Quantum Lambda-Calculus: Realizability and Emerging Logical Properties

Benoît Valiron (CentraleSupélec / LMF)

Joint work with Alejandro Díaz-Caro (UBA / Inria / LORIA)

May 13, 2025

IRN LI, Roma

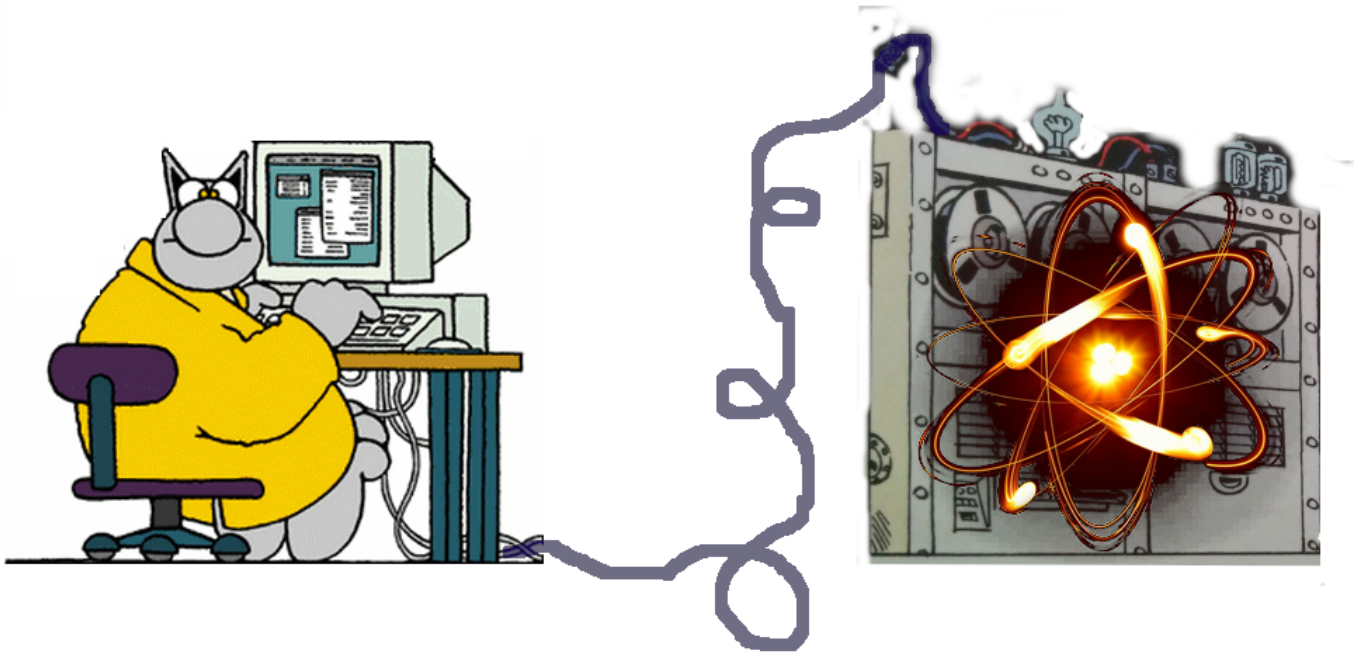
Quantum Computation

	Classical	Quantum
Basic register	bit	quantum bit = qbit
Math model	Set	Hilbert Spaces and Unitary maps
Pure states	$\mathcal{B} = \{\texttt{true}, \texttt{false}\}$	$\mathbb{C}^2 \equiv \langle \mathcal{B} \rangle \qquad \alpha \cdot \lvert 0 \rangle + \beta \cdot \lvert 1 \rangle$
3 registers	$\mathcal{B} \times \mathcal{B} \times \mathcal{B}$	$\mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \mathbb{C}^2 = \mathbb{C}^8 \equiv \langle \mathcal{B}^3 \rangle$
Reading	Deterministic	Probabilistic
Duplicable	Yes	No

Entangled state Cannot be written a pair of states

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \quad \neq \quad |\phi\rangle \otimes |\psi\rangle$$

