



QML: A functional quantum programming language

quantum control and orthogonality

Jonathan Grattage

with Thorsten Altenkirch & Alex Green

sneezy.cs.nott.ac.uk/qml

School of Computer Science & IT,
The University of Nottingham



What is QML?

- A high-level quantum programming language with a structure familiar to functional programmers, which supports reasoning and algorithm design



What is QML?

- A high-level quantum programming language with a structure familiar to functional programmers, which supports reasoning and algorithm design
- Simplifying the design of quantum programs by:
 - Allowing formal reasoning principles for quantum programs
 - Giving a more intuitive understanding of quantum algorithms



What is QML?

- A high-level quantum programming language with a structure familiar to functional programmers, which supports reasoning and algorithm design
- Simplifying the design of quantum programs by:
 - Allowing formal reasoning principles for quantum programs
 - Giving a more intuitive understanding of quantum algorithms
- *A functional quantum programming language*, LICS 2005 with Thorsten Altenkirch



What is QML?

- A high-level quantum programming language with a structure familiar to functional programmers, which supports reasoning and algorithm design
- Simplifying the design of quantum programs by:
 - Allowing formal reasoning principles for quantum programs
 - Giving a more intuitive understanding of quantum algorithms
- *A functional quantum programming language*, LICS 2005 with Thorsten Altenkirch
- *An Algebra of Pure Quantum Programming*, QPL 2005 with Thorsten Altenkirch, Juliana K. Vizzotto, & Amr Sabry



What is QML?

- A high-level quantum programming language with a structure familiar to functional programmers, which supports reasoning and algorithm design
- Simplifying the design of quantum programs by:
 - Allowing formal reasoning principles for quantum programs
 - Giving a more intuitive understanding of quantum algorithms
- *A functional quantum programming language*, LICS 2005 with Thorsten Altenkirch
- *An Algebra of Pure Quantum Programming*, QPL 2005 with Thorsten Altenkirch, Juliana K. Vizzotto, & Amr Sabry
- Project Site: QML@CS.Nott – sneezy.cs.nott.ac.uk/qml



Quantum Languages

- P. Zuliani, PhD 2001, *Quantum Programming* (qGCL)
- P. Selinger, MSCS 2003, *Towards a Quantum Programming Language* (QPL)
- A. van Tonder, SIAM 2003, *A Lambda Calculus for Quantum Computation*
- A. Sabry, Haskell 2003, *Modeling quantum computing in Haskell*
- P. Selinger and B. Valiron, TLCA 2005, *A lambda calculus for quantum computation with classical control*
- ...



Quantum Languages

- P. Zuliani, PhD 2001, *Quantum Programming* (qGCL)
- P. Selinger, MSCS 2003, *Towards a Quantum Programming Language* (QPL)
- A. van Tonder, SIAM 2003, *A Lambda Calculus for Quantum Computation*
- A. Sabry, Haskell 2003, *Modeling quantum computing in Haskell*
- P. Selinger and B. Valiron, TLCA 2005, *A lambda calculus for quantum computation with classical control*
- ...
- *Quantum data, Classical control*

QML Overview



- A first-order functional language for quantum computations on finite types



QML Overview

- A first-order functional language for quantum computations on finite types
- “Quantum Data *and* Control”



QML Overview

- A first-order functional language for quantum computations on finite types
- “Quantum Data *and* Control”
- Based on strict linear logic - controlled, explicit, weakening



QML Overview

- A first-order functional language for quantum computations on finite types
- “Quantum Data *and* Control”
- Based on strict linear logic - controlled, explicit, weakening
- Operational Semantics: Quantum Circuit Model



QML Overview

- A first-order functional language for quantum computations on finite types
- “Quantum Data *and* Control”
- Based on strict linear logic - controlled, explicit, weakening
- Operational Semantics: Quantum Circuit Model
- Denotational Semantics: Superoperators



QML Overview

- A first-order functional language for quantum computations on finite types
- “Quantum Data *and* Control”
- Based on strict linear logic - controlled, explicit, weakening
- Operational Semantics: Quantum Circuit Model
- Denotational Semantics: Superoperators

- Notion of Finite Quantum Computations (FQC) developed by analogy with Finite Classical Computations (FCC)

