From Symmetric Pattern Matching to Quantum Control

Benoît Valiron
LRI – CentraleSupélec, France

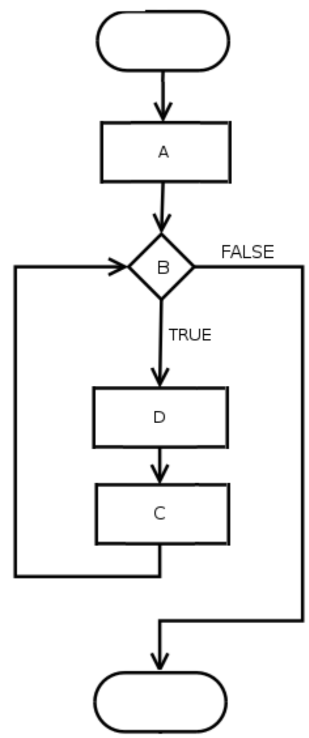
Joint work with Amr Sabry and Juliana Vizzotto

CVQT — March 21, 2018

Notion of Control

Control in quantum computation has two meanings

- The control of a unitary, as in the control-not gate
- The control-flow of a program: made of primitives such as
  - Sequences of operations
  - Tests and branchings
  - Loops/fixpoints



The former is an instance of the latter:
a quantum control-flow primitive.
(albeit limited)

General quantum control-flow
is the purpose of this talk

Plan

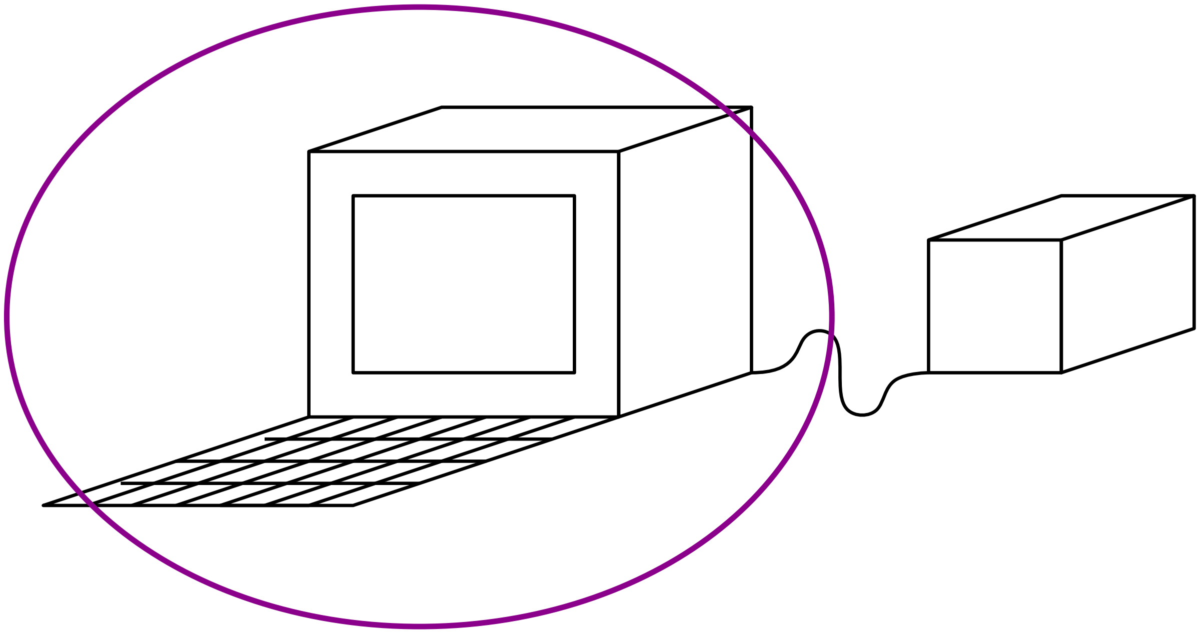
1. Classical Control
2. The bumpy road towards quantum control
3. Detour via reversible computing
4. The elusive quantum loop