Standard Computational Model: Co-Processor



Main Computer	Interface	Co-processor
Program	Instructions	Quantum memory
Execution flow	⇄	Quantum operations
	Feedback	

Standard Computational Model: Co-Processor

The quantum memory

- A set of individually addressable quantum registers

Actions through the interface

- Initialize registers
- Apply quantum operations
  - Linear, unitary transformations on the state space
  - No cloning of quantum states
- Read register
  - Through probabilistic measurement
  - Only access to quantum information

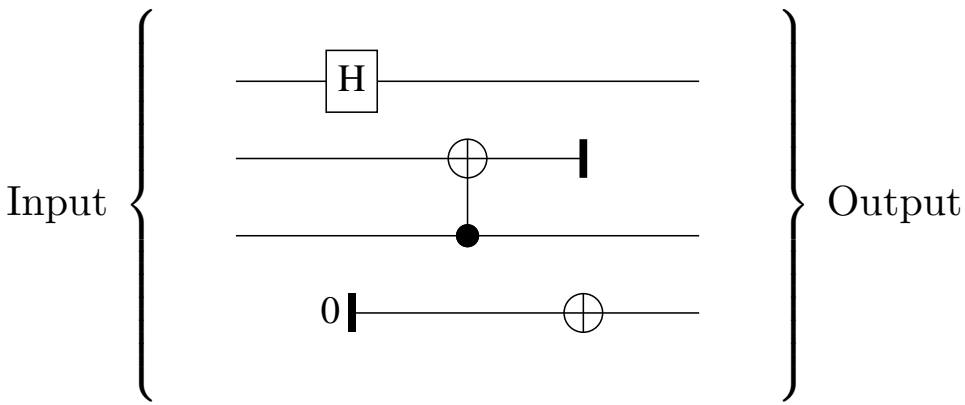
From the programmer's perspective

- A probabilistic read operation
- Non-duplicable data

Standard Computational Model: Co-Processor

Local Operations

- Batch of low-level instructions
- Elementary operations applied on the quantum memory
- Written as a quantum circuit



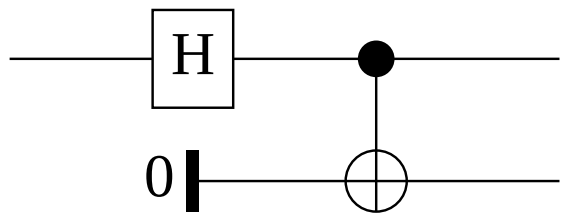

```
input 0 1 2
init 3
H 0
CNOT 2 1
NOT 3
discard 1
output 0 2 3
```

Quantum Circuits as Functions

Wires

- One wire: one qbit, encapulated in a type `qbit`
- Several wires: array of qbits, one per wire
  $\texttt{qbit} \otimes \texttt{qbit} \otimes \cdots \otimes \texttt{qbit}$

Circuits



- Inputs one qbit
- Outputs a pair of qbits
- Function from `qbit` to $\texttt{qbit} \otimes \texttt{qbit}$
- Side-effect: gates acting on the quantum memory

Quantum Lambda-Calculus

Terms

- Pairing constructs and fixpoints
- Boolean `true` and `false`, if-then-else
- Constant, opaque terms: `qinit`, `measure`, `H`, `CNOT`, ...
- Quantum states not in the language
  $\rightarrow$ included as pointers

Operational semantics

- Abstract machine encapsulating the quantum memory:

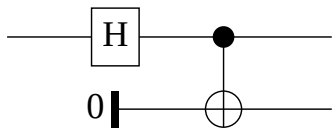
$$\Big(\underbrace{\tfrac{1}{\sqrt{2}}(|00\rangle + |11\rangle)}_{\text{state vector}}, \quad \underbrace{|xy\rangle}_{\text{“linking function”}}, \quad \underbrace{\lambda f.f\langle x, y\rangle}_{\text{lambda-term}}\Big)$$

- Call-by-value evaluation strategy
- Quantum operations through the evaluation strategy