Classical v. Quantum



The University of
Nottingham



Classical v. Quantum

Classical Case (FCC)

Quantum Case (FQC)



Classical v. Quantum

Classical Case (FCC)	Quantum Case (FQC)
Finite sets	



Classical v. Quantum

Classical Case (FCC)	Quantum Case (FQC)
Finite sets	Finite dimensional Hilbert spaces



Classical v. Quantum

Classical Case (FCC)	Quantum Case (FQC)
Finite sets	Finite dimensional Hilbert spaces
Cartesian product (\times)	



Classical v. Quantum

Classical Case (FCC)	Quantum Case (FQC)
Finite sets	Finite dimensional Hilbert spaces
Cartesian product (\times)	Tensor product (\otimes)



Classical v. Quantum

Classical Case (FCC)	Quantum Case (FQC)
Finite sets	Finite dimensional Hilbert spaces
Cartesian product (\times)	Tensor product (\otimes)
Bijections	



Classical v. Quantum

Classical Case (FCC)	Quantum Case (FQC)
Finite sets	Finite dimensional Hilbert spaces
Cartesian product (\times)	Tensor product (\otimes)
Bijections	Unitary operators



Classical v. Quantum

Classical Case (FCC)	Quantum Case (FQC)
Finite sets	Finite dimensional Hilbert spaces
Cartesian product (\times)	Tensor product (\otimes)
Bijections	Unitary operators
Functions	



Classical v. Quantum

Classical Case (FCC)	Quantum Case (FQC)
Finite sets	Finite dimensional Hilbert spaces
Cartesian product (\times)	Tensor product (\otimes)
Bijections	Unitary operators
Functions	Superoperators



Classical v. Quantum

Classical Case (FCC)	Quantum Case (FQC)
Finite sets	Finite dimensional Hilbert spaces
Cartesian product (\times)	Tensor product (\otimes)
Bijections	Unitary operators
Functions	Superoperators
Injective functions	



Classical v. Quantum

Classical Case (FCC)	Quantum Case (FQC)
Finite sets	Finite dimensional Hilbert spaces
Cartesian product (\times)	Tensor product (\otimes)
Bijections	Unitary operators
Functions	Superoperators
Injective functions	Isometries



Classical v. Quantum

Classical Case (FCC)	Quantum Case (FQC)
Finite sets	Finite dimensional Hilbert spaces
Cartesian product (\times)	Tensor product (\otimes)
Bijections	Unitary operators
Functions	Superoperators
Injective functions	Isometries
Projections	



Classical v. Quantum

Classical Case (FCC)	Quantum Case (FQC)
Finite sets	Finite dimensional Hilbert spaces
Cartesian product (\times)	Tensor product (\otimes)
Bijections	Unitary operators
Functions	Superoperators
Injective functions	Isometries
Projections	Partial trace

QML Syntax



- Types:

$$\sigma = Q_1 \mid Q_2 \mid \sigma \otimes \tau$$



QML Syntax

- Types:

$$\sigma = Q_1 \mid Q_2 \mid \sigma \otimes \tau$$

- Syntax:

(Variables) $x, y, \dots \in Vars$

(Prob.amplitudes) $\kappa, \iota, \dots \in \mathbb{C}$

(Patterns) $p, q ::= x \mid (x, y)$

(Terms) $t, u, e ::= x \mid x^{\vec{y}} \mid () \mid (t, u)$

| **let** $p = t$ **in** u

| **if** t **then** u **else** u'

| **if**^o t **then** u **else** u'

| $false \mid true \mid \vec{0} \mid \kappa \times t \mid t + u$



QML Syntax

- Types:

$$\sigma = Q_1 \mid Q_2 \mid \sigma \otimes \tau$$

- Syntax:

(Variables) $x, y, \dots \in Vars$

(Prob.amplitudes) $\kappa, \iota, \dots \in \mathbb{C}$

(Patterns) $p, q ::= x \mid (x, y)$

(Terms) $t, u, e ::= x \mid x^{\vec{y}} \mid () \mid (t, u)$

| **let** $p = t$ **in** u

| **if** t **then** u **else** u'

| **if**^o t **then** u **else** u'

| $false \mid true \mid \vec{0} \mid \kappa \times t \mid t + u$

- EPR State: $\frac{1}{\sqrt{2}} \times (false, false) + \frac{1}{\sqrt{2}} \times (true, true)$



