Quantum Programming Languages

Benoît Valiron (CentraleSupélec / LMF)

Nov 2026

2nd QCOMICAL School @ Pisa

First, some definitions !

Language?

- Expressing computation

Computation?

- Manipulating data
- Packaged in an algorithm

Plan

Quantum data and Quantum computation 2

Structure of Quantum Algorithms 20

Design Choices for Quantum Programming Languages 49

Modeling Functional Quantum Languages 79

Conclusion 112

Plan

Quantum data and Quantum computation 2

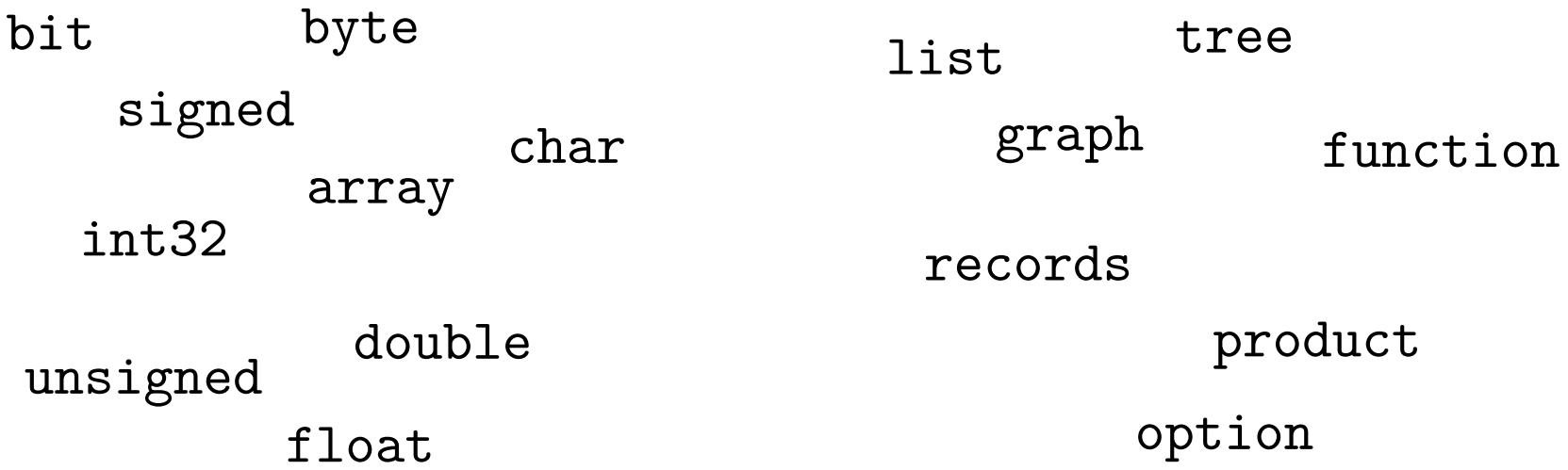
Structure of Quantum Algorithms 20

Design Choices for Quantum Programming Languages 49

Modeling Functional Quantum Languages 79

Conclusion 112

Datatypes



Properties of classical datatypes

- Discrete: sets of elements
- Mostly low-level, linked to the underlying physics

Let us go back to physics then

Dataypes for physicists

At the hardware level Encoding data requires

- a medium
- At least two states of this medium

Examples

- coin – head / tail
- coin – silver / copper
- magnet – north is up / north is down
- light – on / off
- paper – black / white

Beware!

- Easiness to set up, to modify, to read:
  How distinct are the states?
- Stability witu time: noise / error

Low-Level Quantum Datatypes

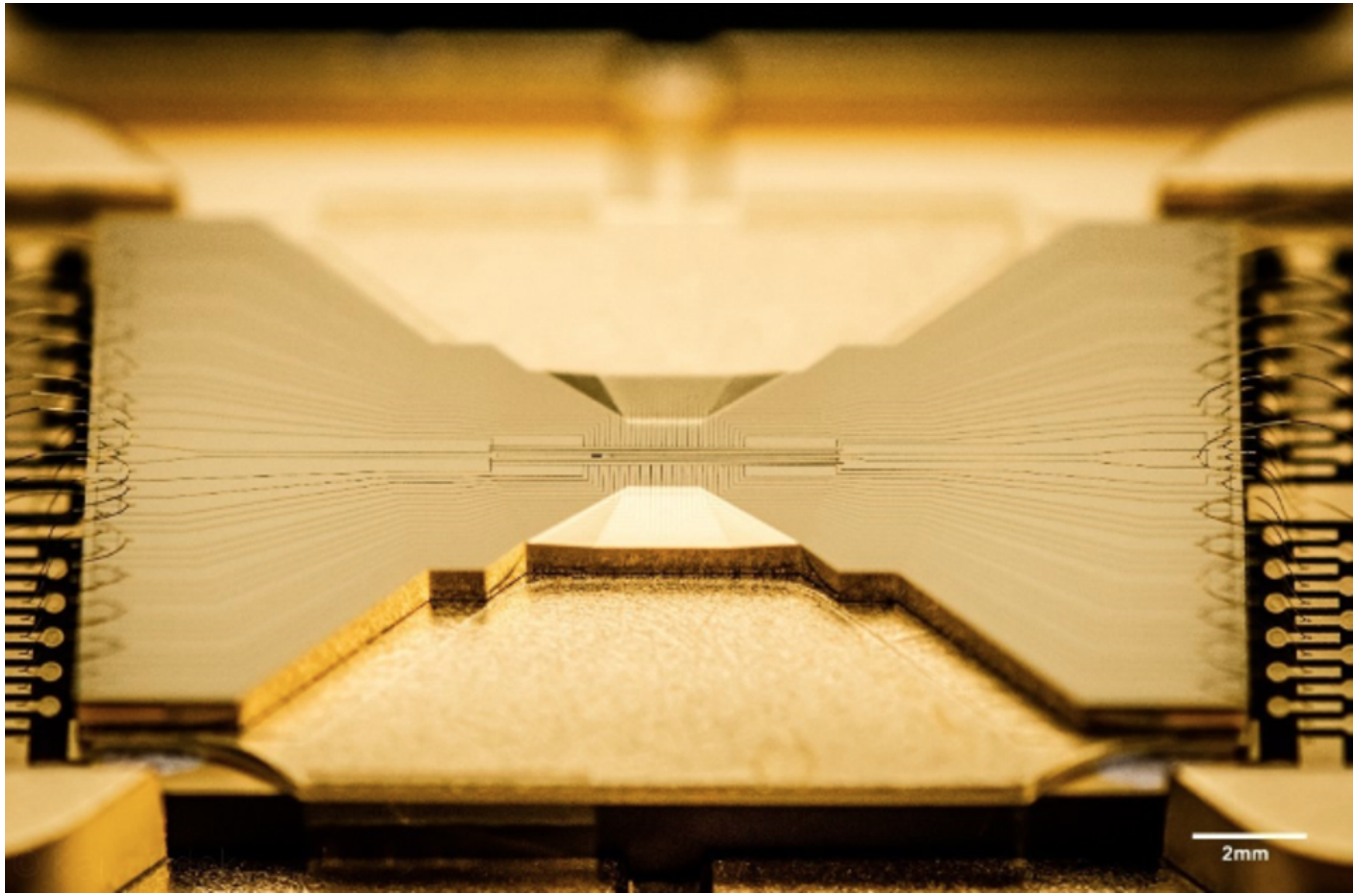
In quantum computation Use state of quantum objects

- photon – polarization (vertical / horizontal)
- photon – location (here / there)
- ion - electronic state
- atom - energy level

Same problems as in the classical case!

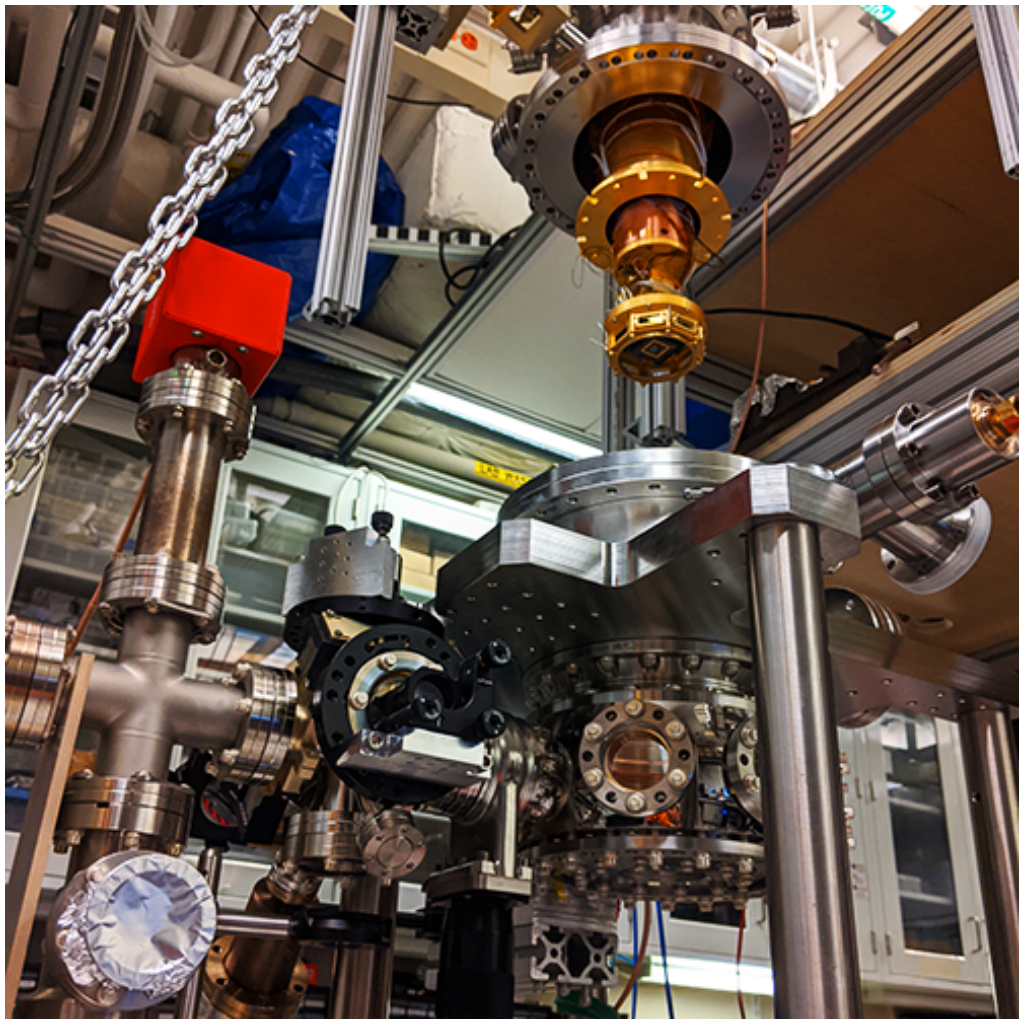
- Easiness to set-up, to read, to modify
- Stability (linked to decoherence)

Example of Quantum Memory



(IonQ memory, 32 qubits)

Example of Quantum Memory



(Same, at MIT: in a really big fridge)

Quantum Memory, Abstraction

	Classical	Quantum
Basic Register	bit	quantum bit = qubit
State	$\mathcal{B} = \{0, 1\}$	$\mathbb{C}^2 = \langle \mathcal{B} \rangle \qquad \alpha \cdot \lvert 0 \rangle + \beta \cdot \lvert 1 \rangle$
3 registers	$\mathcal{B} \times \mathcal{B} \times \mathcal{B}$	$\mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \mathbb{C}^2 = \mathbb{C}^8 = \langle \mathcal{B}^3 \rangle$
Distinguishable?	$\neq$	$\perp$
Math model	Set	Hilbert Spaces

In Quantum

- No-Cloning
- Entanglement:

$$\frac{1}{\sqrt{2}} \cdot (\lvert 00 \rangle + \lvert 11 \rangle)$$

Quantum Memory

Quantum Register on n qubits

- Complex linear combination
- of n-bitstrings

Example with $n = 3$

$$\begin{aligned} & -\tfrac{1}{2} \cdot |0\,0\,0\rangle \\ + \quad & \tfrac{1}{2} \cdot |0\,0\,1\rangle \\ + \quad & \tfrac{i}{2} \cdot |1\,1\,0\rangle \\ - \quad & \tfrac{i}{2} \cdot |1\,1\,1\rangle \end{aligned}$$

with a norm condition.

The state of an n-qubit register is in $\mathbb{C}^{2^n}$.

- superposition of n-bitstrings.
- Unlike probabilistic distributions,

all are available at the same time.

Quantum Memory

Joint state of two quantum registers

In the Kronecker product of the corresponding spaces.

$$\begin{pmatrix} & \frac{1}{\sqrt{2}} \cdot |0\,0\rangle \\ + & \frac{i}{\sqrt{2}} \cdot |1\,1\rangle \end{pmatrix} \otimes \begin{pmatrix} & \frac{1}{\sqrt{3}} \cdot |0\,0\,1\rangle \\ + & \frac{i}{\sqrt{3}} \cdot |1\,0\,0\rangle \\ + & \frac{i}{\sqrt{3}} \cdot |0\,1\,0\rangle \end{pmatrix}$$

Quantum Memory

Joint state of two quantum registers

In the Kronecker product of the corresponding spaces.

$$\begin{aligned} & \frac{1}{\sqrt{2}} \cdot |0\,0\rangle \otimes \left(\begin{array}{cc} & \frac{1}{\sqrt{3}} \cdot |0\,0\,1\rangle \\ + & \frac{i}{\sqrt{3}} \cdot |1\,0\,0\rangle \\ + & \frac{i}{\sqrt{3}} \cdot |0\,1\,0\rangle \end{array} \right) \\ + \quad & \frac{i}{\sqrt{2}} \cdot |1\,1\rangle \otimes \left(\begin{array}{cc} & \frac{1}{\sqrt{3}} \cdot |0\,0\,1\rangle \\ + & \frac{i}{\sqrt{3}} \cdot |1\,0\,0\rangle \\ + & \frac{i}{\sqrt{3}} \cdot |0\,1\,0\rangle \end{array} \right) \end{aligned}$$

Quantum Memory

Joint state of two quantum registers

In the Kronecker product of the corresponding spaces.

$$
\begin{aligned}
& \frac{1}{\sqrt{6}} \cdot |0\,0\rangle \otimes |0\,0\,1\rangle \\
& \frac{i}{\sqrt{6}} \cdot |0\,0\rangle \otimes |1\,0\,0\rangle \\
& \frac{i}{\sqrt{6}} \cdot |0\,0\rangle \otimes |0\,1\,0\rangle \\
+ \quad & \frac{i}{\sqrt{6}} \cdot |1\,1\rangle \otimes |0\,0\,1\rangle \\
& \frac{-1}{\sqrt{6}} \cdot |1\,1\rangle \otimes |1\,0\,0\rangle \\
& \frac{-1}{\sqrt{6}} \cdot |1\,1\rangle \otimes |0\,1\,0\rangle
\end{aligned}
$$

Quantum Memory

Joint state of two quantum registers

In the Kronecker product of the corresponding spaces.

$$
\begin{aligned}
& \frac{1}{\sqrt{6}} \cdot |0\,0\,0\,0\,1\rangle \\
& \frac{i}{\sqrt{6}} \cdot |0\,0\,1\,0\,0\rangle \\
& \frac{i}{\sqrt{6}} \cdot |0\,0\,0\,1\,0\rangle \\
+ \quad & \frac{i}{\sqrt{6}} \cdot |1\,1\,0\,0\,1\rangle \\
& \frac{-1}{\sqrt{6}} \cdot |1\,1\,1\,0\,0\rangle \\
& \frac{-1}{\sqrt{6}} \cdot |1\,1\,0\,1\,0\rangle
\end{aligned}
$$

Quantum Memory

Joint state of two quantum registers
In the Kronecker product of the corresponding spaces.

$$
\begin{aligned}
& \frac{1}{\sqrt{6}} \cdot |0\,0\,0\,0\,1\rangle \\
& \frac{i}{\sqrt{6}} \cdot |0\,0\,1\,0\,0\rangle \\
& \frac{i}{\sqrt{6}} \cdot |0\,0\,0\,1\,0\rangle \\
+ \quad & \frac{i}{\sqrt{6}} \cdot |1\,1\,0\,0\,1\rangle \\
& \frac{-1}{\sqrt{6}} \cdot |1\,1\,1\,0\,0\rangle \\
& \frac{-1}{\sqrt{6}} \cdot |1\,1\,0\,1\,0\rangle
\end{aligned}
$$

of dimension $2^2 \times 2^3 = 2^5 = 32$.

Quantum Memory

Separable state

Of the form $|\texttt{somestate}\rangle \otimes |\texttt{someotherstate}\rangle$

Entangled state

Not separable...

Example

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

Actions on the Quantum Memory

The operations one can perform on the memory are of three kinds:

1. Initialization/creation of a new quantum bit in a given *state*:

$$\begin{array}{rl} & \alpha_0 \cdot |00\rangle \\ + & \alpha_1 \cdot |01\rangle \\ + & \alpha_2 \cdot |10\rangle \\ + & \alpha_3 \cdot |11\rangle \end{array} \longmapsto \begin{array}{rl} & \alpha_0 \cdot |000\rangle \\ + & \alpha_1 \cdot |010\rangle \\ + & \alpha_2 \cdot |100\rangle \\ + & \alpha_3 \cdot |110\rangle \end{array}$$

Actions on the Quantum Memory

The operations one can perform on the memory are of three kinds:

1. Initialization/creation of a new quantum bit in a given *state*:

$$\begin{array}{rl} & \alpha_0 \cdot |00\rangle \\ + & \alpha_1 \cdot |01\rangle \\ + & \alpha_2 \cdot |10\rangle \\ + & \alpha_3 \cdot |11\rangle \end{array} \longmapsto \begin{array}{rl} & \alpha_0 \cdot |001\rangle \\ + & \alpha_1 \cdot |011\rangle \\ + & \alpha_2 \cdot |101\rangle \\ + & \alpha_3 \cdot |111\rangle \end{array}$$

Actions on the Quantum Memory

The operations one can perform on the memory are of three kinds:

2. Measurement. Measuring first qubit:

$$\begin{array}{rl} & \alpha_0 \cdot |00\rangle \\ + & \alpha_1 \cdot |01\rangle \\ & \\ & \\ & \end{array} \qquad \begin{array}{rl} & \\ & \\ + & \alpha_2 \cdot |10\rangle \\ + & \alpha_3 \cdot |11\rangle \\ & \end{array} \qquad \longmapsto \qquad \left\{ \begin{array}{rll} & \alpha_0 \cdot |00\rangle & \text{Reads the bit “0”} \\ + & \alpha_1 \cdot |01\rangle & \text{(prob. } |\alpha_0|^2 + |\alpha_1|^2\text{)} \\ & & \\ & & \\ & & \\ & \alpha_2 \cdot |10\rangle & \text{Reads the bit “1”} \\ + & \alpha_3 \cdot |11\rangle & \text{(prob. } |\alpha_2|^2 + |\alpha_3|^2\text{)} \end{array} \right.$$

modulo renormalization.