Quantum Lambda-Calculus

Some terms are duplicable

- Boolean constants: `true`
- Regular, pure lambda-terms: $\lambda x.x$
- Circuit-descriptions: $\lambda x.(\text{let } z = \texttt{H}\, x \text{ in } \texttt{CNOT}\, \langle z, \texttt{qinit false}\rangle)$



Some terms should not be duplicable

- Qbits: $\texttt{H}\,(\texttt{qinit false})$
- Tuples containing a qbit: $\langle \lambda x.x,\ \texttt{H}\,(\texttt{qinit false})\rangle$
- Functions containing a qbit: $\text{let } y = \texttt{H}\,(\texttt{qinit false}) \text{ in } \lambda f.f\,y$

Distinction between

- Procedure for generating a qbit: duplicable
- End result of the procedure: qbit value, non-duplicable

Quantum Lambda-Calculus: Historical Type System

Standard type system

$$A, B \ ::= \ \texttt{qbit} \mid \texttt{bit} \mid \top \mid A \otimes B \mid A \multimap B \mid !(A \multimap B)$$

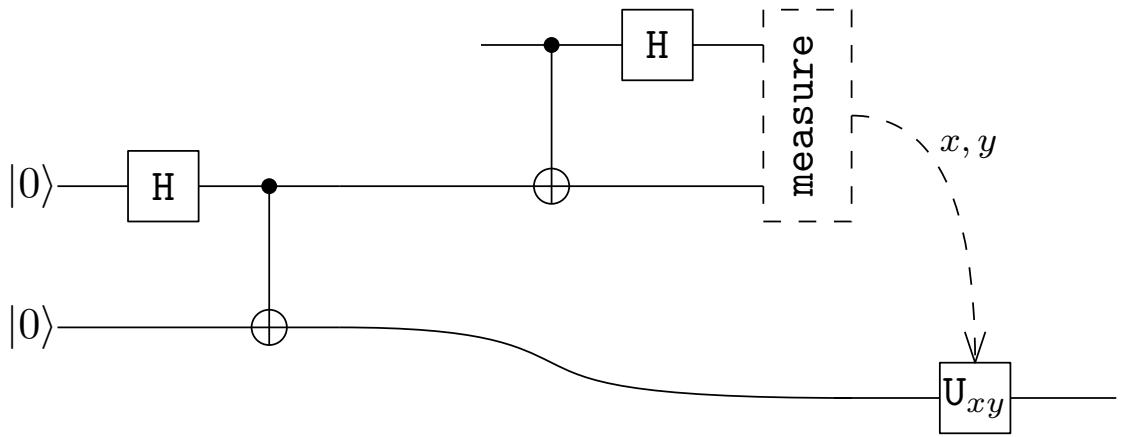
- Based on linear logic
- Non-duplicable functions with $A \multimap B$
- Duplicable functions with $!(A \multimap B)$
- Quantum operations are e.g. $\texttt{measure} : !(\texttt{qbit} \multimap \texttt{bit})$

Non-trivial mix

- Classical and quantum data, probabilistic setting
- Entanglement at higher-order

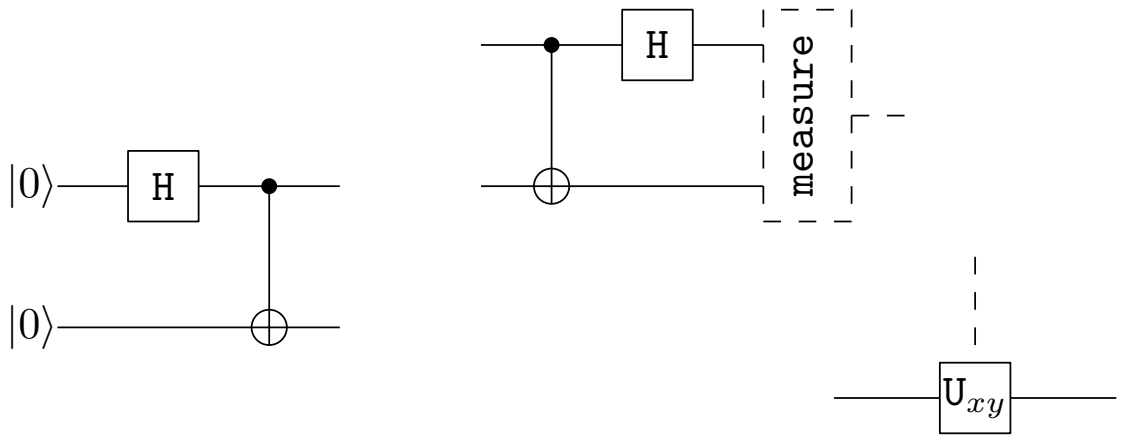
Quantum Lambda-Calculus: Historical Type System

