Can automatically construct more complicated sets (on demand):

\[
\text{instance } (\text{Basis} \; a, \text{Basis} \; b) \Rightarrow \text{Basis}(a, b) \quad \text{where}
\]

\[
\text{basis} = [((a, b) \mid a \leftarrow \text{basis}, \; b \leftarrow \text{basis}]
\]

- Programs at the end produce classical observable values: \( \text{False}, (\text{True}, \text{False}) \), etc
State vectors

- A state vector associates a complex probability amplitude with each basis element:

\[
\text{type} \ PA = \text{Complex Double}
\]
\[
\text{type} \ Vec a = a \rightarrow PA
\]

- Can add, subtract, and multiply vectors: (definitions omitted)

\[
\text{vzero} :: Vec a
\]
\[
(+): Vec a \rightarrow Vec a \rightarrow Vec a
\]
\[
(-): Vec a \rightarrow Vec a \rightarrow Vec a
\]
\[
(*): PA \rightarrow Vec a \rightarrow Vec a
\]
\[
(\cdot): \text{Basis } a \Rightarrow Vec a \rightarrow Vec a \rightarrow PA
\]
Examples of vectors over $\text{Bool}$

- The simplest vector is a unit representing a basis element:

\[
\text{unit} :: \text{Basis } a \Rightarrow a \rightarrow \text{Vec } a \\
\text{unit } a = (\forall b \rightarrow \text{if } a = b \text{ then } 1 \text{ else } 0)
\]

- The two basic unit vectors:

\[
\text{qFalse} = \text{unit } \text{False} \quad \text{— in Dirac notation } |0\rangle \\
\text{qTrue} = \text{unit } \text{True} \quad \text{— in Dirac notation } |1\rangle
\]

- Vector representing superpositions:

\[
\text{qFT} = (1 / \sqrt{2}) \star (\text{qFalse } \langle + \rangle \text{ qTrue}) \quad \text{—} \quad \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\
\text{qFmT} = (1 / \sqrt{2}) \star (\text{qFalse } \langle - \rangle \text{ qTrue}) \quad \text{—} \quad \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)
\]
Examples of vectors over \((\text{Bool, Bool})\)

- Using the tensor product:

\[
p_1 = qFalse \langle \ast \rangle qFT \quad \text{— in Dirac notation} \quad |0\rangle \ast \left(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right)
\]

- Using the classical product over the basis:

\[
qFF = \text{unit} (False, False) \quad \text{— in Dirac notation} \quad |00\rangle
qTT = \text{unit} (True, True) \quad \text{— in Dirac notation} \quad |11\rangle
\]

- A vector representing the EPR pair \(\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)\):

\[
epr = (1 / \sqrt{2}) \ast (qFF \langle + \rangle qTT)
\]
Linear operators

- Given a function \( f : a \rightarrow Vec\ b \), we can produce the linear operator of type \( Vec\ a \rightarrow Vec\ b \)

- Apply \( f \) to each basis element and accumulate the results:

\[
\text{linop} \quad :: \quad Basis\ a \Rightarrow (a \rightarrow Vec\ b) \rightarrow (Vec\ a \rightarrow Vec\ b)
\]
\[
\text{linop} f\ va = (\forall\ b \rightarrow \sum [(va\ a) \ast (f\ a\ b) | a \leftarrow basis])
\]

- So we can define:

\[
\text{type Lin\ a\ b} = a \rightarrow Vec\ b
\]
Examples of linear operators

- Construct a linear operator from any pure function:

\[
\begin{align*}
\text{fun2lin} &: (\text{Basis } a, \text{Basis } b) \Rightarrow (a \rightarrow b) \rightarrow \text{Lin } a \ b \\
\text{fun2lin } f \ a &= \text{unit } (f \ a)
\end{align*}
\]

- Common linear operators on booleans:

\[
\begin{align*}
\text{qnot} &= \text{fun2lin } \text{not} \\
\text{hadamard } \text{False} &= \text{qFT} \\
\text{hadamard } \text{True} &= \text{qFmT}
\end{align*}
\]
More linear operations

- Outer product:

\[
\langle \rangle \cdot \langle \rangle :: \text{Basis } a \Rightarrow \text{Vec } a \rightarrow \text{Vec } a \rightarrow \text{Lin } a \ a \\
\langle v_1 \rangle \cdot \langle v_2 \rangle \ a_1 \ a_2 = v_1 \ a_1 \ast \text{conjugate}(v_2 \ a_2)
\]