This operation is probabilistic

Actions on the Quantum Memory

The operations one can perform on the memory are of three kinds:

2. Measurement. Measuring second qubit:

$$\begin{array}{lll} & \alpha_0 \cdot |00\rangle & \\ & & + \quad \alpha_1 \cdot |01\rangle \\ + \quad \alpha_2 \cdot |10\rangle & & \\ & & + \quad \alpha_3 \cdot |11\rangle \end{array} \longmapsto \begin{cases} \begin{array}{ll} & \alpha_0 \cdot |00\rangle \\ + & \alpha_2 \cdot |10\rangle \end{array} & \text{Reads the bit “0”} \\ & \text{(prob. } |\alpha_0|^2 + |\alpha_2|^2\text{)} \\ \\ \begin{array}{ll} & \alpha_1 \cdot |01\rangle \\ + & \alpha_3 \cdot |11\rangle \end{array} & \text{Reads the bit “1”} \\ & \text{(prob. } |\alpha_1|^2 + |\alpha_3|^2\text{)} \end{cases}$$

modulo renormalization.

This operation is probabilistic

Actions on the Quantum Memory

The operations one can perform on the memory are of three kinds:

3. Unitary operations. Linear maps
   - preserving norms,
   - preserving orthogonality,
   - reversible.

   E.g. the N-gate on one quantum bit (flip). On the first qubit:

$$\begin{array}{rl} & \alpha_0 \cdot |00\rangle \\ + & \alpha_1 \cdot |01\rangle \\ + & \alpha_2 \cdot |10\rangle \\ + & \alpha_3 \cdot |11\rangle \end{array} \longmapsto \begin{array}{rl} & \alpha_0 \cdot |10\rangle \\ + & \alpha_1 \cdot |11\rangle \\ + & \alpha_2 \cdot |00\rangle \\ + & \alpha_3 \cdot |01\rangle \end{array}$$

Actions on the Quantum Memory

The operations one can perform on the memory are of three kinds:

3. Unitary operations.
   E.g. the Hadamard gate on one quantum bit. Sends

$$\begin{aligned} |0\rangle &\longmapsto \frac{\sqrt{2}}{2}\cdot|0\rangle + \frac{\sqrt{2}}{2}\cdot|1\rangle \\ |1\rangle &\longmapsto \frac{\sqrt{2}}{2}\cdot|0\rangle - \frac{\sqrt{2}}{2}\cdot|1\rangle \end{aligned}$$

   When applied on the first qubit:

$$\begin{array}{rl} & \alpha_0\cdot|01\rangle \\ + & \alpha_1\cdot|10\rangle \end{array} \longmapsto \begin{array}{rl} & \alpha_0\cdot\left(\frac{\sqrt{2}}{2}\cdot|01\rangle + \frac{\sqrt{2}}{2}\cdot|11\rangle\right) \\ + & \alpha_1\cdot\left(\frac{\sqrt{2}}{2}\cdot|00\rangle - \frac{\sqrt{2}}{2}\cdot|10\rangle\right) \end{array}$$

Actions on the Quantum Memory

The operations one can perform on the memory are of three kinds:

3. Unitary operations.
   E.g. the Hadamard gate on one quantum bit. Sends

$$\begin{aligned} |0\rangle &\longmapsto \frac{\sqrt{2}}{2}\cdot|0\rangle + \frac{\sqrt{2}}{2}\cdot|1\rangle \\ |1\rangle &\longmapsto \frac{\sqrt{2}}{2}\cdot|0\rangle - \frac{\sqrt{2}}{2}\cdot|1\rangle \end{aligned}$$

When applied on the first qubit:

$$\begin{array}{rl} & \alpha_0\cdot|01\rangle \\ + & \alpha_1\cdot|10\rangle \end{array} \longmapsto \begin{array}{rl} & \alpha_0\frac{\sqrt{2}}{2}\cdot|01\rangle \\ + & \alpha_0\frac{\sqrt{2}}{2}\cdot|11\rangle \\ + & \alpha_1\frac{\sqrt{2}}{2}\cdot|00\rangle \\ + & \alpha_1\frac{\sqrt{2}}{2}\cdot|10\rangle \end{array}$$

Actions on the Quantum Memory

The operations one can perform on the memory are of three kinds:

3. Unitary operations.
   They can create superposition. . .

$$|1100\rangle \longmapsto \begin{array}{l} \frac{\sqrt{2}}{2} \cdot |1100\rangle \\ + \quad \frac{\sqrt{2}}{2} \cdot |1110\rangle \end{array}$$

. . . or remove it

$$\begin{array}{l} \frac{\sqrt{2}}{2} \cdot |1100\rangle \\ + \quad \frac{\sqrt{2}}{2} \cdot |1110\rangle \end{array} \longmapsto |1100\rangle$$

Actions on the Quantum Memory

The operations one can perform on the memory are of three kinds:

3. Unitary operations.
   They can simulate classical operations:
   - Bit-flip (N-gate).
   - Tests (Controlled operations).
     E.g. Controlled-not. Second qubit is controlling:

$$\begin{array}{rl} & \alpha_0 \cdot |00\rangle \\ + & \alpha_1 \cdot |01\rangle \\ + & \alpha_2 \cdot |10\rangle \\ + & \alpha_3 \cdot |11\rangle \end{array} \longmapsto \begin{array}{rl} & \alpha_0 \cdot |00\rangle \\ + & \alpha_1 \cdot |11\rangle \\ + & \alpha_2 \cdot |10\rangle \\ + & \alpha_3 \cdot |01\rangle \end{array} = \begin{array}{rl} & \alpha_0 \cdot |00\rangle \\ + & \alpha_3 \cdot |01\rangle \\ + & \alpha_2 \cdot |10\rangle \\ + & \alpha_1 \cdot |11\rangle \end{array}$$

Plan

Quantum data and Quantum computation 2

Structure of Quantum Algorithms 20
- The Computational Model 20
- Internal of Quantum Algorithms 24
- Case Studies 38

Design Choices for Quantum Programming Languages 49

Modeling Functional Quantum Languages 79

Conclusion 112

Plan

Quantum data and Quantum computation 2

Structure of Quantum Algorithms 20
- The Computational Model 20
- Internal of Quantum Algorithms 24
- Case Studies 38

Design Choices for Quantum Programming Languages 49

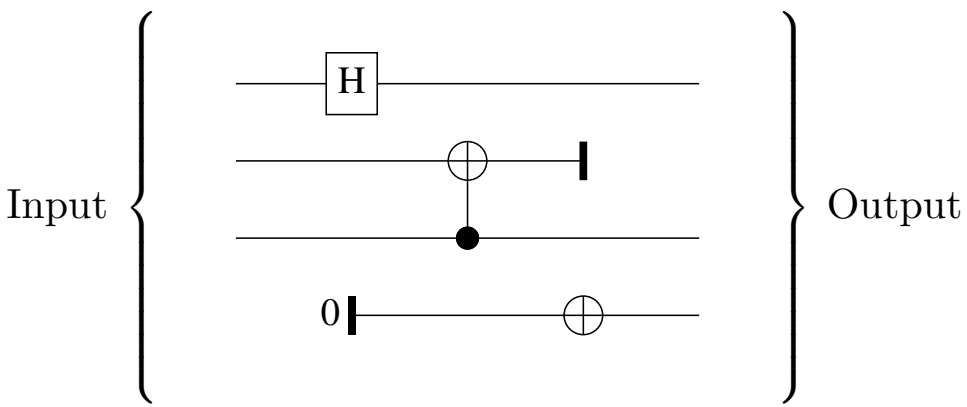
Modeling Functional Quantum Languages 79

Conclusion 112

Circuit Model

Local Actions

- On-the-fly instructions
- Elementary unitaries
- Described by a quantum circuit.




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

No “loop” nor “branching”.

Standard Model: Quantum Co-Processor

The co-processor: A quantum memory

- Quantum registers (qubits) individually addressable

An interface for quantum circuits

- qubit Initializations
  - Return: pointer/address/identification local to the co-processor
- Local, elementary in-place modifications
  - Unitaries on one or two qubits at a time
- Reading of the memory
  - Through mesurement
  - Probabilistic operation
  - Return a classical bitstring

From the perspective of the programmer

- A memory with a strange behavior: no-cloning!
- With a probabilistic reading behavior

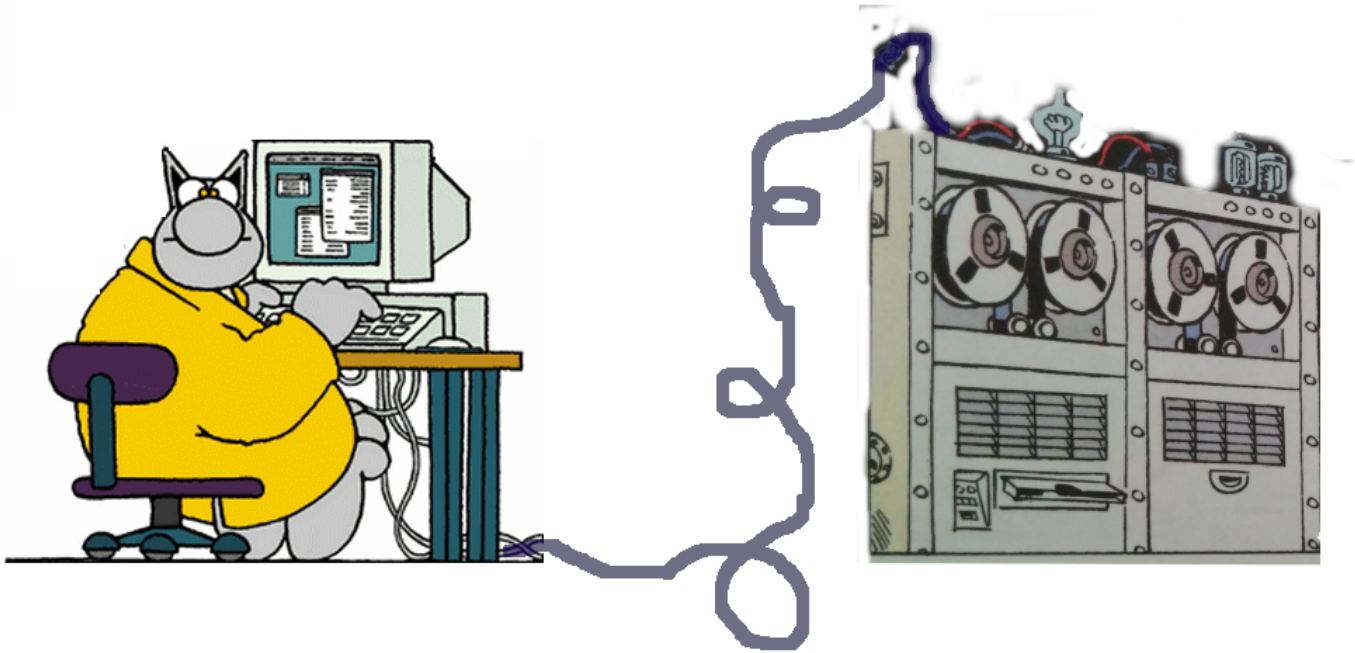
Standard Model: Quantum Co-Processor



Programmer with a laptop

Quantum memory (in a really really large fridge)

Standard Model: Quantum Co-Processor



Computer
Program
Execution Flow

Interface
Initialization
Modifications
Readings

Co-processor
Quantum Memory

Plan

Quantum data and Quantum computation 2

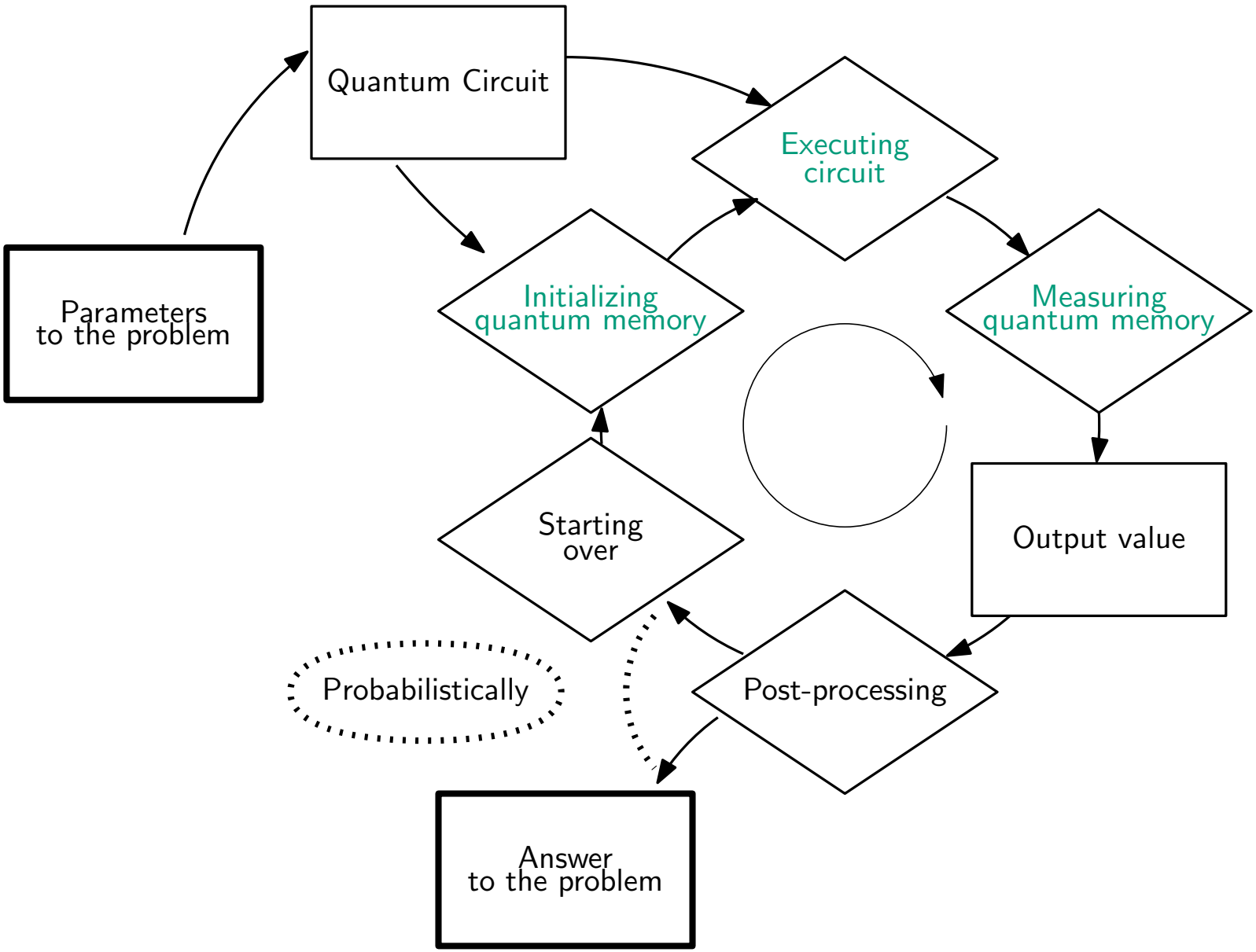
Structure of Quantum Algorithms 20
- The Computational Model 20
- Internal of Quantum Algorithms 24
- Case Studies 38

Design Choices for Quantum Programming Languages 49

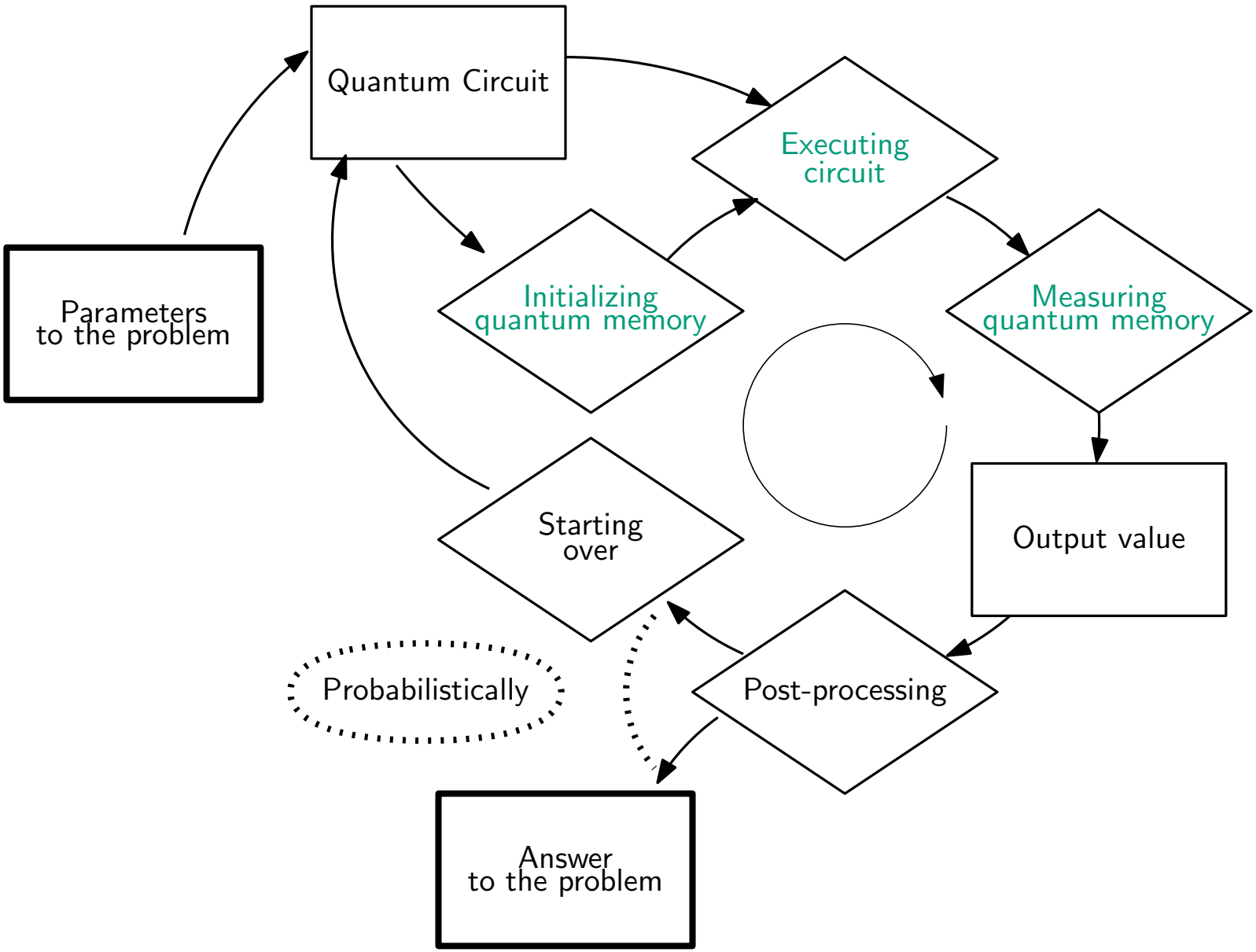
Modeling Functional Quantum Languages 79

Conclusion 112

Structure of (Static) Quantum Algorithms



Structure of (Variational) Quantum Algorithms



Quantum Algorithm, Probabilistic Algorithm

Simple probabilistic algorithm to factor 289884400687823

- Fair draw of a number among $2, 3, 4, 5, \ldots$
- Test: Euclidian division
- Found a factor: success. Otherwise: start over.