Example of higher-order entanglement: Teleportation



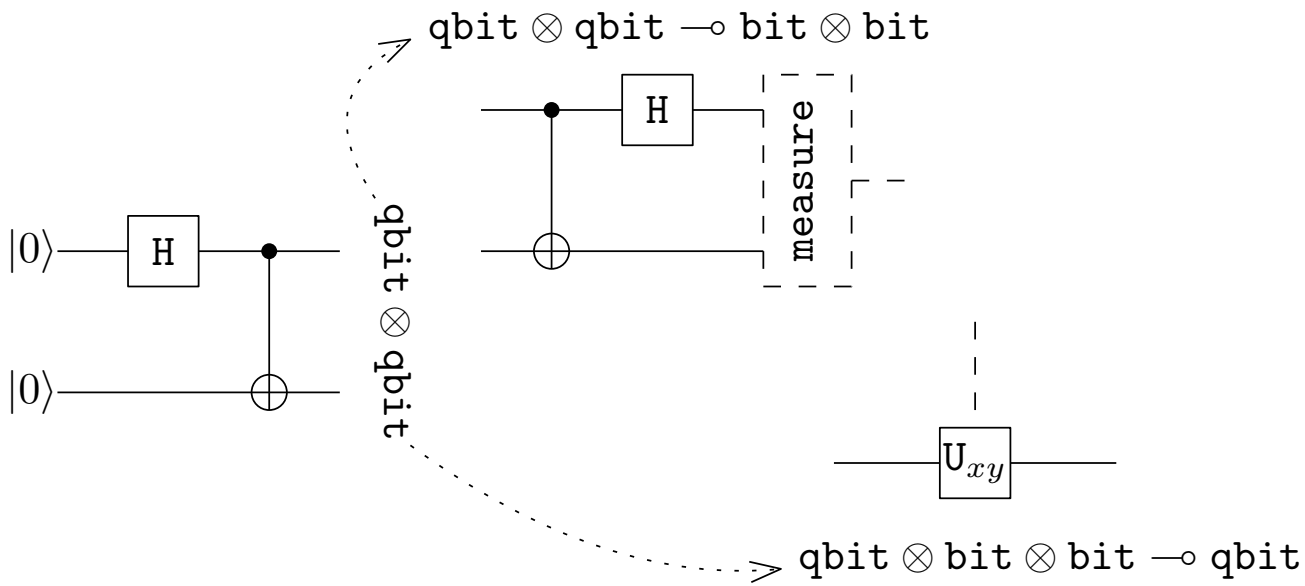
Quantum Lambda-Calculus: Historical Type System