- Composition:

\[
o :: (\text{Basis } a, \text{Basis } b, \text{Basis } c) \Rightarrow \\
\text{Lin } a \ b \rightarrow \text{Lin } b \ c \rightarrow \text{Lin } a \ c \\
o \ f \ g \ a = \text{linop } g(f \ a)
\]

- Controlled-operations:

\[
\text{controlled} :: \text{Basis } a \Rightarrow \\
\text{Lin } a \ a \rightarrow \text{Lin} (\text{Bool}, \ a)(\text{Bool}, \ a) \\
\text{controlled } f(b_1, \ b_2) = \langle \text{unit } b_1 \rangle (\ast)(\text{if } b_1 \text{ then } f \ b_2 \text{ else } \text{unit } b_2)
\]
Almost a monad!

- We can define:

\[
\begin{align*}
\text{return} & \quad :: \quad \text{Basis } a \Rightarrow a \rightarrow \text{Vec } a \\
\text{return} & \quad = \quad \text{unit} \\
(\ggg) & \quad :: \quad \text{Basis } a \Rightarrow \text{Vec } a \rightarrow (a \rightarrow \text{Vec } b) \rightarrow \text{Vec } b \\
(va \ggg f) & \quad = \quad \text{linop } f \ va
\end{align*}
\]

- The right equations are satisfied

- The types are wrong: the extra constraints mean that the construction is not universal. In Haskell terms, we cannot use the \textbf{do}-notation

- Already observed [Mu and Bird, 2001] but in a system restricted to manipulating lists of qubits
Toffoli circuit

\[
\text{toffoli :: Lin (Bool, Bool, Bool) (Bool, Bool, Bool)} \\
\text{toffoli (top, middle, bottom) =} \\
\text{let cnot = controlled qnot} \\
\text{cphase = controlled phase} \\
\text{in hadamard bottom} \\
\text{cphase (middle, \(b_1\))} \\
\text{cnot (top, \(m_1\))} \\
\text{controlled (adjoint phase) (\(m_2\), \(b_2\))} \\
\text{cnot (\(t_1\), \(m_3\))} \\
\text{cphase (\(t_2\), \(b_3\))} \\
\text{hadamard \(b_4\)} \\
\text{return (\(t_3\), \(m_4\), \(b_5\))}
\]
So far

- (Almost) monads can be used to structure quantum parallelism: no explicit global state and pointers

- Connections to category theory, etc

- That’s the easy part . . . How do we deal with measurement?
Measurement
Measurement & collapse

Measuring $qFT \ (= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle))$:

- returns 0 with probability 1/2
  and *as a side-effect* collapses $qFT$ to $|0\rangle$, or

- returns 1 with probability 1/2
  and *as a side-effect* collapses $qFT$ to $|1\rangle$
Measurement & spooky action at a distance

Measuring the left qubit of $epr$: ($= \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$):

- returns 0 with probability $1/2$
  and as a side-effect collapses $epr$ to $|00\rangle$, or

- returns 1 with probability $1/2$
  and as a side-effect collapses $epr$ to $|11\rangle$

- The right qubit is affected even if physically distant!
Ignore measurement?

• Measurements can always be delayed to the end; many formalisms ignore them

• Mu and Bird use the IO monad to explain measurement; cannot mix measurement with linear operations

• Can we deal with measurements in the formalism?
Teleportation

Communication uses a \textit{classical} channel, sending classical bits.
Quantum Computing (II)
State vectors have too much information

- Perhaps vectors are not expressive enough?
- Vector is exact state of the system but much of the information in the state is not observable
- Take \( qFT \) and measure it. The result is either:
  \[
  \frac{1}{\sqrt{2}} |0\rangle \quad \text{or} \quad \frac{1}{\sqrt{2}} |1\rangle
  \]
- Apply Hadamard to the result:
  \[
  \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \quad \text{or} \quad \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)
  \]
- The two configurations are indistinguishable (observationally equivalent)
Density matrices

- Statistical perspective of the state vector
- Technically, we use the outer product:

\[
\text{type } \text{Dens } a = \text{Vec } (a, a)
\]

\[
\text{pureD} :: \text{Basis } a \Rightarrow \text{Vec } a \rightarrow \text{Dens } a
\]

\[
\text{pureD } v = \text{lin2vec } (v \rangle \langle v)
\]