Chapter 1

Classical Control

Quantum with Classical Control

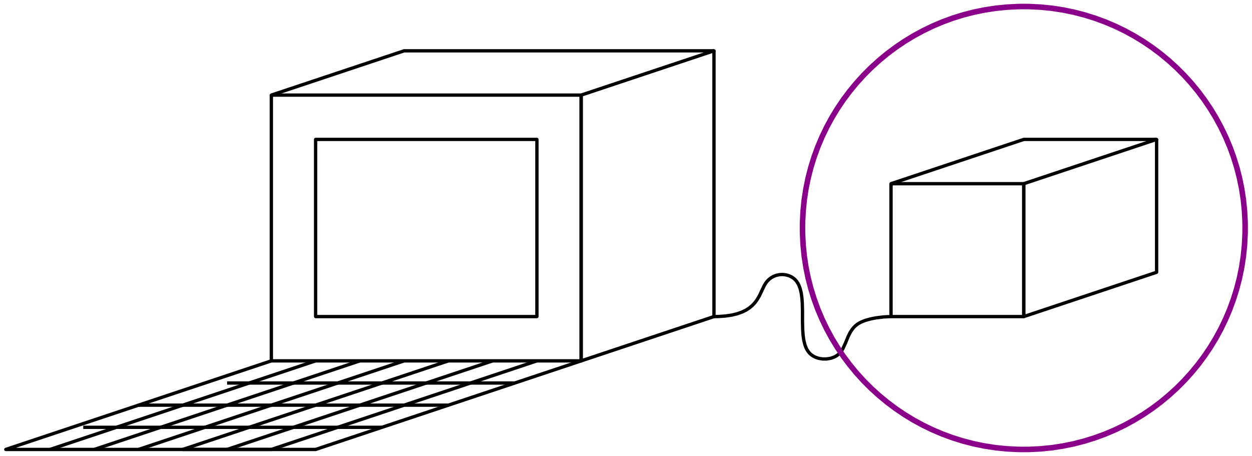
Model of computation



The program lives here

Quantum with Classical Control

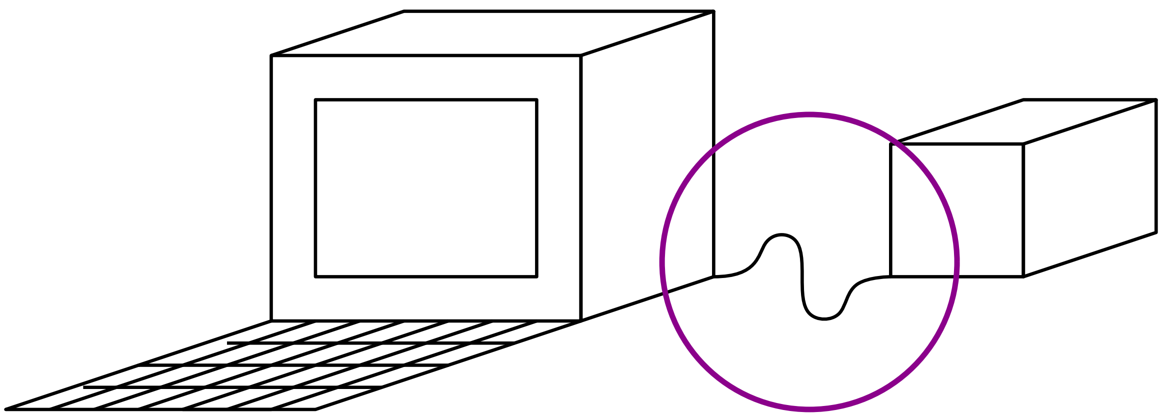
Model of computation



This only holds quantum data

Quantum with Classical Control

Model of computation



Series of instructions/circuit and feedback

A First-Order Quantum Language

Based on the QRAM model [Knill,1996]

- Memory cells are addressable separately
- Instructions/circuit sent down the wire

A flow-chart language [Selinger,2004]

- Global environment of bits and qubits
- Simple actions:
  - Local operations
  - Tests of bit-value, loops using control points

Denotational semantics

- Positive matrices and norm-non-increasing superoperators

Higher-Order Approach

Completely positive maps [SELINGER,V]

- Superoperators without the norm condition
- Compact closed structure

Quantum lambda-calculus [SELINGER,PAGANI,V]

- Adds to the previous approach
  - Higher-order
  - Opaque type `qbit`
  - Duplicable/non-duplicable data
  - Recursive types (e.g. lists)
  - Fixpoints
- Various categorical models extending CPM
- But only low-level QRAM op. for quantum