Example of higher-order entanglement: Teleportation



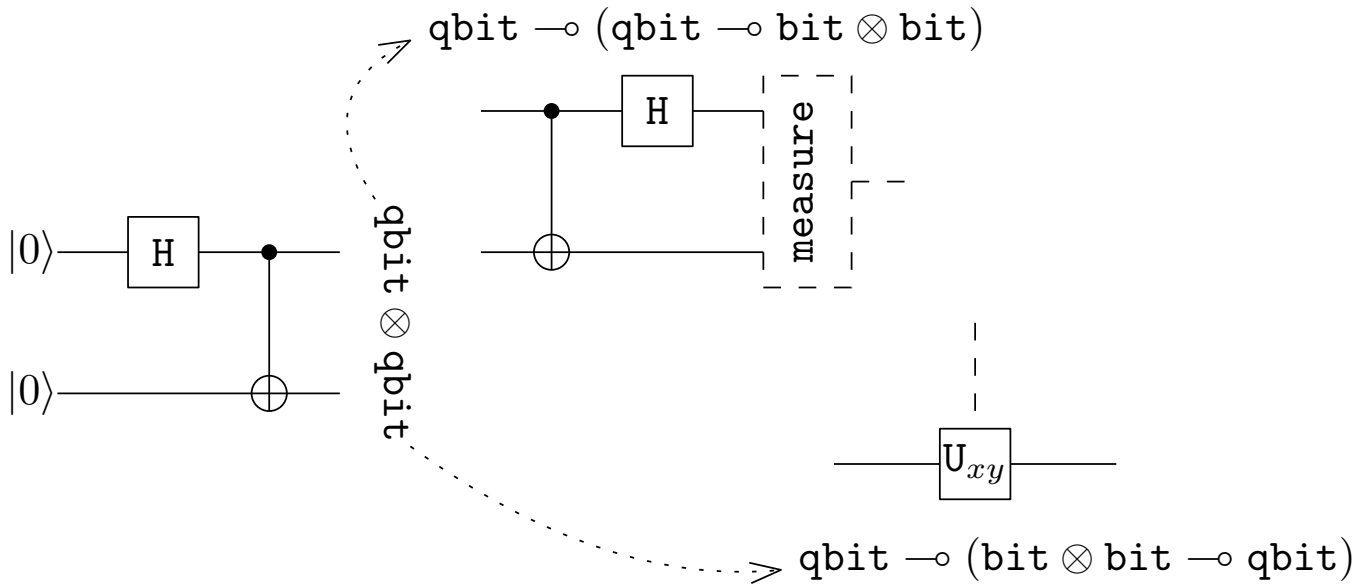
Quantum Lambda-Calculus: Historical Type System

Example of higher-order entanglement: Teleportation



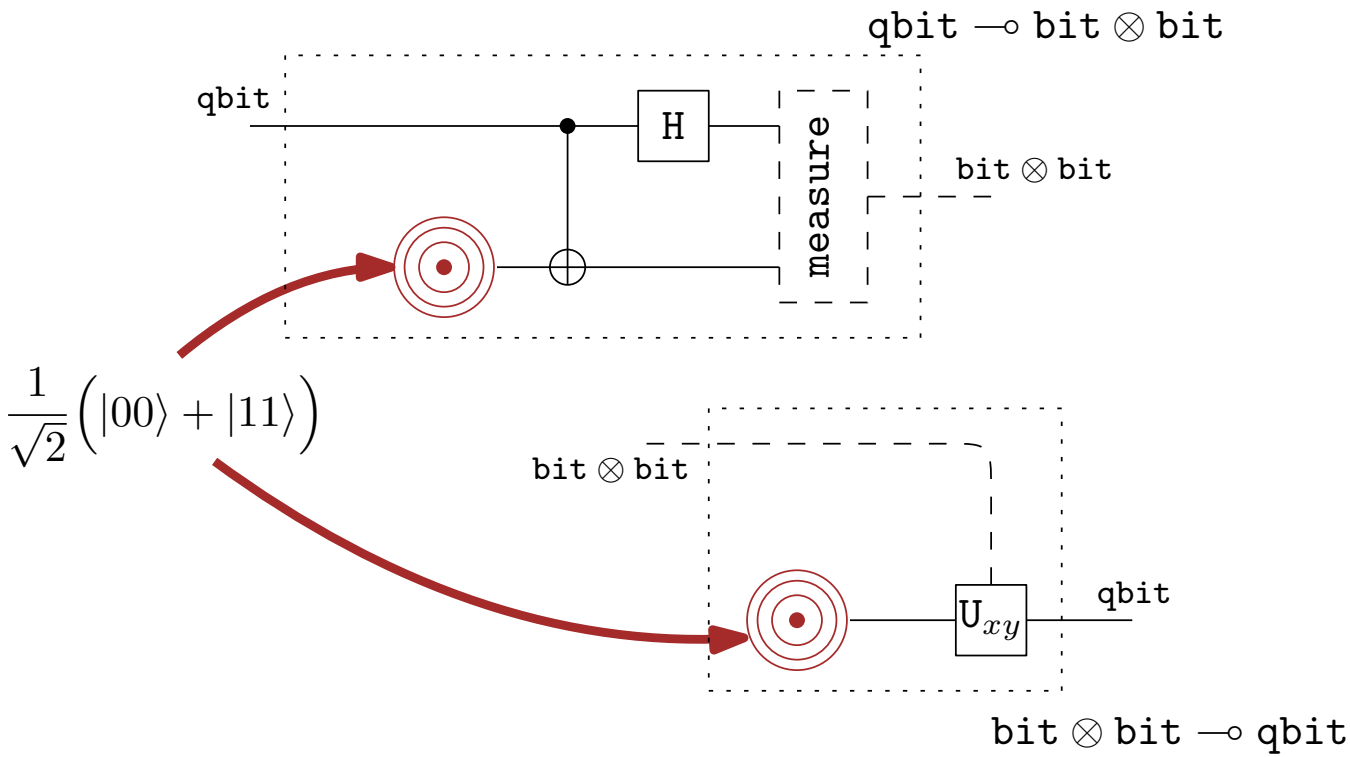
Quantum Lambda-Calculus: Historical Type System