- Examples:

<table>
<thead>
<tr>
<th>$qFalse$</th>
<th>$qTrue$</th>
<th>$qFT$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>0\rangle$</td>
<td>$</td>
</tr>
</tbody>
</table>

\[
\begin{pmatrix}
1 & 0 \\
0 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 0 \\
0 & 1
\end{pmatrix}
\begin{pmatrix}
1/2 & 1/2 \\
1/2 & 1/2
\end{pmatrix}
\]

Yale CS Colloquium
Density matrices and measurement

• When we measure $qFT \left( \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \right)$ we get:
  *False* with probability $1/2$, or *True* with probability $1/2$:

\[
\begin{pmatrix}
0.5 & 0 \\
0 & 0 \\
0 & 0
\end{pmatrix} + \begin{pmatrix}
0 & 0 \\
0 & 0.5
\end{pmatrix} = \begin{pmatrix}
0.5 & 0 \\
0 & 0.5
\end{pmatrix}
\]

• The density matrix can represent a *mixed state*

• Operations are linear:

\[
H \begin{pmatrix}
0.5 & 0 \\
0 & 0.5
\end{pmatrix} = H \begin{pmatrix}
0.5 & 0 \\
0 & 0
\end{pmatrix} + H \begin{pmatrix}
0 & 0 \\
0 & 0.5
\end{pmatrix} = \begin{pmatrix}
0.5 & 0 \\
0 & 0.5
\end{pmatrix}
\]

• The two states are indeed *observationally equivalent*. 
Superoperators

- Every linear operator can be lifted to an operator on density matrices

- Such operators are called superoperators:

```haskell
type Super a b = (a, a) -> Dens b

lin2super :: (Basis a, Basis b) -> Lin a b -> Super a b
lin2super f (a1, a2) = (f a1) \langle * \rangle (dual (adjoint f) a2)
  where dual f a b = f b a
```
Tracing and measurement

- $trL$ measures and "forgets" the result of measurement $meas$ measures and returns the result of measurement

$trL : : (\text{Basis } a, \text{Basis } b) \Rightarrow \text{Super } (a, b) \ b$

$trL ((a_1, b_1), (a_2, b_2)) = \text{if } a_1 = a_2 \text{ then return } (b_1, b_2) \text{ else } vzero$

$meas : : \text{Basis } a \Rightarrow \text{Super } a (a, a)$

$meas (a_1, a_2) = \text{if } a_1 = a_2 \text{ then return } ((a_1, a_1), (a_1, a_1)) \text{ else } vzero$

- Measuring $qFT$ and forgetting the collapsed quantum state:

$$\text{pureD } qFT \gg meas \gg trL$$

evaluates to:

$$\begin{pmatrix} 1/2 & 0 \\ 0 & 1/2 \end{pmatrix}$$
No longer a monad

• At least we can’t prove it is a monad

• Superoperators do not form a basis

• We seem to have lost all our structure
Arrows
A generalization of monads

\[
\begin{align*}
\text{class } &\text{Arrow } a \text{ where} \\
\text{arr} &:: (b \to c) \to a \ b \ c \\
(\ggg) &:: a \ b \ c \to a \ c \ d \to a \ b \ d \\
\text{first} &:: a \ b \ c \to a \ (b, d) \ (c, d)
\end{align*}
\]

(a) \hspace{2cm} (b) \hspace{2cm} (c)
More about arrows

Look up excellent work at Yale
Embedding in Haskell using arrows
Superoperators are arrows

Well ... almost: the types have additional constraints

\[
\text{arr} :: \text{(Basis } b, \text{ Basis } c) \Rightarrow (b \rightarrow c) \rightarrow \text{Super } b \ c \\
\text{arr} \ f = \text{fun2lin} (\ \backslash (b_1, b_2) \rightarrow (f \ b_1, \ f \ b_2)) \\
\]

\[
(\Rightarrow) :: \text{(Basis } b, \text{ Basis } c, \text{ Basis } d) \Rightarrow \\
\text{Super } b \ c \rightarrow \text{Super } c \ d \rightarrow \text{Super } b \ d \\
(\Rightarrow) = o \\
\]

\[
\text{first} :: \text{(Basis } b, \text{ Basis } c, \text{ Basis } d) \Rightarrow \text{Super } b \ c \rightarrow \text{Super } (b, d) (c, d) \\
\text{first} \ f ((b_1, d_1), (b_2, d_2)) = \text{permute} ((f \ b_1, b_2)) (\star) (\text{return} (d_1, d_2)) \\
\text{where} \ \text{permute} \ v ((b_1, b_2), (d_2, d_2)) = v ((b_1, d_1), (b_2, d_2))
\]