Circuit Description Languages

In real quantum algorithms

- Circuits described with
  - Explicit series of gates
  - Circuit combinators: Inversion, control, repetition, *etc*
- Need circuits as first-class objects
- QRAM model not enough

Circuit Description Languages

Extending the quantum λ-calculus with [QUIPPER,2013],[QWIRE,2017]

- A new opaque type for circuits: $\texttt{Circ}(A, B)$
- Box and unbox constructions

$$(A \multimap B) \underset{\texttt{unbox}}{\overset{\texttt{box}}{\rightleftarrows}} \texttt{Circ}(A, B)$$

  - Box: instantiate a new circuit
  - Unbox: evaluate a circuit
- A list of fixed, opaque circuits combinators such as

$$\begin{array}{lcl} \texttt{ctl} & : & \texttt{Circ}(A, B) \multimap \texttt{Circ}(\texttt{qbit} \otimes A, \texttt{qbit} \otimes B) \\ \texttt{rev} & : & \texttt{Circ}(A, B) \multimap \texttt{Circ}(B, A) \end{array}$$

- Nice arrow-like, categorical semantics [SELINGER&RIOS,2017]

Circuit Description Languages

The circuit construction is CLASSICAL

- The circuit is built on the classical machine
- It is instantiated on one particular set of qubits
- and applied regardless of the state of the memory.
- The type $\texttt{Circ}(A, B)$ and the circuit combinators are
  - opaque
  - non-programmable

Trying to build circuit combinators

- requires the non-available quantum control

Chapter 2

The Bumpy Road Towards Quantum Control

Quantum Control

Exhibiting the "control flow" hidden in circuits

- "Quantum" tests
- "Quantum" loops/fixpoints

Long-term objective

- Understand the structure of quantum operations
- Syntax for building circuit combinators

Measure of success: How can we say we got quantum control

- Compilation to circuits?
- Modeled as unitary in some Hilbert space?

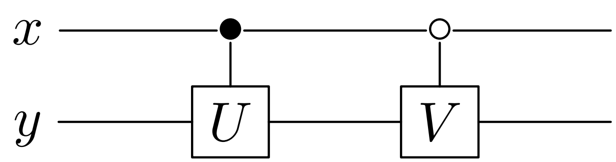
QML

The first successful attempt at implementing a quantum test:

$$\texttt{qif}^{\circ}\ x\ \texttt{then}\ U\,y\ \texttt{else}\ V\,y$$

Perform U or V on y conditionally on x without measuring.

- A naïve compilation approach would do



- if U and V are "orthogonal", one can get rid of x
- The orthogonality property is hard to state
- But QML compiles down to circuits: fully quantum

[Altenkirch&Grattage,2005]

van Tonder's Quantum λ-Calculus

Programs in superposition: [vanTonder,2004]

van Tonder defines a syntactic λ-calculus with

- λ-terms in quantum registers
- β-reduction as unitary operation
- Constants such as 0, 1 and H:

$$H\,0 \quad \longrightarrow \quad \frac{1}{\sqrt{2}}(0+1).$$

The unitarity constraints are too strong

- The terms in superpositions are morally the same
- Turning the language into a purely classical one

Linear algebraic lambda-calculi

A side track to overcome the issue [Arrighi&Dowek,2008],[DiazCaro&al]

- Allow linear combinations of terms (aka "superposition")
  - $\lambda x.M$ is an operator where M can be a linear combination
  - $N(\alpha V + \beta W) \to \alpha(NV) + \beta(NW)$
- Relax the constraints on orthogonality and norm

Advantages

- Full power of λ-calculus
- The β-reduction works fine
- Isolate and study separately problems and solutions

Inconvenient (for this talk)

- Not completely quantum anymore:

  No unitarity nor compilation to circuits

See Alejandro's talk!