Control of Decoherence

- Projection Function

$$\pi_1 \in (\mathcal{Q}_2, \mathcal{Q}_2) \rightarrow \mathbf{C}_2$$

$$\pi_1(x, y) = x$$

$$\begin{array}{ccc} \mathcal{Q}_2 & \xrightarrow{\quad} & \mathcal{Q}_2 \\ \mathcal{Q}_2 & \xrightarrow{\quad} & \mathcal{Q}_2 \\ & \phi_{\pi_1} & \end{array}$$

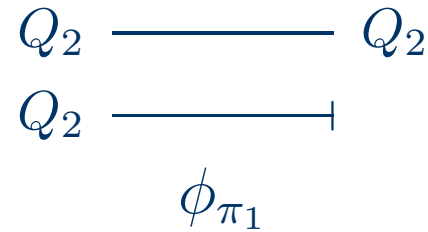


Control of Decoherence

- Projection Function

$$\pi_1 \in (\mathcal{Q}_2, \mathcal{Q}_2) \rightarrow \mathbf{C}_2$$

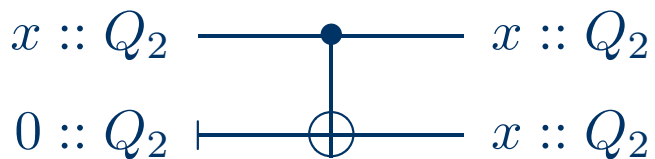
$$\pi_1(x, y) = x$$



- Diagonal Function

$$\delta \in \mathcal{Q}_2 \rightarrow (\mathcal{Q}_2, \mathcal{Q}_2) \quad x :: \mathcal{Q}_2$$

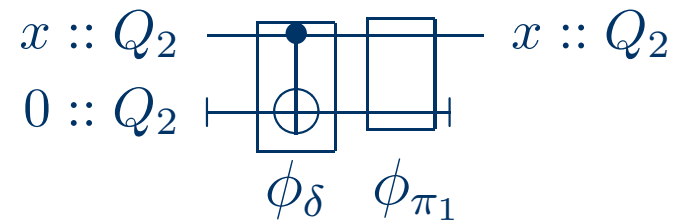
$$\delta x = (x, x)$$





Control of Decoherence

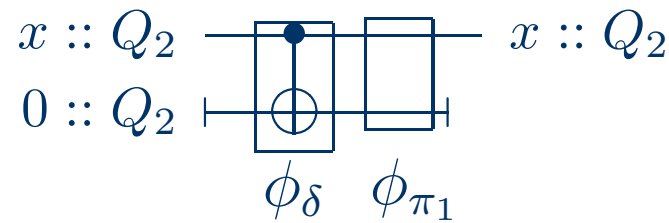
- $\pi_1.\delta \in \mathcal{Q}_2 \rightarrow \mathcal{Q}_2$





Control of Decoherence

- $\pi_1.\delta \in \mathcal{Q}_2 \rightarrow \mathcal{Q}_2$



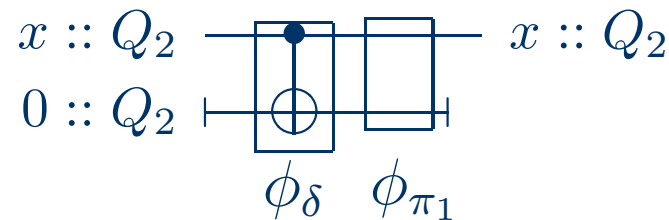
- Classical Case:





Control of Decoherence

- $\pi_1.\delta \in \mathcal{Q}_2 \rightarrow \mathcal{Q}_2$



- Classical Case:



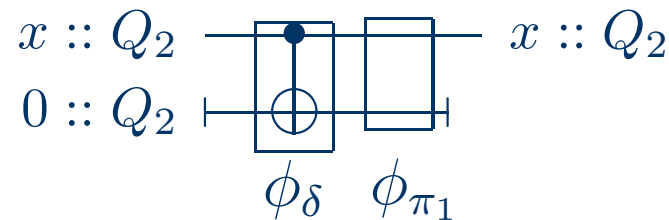
- Quantum Case:

Input = $\frac{1}{\sqrt{2}} \times false + \frac{1}{\sqrt{2}} \times true$ (equal superposition)



Control of Decoherence

- $\pi_1.\delta \in \mathcal{Q}_2 \rightarrow \mathcal{Q}_2$



- Classical Case:



- Quantum Case:

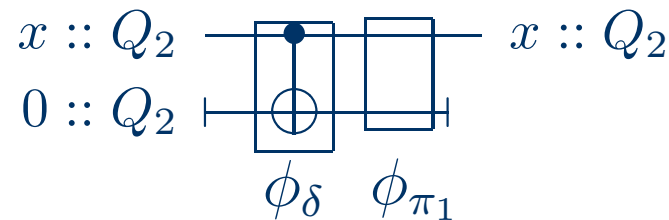
Input = $\frac{1}{\sqrt{2}} \times false + \frac{1}{\sqrt{2}} \times true$ (equal superposition)

Output = $\{\frac{1}{2}\} false + \{\frac{1}{2}\} true$ (probability distribution)



Control of Decoherence

- $\pi_1.\delta \in \mathcal{Q}_2 \rightarrow \mathcal{Q}_2$



- Classical Case:

$$Q_2 \text{ ————— } Q_2$$

- Quantum Case:

Input = $\frac{1}{\sqrt{2}} \times false + \frac{1}{\sqrt{2}} \times true$ (equal superposition)

Output = $\{\frac{1}{2}\} false + \{\frac{1}{2}\} true$ (probability distribution)

Decoherence! Not the identity function

More Decoherence



The University of
Nottingham



More Decoherence

- *forget* mentions x
 $forget \in Q_2 \multimap Q_2$
 $forget\ x = \mathbf{if\ } x \mathbf{\ then\ } true \mathbf{\ else\ } true$



More Decoherence

- *forget* mentions x
 $forget \in Q_2 \multimap Q_2$
 $forget\ x = \mathbf{if\ } x \mathbf{\ then\ } true \mathbf{\ else\ } true$
- but doesn't use it.



More Decoherence

- *forget* mentions x
 $forget \in Q_2 \multimap Q_2$
 $forget\ x = \text{if } x \text{ then } true \text{ else } true$
- but doesn't use it.
- Hence, it **has** to measure it!



More Decoherence

- *forget* mentions x
 $forget \in Q_2 \multimap Q_2$
 $forget\ x = \mathbf{if}\ x\ \mathbf{then}\ true\ \mathbf{else}\ true$
- but doesn't use it.
- Hence, it **has** to measure it!
- **if** always measures the conditional



More Decoherence

- *forget* mentions x
 $forget \in Q_2 \multimap Q_2$
 $forget\ x = \mathbf{if}\ x\ \mathbf{then}\ true\ \mathbf{else}\ true$
- but doesn't use it.
- Hence, it **has** to measure it!
- **if** always measures the conditional
- Not, using **if**
 $not \in Q_2 \multimap Q_2$
 $not\ x = \mathbf{if}\ x\ \mathbf{then}\ false\ \mathbf{else}\ true$

if^o – Quantum control



The University of
Nottingham



if° – Quantum control

-

$\text{forget}' \in \mathcal{Q}_2 \multimap \mathcal{Q}_2$

$\text{forget}' x = \text{if}^\circ x \text{ then } \text{true} \text{ else } \text{true}$



if° – Quantum control

- $\text{forget}' \in \mathcal{Q}_2 \multimap \mathcal{Q}_2$
 $\text{forget}' x = \text{if}^\circ x \text{ then } \text{true} \text{ else } \text{true}$
- This program has a type error, because $\text{true} \not\vdash \text{true}$.



if° – Quantum control

- $\text{forget}' \in \mathcal{Q}_2 \multimap \mathcal{Q}_2$
 $\text{forget}' x = \text{if}^\circ x \text{ then } \text{true} \text{ else } \text{true}$
- This program has a type error, because $\text{true} \not\vdash \text{true}$.