Very poor probability of success!

Shor's factorization algorithm

- Probabilistic sampling performed with measurement
- The quantum circuit build a "good" probability distribution.
  $\rightarrow$ boosts factors!

Quantum programming means building circuits
Circuits describe probabilistic distributions

(In case you're wondering: $315697 \cdot 918236159$)

Internals of Current Quantum Algorithms

The techniques used to described quantum algorithms are diverse.

1. Quantum primitives.
   - Quantum Fourier Transform
     Assuming $\omega = 0.\mathtt{xy}$, we want

$$\begin{array}{rl} & (e^{2\pi i\omega})^0 \cdot 00 \\ + & (e^{2\pi i\omega})^1 \cdot 01 \\ + & (e^{2\pi i\omega})^2 \cdot 10 \\ + & (e^{2\pi i\omega})^3 \cdot 11 \end{array} \qquad \longmapsto \qquad 1 \cdot \mathtt{xy}$$

Internals of Current Quantum Algorithms

The techniques used to described quantum algorithms are diverse.

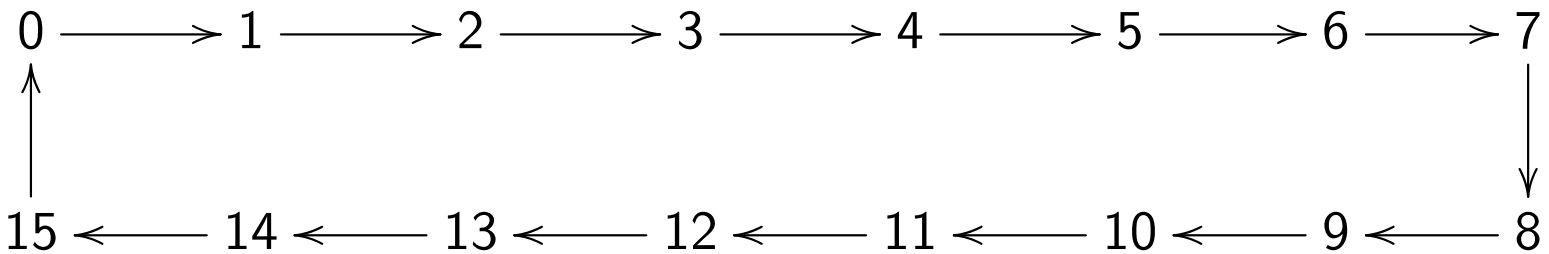
1. Quantum primitives.
   - Phase estimation.
   - Amplitude amplification.
     Qubit 3 in state 1 means good.

$$\begin{array}{cc} & \alpha_0 \cdot 000 \\ + & \alpha_1 \cdot 011 \\ + & \alpha_2 \cdot 100 \\ + & \alpha_3 \cdot 110 \end{array} \longmapsto \begin{array}{cc} & \alpha_0 \cdot 000 \\ + & \alpha_1 \cdot 011 \\ + & \alpha_2 \cdot 100 \\ + & \alpha_3 \cdot 110 \end{array}$$

Internals of Current Quantum Algorithms

The techniques used to described quantum algorithms are diverse.

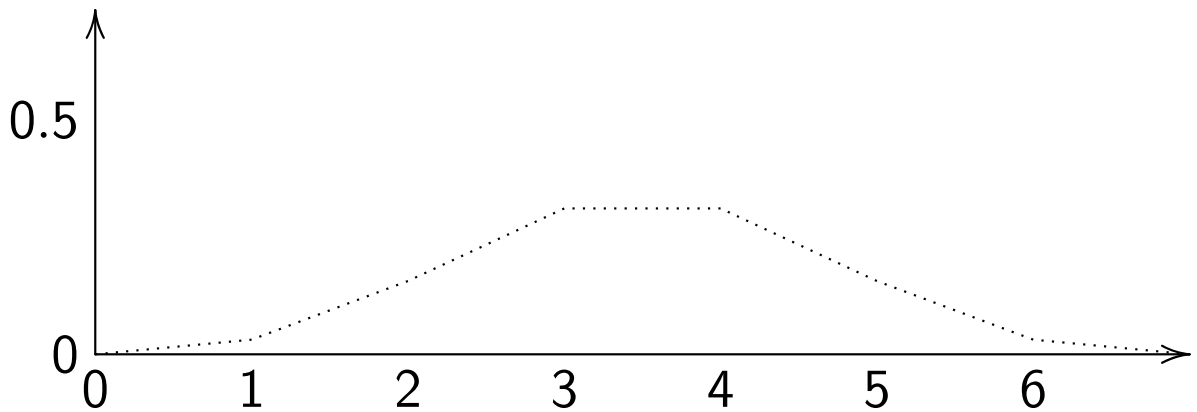
1. Quantum primitives.
   - Quantum Fourier Transform
   - Amplitude amplification.
   - Quantum walk.



Internals of Current Quantum Algorithms

The techniques used to described quantum algorithms are diverse.

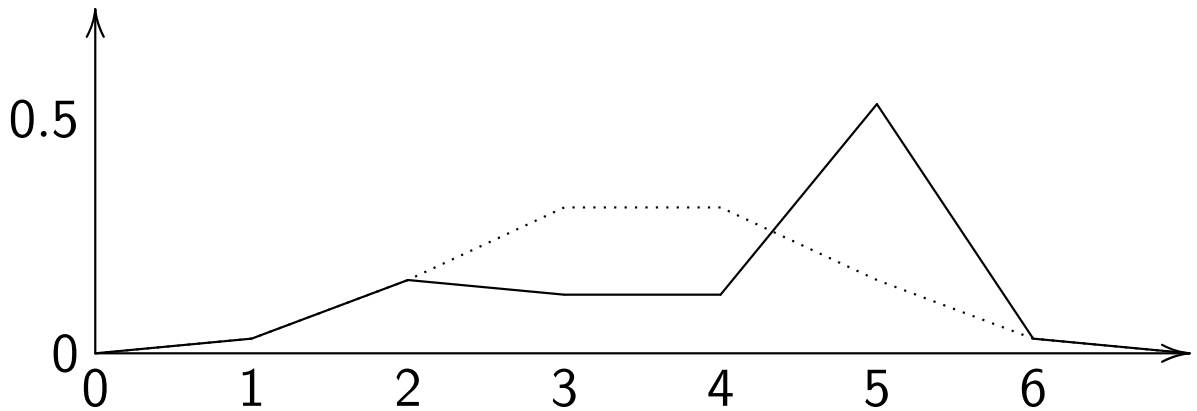
1. Quantum primitives.
   - Quantum Fourier Transform
   - Amplitude amplification.
   - Quantum walk.
     After 5 steps of a probabilistic walk:



Internals of Current Quantum Algorithms

The techniques used to described quantum algorithms are diverse.

1. Quantum primitives.
   - Quantum Fourier Transform
   - Amplitude amplification.
   - Quantum walk.
     After 5 steps of a quantum walk:



Internals of Current Quantum Algorithms

The techniques used to described quantum algorithms are diverse.

1. Quantum primitives.
   - Quantum Fourier Transform
   - Amplitude amplification
   - Quantum walk
   - Hamiltonian simulation
   - ...

They are given as circuit templates

Internals of Current Quantum Algorithms

The techniques used to described quantum algorithms are diverse.

2. Oracles.
   - Take a classical function $f : \mathtt{Bool}^n \to \mathtt{Bool}^m$.
   - Construct

$$\begin{array}{rccc} \overline{f} : & \mathtt{Bool}^{n+m} & \longrightarrow & \mathtt{Bool}^{n+m} \\ & (x, y) & \longmapsto & (x, y \oplus f(x)) \end{array}$$

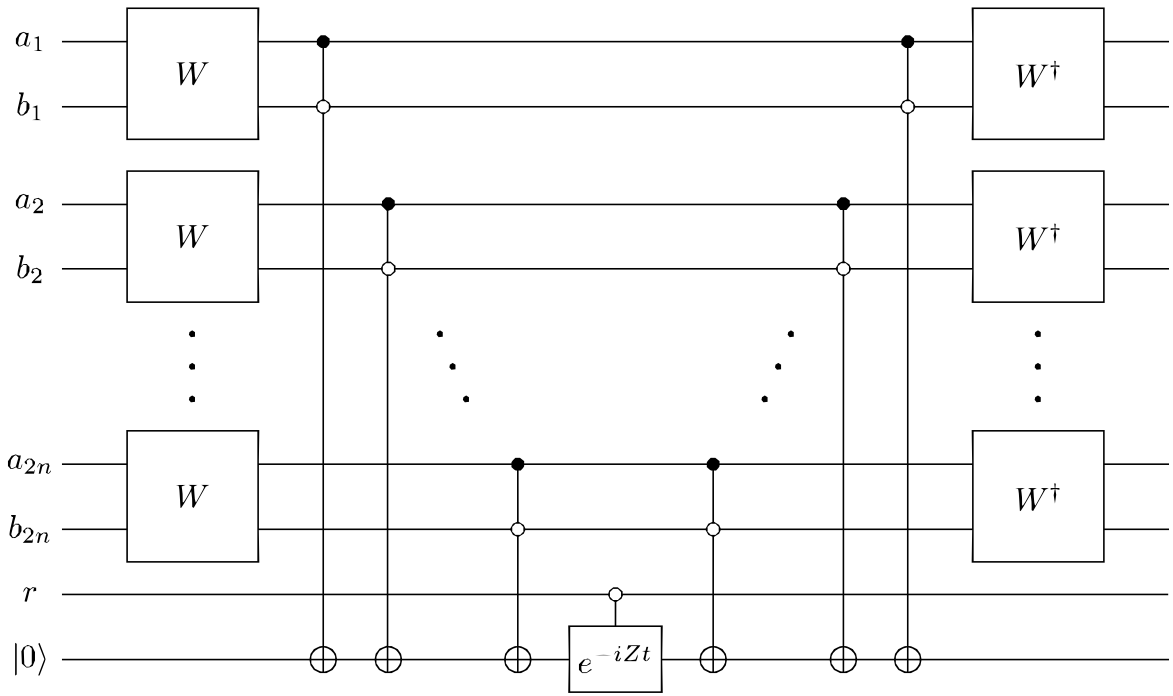
   - Build the unitary U_f acting on $n + m$ qubits computing $\overline{f}$.

Building the circuit depends on how f is given

Internals of Current Quantum Algorithms

The techniques used to described quantum algorithms are diverse.

3. Blocks of loosely-defined low-level circuits.

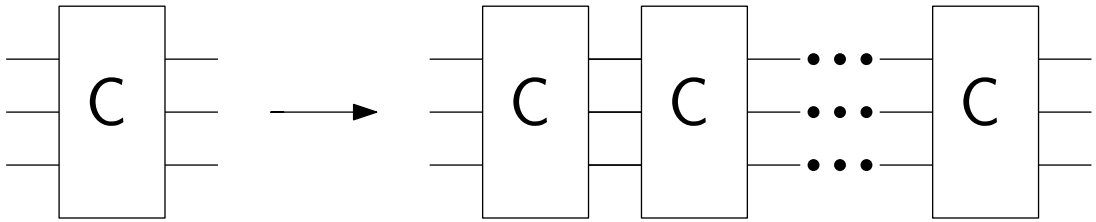


This is not a formal specification!

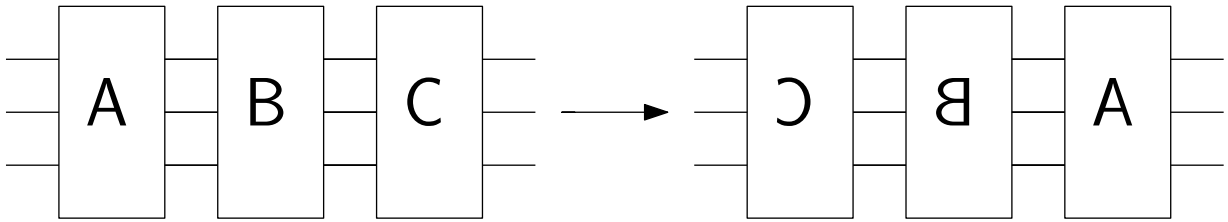
Internals of Current Quantum Algorithms

The techniques used to described quantum algorithms are diverse.

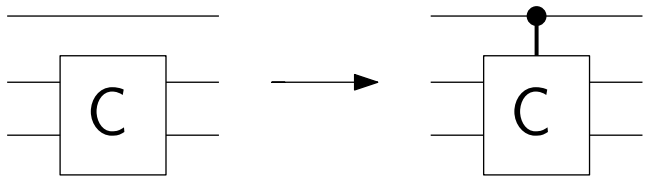
4. High-level operations on circuit:
   - Repetition



   - Inversion



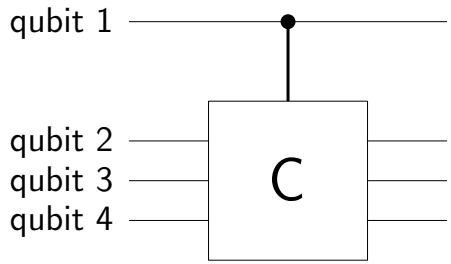
   - Control



Internals of Current Quantum Algorithms

The techniques used to described quantum algorithms are diverse.

4. High-level operations on circuit:
   - Control : conditional action of a circuit



C is applied on qubits 2-4 only when qubit 1 is true:
Suppose that C flips its input bits. The above circuit does

$$\begin{array}{rl} \texttt{qbit} & \texttt{1 234} \\ \frac{1}{\sqrt{2}} & \texttt{1 010} \\ + \frac{1}{\sqrt{2}} & \texttt{0 110} \end{array} \longmapsto \begin{array}{rl} \texttt{qbit} & \texttt{1 234} \\ \frac{1}{\sqrt{2}} & \texttt{1 101} \\ + \frac{1}{\sqrt{2}} & \texttt{0 110} \end{array}$$

This acts as a form of “quantum test”

Internals of Current Quantum Algorithms

The techniques used to described quantum algorithms are diverse.

5. Classical processing.
   - Generating the circuit...
   - Computing the input to the circuit.
   - Processing classical feedback in the middle of the computation.
   - Analyzing the final answer (and possibly starting over).

Plan

Quantum data and Quantum computation 2

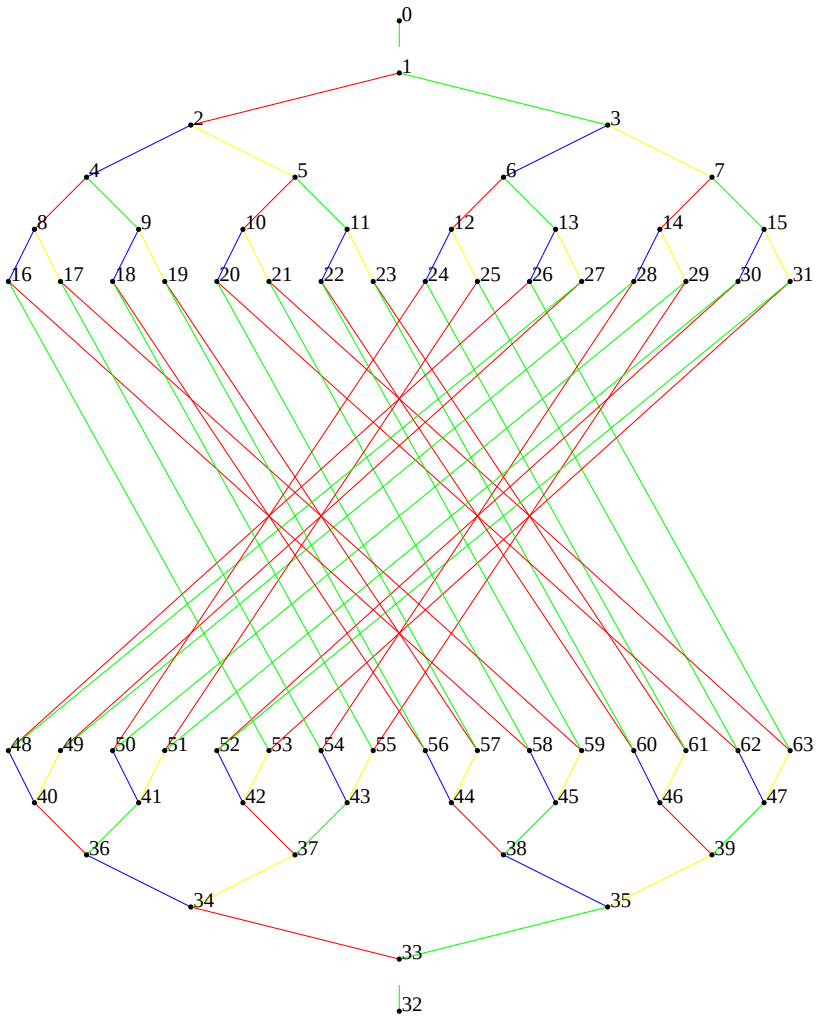
Structure of Quantum Algorithms 20
- The Computational Model 20
- Internal of Quantum Algorithms 24
- Case Studies 38

Design Choices for Quantum Programming Languages 49

Modeling Functional Quantum Languages 79

Conclusion 112

Case study: BWT algorithm



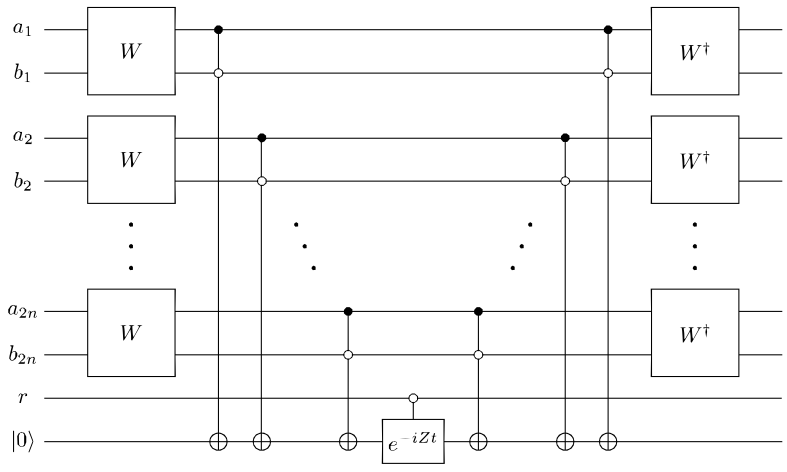
- Start at entrance, look for exit
- Description of the graph:

$$
\begin{array}{lcl}
I & : & \texttt{Node} \\
G & : & \texttt{Color} \times \texttt{Node} \to \texttt{Maybe Node} \\
O & : & \texttt{Node} \to \texttt{Bool}
\end{array}
$$

- Random/Quantum walk
- Parameters:
  height of tree ; number of steps.