Example of higher-order entanglement: Teleportation



Quantum Lambda-Calculus: Historical Type System

Example of higher-order entanglement: Teleportation



A pair of two entangled, non-duplicable, functions

$$(\texttt{qbit} \multimap \texttt{bit} \otimes \texttt{bit}) \otimes (\texttt{bit} \otimes \texttt{bit} \multimap \texttt{qbit})$$

inverses of each other.

Type System Extensions: The Easy Part

Co-product

- $A \oplus B$
- Left and right injections

Lists

- Identify $[A]$ with $I \oplus (A \otimes [A])$

Natural Numbers

- Constants $\overline{0}$, $\overline{1}$, $\overline{2}$, ... of type $\mathbb{N}$
- Arithmetics operations and tests

Type System Extensions: The Hard Part

Dependent types

- $[A]_n$: lists of size n
- $\forall n : \mathbb{N},\ [\texttt{qbit}]_n \multimap [\texttt{qbit}]_n$
- Categorical model from Selinger *et al.*
  $\rightarrow$ Fibrations over a monoidal category
- Dependency limited to classical types

We aim at general dependency over any type

- MVP: Spell out the specification of the teleportation algorithm

  "inverses of each other"
- Exploration tool: (intuitionistic) realizability

(Intuitionistic, Standard) Realizability

Semantic Types (for standard lambda-calculus)

- A (semantic) type is defined as a set of closed values
- Booleans : literally the terms "true" and "false"
- $A \times B$ is... literally pairs of elements of A and elements of B
- A realizer : $M \Vdash A$ when $M \to^* V \in A$
- $A \Rightarrow B$: set of values V such that $\forall W \in A,\ VW \Vdash B$

Syntactic Types

- Formal grammar:

$$A, B ::= \dot{S} \mid X \mid A \Rightarrow B \mid A \times B \mid \forall X.A$$

  with S a semantic type.

- Denotation of syntactic types: $[\![A]\!]_\theta$

$$[\![X]\!]_\theta = \theta(X) \qquad [\![\forall X.A]\!]_\theta = \bigcap_S [\![A]\!]_{\theta \cup \{X := \dot{S}\}}$$

$\theta : TVAr \to \mathcal{P}\{\text{sem types}\}$

(Intuitionistic, Standard) Realizability

Typing judgements

- Based on the notion of syntactic types
- $x : A, y : B \vdash M : C$ is defined as a relation meaning

$$\forall \theta, \forall V \in [\![A]\!]_\theta, \forall W \in [\![B]\!]_\theta, \quad M[x := V, y := W] \Vdash [\![C]\!]_\theta$$

- Typing rules becomes lemmas
  $\rightarrow$ Logical properties driven by the computational behavior

Realizability for the Quantum Lambda-Calculus

Problems to solve

- Probabilistic evaluation strategy
- Evaluation uses an abstract machine $[Q, L, M]$
- There are no closed values of type qubits.