- $\text{qnot} \in \mathcal{Q}_2 \multimap \mathcal{Q}_2$
 $\text{qnot } x = \text{if}^\circ x \text{ then } \text{false} \text{ else } \text{true}$



if° – Quantum control

- $\text{forget}' \in \mathcal{Q}_2 \multimap \mathcal{Q}_2$
 $\text{forget}' x = \text{if}^\circ x \text{ then } \text{true} \text{ else } \text{true}$
- This program has a type error, because $\text{true} \not\perp \text{true}$.
- $\text{qnot} \in \mathcal{Q}_2 \multimap \mathcal{Q}_2$
 $\text{qnot } x = \text{if}^\circ x \text{ then } \text{false} \text{ else } \text{true}$
- This program typechecks, because $\text{false} \perp \text{true}$.



if° – Quantum control

- $\text{forget}' \in \mathcal{Q}_2 \multimap \mathcal{Q}_2$
 $\text{forget}' x = \text{if}^\circ x \text{ then } \text{true} \text{ else } \text{true}$
- This program has a type error, because $\text{true} \not\perp \text{true}$.
- $\text{qnot} \in \mathcal{Q}_2 \multimap \mathcal{Q}_2$
 $\text{qnot } x = \text{if}^\circ x \text{ then } \text{false} \text{ else } \text{true}$
- This program typechecks, because $\text{false} \perp \text{true}$.
- $\text{cnot } c x = \text{if}^\circ c \text{ then } (\text{true}, \text{qnot } x) \text{ else } (\text{false}, x)$



if° – Quantum control

- $\text{forget}' \in \mathcal{Q}_2 \multimap \mathcal{Q}_2$
 $\text{forget}' x = \text{if}^\circ x \text{ then } \text{true} \text{ else } \text{true}$
- This program has a type error, because $\text{true} \not\perp \text{true}$.
- $\text{qnot} \in \mathcal{Q}_2 \multimap \mathcal{Q}_2$
 $\text{qnot } x = \text{if}^\circ x \text{ then } \text{false} \text{ else } \text{true}$
- This program typechecks, because $\text{false} \perp \text{true}$.
- $\text{cnot } c x = \text{if}^\circ c \text{ then } (\text{true}, \text{qnot } x) \text{ else } (\text{false}, x)$
- Deutsch-Joza Algorithm, Quantum Teleport Algorithm, ...



Quantum Teleport Algorithm

$pZed \in Q_2 \multimap Q_2$

$pZed\ x = \mathbf{if}^\circ\ x\ \mathbf{then}\ (-1) \times\ true\ \mathbf{else}\ false$

$had \in Q_2 \multimap Q_2$

$had\ x = \mathbf{if}^\circ\ x\ \mathbf{then}\ (-1) \times\ true +\ false\ \mathbf{else}\ true +\ false$

$tel \in Q_2 \multimap Q_2$

$tel\ x = \mathbf{let}\ (a, b) = (false, false) + (true, true)$

$(a', x') = cnot\ a\ x$

$b' = \mathbf{if}\ a'\ \mathbf{then}\ qnot\ b\ \mathbf{else}\ b$

$b'' = \mathbf{if}^\circ\ had\ x'\ \mathbf{then}\ pZed\ b'\ \mathbf{else}\ b'$

$\mathbf{in}\ b''$



Inner Product & \perp

- We define the inner product of terms, which to any pair of terms $\Gamma \vdash t, u : \sigma$ assigns $\langle t|u \rangle \in \mathbb{C} \cup \{?\}$.



Inner Product & \perp

- We define the inner product of terms, which to any pair of terms $\Gamma \vdash t, u : \sigma$ assigns $\langle t|u \rangle \in \mathbb{C} \cup \{?\}$.
- $t \perp u$ holds if $\langle t|u \rangle = 0$.