Case study: BWT algorithm

- Initialization of a register to the input node (using I)
- 10^6 iterations:
  - Diffuse
  - Call oracle for red
  - Diffuse
  - Call oracle for green
  - Diffuse
  - Call oracle for blue
  - Diffuse
  - Call oracle for yellow



- Measure the node we sit on
- Test with O that we reached the output node.

Case study: QLS algorithm

Considering a vector $\vec{b}$ and the system

$$A \cdot \vec{x} = \vec{b},$$

compute the value of $\langle \vec{x} \,|\, \vec{r} \rangle$ for some vector $\vec{r}$.

Practical situation: the matrix A corresponds to the finite-element approximation of the scattering problem:

Case study: QLS algorithm

For more precision: `arXiv:1505.06552`

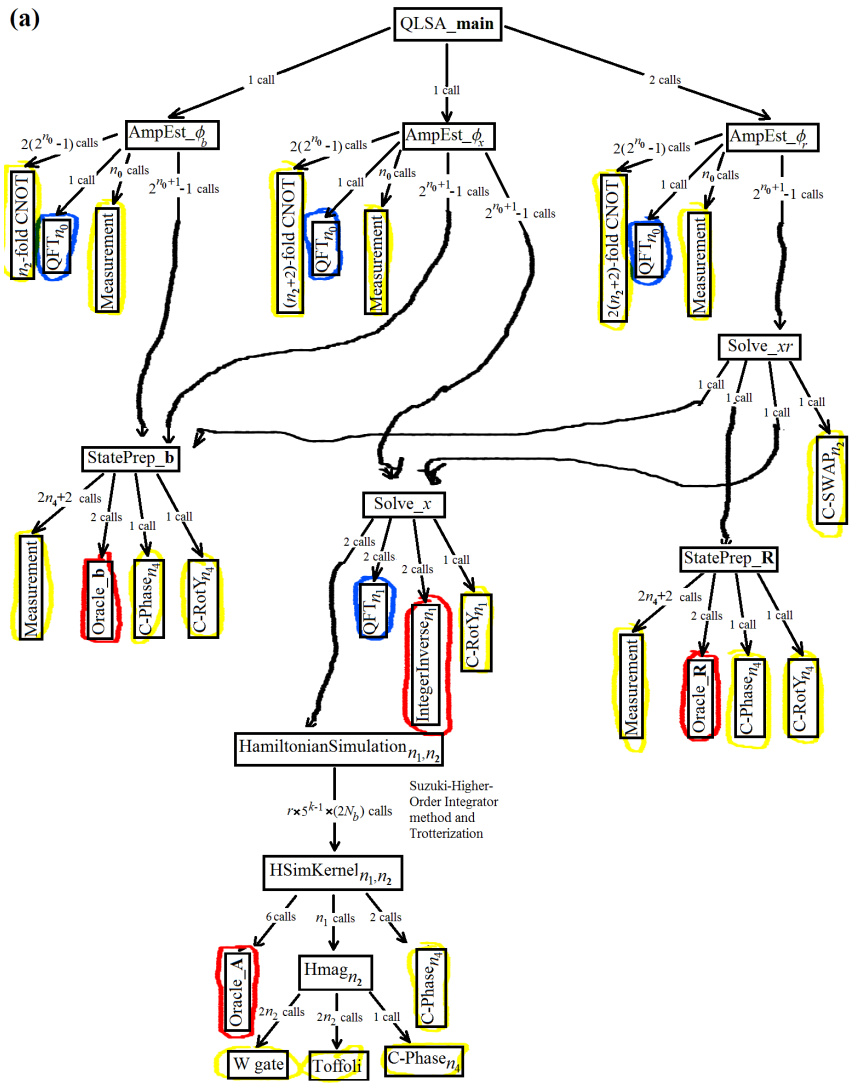
Three oracles:

- for $\vec{r}$ and for $\vec{b}$: input an index, output (the representation of) a complex number
- for A: input two indexes, output also a complex number

It uses many quantum primitives

- Amplitude estimation
- Phase estimation
- Amplitude amplification
- Hamiltonian simulation

Case study: QLS algorithm



- Yellow: Elementary gates.
- Red: Oracles.
- Blue: QFT's.
- Black: Subroutines.
- *Parameters:*
  Dimensions of the space;
  Precision for each of the vectors;
  Allowed error;
  Various parameters for A...
  In total, 19 parameters.

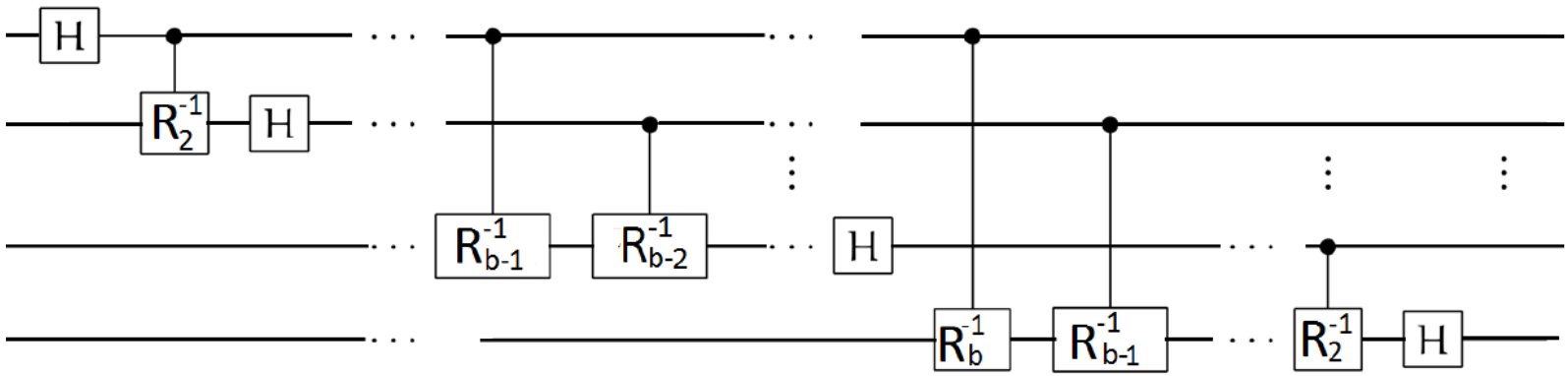
Case study: QLS algorithm

Oracle R is given by the function

```
calcRweights y nx ny lx ly k theta phi =
    let (xc',yc') = edgetoxy y nx ny in
    let xc = (xc'-1.0)*lx - ((fromIntegral nx)-1.0)*lx/2.0 in
    let yc = (yc'-1.0)*ly - ((fromIntegral ny)-1.0)*ly/2.0 in
    let (xg,yg) = itoxy y nx ny in
    if (xg == nx) then
        let i = (mkPolar ly (k*xc*(cos phi)))*(mkPolar 1.0 (k*yc*(sin phi)))*
                ((sinc (k*ly*(sin phi)/2.0)) :+ 0.0) in
        let r = ( cos(phi) :+ k*lx )*((cos (theta - phi))/lx :+ 0.0) in i * r
    else if (xg==2*nx-1) then
        let i = (mkPolar ly (k*xc*cos(phi)))*(mkPolar 1.0 (k*yc*sin(phi)))*
                ((sinc (k*ly*sin(phi)/2.0)) :+ 0.0) in
        let r = ( cos(phi) :+ (- k*lx))*((cos (theta - phi))/lx :+ 0.0) in i * r
    else if ( (yg==1) && (xg<nx) ) then
        let i = (mkPolar lx (k*yc*sin(phi)))*(mkPolar 1.0 (k*xc*cos(phi)))*
                ((sinc (k*lx*(cos phi)/2.0)) :+ 0.0) in
        let r = ( (- sin phi) :+ k*ly )*((cos(theta - phi))/ly :+ 0.0) in i * r
    else if ( (yg==ny) && (xg<nx) ) then
        let i = (mkPolar lx (k*yc*sin(phi)))*(mkPolar 1.0 (k*xc*cos(phi)))*
                ((sinc (k*lx*(cos phi)/2.0)) :+ 0.0) in
        let r = ( (- sin phi) :+ (- k*ly) )*((cos(theta - phi)/ly) :+ 0.0) in i * r
    else 0.0 :+ 0.0
```

Case study: circuit snippets

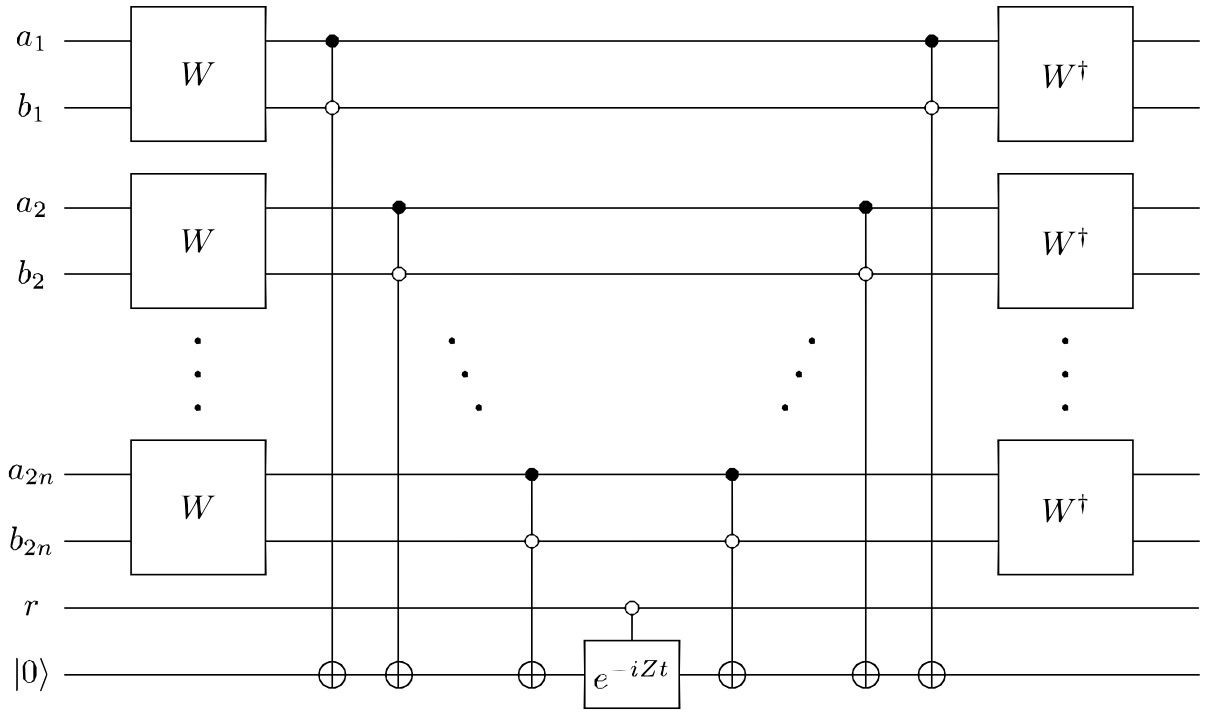
The algorithms create circuits whose sizes and shapes depend on the parameters. E.g. the size of the input register:



(QFT)

Case study: circuit snippets

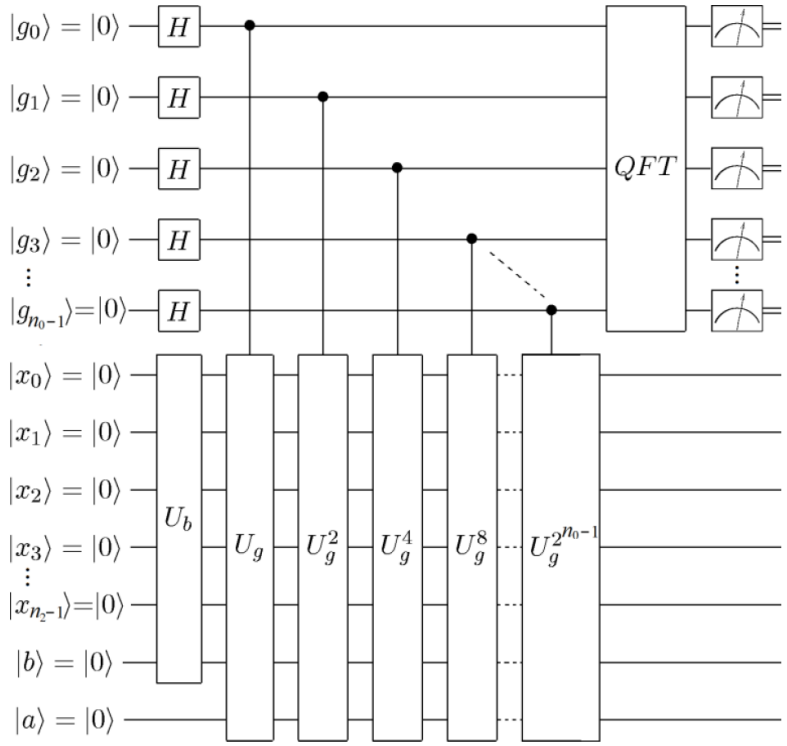
The algorithms create circuits whose sizes and shapes depend on the parameters. E.g. the size of the input register:



(diffusion step in BWT)

Case study: circuit snippets

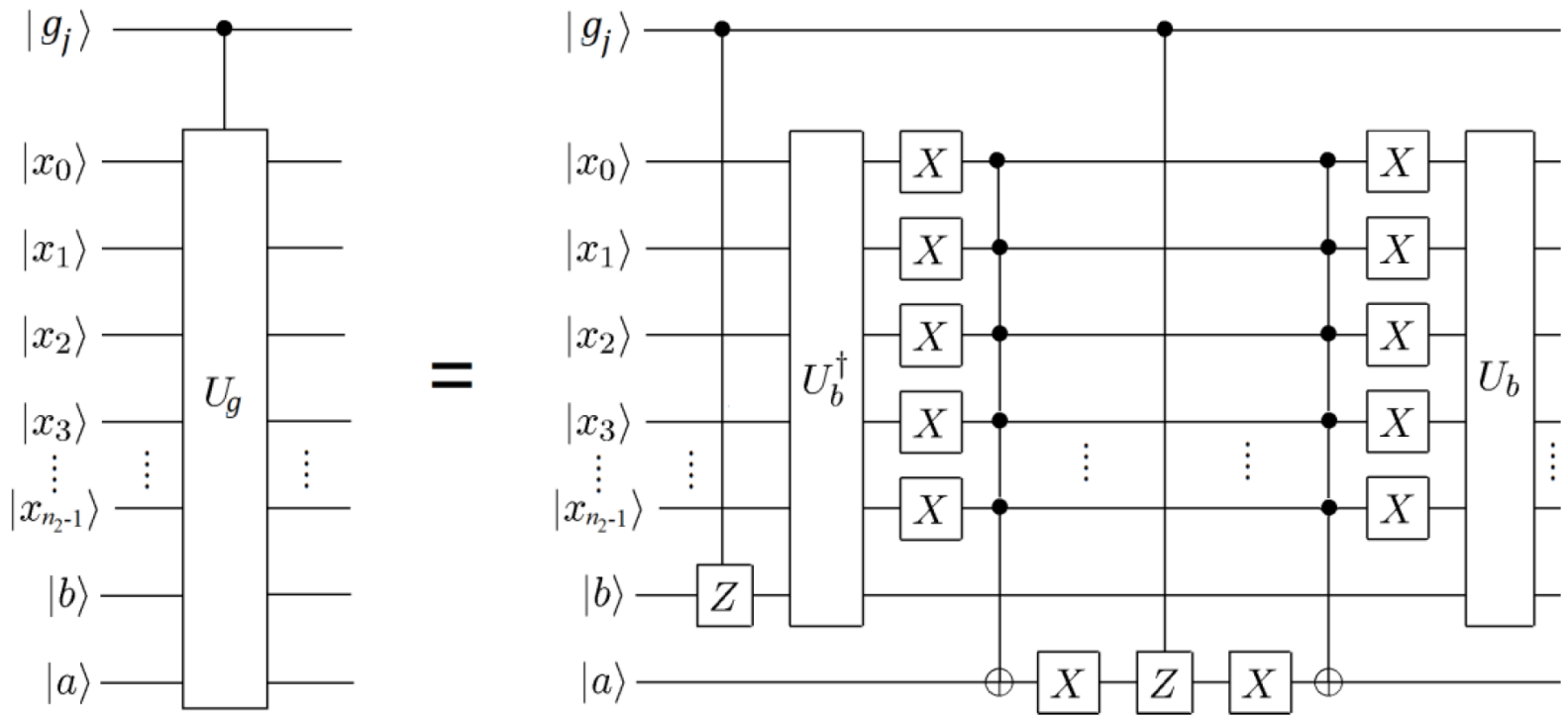
The algorithms create circuits whose sizes and shapes depend on the parameters. E.g. the size of the input register:



(piece of one subroutine of QLS)