[72x248]Superoperators are arrows

Well ... almost: the types have additional constraints

\[
\text{arr} :: \text{(Basis } b, \text{ Basis } c) \Rightarrow (b \rightarrow c) \rightarrow \text{Super } b \ c \\
\text{arr} \ f = \text{fun2lin} (\ \backslash (b_1, b_2) \rightarrow (f \ b_1, \ f \ b_2)) \\
\]

\[
(\Rightarrow) :: \text{(Basis } b, \text{ Basis } c, \text{ Basis } d) \Rightarrow \\
\text{Super } b \ c \rightarrow \text{Super } c \ d \rightarrow \text{Super } b \ d \\
(\Rightarrow) = o \\
\]

\[
\text{first} :: \text{(Basis } b, \text{ Basis } c, \text{ Basis } d) \Rightarrow \text{Super } b \ c \rightarrow \text{Super } (b, d) (c, d) \\
\text{first} \ f ((b_1, d_1), (b_2, d_2)) = \text{permute} ((f \ b_1, b_2)) (\star) (\text{return} (d_1, d_2)) \\
\text{where} \ \text{permute} \ v ((b_1, b_2), (d_2, d_2)) = v ((b_1, d_1), (b_2, d_2))
\]
Superoperators as a model of quantum computing

- The category of superoperators is known to be an adequate model of quantum computation [Selinger]

- This work suggests that this category corresponds to a functional language with arrows

- Can we accurately express quantum computation in a functional language with arrows?
Toffoli

\(\text{toffoli} :: \text{Super (Bool, Bool, Bool)(Bool, Bool, Bool)}\)

\[
\begin{align*}
to\text{ffoli} & = \text{let hadS} = \text{lin2super hadamard} \\
cnotS & = \text{lin2super (controlled qnot)} \\
c\text{phaseS} & = \text{lin2super (controlled phase)} \\
c\text{aphaseS} & = \text{lin2super (controlled (adjoint phase))}
\end{align*}
\]

in proc \((a_0, b_0, c_0) \rightarrow \text{do} \)

\[
\begin{align*}
c_1 & \leftarrow \text{hadS} \prec c_0 \\
(b_1, c_2) & \leftarrow \text{cphaseS} \prec (b_0, c_1) \\
(a_1, b_2) & \leftarrow \text{cnotS} \prec (a_0, b_1) \\
(b_3, c_3) & \leftarrow \text{caphaseS} \prec (b_2, c_2) \\
(a_2, b_4) & \leftarrow \text{cnotS} \prec (a_1, b_3) \\
(a_3, c_4) & \leftarrow \text{cphaseS} \prec (a_2, c_3) \\
c_5 & \leftarrow \text{hadS} \prec c_4 \\
\text{returnA} & \prec (a_3, b_4, c_5)
\end{align*}
\]
Teleportation (I)

- Can write, type, reason about each component separately.

- Can incorporate measurement in the computation

- Main:

```plaintext
teleport :: Super (Bool, Bool, Bool) Bool
teleport = proc (eprL, eprR, q) → do
           (m₁, m₂) ← alice < (eprL, q)
           q' ← bob < (eprR, m₁, m₂)
           returnA < q'
```

Yale CS Colloquium
Teleportation (II)