Inner Product & \perp

- We define the inner product of terms, which to any pair of terms $\Gamma \vdash t, u : \sigma$ assigns $\langle t|u \rangle \in \mathbb{C} \cup \{?\}$.
- $t \perp u$ holds if $\langle t|u \rangle = 0$.
- $\langle t|t \rangle = 1$, $\langle \text{false}|\text{true} \rangle = 0$, $\langle \text{true}|\text{false} \rangle = 0$



Inner Product & \perp

- We define the inner product of terms, which to any pair of terms $\Gamma \vdash t, u : \sigma$ assigns $\langle t|u \rangle \in \mathbb{C} \cup \{?\}$.
- $t \perp u$ holds if $\langle t|u \rangle = 0$.
- $\langle t|t \rangle = 1$, $\langle \text{false}|\text{true} \rangle = 0$, $\langle \text{true}|\text{false} \rangle = 0$
- $\langle \vec{0}|\text{true} \rangle = 0 = \langle \text{true}|\vec{0} \rangle$, $\langle \vec{0}|\text{false} \rangle = 0 = \langle \text{false}|\vec{0} \rangle$,
 $\langle \vec{0}|\mathbf{x} \rangle = 0 = \langle \mathbf{x}|\vec{0} \rangle$



Inner Product & \perp

- We define the inner product of terms, which to any pair of terms $\Gamma \vdash t, u : \sigma$ assigns $\langle t|u \rangle \in \mathbb{C} \cup \{?\}$.
- $t \perp u$ holds if $\langle t|u \rangle = 0$.
- $\langle t|t \rangle = 1$, $\langle \text{false}|\text{true} \rangle = 0$, $\langle \text{true}|\text{false} \rangle = 0$
- $\langle \vec{0}|\text{true} \rangle = 0 = \langle \text{true}|\vec{0} \rangle$, $\langle \vec{0}|\text{false} \rangle = 0 = \langle \text{false}|\vec{0} \rangle$,
 $\langle \vec{0}|\mathbf{x} \rangle = 0 = \langle \mathbf{x}|\vec{0} \rangle$
- $\langle (t, t') | (u, u') \rangle = \langle t|u \rangle * \langle t'|u' \rangle$



Inner Product & \perp

- We define the inner product of terms, which to any pair of terms $\Gamma \vdash t, u : \sigma$ assigns $\langle t|u \rangle \in \mathbb{C} \cup \{?\}$.
- $t \perp u$ holds if $\langle t|u \rangle = 0$.
- $\langle t|t \rangle = 1$, $\langle \text{false}|\text{true} \rangle = 0$, $\langle \text{true}|\text{false} \rangle = 0$
- $\langle \vec{0}|\text{true} \rangle = 0 = \langle \text{true}|\vec{0} \rangle$, $\langle \vec{0}|\text{false} \rangle = 0 = \langle \text{false}|\vec{0} \rangle$,
 $\langle \vec{0}|\mathbf{x} \rangle = 0 = \langle \mathbf{x}|\vec{0} \rangle$
- $\langle (t, t') | (u, u') \rangle = \langle t|u \rangle * \langle t'|u' \rangle$
- $\langle \lambda * t + \lambda' * t' | u \rangle = \lambda * \langle t|u \rangle + \lambda' * \langle t'|u \rangle$,
 $\langle t | \kappa * u + \kappa' * u' \rangle = \kappa * \langle t|u \rangle + \kappa' * \langle t|u' \rangle$



Inner Product & \perp

- We define the inner product of terms, which to any pair of terms $\Gamma \vdash t, u : \sigma$ assigns $\langle t|u \rangle \in \mathbb{C} \cup \{?\}$.
- $t \perp u$ holds if $\langle t|u \rangle = 0$.
- $\langle t|t \rangle = 1$, $\langle \text{false}|\text{true} \rangle = 0$, $\langle \text{true}|\text{false} \rangle = 0$
- $\langle \vec{0}|\text{true} \rangle = 0 = \langle \text{true}|\vec{0} \rangle$, $\langle \vec{0}|\text{false} \rangle = 0 = \langle \text{false}|\vec{0} \rangle$,
 $\langle \vec{0}|\mathbf{x} \rangle = 0 = \langle \mathbf{x}|\vec{0} \rangle$
- $\langle (t, t') | (u, u') \rangle = \langle t|u \rangle * \langle t'|u' \rangle$
- $\langle \lambda * t + \lambda' * t' | u \rangle = \lambda * \langle t|u \rangle + \lambda' * \langle t'|u \rangle$,
 $\langle t | \kappa * u + \kappa' * u' \rangle = \kappa * \langle t|u \rangle + \kappa' * \langle t|u' \rangle$
- ...