Case study: circuit snippets

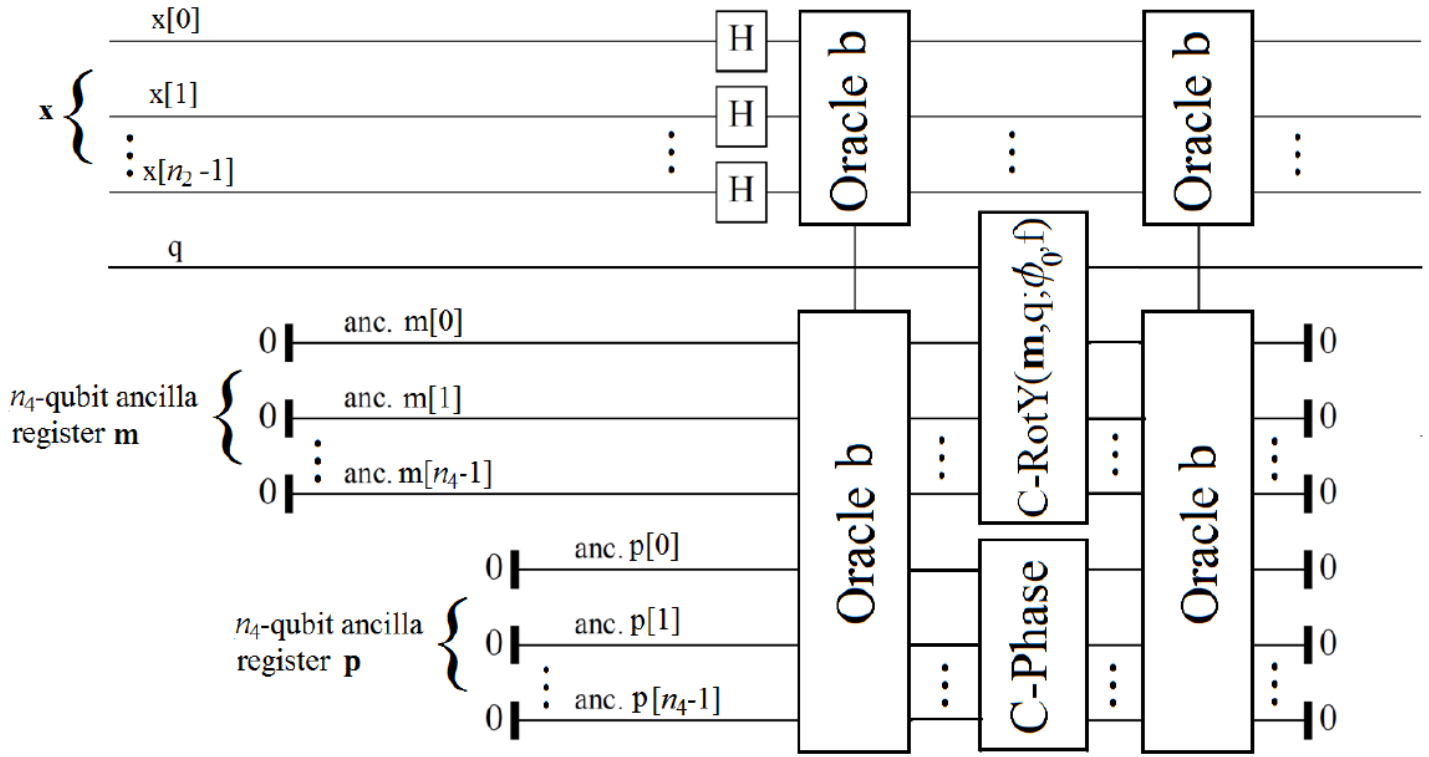
The algorithms create circuits whose sizes and shapes depend on the parameters. E.g. the size of the input register:



(the subroutine U_g)

Case study: circuit snippets

The algorithms create circuits whose sizes and shapes depend on the parameters. E.g. the size of the input register:



(the subroutine U_b)

Plan

Quantum data and Quantum computation 2

Structure of Quantum Algorithms 20

Design Choices for Quantum Programming Languages 49
- Accessing Qubits 50
- Circuits as Functions 54
- Handling Parametricity 66
- Example with BWT 71
- Discussion 74

Modeling Functional Quantum Languages 79

Conclusion 112

Lessons learned

- Circuit construction
  - Procedural: Instruction-based, one line at a time
  - Declarative: Circuit combinators
    - Inversion
    - Repetition
    - Control
    - Computation/uncomputation
- Circuits as inputs to other circuits
- Regularity with respect to the size of the input
- Distinction parameter / input
- Need for automation for oracle generation

Plan

Quantum data and Quantum computation 2

Structure of Quantum Algorithms 20

Design Choices for Quantum Programming Languages 49
- Accessing Qubits 50
- Circuits as Functions 54
- Handling Parametricity 66
- Example with BWT 71
- Discussion 74

Modeling Functional Quantum Languages 79

Conclusion 112

Programming framework

Two approaches

- Circuit as a record
  - One type `circuit`
  - Qubits $\equiv$ wire numbers
  - Native: vertical/horizontal concatenation, gate addition
- Circuit as a function
  - Qubits $\equiv$ first-order objects
  - Input wires $\equiv$ function input
  - Output wires $\equiv$ function output

Circuits as Records

Simplest model: an object holding all of the circuit structure

- Classical wires
- Quantum wires
- List of gates (or directed acyclic graph)
- This is for instance QisKit/QASM model

In this system

- Static circuit
- No high-level hybrid interaction: sequence
  1. circuit generation
  2. circuit evaluation
  3. measure
  4. classical post-processing
  5. back to (1)

Circuits as Records

Procedural construction (QisKit)

```
q = QuantumRegister(5)
c = ClassicalRegister(1)
circ = QuantumCircuit(q,c)

circ.h(q[0])
for i in range(1,5):
  circ.cx(q[0], q[i])
circ.meas(q[4],c[0])
```

- Static ID For registers
- Wires are numbers
- Gate $\equiv$ instruction
- Classical control: Circuit building
- Explicit "run" of circuit

Combinators: return a record `circuit`

- `circ.control(4)`
- `circ.inverse()`
- `circ.append(other-circuit)`

Circuits as Functions

A function (Quipper)

```
a -> Circ b
```

- Inputs something of type `a`
- Outputs something of type `b`
- As a side-effect, generates a circuit snippet.

Or

- Inputs a value of type `a`
- Outputs a computation of type `b`

Plan

Quantum data and Quantum computation 2

Structure of Quantum Algorithms 20

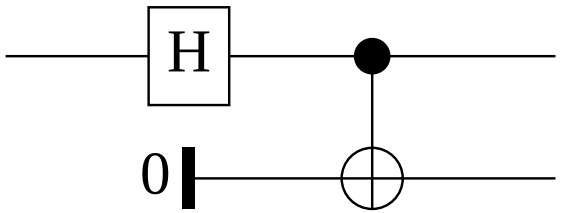
Design Choices for Quantum Programming Languages 49
- Accessing Qubits 50
- Circuits as Functions 54
- Handling Parametricity 66
- Example with BWT 71
- Discussion 74

Modeling Functional Quantum Languages 79

Conclusion 112

Circuits as Functions

The circuit



can be typed with

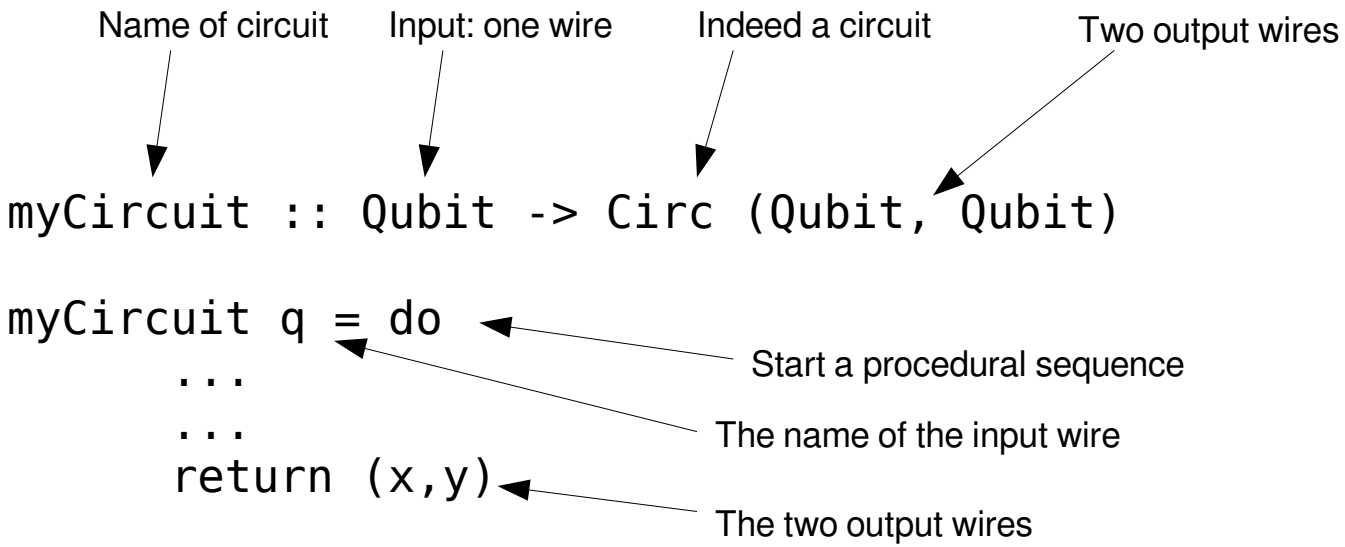
```
Qubit -> Circ (Qubit,Qubit)
```

- Inputs one qubit
- Outputs a pair of qubits
- Spits out some gates when evaluated

The gates are however encapsulated in the function

Circuits as Functions

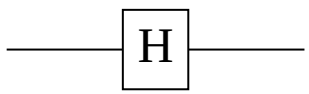
Representing circuits (Quipper)



Circuits as Functions

Procedural presentation of circuits:

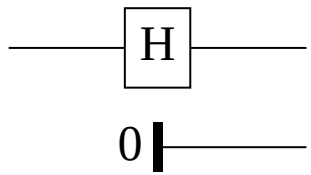
```
prog :: Qubit -> Circ (Qubit,Qubit)
prog q = do
  hadamard_at q
  r <- qinit False
  qnot_at r `controlled` q
  return (q,r)
```



Circuits as Functions

Procedural presentation of circuits:

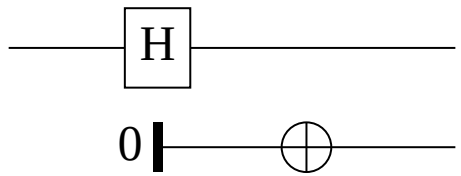
```
prog :: Qubit -> Circ (Qubit,Qubit)
prog q = do
  hadamard_at q
  r <- qinit False
  qnot_at r `controlled` q
  return (q,r)
```



Circuits as Functions

Procedural presentation of circuits:

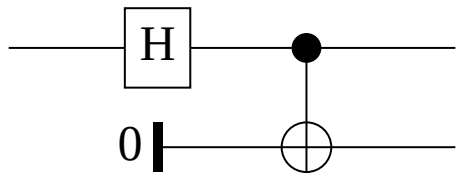
```
prog :: Qubit -> Circ (Qubit,Qubit)
prog q = do
  hadamard_at q
  r <- qinit False
  qnot_at r `controlled` q
  return (q,r)
```



Circuits as Functions

Procedural presentation of circuits:

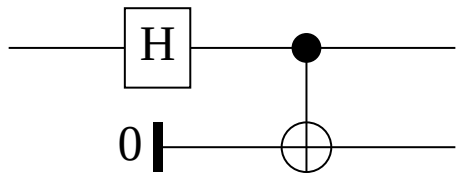
```
prog :: Qubit -> Circ (Qubit,Qubit)
prog q = do
  hadamard_at q
  r <- qinit False
  qnot_at r `controlled` q
  return (q,r)
```



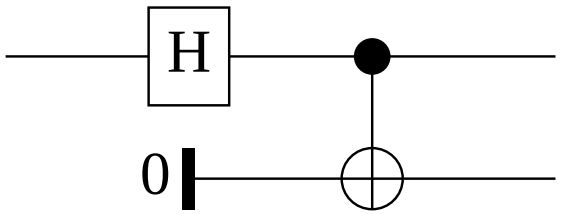
Circuits as Functions

Procedural presentation of circuits:

```
prog :: Qubit -> Circ (Qubit,Qubit)
prog q = do
  hadamard_at q
  r <- qinit False
  qnot_at r `controlled` q
  return (q,r)
```



Circuits as Functions



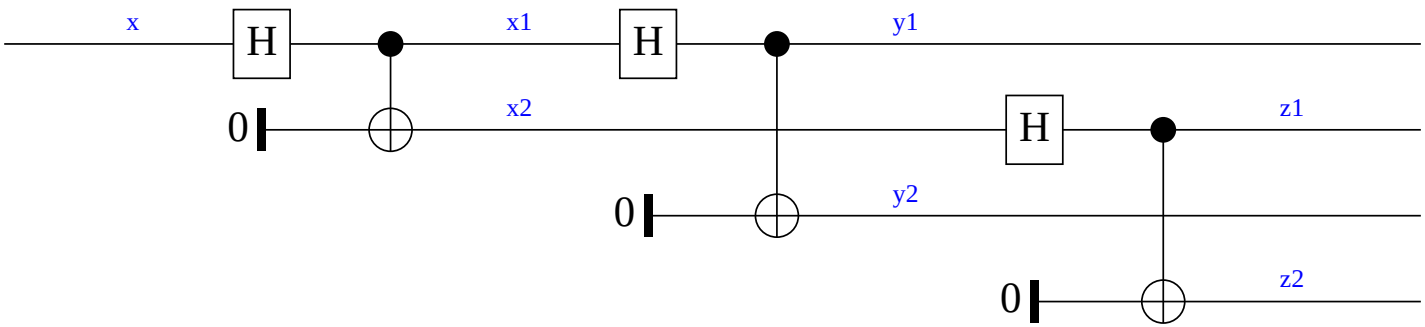

```
import Quipper

circ ::
   Qubit -> Circ (Qubit,Qubit)
circ x = do
   y <- qinit False
   hadamard_at x
   qnot_at y `controlled` x
   return (x,y)
```

- Qubits ≡ first-class variable
- Circuit ≡ function
- Wires ≡ inputs and outputs
- Mix classical/quantum

Circuits as Functions

Wires do not have “fixed” location

```
circ2 :: Qubit -> Circ ()
circ2 x = do
   (x1,x2) <- circ x
   (y1,y2) <- circ x1
   (z1,z2) <- circ x2
   return ()
```



- Qubit $\not\equiv$ Wire number
- Circuits as functions: can be applied
- More expressive types

Circuit Combinators: exercise !

No circuits first What could be the corresponding operations?

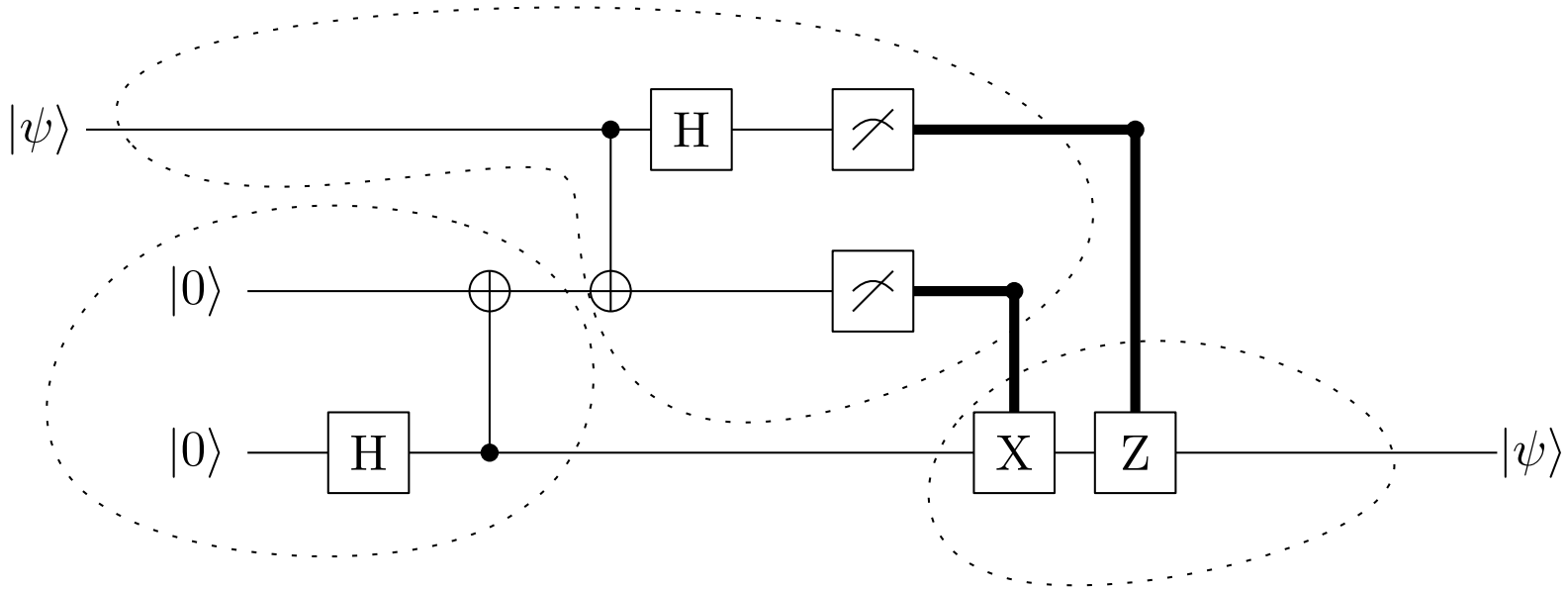
1. `(a -> b) -> (b -> c) -> (a -> c)`
2. `((a,b) -> c) -> (a -> b -> c)`
3. `(a -> b) -> (c -> d) -> ((a,c) -> (b,d))`

With circuits What could be the corresponding operations?