\[
\begin{align*}
\text{alice} & : \text{Super} \ (\text{Bool}, \text{Bool}) \ (\text{Bool}, \text{Bool}) \\
\text{alice} & = \textbf{proc} \ (\text{eprL}, q) \rightarrow \textbf{do} \\
& \quad (q_1, e_1) \leftarrow \text{lin2super} \ (\text{controlled qnot}) \prec (q, \text{eprL}) \\
& \quad q_2 \leftarrow \text{lin2super hadamard} \prec q_1 \\
& \quad ((q_3, e_2), (m_1, m_2)) \leftarrow \text{meas} \prec (q_2, e_1) \\
& \quad (m'_1, m'_2) \leftarrow \text{trL} \ ((q_3, e_2), (m_1, m_2)) \\
& \quad \text{returnA} \prec (m'_1, m'_2) \\
\text{bob} & : \text{Super} \ (\text{Bool}, \text{Bool}, \text{Bool}) \ \text{Bool} \\
\text{bob} & = \textbf{proc} \ (\text{eprR}, m_1, m_2) \rightarrow \textbf{do} \\
& \quad (m'_1, e_1) \leftarrow \text{lin2super} \ (\text{controlled qnot}) \prec (m_2, \text{eprR}) \\
& \quad (m'_1, e_2) \leftarrow \text{lin2super} \ (\text{controlled z}) \prec (m_1, e_1) \\
& \quad q' \leftarrow \text{trL} \prec ((m'_1, m'_2), e_2) \\
& \quad \text{returnA} \prec q'
\end{align*}
\]
QML
Why Haskell is not adequate

- There is more to quantum computation than a functional language with arrows
- Cloning?

\[
\delta :: Super \, \text{Bool} \, (\text{Bool}, \text{Bool}) \\
\delta = \text{arr} (\lambda x \to (x, x))
\]

- Weakening

\[
\text{weaken} :: Super \, (\text{Bool}, \text{Bool}) \, \text{Bool} \\
\text{weaken} = \text{arr} (\lambda (x, y) \to y)
\]
Cloning

- Well-known “non-cloning” property of quantum states!

\[
\delta :: Super \ Bool (Bool, \ Bool) \\
\delta = arr (\ x \rightarrow (x, x))
\]

- \(\delta\) only clones classical information encoded in quantum data

- Applying \(\delta\) to \(qFalse\) will give \(qFalse \langle * \rangle qFalse\)

- But applying \(\delta\) to \(qFT\) does not produce \(qFT \langle * \rangle qFT\); rather it produces \(epr\)

- One can think of it as cloning a pointer; and sharing the quantum data.
Weakening

- The definition \textit{weaken} allows us to drop some values

\[
\text{weaken :: Super (Bool, Bool) Bool} \\
\text{weaken = arr } \lambda (x, y) \rightarrow y
\]

- Applying \textit{weaken} to \textit{epr} gives \textit{qFT}

- But dropping a value amounts to measuring it, and if we measure the left qubit of \textit{epr}, we should be getting either \textit{qFalse} or \textit{qTrue} or the mixed state of both measurements, but never \textit{qFT}.

- \textbf{Must prevent weakening}
QML [Altenkirch and Grattage]

- A functional language to model quantum computation

- Prevent weakening using strict linear logic

- Two semantics: translation to quantum circuits and translation to superoperators

- Source language models irreversible computations; semantics (compiler) takes care of making everything reversible (by adding a heap input and a garbage output)
QML type system

• Must keep track of uses of variables

• Variables in the context that are not used must be measured:

\[ x: \sigma, y: \tau \vdash x\{y\}: \sigma \]

• Variables in the context can be used more than once (by essentially applying \( \delta \))
Using variables

• Does $f$ use $x$?

\[
 f \ x = \ \text{if} \ x \ \text{then} \ q\text{True} \ \text{else} \ q\text{True}
\]

• Depends on the semantics of \texttt{if}

• Classical control: measure the qubit $x$ to get a classical boolean value and then select appropriate branch

• Quantum control ($\texttt{if}^\circ$): return the superposition of the branches

• Quantum control returns $q\text{True}$ without using $x$

• The version with $\texttt{if}^\circ$ must not be allowed to typecheck
Quantum control & orthogonality

- In an if° expression, the superposition of $e_1$ and $e_2$ is calculated using the probability amplitudes of the superposition of $x$:

$$\text{if}° x \text{ then } e_1 \text{ else } e_2$$

- Basically if $e_1$ and $e_2$ are orthogonal then it is safe to replace the superposition in $x$ with the superposition of $e_1$ and $e_2$.

- This is ok:

$$f° x = \text{if}° x \text{ then } qTrue \text{ else } qFalse$$
Conclusions

• Fairly elegant semantic analysis

• Quantum computing =
  functional language +
  arrows (for parallelism and measurement) +
  some kind of linear type system (to control weakening)

• Formalize the connections between QML and a functional languages with superoperators as arrows

• Still need to explain a few open issues

• Higher-order programs, infinite datatypes, etc