Ying's quantum loops

Mingsheng Ying interprets quantum walk using [Ying,2016]

- a quantum, "regular" while-loop
- based on "quantum coins":

```
while (coin toss yields true) do something end
```

The quantum coin is implemented as an element of a Fock space

$$\bigoplus_{n=0}^{\infty} \left(\mathtt{qubit}^{\odot n}\right)$$

An element of $\mathtt{qubit}^{\odot n}$ is a qubit that can be used n times

A while-loop on $\mathtt{qubit}^{\odot n}$

- will loop at most n times
- can behave differently on each superposed qubit state
- the loop "stops quantumly"

This construction still calls for a concrete language

Limits of current approaches

Quantum control flow

- Quantum tests are fine: can be done with regular control
- Quantum loops are the main road block

The following seem slightly incompatible

- Reasonnably expressive language : linear alg. λ-calc
- Satisfactory loops : Ying's approach
- Preservation of unitarity : QML

Chapter 3

Detour via Reversible Computation

[FoSSaCS,2018]

Test with Pattern Matching

```
f : Nat + Nat -> ...
f (Left 0)     -> ...
f (Left n+1)   -> ...
f (Right n)    -> ...
```

```
g : (Bool x Nat) -> ...
g (True, 0)    -> ...
g (False, n)   -> ...
g (True, n+1)  -> ...
```

Test with Pattern Matching

```
f : Nat + Nat -> ...
f (Left 0)     -> ...
f (Left n+1)   -> ...
f (Right n)    -> ...
```

```
g : (Bool x Nat) -> ...
g (True, 0)     -> ...
g (False, n)    -> ...
g (True, n+1)   -> ...
```

```
h : Nat + Nat <-> (Bool x Nat)
h (Left 0)    <-> (True, 0)
h (Left n+1)  <-> (False, n)
h (Right n)   <-> (True, n+1)
```

Conditions for reversible tests [James&Sabry,2014]

- Exhaustivity
- Non-overlapping

Reversible Pattern Matching

Consider the isos $U : a \leftrightarrow c$ and $V : b \leftrightarrow d$.

Then

$$\left\{ \begin{array}{lcl} in_l(x) & \leftrightarrow & in_l(U\,x) \\ in_r(y) & \leftrightarrow & in_r(V\,y) \end{array} \right\} : a \oplus b \leftrightarrow c \oplus d$$

Apply U on values of tpe a and V on values of type b.

Reversible Pattern Matching

Consider the isos $U : a \leftrightarrow c$ and $V : b \leftrightarrow d$.

Written as a circuit: $a \oplus b \leftrightarrow c \oplus d$

$$\left\{ \begin{array}{lcl} in_l(x) & \leftrightarrow & \texttt{do}\ x \text{ —}\boxed{U}\text{— } x' \ \texttt{return}\ in_l(x') \\ in_r(y) & \leftrightarrow & \texttt{do}\ y \text{ —}\boxed{V}\text{— } y' \ \texttt{return}\ in_r(y') \end{array} \right\}$$

Generalized notion of controlled operation

Reversible Pattern Matching

Consider the isos $U : a \leftrightarrow c$ and $V : b \leftrightarrow d$.

Inverting is trivial:

$$\left\{ \begin{array}{lcl} in_l(x') & \leftrightarrow & \texttt{do}\ \ x' \,\text{---}\boxed{U^{-1}}\text{---}\, x\ \ \texttt{return}\ in_l(x) \\ in_r(y') & \leftrightarrow & \texttt{do}\ \ y' \,\text{---}\boxed{V^{-1}}\text{---}\, y\ \ \texttt{return}\ in_r(y) \end{array} \right\}$$

as long as U and V are invertible.

Reversible Pattern Matching

Reversible pattern matching: a syntax for circuits. . .

- with type constructors $\oplus$ and $\otimes$
- generalized notion of control

Following Quipper/QWIRE we add

- recursive types, e.g. $[a] \equiv 1 \oplus (a \otimes [a])$
- higher-order on isos :
  - iso-variables
  - boxes in circuits can be iso-variables
  - lambda-abstractions $\lambda f.\{\cdots\}$ and application
  - fixpoints : $\mu f.\{\cdots\}$

Reversible Pattern Matching

Example of "complex" program: the map operation

Let $f : a \leftrightarrow b$

Define `map` $f : [a] \leftrightarrow [b]$ as

$$\mu \mathbf{g}^{[a]\leftrightarrow[b]}. \left\{ \begin{array}{lcl} [] & \leftrightarrow & [] \\ h : t & \leftrightarrow & \texttt{do} \begin{array}{l} h \;\text{—}\boxed{f}\text{—}\; h' \\ t \;\text{—}\boxed{\mathbf{g}}\text{—}\; t' \end{array} \;\texttt{return}\ h' : t' \end{array} \right\}$$

Reversible Pattern Matching

Operational semantics:

- Substituting and circuit unfolding
- The "generated circuit" depends on the shape of the input value
  - `map f` on a list of size 0 : the identity
  - `map f` on a list of size 1 : the map `f`
  - `map f` on a list of size n : `f` applied on each wire
- Inversion is still obtained syntactically

Reversible Pattern Matching

In summary, non-overlapping and exhaustivity give

- syntactic inverses
- generalized controls and tests
- fixpoints

Is everything done?