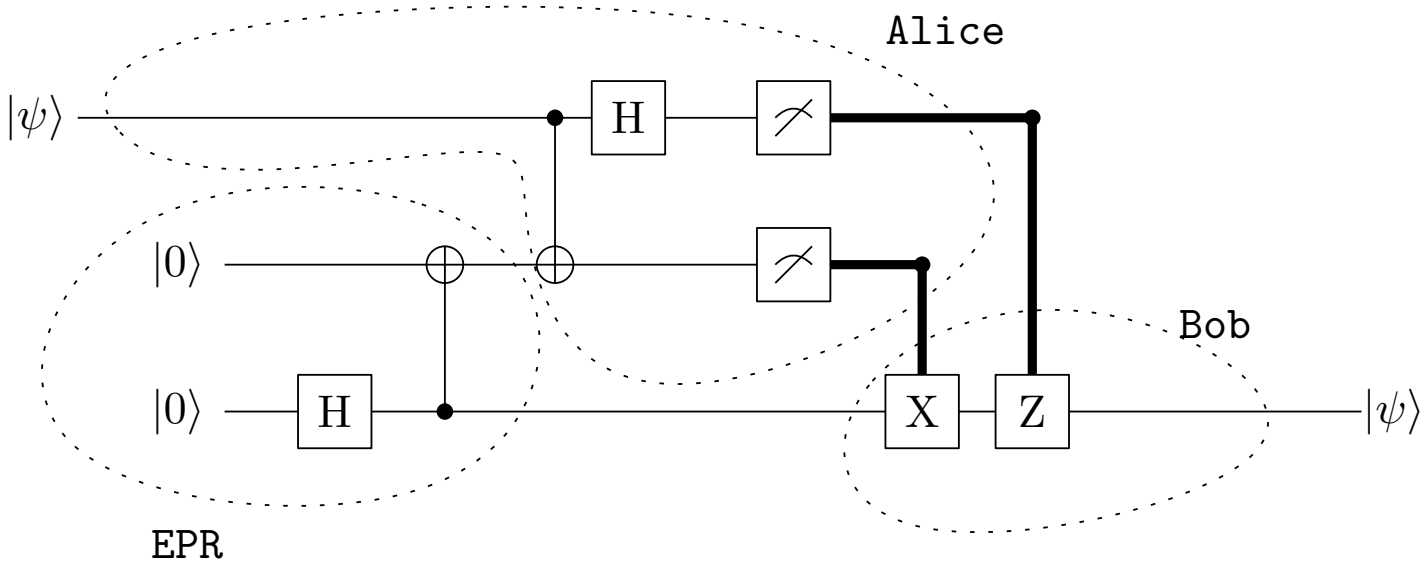
1. `(a -> Circ b) -> (b -> Circ c) -> (a -> Circ c)`
2. `(Circ b) -> (b -> c) -> (Circ c)`
3. `(a -> Circ b) -> (b -> Circ a)`
4. `(a -> Circ b) -> (c -> Circ d)`
   `-> ((a,c) -> Circ (b,d))`
5. `(a -> Circ b) -> ((a,Qubit) -> Circ (b,Qubit))`

Example with Teleportation

Exercise: Decompose according to the dashed sections



Example with Teleportation



can be typed as

- `EPR :: Circ (Qubit, Qubit)`
- `Alice :: Qubit -> Qubit -> Circ (Bit,Bit)`
- `Bob :: Qbit -> (Bit,Bit) -> Circ Qubit`

Composing, we get

```
Circ (Qubit -> Circ (Bit,Bit), (Bit,Bit) -> Circ Qubit)
```

Plan

Quantum data and Quantum computation 2

Structure of Quantum Algorithms 20

Design Choices for Quantum Programming Languages 49
- Accessing Qubits 50
- Circuits as Functions 54
- Handling Parametricity 66
- Example with BWT 71
- Discussion 74

Modeling Functional Quantum Languages 79

Conclusion 112

Families of Circuits

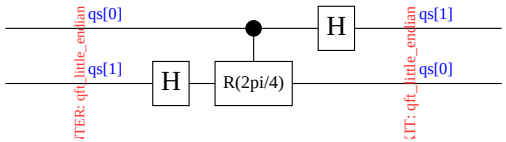
A program

- Inputs classical parameters
- Construct a circuit from these parameters
- Run the circuit

Circuits are parametrized families!

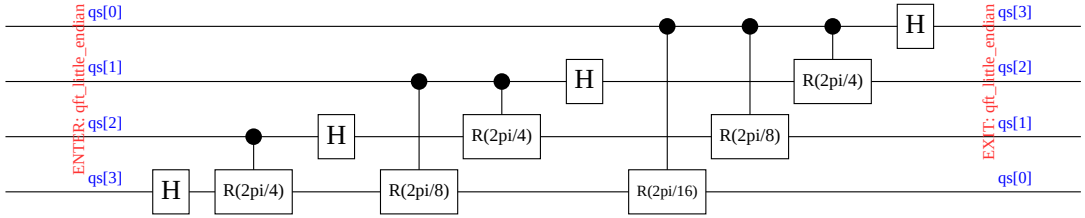
Families of Circuits

Example: QFT



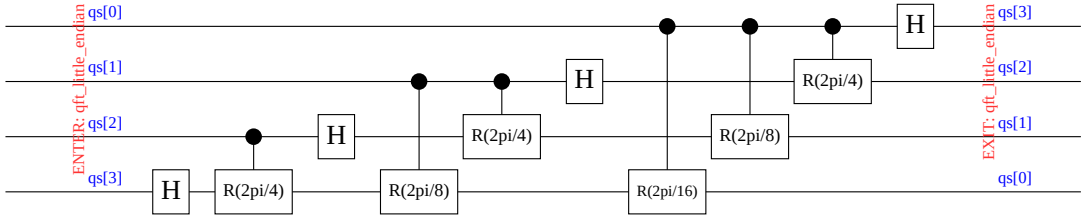
Families of Circuits

Example: QFT



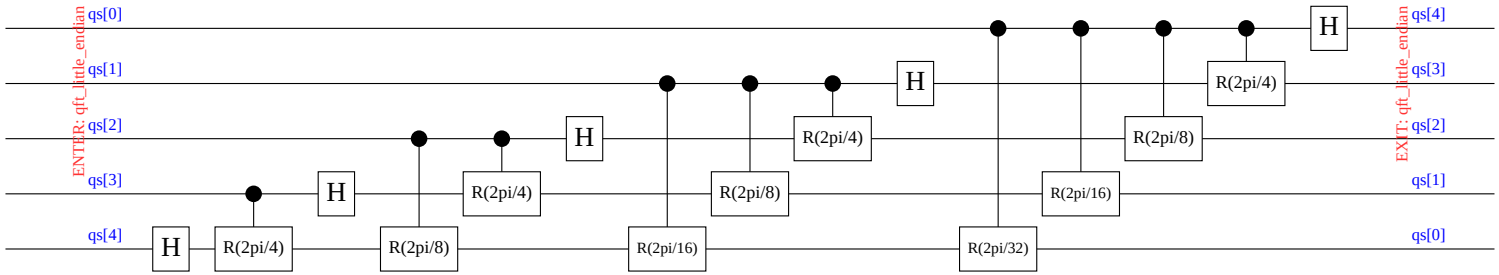
Families of Circuits

Example: QFT



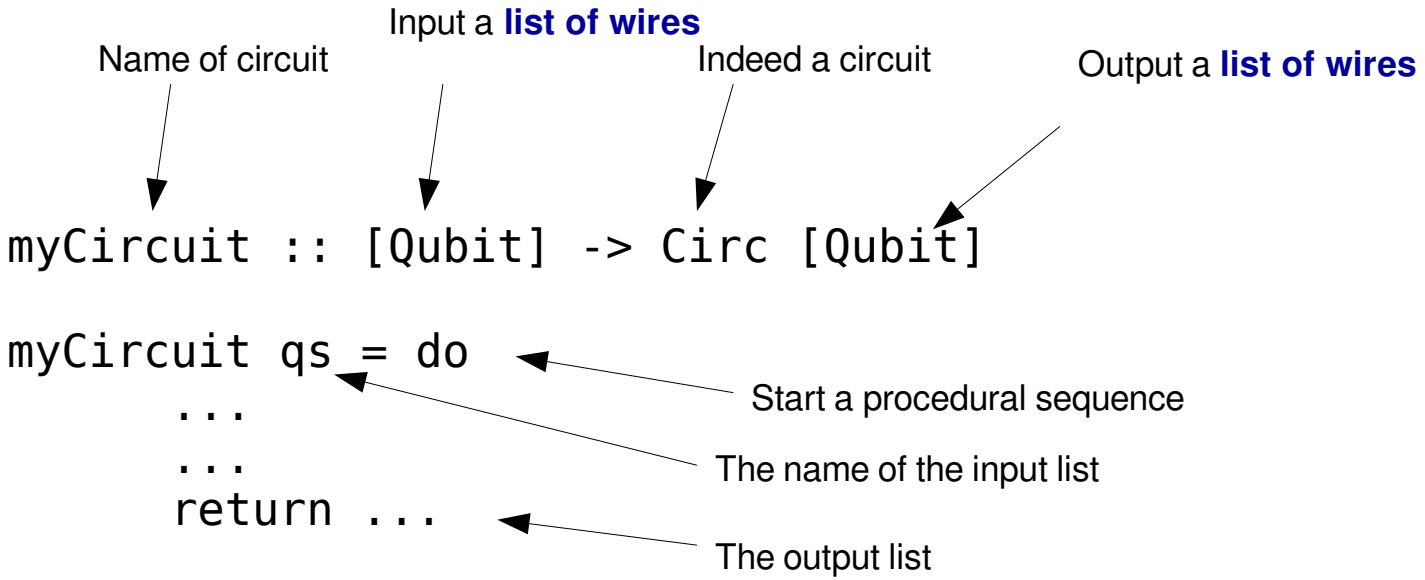
Families of Circuits

Example: QFT



Families of Circuits

With the help of lists:



Families of Circuits

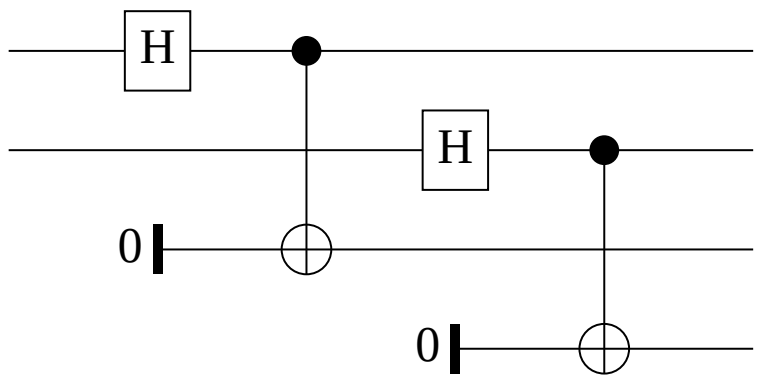
List combinators, e.g.

```
mapM ::  (a -> Circ b) -> [a] -> Circ [b]
```

Mixed presentation of circuits:

```
prog :: Qubit -> Circ (Qubit,Qubit)
prog q = do
  hadamard_at q
  r <- qinit False
  qnot_at r `controlled` q
  return (q,r)

prog2 :: [Qubit] -> Circ [(Qubit,Qubit)]
prog2 l = mapM prog l
```

List of size 2:



Families of Circuits

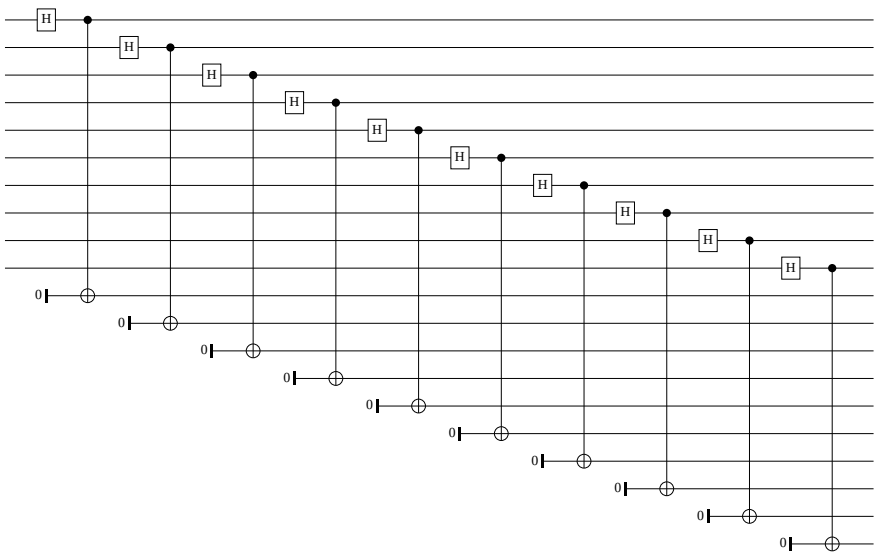
List combinators, e.g.

```
mapM ::  (a -> Circ b) -> [a] -> Circ [b]
```

Mixed presentation of circuits:

```
prog :: Qubit -> Circ (Qubit,Qubit)
prog q = do
  hadamard_at q
  r <- qinit False
  qnot_at r `controlled` q
  return (q,r)

prog2 :: [Qubit] -> Circ [(Qubit,Qubit)]
prog2 l = mapM prog l
```

List of size 10:



Plan

Quantum data and Quantum computation 2

Structure of Quantum Algorithms 20

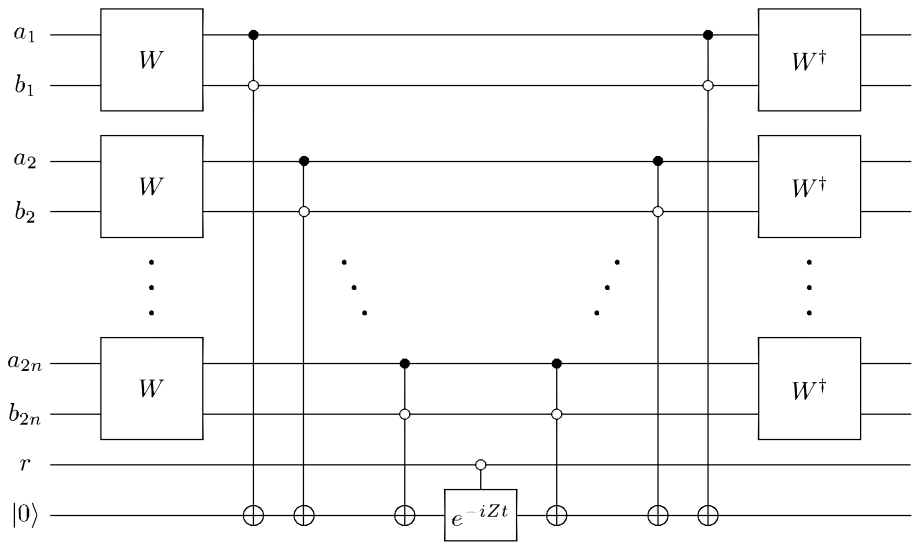
Design Choices for Quantum Programming Languages 49
- Accessing Qubits 50
- Circuits as Functions 54
- Handling Parametricity 66
- Example with BWT 71
- Discussion 74

Modeling Functional Quantum Languages 79

Conclusion 112

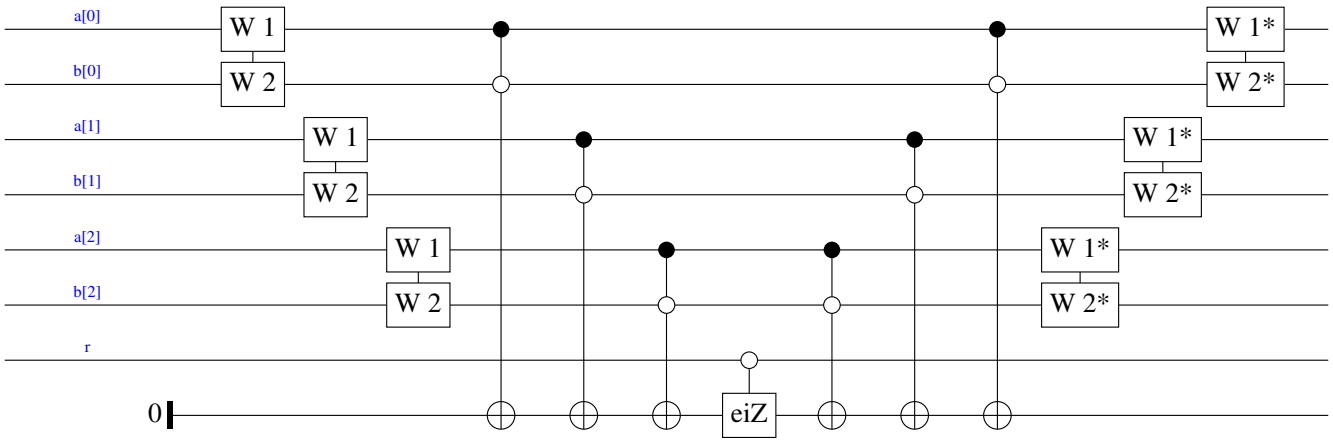
Example: Quipper Code

```
import Quipper

w :: (Qubit,Qubit) -> Circ (Qubit,Qubit)
w = named_gate "W"

toffoli :: Qubit -> (Qubit,Qubit) -> Circ Qubit
toffoli d (x,y) =
  qnot d `controlled` x .==. 1 .&&. y .==. 0

eiz_at :: Qubit -> Qubit -> Circ ()
eiz_at d r =
  named_gate_at "eiZ" d `controlled` r .==. 0

circ :: [(Qubit,Qubit)] -> Qubit -> Circ ()
circ ws r = do
  label (unzip ws,r) (("a","b"),"r")
  d <- qinit 0
  mapM_ w ws
  mapM_ (toffoli d) ws
  eiz_at d r
  mapM_ (toffoli d) (reverse ws)
  mapM_ (reverse_generic w) (reverse ws)
  return ()

main = print_generic EPS circ (replicate 3 (qubit,qubit)) qubit
```



Example: BWT

Result (3 wires):



Example: BWT

Result (30 wires):



Plan

Quantum data and Quantum computation 2

Structure of Quantum Algorithms 20

Design Choices for Quantum Programming Languages 49
- Accessing Qubits 50
- Circuits as Functions 54
- Handling Parametricity 66
- Example with BWT 71
- Discussion 74

Modeling Functional Quantum Languages 79

Conclusion 112

Design Choices: Summary

Requirements for coding Circuits

- Classical structures!
- Hierarchical Representation
- Parametricity
- Non-trivial Combinators

A natural view

- Low Key, First-Order Functions
- Circuit construction seen as a monad

Circuit Construction as a Monad?

Monad: a type constructor `M` equipped with

- `return :: a -> M a`
- `app :: M a -> (a -> M b) -> M b`

Example: the list monad

- Type `[a]` : for lists of elements of type `a`
- `return x = [x]`
- `app [x1, x2, x3] f = (f x1) ++ (f x2) ++ (f x3)`

Circuit Construction as a Monad?

Monad: a type constructor `M` equipped with

- `return :: a -> M a`
- `app :: M a -> (a -> M b) -> M b`

Example: state monad `M a = Int -> (a, Int)`

- `return x = λ n . (x, n)`
- `app g f = λ n . let (y,m) = g n in f y m`

Special combinators

```
get :: M Int
get = λ n . (n,n)

inc :: Int -> M ()
inc n = λ m . ((), m+n)
```

do-notation

```
double = do
    n <- get
    inc n
```

Circuit Construction as a Monad?