Solution we follow

- Semantic types are possibly open values
  $\rightarrow$ Free variables assimilated with qbits
- Type of qbits: set of term variables!
- $M \Vdash A$ means (provided that L captures all $FV(M)$)

$$\forall Q, \quad [Q, L, M] \rightarrow^* \sum_i p_i \cdot [Q_i, L_i, V_i] \text{ with } \forall i, V_i \in A$$

- Limitation here: no reasonning on the structure of qubits

Realizability for the Quantum Lambda-Calculus

Recovering linear logic

- $!A = \{V \in A \mid FV(V) = \emptyset\}$
- Additive and multiplicative pairings

Additive Constructions

- $A \times B = \{\langle V, W\rangle \mid V \in A, W \in B\}$
- $\langle x, x\rangle \in \texttt{qbit} \times \texttt{qbit}$
- $A \Rightarrow B = \{V \mid \forall W \in A, \forall Q,\ [Q, L, VW] \Vdash B\}$

Multiplicative Constructions

- $A \otimes B = \{\langle V, W\rangle \mid V \in A, W \in B, FV(V) \cap FV(W) = \emptyset\}$
- $A \multimap B = \{V \mid \forall W \in A \text{ with } FV(V) \cap FV(W) = \emptyset,$
  $\forall Q,\ [Q, L, VW] \Vdash B\}$
- A feeling of separation logic!

Syntactic Types and Typing Judgements

We can reuse the standard definition

- Syntactic types (with our new constructors)

$$A, B ::= \dot{S} \mid X \mid A \times B \mid A \otimes B \mid A \Rightarrow B \mid A \multimap B \mid !A \mid \forall X.A$$

- Typing judgements: $x : A, y : B \vdash M : C$ defined as

$$\forall \langle V, W \rangle \in A \otimes B, \forall Q,\ [Q, L, M[x := V, y := W]] \Vdash C$$

- We recover the original typing rules of the quantum lambda-calculus
- With second-order quantifiers for free!

Some "Interesting" Dependent Types

Equality Type for Qbits (one possibility)

$$(M =_q N) \triangleq \begin{cases} \{\star\} & \text{if } FV(M) = FV(N) \text{ and if } \forall Q, \\ & [Q, L, M] \text{ and } [Q, L, N] \text{ reduce to registers} \\ & \text{in the same mixed state} \\ \emptyset & \text{otherwise} \end{cases}$$

Probability of Success

$$P^p_{tt}(M) \triangleq \begin{cases} \{\star\} & \text{if } \forall Q, [Q, L, M] \to^* p \cdot tt + \rho \\ \emptyset & \text{otherwise} \end{cases}$$

A Dependent Type System

Built as a second layer on top of the previous one

- Syntactic types:

$$\sigma, \tau \quad ::= \quad A \mid \sigma \multimap \tau \mid \sigma \otimes \tau \mid !\sigma \mid \forall x : A, \sigma \mid$$
$$M =_q N \mid P^p_{tt}(M) \mid \ \ldots$$

- The corresponding semantics type $[\![\sigma]\!]_{\theta,\xi}$ parameterized by
  - θ : mapping type variables to semantic types
  - ξ : mapping term variables to terms

Semantics Types for the Quantifiers

$$[\![\forall x : A, \sigma]\!]_{\theta,\xi} := \bigcap_{V \in [\![A]\!]_\theta} [\![\sigma]\!]_{\theta,\, \xi \cup \{x := V\}}$$

Example

The term $\lambda z.\star$ realizes

$$\forall x : \mathtt{qbit}, \forall y : \mathtt{qbit}, (x =_q y) \multimap (y =_q x)$$