Reversible Pattern Matching

Wait! Why are fixpoints yielding exhaustive patterns?

They are not in general but

- If the fixpoint is non-terminating
  - partial injective maps
  - original semantics of Theseus [James&Sabry,2014]
  - link with inverse categories [Kaasgaard&al 2017]
- As long as the fixpoint is terminating on all inputs:

  The iso describes a bijective map on the sets of values

Isos as Unitary Maps

How to go from classical reversible to quantum?

- linear combination of data $\equiv$ some sort of side effect
- In the spirit of the monad steaming from the adjunction

$$\begin{tikzcd} \mathrm{FinSet} \arrow[r, bend left] & \mathrm{FinVec} \arrow[l, bend left] \end{tikzcd}$$

- Following what is done in the linear algebraic lambda-calculi

$$M\,(\alpha \cdot u + \beta \cdot v) \equiv \alpha \cdot (M\,u) + \beta \cdot (M\,v)$$

- Caveat: vector spaces + unitary maps is not a coKleisli category
  - (If you found a way, tell me, I'm eager to know!)

Isos as Unitary Maps

Consider the finite types a, b and their sets of values $|a|$ and $|b|$:

- An iso $U : a \leftrightarrow b$ is a bijection between $|a|$ and $|b|$

Consider the vector space $\mathbb{C}^{|a|}$.

- A bijection $|a| \to |b|$ is a 0/1, unitary matrix $\mathbb{C}^{|a| \times |b|}$
- The iso $U : a \leftrightarrow b$ then yields a unitary map $\mathbb{C}^{|a|} \to \mathbb{C}^{|b|}$

Isos as Unitary Maps

Consider again isos $U : a \leftrightarrow c$ and $V : b \leftrightarrow d$, and the map

$$\left\{ \begin{array}{ccc} in_l(x) & \leftrightarrow & in_l(U\,x) \\ in_r(y) & \leftrightarrow & in_r(V\,y) \end{array} \right\} : a \oplus b \leftrightarrow c \oplus d$$

Control over the branch taken for the $\oplus$, yields the matrix:

$$\begin{pmatrix} U & 0 \\ 0 & V \end{pmatrix} : \quad a \oplus b \quad \longrightarrow \quad c \oplus d.$$

Allowing sum and scalar products over terms, one can generalize.

Isos as Unitary Maps

Consider isos

$$U_{11} : a \leftrightarrow c \qquad\qquad U_{21} : b \leftrightarrow c$$
$$U_{12} : a \leftrightarrow d \qquad\qquad U_{22} : b \leftrightarrow d$$

the map $\quad a \oplus b \leftrightarrow c \oplus d$

$$\left\{ \begin{array}{lcl} in_l(x) & \leftrightarrow & \alpha_{11} \cdot in_l(U_{11}\, x) + \alpha_{12} \cdot in_r(U_{12}\, x) \\ in_r(y) & \leftrightarrow & \alpha_{21} \cdot in_l(U_{21}\, y) + \alpha_{22} \cdot in_r(U_{22}\, y) \end{array} \right\}$$

yields a matrix:

$$\begin{pmatrix} \alpha_{11} \cdot U_{11} & \alpha_{21} \cdot U_{21} \\ \alpha_{12} \cdot U_{12} & \alpha_{22} \cdot U_{22} \end{pmatrix} : \quad a \oplus b \quad \longrightarrow \quad c \oplus d.$$

Isos as Unitary Maps

With finite types and the absence of fixpoints

- The previous analysis carries over
- Asking for the unitary of

$$\begin{pmatrix} \alpha_{11} & \alpha_{21} \\ \alpha_{12} & \alpha_{22} \end{pmatrix}$$

  entails that isos describe unitaries
- $|a|$ forms an orthonormal basis for $\mathbb{C}^{|a|}$
- Pattern matching still describes a generalized control

But everything is finite

Chapter 4

The Elusive Quantum Loop

[FoSSaCS,2018]

Recursive Types and Unitary Maps

What about recursive types?