Monad: a type constructor `M` equipped with

- `return :: a -> M a`
- `app :: M a -> (a -> M b) -> M b`

Circuit monad: `M a = GateList -> (a, GateList)`

- `return x = λ n .(x, n)`
- `app g f = λ n .let (y,m) = g n in f y m`

Special combinators

```
addGate :: Gate -> Wire -> M ()
addGate g w = λ gs .((), [(g w) added to gs])

qinit :: M Wire
qinit = λ gs .([fresh wire not in gs], gs)
```

Interaction only performed through these combinators

Quantum PL in the wild

Just to name a few

- Quipper (Academic project)
- Q# (Microsoft)
- Silq (ETH Zurich)

And a wealth of Python's libraries

- Cirq (Google)
- myQLM (Eviden)
- Perceval (Quandela)
- Qiskit (IBM)
- and one for about every single company out there

Plan

Quantum data and Quantum computation 2

Structure of Quantum Algorithms 20

Design Choices for Quantum Programming Languages 49

Modeling Functional Quantum Languages 79
- Lambda-Calculus 79
- Incorporating Quantum 95
- Linear Type System 99
- Typing Circuit Combinators 108

Conclusion 112

Plan

Quantum data and Quantum computation 2

Structure of Quantum Algorithms 20

Design Choices for Quantum Programming Languages 49

Modeling Functional Quantum Languages 79

- Lambda-Calculus 79
- Incorporating Quantum 95
- Linear Type System 99
- Typing Circuit Combinators 108

Conclusion 112

Goal

Formalizing

- Functional programming
- Higher-order combinators
- Capable of manipulating quantum information
- And quantum circuits

The first two items

- Realm of lambda-calculus

Lambda-Calculus

- Formal system from Alonzo Church, ∼1930
- Concept of Function and Application
  - "Every term is a function!"
  - Core of functional programming
  - For example: Haskell, OCaml, F#, Lisp, Erlang, *etc.*
- Very simple grammar:
  - Variables $x_1, x_2, x_3, \ldots$
  - Application (binary, infix)
  - Abstraction: $\lambda x.t$ (where t is a term)
- $\lambda x.t$: the function "$x \mapsto t$"

Lambda-Calculus

- Extension of a first-order system
  - Can be extended to other first-order symbols
  - Also to other second-order constructions (like μ-calculus...)
  - But universal
- Notation
  - $\lambda x.t_1\ t_2\ t_3 = \lambda x.((t_1\ t_2)\ t_3)$
  - $\lambda xy.t = \lambda x.\lambda y.t$

Lambda-Calculus

- Notion of bound and free variables
- In $\lambda y.x(\lambda z.z)$:
  - x is free
  - y, z are bound
- Each bound variable is attached to a λ
  - λz . x λx . x (λx . x z)
- The name of bound variables does not matter
  - $\lambda x.x = \lambda y.y$
  - $\lambda xy.x\,(y\,z) = \lambda ab.a\,(b\,z)$
  - Careful! $\lambda x.y \neq \lambda y.y$

Rewriting Rules

- β-reduction: $(\lambda x.t)u \longrightarrow_\beta t[x := u]$
  (! only the x bound by the corresponding λ are replaced)

- A rule that can be added:
  η-reduction: $(\lambda x.tx) \longrightarrow_\eta t$
  (! when x is not free in t)

- Congruence:
  Reduction can occur within a term
  If $M \longrightarrow M'$, then $MN \longrightarrow M'N$
  (Context-free rules)

Example

What are the behaviors of

- $\Omega = (\lambda x.xx)(\lambda x.xx)$?
- $ZZ\ V$ when $Z = \lambda zx.x(zzx)$? (Turing's fixed point combinator)

Church numerals are defined as

- $\overline{0} \triangleq \lambda xy.y$
- $\overline{1} \triangleq \lambda xy.xy$
- $\overline{2} \triangleq \lambda xy.x(xy)$
- $\overline{3} \triangleq \lambda xy.x(x(xy))$

When fed with $\overline{m}$ and $\overline{n}$, what are the behaviors of

- $M = \lambda mn.\lambda xy.mx(nxy)$?
- $N = \lambda mn.m(Mn)(\lambda xy.y)$?

Pure lambda calculus is Turing complete

Link with Functional Programming

$(\lambda x.t)u$ can be interpreted as $\text{let}\, x = u \,\text{in}\, t$

Example with

```
let a = 1 + 2 in
let b = 5 * a * a in
let f = λx . a * x * x in f b
```

(Here we assume that we have integers available somehow)

Evaluation Strategy

- Imagine that "`tic`" evaluates to 0 while emitting... a tic.
- Consider the term:

$$(\lambda x.xx)((\lambda yz.z)\,\texttt{tic})$$

- How many tics does the term emit?

Two standard strategies

- No rewriting under λ's
- Call by name: "As far left as possible" / "As early as possible"
  This is called lazy evaluation
  $\rightarrow$ Evaluation used in Haskell, for example
- Call by value: "As far right as possible"
  Evaluation starts with the arguments
  $\rightarrow$ The standard evaluation: OCaml, F#, etc.

Call-by-Value

```
let a = 1 + 2 in
let b = 5 * a * a in
let f = λx . a * x * x in f b
```

corresponds to the lambda term:

$$(\lambda a.(\lambda b.(\lambda f.fb)(\lambda x.a * x * x))(5 * a * a))(1 + 2)$$

Call by value evaluation:

$$\begin{aligned}
&\to (\lambda a.(\lambda b.(\lambda f.fb)(\lambda x.a * x * x))(5 * a * a))3 \\
&\to (\lambda b.(\lambda f.fb)(\lambda x.3 * x * x))(5 * 3 * 3) \\
&\to (\lambda b.(\lambda f.fb)(\lambda x.3 * x * x))45 \\
&\to (\lambda f.f\ 45)(\lambda x.3 * x * x) \\
&\to (\lambda x.3 * x * x)45 \\
&\to 3 * 45 * 45 \\
&\to 6075
\end{aligned}$$

No reduction under lambdas

Call-by-Value

```
let a = 1 + 2 in
let b = 5 * a * a in
let f = λx . a * x * x in f b
```

corresponds to the code transformation

```
let a = 3 in
let b = 5 * a * a in
let f = λx . a * x * x in f b
```

which rewrites to

```
let b = 5 * 3 * 3 in
let f = λx . 3 * x * x in f b
```

which rewrites to

```
let b = 45 in
let f = λx . 3 * x * x in f b
```

...

Functional Purity

```
let a = 1 + 2 in
let b = 5 * a * a in
let f = λx . a * x * x in
let a = 0 in f b
```

The second a has nothing to do with the first one

- Immutable variables
- Notion of purely functional language
- The environment does not affect the behavior of a function

Functional Purity

```
let f = λx . λy . x * y in (f (1 + 2)) (3 + 4)
```

or

```
let f = λy . λx . x * y in (f (3 + 4)) (1 + 2)
```

The order of argument evaluation does not matter

- First `a` then `b`, or the opposite
- Again linked to functional purity

Simple Type System

Similar a very minimal logic

$$A, B \quad ::= \quad \texttt{nat} \mid A \to B$$

with an opaque type `nat`.

Intuition

- $\lambda x.M$ is typed with $A \to B$ (if well-typed)
- 2 is typed with `nat`
- $+$ is typed with $\texttt{nat} \to \texttt{nat} \to \texttt{nat}$ (modulo infix notation)

Simple Type System

Typing context

$$x_1 : A_1, x_2 : A_2, \ldots x_n : A_n \vdash M : B$$

Typing rules

$$\frac{\Delta, x : A \vdash M : B}{\Delta \vdash \lambda x.M : A \to B} \qquad \frac{\Delta \vdash M : A \to B \quad \Delta \vdash N : A}{\Delta \vdash MN : B}$$

$$\frac{}{\Delta, x : A \vdash x : A}$$

$$\frac{}{+ : \mathtt{nat} \to \mathtt{nat} \to \mathtt{nat}} \qquad \frac{}{2 : \mathtt{nat}} \qquad \text{(one for each } n\text{)}$$

Simple Type System

Typing context

$$A_1, \quad A_2, \ldots \quad A_n \vdash \quad B$$

Typing rules

$$\frac{\Delta, \quad A \vdash \quad B}{\Delta \vdash \quad A \to B} \qquad \frac{\Delta \vdash \quad A \to B \quad \Delta \vdash \quad A}{\Delta \vdash \quad B}$$

$$\frac{}{\Delta, \quad A \vdash \quad A}$$

Curry-Howard correspondence: well-typed terms are proofs

Extensions

Example: Pairing, Boolean values:

$$A, B \quad ::= \quad \texttt{nat} \mid A \to B \mid A \times B \mid \texttt{bit}$$

Typing rules with new term constructs

$$\frac{\Delta, x : A \vdash M : B}{\Delta \vdash \lambda x.M : A \to B} \qquad \frac{\Delta \vdash M : A \to B \quad \Delta \vdash N : A}{\Delta \vdash MN : B}$$

$$\frac{\Delta \vdash M : A \quad \Delta \vdash N : B}{\Delta \vdash \langle M, N \rangle : A \times B} \qquad \frac{\Delta \vdash M : A_1 \times A_2}{\Delta \vdash \pi_i M : A_i}$$

$$\frac{\Delta \vdash P : \texttt{bit} \quad \Delta \vdash M, N : C}{\Delta \vdash \texttt{if } P \texttt{ then } M \texttt{ else } N : C} \qquad \frac{}{\Delta \vdash \texttt{true}, \texttt{false} : \texttt{bit}}$$

Safety Properties

- Subject reduction: type is preserves by reduction
- Progress: well-typed terms either reduce or reached a value

Plan

Quantum data and Quantum computation 2

Structure of Quantum Algorithms 20

Design Choices for Quantum Programming Languages 49

Modeling Functional Quantum Languages 79

- Lambda-Calculus 79
- Incorporating Quantum 95
- Linear Type System 99
- Typing Circuit Combinators 108

Conclusion 112

One problem: Entanglement

Let us add

- an opaque type `qbit`
- constants $|\phi\rangle$ for all possible states

We can do

$$\lambda f.(f\,|0\rangle)\,|1\rangle : (\texttt{qbit} \to \texttt{qbit} \to A) \to A$$

but what if the two states are entangled:

$$\lambda f.(f q_1) q_2 : (\texttt{qbit} \to \texttt{qbit} \to A) \to A$$

where q_1, q_2 is in state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$?

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:

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

- Call-by-value evaluation strategy

  (Reduction strategy linked to the type system!)

- Quantum operations through the evaluation strategy

Quantum Lambda-Calculus

$$[\alpha\,|0\rangle + \beta\,|1\rangle\,, |x\rangle\,, \texttt{let y = qinit false in CNOT } \langle \texttt{x, y}\rangle]$$

reduces to

$$[\alpha\,|00\rangle + \beta\,|11\rangle\,, |xy\rangle\,, \langle x, y\rangle]$$

Another Problem: Non-Duplicability

Consider the following

$$\left[\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), |x\rangle\,, \langle \mathtt{meas}\, x, \mathtt{had}\, x\rangle\right]$$

In a purely functional world, the order should not matter!

Notion of linear type system

- Quantum data is non-duplicable
- Type system based on linear logic

Plan

Quantum data and Quantum computation 2

Structure of Quantum Algorithms 20

Design Choices for Quantum Programming Languages 49

Modeling Functional Quantum Languages 79
- Lambda-Calculus 79
- Incorporating Quantum 95
- Linear Type System 99
- Typing Circuit Combinators 108

Conclusion 112

Quantum Memory

Mathematical Structure

- Quantum register $\equiv$ (finite) Hilbert space
- Juxtaposition $\equiv$ Kronecker (tensor) product
- Reading $\equiv$ measure $\equiv$ getting a bit, probabilistic
- Quantum information is non-duplicable

Type Structure

- Based on (Intuitionistic) Multiplicative Linear Logic

$$A, B \quad ::= \quad \texttt{qbit} \mid \texttt{bit} \mid A \otimes B$$

- Entanglement:

$$[\tfrac{1}{\sqrt{2}}(|00\rangle + |11\rangle), |xy\rangle, \langle x, y\rangle] \quad : \quad \texttt{qbit} \otimes \texttt{qbit}$$

Quantum Memory

Mathematical Structure

- Quantum register $\equiv$ (finite) Hilbert space
- Juxtaposition $\equiv$ Kronecker (tensor) product
- Reading $\equiv$ measure $\equiv$ getting a bit, probabilistic
- Quantum information is non-duplicable

Type Structure

- Based on (Intuitionistic) Multiplicative Linear Logic

$$A, B \quad ::= \quad \texttt{qbit} \mid \texttt{bit} \mid A \otimes B \mid A \multimap B$$

Type of linear functions

- Entanglement:

$$[\tfrac{1}{\sqrt{2}}(|00\rangle + |11\rangle), |xy\rangle, \langle x, y\rangle] \quad : \quad \texttt{qbit} \otimes \texttt{qbit}$$

Linear Type System

Core Typing Rules

$$\frac{\Delta, x : A \vdash M : B}{\Delta \vdash \lambda x.M : A \multimap B} \qquad \frac{\Delta \vdash M : A \multimap B \quad \Gamma \vdash N : A}{\Delta, \Gamma \vdash MN : B}$$

$$\frac{\Delta \vdash M : A \quad \Gamma \vdash N : B}{\Delta, \Gamma \vdash \langle M, N \rangle : A \otimes B} \qquad \frac{\Delta, x : A, y : B \vdash N : C \quad \Gamma \vdash M : A \otimes B}{\Delta, \Gamma \vdash \texttt{let } \langle \texttt{x,y} \rangle \texttt{ = } M \texttt{ in } N : C}$$

$$\frac{}{x : A \vdash x : A}$$

- Non-duplicability
- $\lambda x.\langle x, x \rangle$ is not typable

Quantum Circuit Model

In this model

- Circuit $\equiv$ pure function from input to output

1-qbit unitary	$\texttt{qbit} \multimap \texttt{qbit}$
2-qbit unitary	$\texttt{qbit} \otimes \texttt{qbit} \multimap \texttt{qbit} \otimes \texttt{qbit}$
measurement	$\texttt{qbit} \multimap \texttt{bit}$
initialization	$1 \multimap \texttt{qbit}$
discard	$\texttt{qbit} \multimap 1$

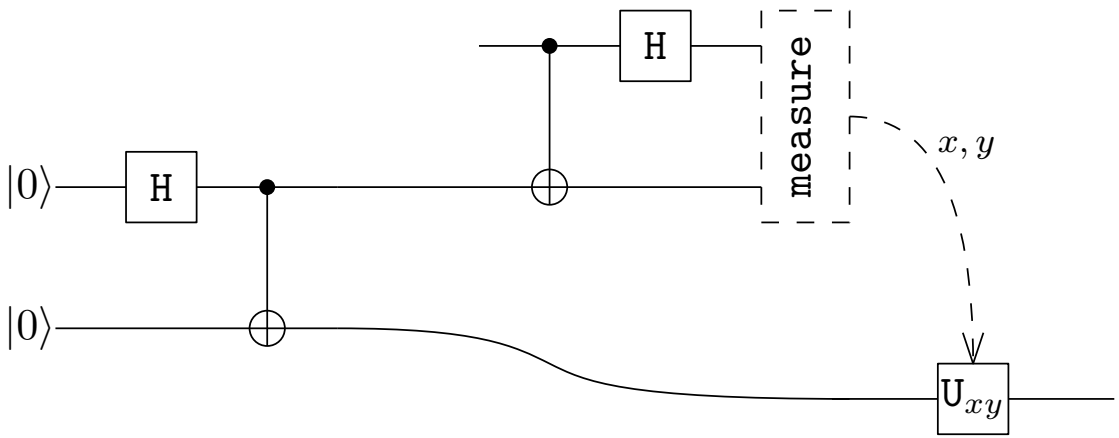
- Vertical composition $\equiv$ tensoring
- Horizontal composition $\equiv$ function composition
- Abstracts away the notion of register