Deriving the Example

The term $\lambda z.\star$ realizes

$$\forall x : \mathtt{qbit}, \forall y : \mathtt{qbit}, (x =_q y) \multimap (y =_q x)$$

Comes from the fact that $\lambda z.\star$ realizes the open formula

$$(x =_q y) \multimap (y =_q x)$$

because

$$\forall \theta,\ \forall \xi,\ \forall U \in [\![x =_q y]\!]_{\theta,\xi},\ \star \Vdash [\![y =_q x]\!]_{\theta,\xi}$$

- U is just $\star$
- $\xi(x)$ and $\xi(y)$ are "just" term variables
- They refer to quantum registers in a quantum memory
- Symmetry of equality comes from the symmetry of the "iff"

Back to Teleportation

Remember

We were able to build a (non-duplicable) pair of types:

$$(\texttt{qbit} \multimap \texttt{bit} \otimes \texttt{bit}) \otimes (\texttt{bit} \otimes \texttt{bit} \multimap \texttt{qbit})$$

Write this lambda-term $\langle alice, bob \rangle$ and the type as $A_{alice} \otimes B_{bob}$
By construction, these two functions are inverses of each other.

Dependent Specification

With our equality type, this is

$$\forall x : \texttt{bit}, bob\,(alice\,x) =_q x$$

But *alice* and *bob* are entangled
$\rightarrow$ they only make sense with their quantum state.

Instead, the pair can be regarded as a witness for the type

$$\forall X.(\forall f : A_{alice}, \forall g : B_{bob}, (\forall x : \texttt{qbit}, g(fx) =_q x) \rightarrow X) \rightarrow X$$

Back to Teleportation

Let us spell it down A term of type

$$\forall X.(\forall f : A_{alice}, \forall g : B_{bob}, (\forall x : \mathtt{qbit}, g(fx) =_q x) \to X) \to X$$

inputs a function of 1 argument, and feed it with

- a witness that $g \circ f$ is the identity
- for some f of type $\mathtt{qbit} \multimap \mathtt{bit} \otimes \mathtt{bit}$
- and some g of type $\mathtt{bit} \otimes \mathtt{bit} \multimap \mathtt{qbit}$

Our pair contains these elements.
The witness is implicit: the property is true by definition.

Said otherwise It is "logically" the same as

$$\neg \forall f : A_{alice}, \forall g : B_{bob}, \neg(\forall x : \mathtt{qbit}, g(fx) =_q x)$$

which is

$$\neg\neg \exists f : A_{alice}, \exists g : B_{bob}, \forall x : qbit, g(fx) =_q x$$

Limit of the approach

We would love the term $\lambda z.\lambda z'.\star$ to realize

$$\forall x : \texttt{qbit}, \forall y : \texttt{qbit}, (x =_q y) \multimap P_{tt}^{\frac{1}{2}}(x) \multimap P_{tt}^{\frac{1}{2}}(y)$$

This should come from that $\lambda z.\lambda z'.\star$ realizes the open formula

$$(x =_q y) \multimap P_{tt}(x, \tfrac{1}{2}) \multimap P_{tt}(y, \tfrac{1}{2})$$

I.e.

$$\forall \theta,\ \forall \xi,\ \forall U \in [\![x =_q y]\!]_{\theta,\xi},\ \forall V \in [\![P_{tt}(x, \tfrac{1}{2})]\!]_{\theta,\xi},\quad \star \Vdash [\![P_{tt}(y, \tfrac{1}{2})]\!]_{\theta,\xi}$$

- U and V are $\star$
- $\xi(x)$ and $\xi(y)$ are "just" term variables
- They refer to quantum registers in a quantum memory
- But each dependent relation uses its own quantum context!
- Elements of type `qbit` only refers to registers...

Conclusion

- Realizability: A versatile framework
- Novel typing construction for the quantum lambda-calculus
- Captures quantification over qubits!
- Still limited...
- But gives hints as of where to search

To be continued...

...and presented at the next IRN-LI seminar?