Inner Product & \perp

- We define the inner product of terms, which to any pair of terms $\Gamma \vdash t, u : \sigma$ assigns $\langle t|u \rangle \in \mathbb{C} \cup \{?\}$.
- $t \perp u$ holds if $\langle t|u \rangle = 0$.
- $\langle t|t \rangle = 1$, $\langle \text{false}|\text{true} \rangle = 0$, $\langle \text{true}|\text{false} \rangle = 0$
- $\langle \vec{0}|\text{true} \rangle = 0 = \langle \text{true}|\vec{0} \rangle$, $\langle \vec{0}|\text{false} \rangle = 0 = \langle \text{false}|\vec{0} \rangle$,
 $\langle \vec{0}|\mathbf{x} \rangle = 0 = \langle \mathbf{x}|\vec{0} \rangle$
- $\langle (t, t') | (u, u') \rangle = \langle t|u \rangle * \langle t'|u' \rangle$
- $\langle \lambda * t + \lambda' * t' | u \rangle = \lambda * \langle t|u \rangle + \lambda' * \langle t'|u \rangle$,
 $\langle t | \kappa * u + \kappa' * u' \rangle = \kappa * \langle t|u \rangle + \kappa' * \langle t|u' \rangle$
- ...
- $\langle t|u \rangle = ?$ otherwise

Summary



- Our semantic ideas proved useful when designing a quantum programming language, analogous concepts are modelled by the same syntactic constructs



Summary

- Our semantic ideas proved useful when designing a quantum programming language, analogous concepts are modelled by the same syntactic constructs
- Our analysis also highlights the differences between classical and quantum programming



Summary

- Our semantic ideas proved useful when designing a quantum programming language, analogous concepts are modelled by the same syntactic constructs
- Our analysis also highlights the differences between classical and quantum programming
- Quantum programming introduces the problem of *control of decoherence*, which we address by making the ‘forgetting’ of variables explicit, and by having two **if** constructs and the orthogonality condition



Summary

- Our semantic ideas proved useful when designing a quantum programming language, analogous concepts are modelled by the same syntactic constructs
- Our analysis also highlights the differences between classical and quantum programming
- Quantum programming introduces the problem of *control of decoherence*, which we address by making the ‘forgetting’ of variables explicit, and by having two **if** constructs and the orthogonality condition
- Other work:



Summary

- Our semantic ideas proved useful when designing a quantum programming language, analogous concepts are modelled by the same syntactic constructs
- Our analysis also highlights the differences between classical and quantum programming
- Quantum programming introduces the problem of *control of decoherence*, which we address by making the ‘forgetting’ of variables explicit, and by having two **if** constructs and the orthogonality condition
- Other work:
- A compiler for QML, in Haskell, that produces (typed) circuits



Summary

- Our semantic ideas proved useful when designing a quantum programming language, analogous concepts are modelled by the same syntactic constructs
- Our analysis also highlights the differences between classical and quantum programming
- Quantum programming introduces the problem of *control of decoherence*, which we address by making the ‘forgetting’ of variables explicit, and by having two **if** constructs and the orthogonality condition
- Other work:
 - A compiler for QML, in Haskell, that produces (typed) circuits
 - Extend algebra and normal form beyond pure subset of QML



Summary

- Our semantic ideas proved useful when designing a quantum programming language, analogous concepts are modelled by the same syntactic constructs
- Our analysis also highlights the differences between classical and quantum programming
- Quantum programming introduces the problem of *control of decoherence*, which we address by making the ‘forgetting’ of variables explicit, and by having two **if** constructs and the orthogonality condition
- Other work:
 - A compiler for QML, in Haskell, that produces (typed) circuits
 - Extend algebra and normal form beyond pure subset of QML
 - Higher-order types, compositionality proofs, implement using measurement calculus ...



Thanks for listening

- Papers on QML can be found at:
- sneezy.cs.nott.ac.uk/qml
- There is also an interactive research diary:
- sneezy.cs.nott.ac.uk/qml/weblog

- Jonathan Grattage (www.cs.nott.ac.uk/~jjg)
Thorsten Altenkirch (www.cs.nott.ac.uk/~txa)