Limitation

- Difficult to implement combinators

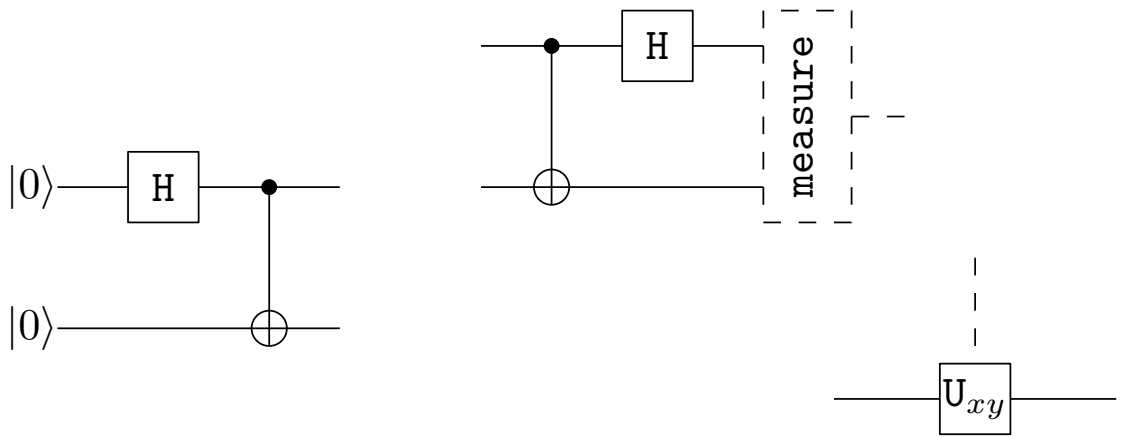
Entanglement of Functions

Example of higher-order entanglement: Teleportation



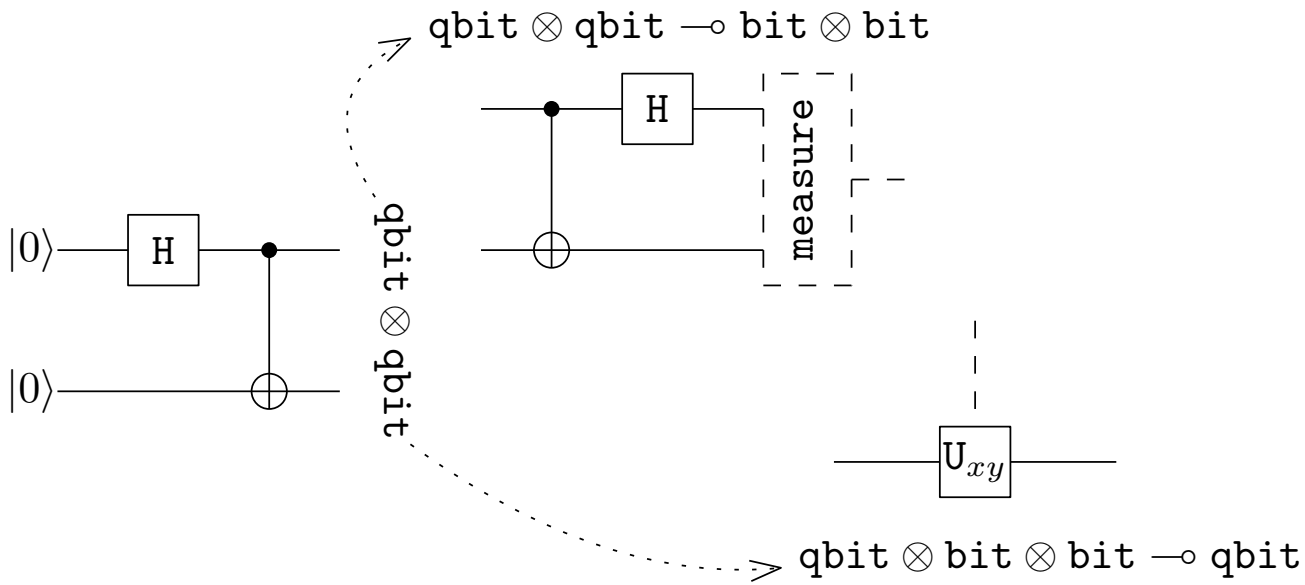
Entanglement of Functions

Example of higher-order entanglement: Teleportation



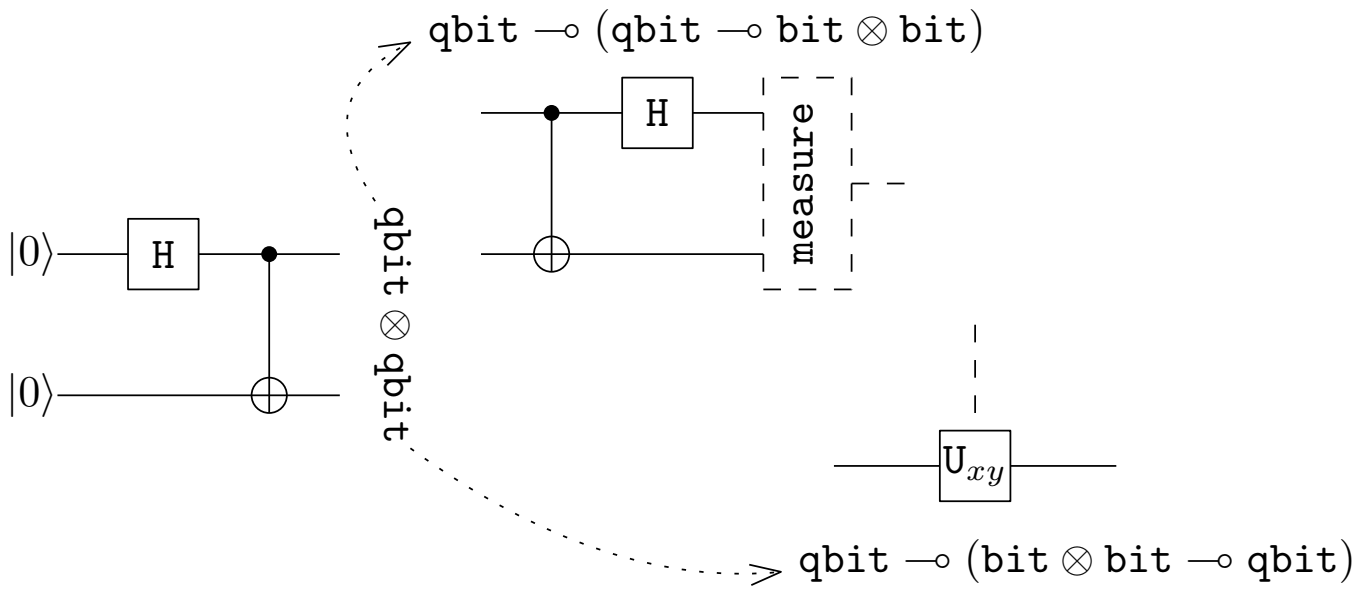
Entanglement of Functions

Example of higher-order entanglement: Teleportation



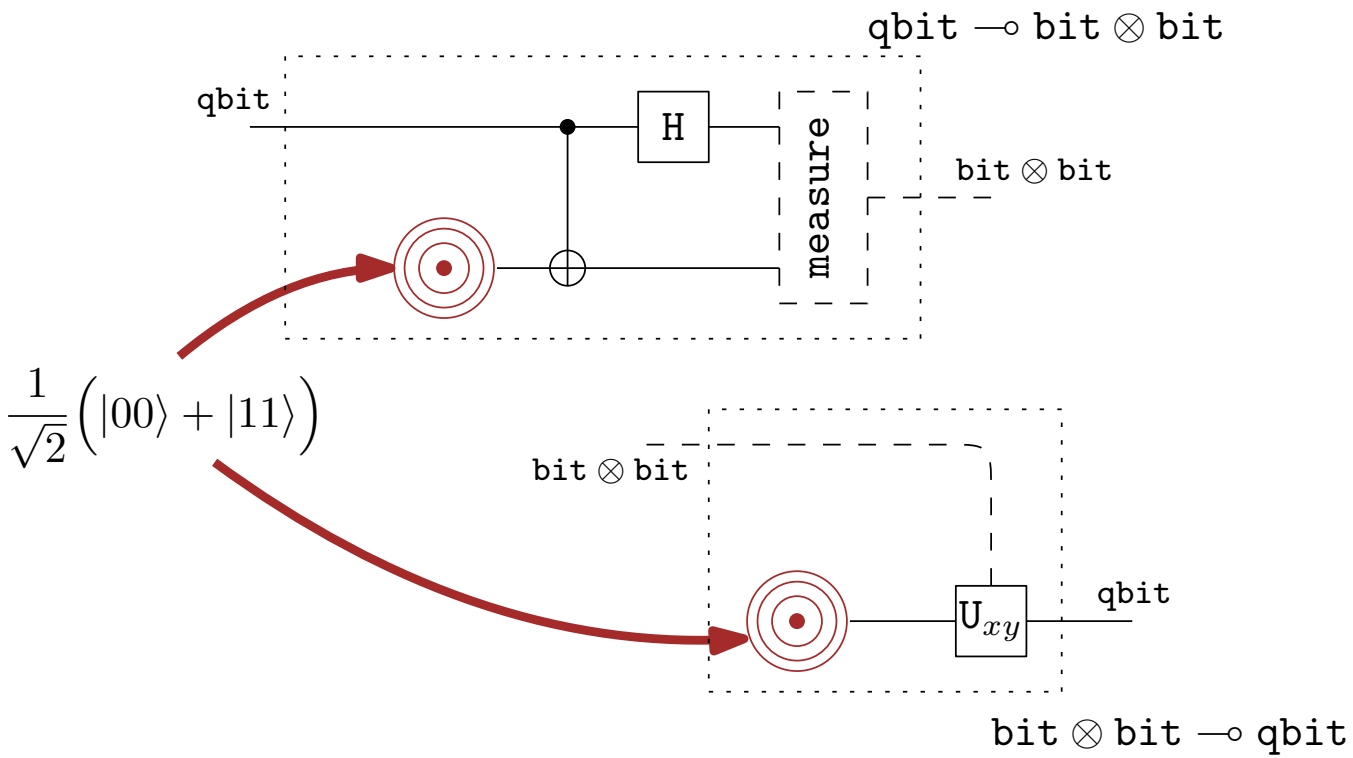
Entanglement of Functions

Example of higher-order entanglement: Teleportation



Entanglement of Functions

Example of higher-order entanglement: Teleportation



A pair of two entangled functions

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

inverses of each other.

Typing Duplication

A new type construct

$$A, B \ ::= \ \texttt{qbit} \mid \texttt{bit} \mid 1 \mid A \otimes B \mid A \multimap B \mid !A$$

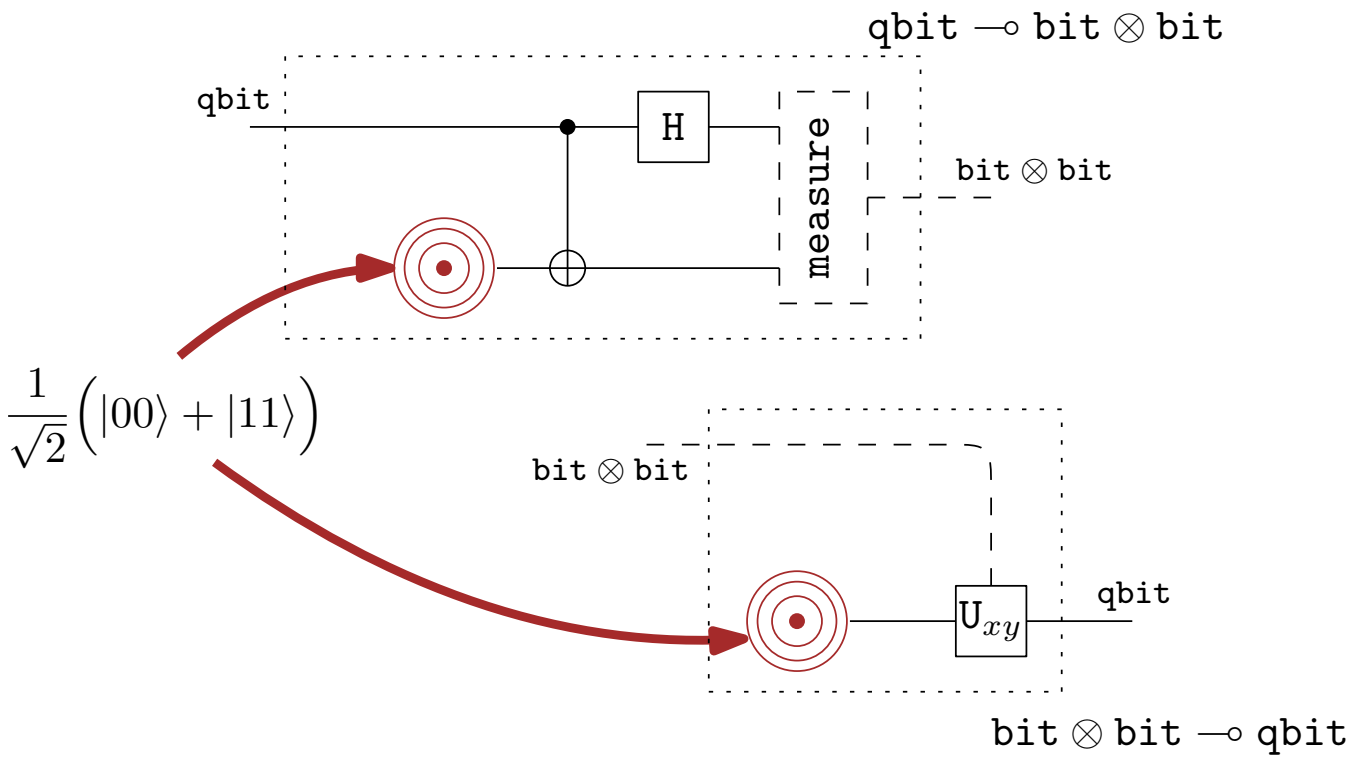
- Based on linear logic
- Non-duplicable functions with $A \multimap B$
- Duplicable functions with $!(A \multimap B)$
- Quantum operations are duplicable
  $\rightarrow$ e.g. $\texttt{measure} : !(\texttt{qbit} \multimap \texttt{bit})$

Non-trivial mix

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

Entanglement of Functions

Example of higher-order entanglement: Teleportation



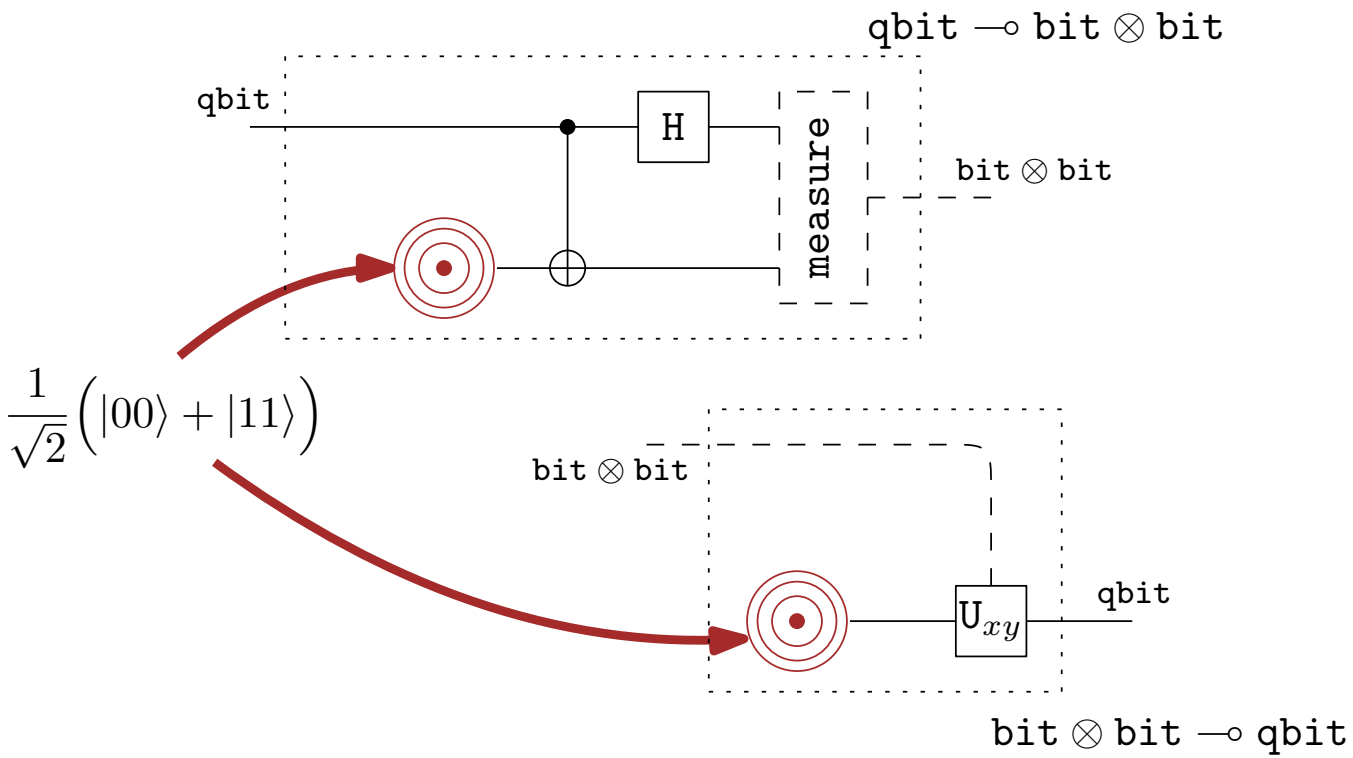
A pair of two entangled non-duplicable functions

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

inverses of each other.

Entanglement of Functions

Example of higher-order entanglement: Teleportation



A duplicable procedure generating non-duplicable functions

$$!(1 \multimap (\texttt{qbit} \multimap \texttt{bit} \otimes \texttt{bit}) \otimes (\texttt{bit} \otimes \texttt{bit} \multimap \texttt{qbit}))$$

inverses of each other.

Typing Duplication

Core typing rules

$$\frac{!\Delta \vdash V : A}{!\Delta \vdash V : !A}\ (P) \qquad \frac{\Delta \vdash M : !A}{\Delta \vdash M : A}\ (D)$$

$$\frac{\Delta, x :!A, y : !A \vdash M : B}{\Delta, x : !A \vdash M[y := x] : B}\ (C)$$

- Only values can be duplicated

(Call-by-value!)

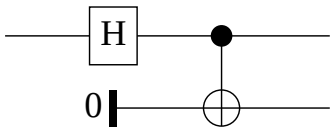
Examples

- $\vdash \texttt{had}\ (\texttt{qinit true}) : !\texttt{qbit}$ (What is wrong?)
- $\vdash \lambda x.\langle x, x\rangle : !A \multimap !A \otimes !A$ (Why?)

Typing Duplication

Type these terms!

- `true`
- $\lambda x.x$
- $\lambda x.(\mathsf{let}\, z = \mathtt{H}\, x\, \mathsf{in}\, \mathtt{CNOT}\, \langle z, \mathtt{qinit\ false} \rangle)$
- $\mathtt{H}\,(\mathtt{qinit\ false})$
- $\langle \mathtt{H}\,(\mathtt{qinit\ false}),\ \lambda x.x \rangle$
- $\mathsf{let}\, y = \mathtt{H}\,(\mathtt{qinit\ false})\, \mathsf{in}\, \lambda f.f\, y$



Distinction between

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

Quantum Lambda-Calculus

Bottom line

- Classical data handled natively
- Quantum data handled through pointers and instructions
- Mix of duplicable and non-duplicable data, with higher-order

Properties

- Type system imposed as axioms
- Safety properties derived "by hand"

(Could have used a realizability approach!)

Limitations

- Gates handled individually
- Far from what is done in quantum algorithm
- Need to consider an extended circuit model

Plan

Quantum data and Quantum computation 2

Structure of Quantum Algorithms 20

Design Choices for Quantum Programming Languages 49

Modeling Functional Quantum Languages 79
- Lambda-Calculus 79
- Incorporating Quantum 95
- Linear Type System 99
- Typing Circuit Combinators 108

Conclusion 112

The problem

Naïve approach for typing combinators:

- Repetition : $\mathbb{N} \multimap (A \multimap A) \multimap (A \multimap A)$ (OK)
- Inversion : $(A \multimap B) \multimap (B \multimap A)$ (WRONG)
- Control : $(A \multimap B) \multimap (\mathtt{qbit} \otimes A \multimap \mathtt{qbit} \otimes B)$ (WRONG)

Does not work

- Would require “reading” the gates.
- But gates can only be sent to the QRAM

Circuit Description Languages

Extending the quantum λ-calculus with

- 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

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

- Nice arrow-like, categorical semantics

Formalization of Quipper

- Proto-Quipper
- Notion of circuit-description language

Circuit Description Languages

Possible extensions

- Inductive types (such as lists)
- First-order quantifiers
  $\rightarrow$ limited to "classical types"

Dependent Proto-Quipper

- Type $[A]_n$ for lists of length n made of elements of type A
- In general, $B(n)$ is a type parameterized by n
- $f : \forall n : A \cdot B(n)$: function A to B, with $f(n)$ of type $B(n)$

Examples

- $\forall n : \mathbb{N} \cdot \texttt{Circ}([\texttt{qbit}]_n, [\texttt{qbit}]_n)$
- $\forall m, n : \mathbb{N} \cdot [[\texttt{qbit}]_m]_n \multimap [\texttt{qbit}]_{mn}$

Extended Quantum Circuit Model

Summary

- Circuits are now first-order citizens
- Close to what is done for "real algorithms"
- Suitable for formalization and extensions

  [Lindenhovius,Mislove,Zamdzhiev, 2018] [Fu, Selinger et al, 2020,2022,2024] [Lee, V et al, 2021] [Colledan, Dal Lago, 2025]

Limitations

- Circuits are built from opaque boxes
- The only control is classical
- What about superposition of execution?

Plan

Quantum data and Quantum computation 2

Structure of Quantum Algorithms 20

Design Choices for Quantum Programming Languages 49

Modeling Functional Quantum Languages 79

Conclusion 112

Conclusion: Quantum Programming Language

Practical aspect

- Coding quantum algorithms!
- Design choices for quantum circuit-description
- Monadic approach for sound semantics

Theoretical aspect

- Foundational work: quantum lambda-calculus
- Type system based on linear logic
- Extensions to capture circuit-description
- Limits in expressivity:
  How do you program circuit combinators?

Questions?