Consider $[a] \;\; \equiv \;\; 1 \oplus (a \otimes [a])$. Unrolling the definition:

$$[a] \;\; \equiv \;\; 1 \oplus a \oplus (a \otimes a) \oplus (a \otimes a \otimes a) \oplus \cdots$$

- Let $U : [a] \leftrightarrow [b]$ defined with
  - Fixpoints
  - linear combinations
  - terminating on all lists v of type a
- For every list $v : a$
  the term Uv reduces to a linear combination of lists of type b.
- $v \perp w$ but do we have $Uv \perp Uw$?
- $\mathbb{C}^{|[a]|}$ is infinite-dimensional...

Isos as Unitary Maps in $\ell^2(|a|)$

A candidate model: The Hilbert space $\ell^2(|[a]|)$

- Space of absolute converging sequences indexed with $|[a]|$.
- Unitary maps U are
  - Surjective
  - Preserving the inner product
  - Bounded: $||Ux|| \leq \rho||x||$ for some $\rho > 0$.

With enough conditions on fixpoints: iterators

- General isos $[a] \leftrightarrow [b]$ yield unitaries $\ell^2(|[a])\to \ell^2(|[b])$

Isos as Unitary Maps in $\ell^2(|a|)$

For example

Consider $\mathtt{map}\,\mathtt{Had}$ of type $[\mathtt{Bool}] \leftrightarrow [\mathtt{Bool}]$

This is the iso

$$\mu \mathbf{g}^{[a]\leftrightarrow[b]}.\left\{\begin{array}{lcl} [] & \leftrightarrow & [] \\ h : t & \leftrightarrow & \mathtt{do}\ \begin{array}{l} h\ \text{---}\boxed{\mathtt{Had}}\text{---}\ h' \\ t\ \text{---}\boxed{\mathbf{g}}\text{---}\ t' \end{array}\ \mathtt{return}\ h' : t' \end{array}\right\}$$

with

$$\mathtt{Had} = \left\{\begin{array}{lcl} \mathtt{true} & \leftrightarrow & \frac{1}{\sqrt{2}} \cdot \mathtt{true} + \frac{1}{\sqrt{2}} \cdot \mathtt{false} \\ \mathtt{false} & \leftrightarrow & \frac{1}{\sqrt{2}} \cdot \mathtt{true} - \frac{1}{\sqrt{2}} \cdot \mathtt{false} \end{array}\right\}$$

Isos as Unitary Maps in $\ell^2(|a|)$

For example

Apply $\mathtt{map}\,\mathtt{Had}$ of type $[\mathtt{Bool}] \leftrightarrow [\mathtt{Bool}]$ on

$$\frac{1}{\sqrt{2}}[\mathtt{true}] + \frac{1}{\sqrt{2}}[\mathtt{true}, \mathtt{true}, \mathtt{true}, \mathtt{true}, \mathtt{true}]$$

and get

$$\frac{1}{\sqrt{2}}\left(\mathtt{map}\,\mathtt{Had}\,[\mathtt{true}]\right) + \frac{1}{\sqrt{2}}\left(\mathtt{map}\,\mathtt{Had}\,[\mathtt{true}, \mathtt{true}, \mathtt{true}, \mathtt{true}, \mathtt{true}]\right)$$

As for Ying's fixpoint, it will "loop"

- 1 time on the first list
- 5 times on the second list

A "natural" notion of quantum loop with "quantum termination"?

Concluding

In this paper

- We reconciliate QML, the linear algebraic lambda-calculus and Ying's approach.
  - Lists of qubits extends quantum coins
  - Quantum while-loops are simply iterators
- We provide a generalized circuit-construction language with quantum control

Concluding

Quantum loops/fixpoints form a meaningful concept

- Indeed, they can be modeled with unitaries

But

Quantum loops/fixpoints makes no sense

- Finite loops are not "real loops": they can be unrolled
- Fixpoints are only interesting on unbounded datatypes
- But circuits on unbounded datatypes require infinite unrolling...
  quantum non-termination is meaningless

Paradoxical?