Quantum Computation
Model and Programming Paradigm

Benoît Valiron
CentraleSupélec – LRI

Journées LTP, 6 Décembre 2018
Big Picture: Quantum Computer

Classical unit = regular computer
Communicates with the coprocessor

Quantum unit = blackbox
Contains a quantum memory

Getting faster algorithms for conventional problems
Big Picture: Quantum Computer

You can access one now!

https://quantumexperience.ng.bluemix.net/qx
Big Picture: Quantum Computer

A small memory-chip inside a big fridge
Big Picture: Quantum Computer

What are quantum algorithms good for?

- **factoring**
  - for breaking modern cryptography

- **simulating quantum systems**
  - for more efficient molecule distillation procedure

- **solving linear systems**
  - for high-performance computing

- **solving optimization problems**
  - for big learning

- ...more than 300 algorithms:
Plan

1. Quantum memory
2. Quantum / Classical interaction
3. Internals of algorithms
4. Coding quantum algorithms
5. The language Quipper
6. A formalization: Proto-Quipper
7. Conclusion
Plan

1. Quantum memory
2. Quantum / Classical interaction
3. Internals of algorithms
4. Coding quantum algorithms
5. The language Quipper
6. A formalization : Proto-Quipper
7. Conclusion
Quantum memory

A quantum memory with \( n \) quantum bits is a complex combination of strings of \( n \) bits. E.g. for \( n = 3 \):

\[
\alpha_0 \cdot 000 + \alpha_1 \cdot 001 + \alpha_2 \cdot 010 + \alpha_3 \cdot 011 + \alpha_4 \cdot 100 + \alpha_5 \cdot 101 + \alpha_6 \cdot 110 + \alpha_7 \cdot 111
\]

with \( |\alpha_0|^2 + |\alpha_1|^2 + |\alpha_2|^2 + |\alpha_3|^2 + |\alpha_4|^2 + |\alpha_5|^2 + |\alpha_6|^2 + |\alpha_7|^2 = 1 \).

(alike probabilities with complex numbers... )
Quantum memory

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

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

\[
\begin{align*}
\alpha_0 \cdot 00 & \quad \alpha_0 \cdot 000 \\
+ \quad \alpha_1 \cdot 01 & \quad + \quad \alpha_1 \cdot 010 \\
+ \quad \alpha_2 \cdot 10 & \quad + \quad \alpha_2 \cdot 100 \\
+ \quad \alpha_3 \cdot 11 & \quad + \quad \alpha_3 \cdot 110
\end{align*}
\]
Quantum memory

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

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

   \[
   \alpha_0 \cdot 00 \quad \rightarrow \quad \alpha_0 \cdot 001 \\
   + \quad \alpha_1 \cdot 01 \\
   + \quad \alpha_2 \cdot 10 \\
   + \quad \alpha_3 \cdot 11 \\
   \]

   \[
   \quad + \quad \alpha_0 \cdot 011 \\
   \quad + \quad \alpha_1 \cdot 011 \\
   \quad + \quad \alpha_2 \cdot 101 \\
   \quad + \quad \alpha_3 \cdot 111 
   \]
Quantum memory

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

2. Measurement. Measuring first qubit:

$$\alpha_0 \cdot 00 + \alpha_1 \cdot 01 + \alpha_2 \cdot 10 + \alpha_3 \cdot 11 \rightarrow \begin{cases} \alpha_0 \cdot 00 + \alpha_1 \cdot 01 \quad (\text{prob. } |\alpha_0|^2 + |\alpha_1|^2) \\ + \alpha_2 \cdot 10 \quad (\text{prob. } |\alpha_2|^2 + |\alpha_3|^2) \\ + \alpha_3 \cdot 11 \end{cases}$$

modulo renormalization.
Quantum memory

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

2. Measurement. Measuring second qubit:

\[
\begin{align*}
\alpha_0 \cdot 00 \\
+ \alpha_1 \cdot 01 \\
+ \alpha_2 \cdot 10 \\
+ \alpha_3 \cdot 11
\end{align*}
\]

\[
\begin{align*}
\alpha_0 \cdot 00 \\
+ \alpha_2 \cdot 10
\end{align*} \quad \text{(prob. } |\alpha_0|^2 + |\alpha_2|^2) \\
\begin{align*}
\alpha_1 \cdot 01 \\
+ \alpha_3 \cdot 11
\end{align*} \quad \text{(prob. } |\alpha_1|^2 + |\alpha_3|^2)
\]

modulo renormalization.
Quantum memory

The operation 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:

\[
\begin{align*}
\alpha_0 \cdot 00 & \quad \rightarrow \quad \alpha_0 \cdot 10 \\
+ \alpha_1 \cdot 01 & \quad + \quad \alpha_1 \cdot 11 \\
+ \alpha_2 \cdot 10 & \quad + \quad \alpha_2 \cdot 00 \\
+ \alpha_3 \cdot 11 & \quad + \quad \alpha_3 \cdot 01
\end{align*}
\]
Quantum memory

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

3. Unitary operations.

E.g. the Hadamard gate on one quantum bit. Sends

\[
\begin{align*}
0 & \mapsto \frac{\sqrt{2}}{2} \cdot 0 + \frac{\sqrt{2}}{2} \cdot 1 \\
1 & \mapsto \frac{\sqrt{2}}{2} \cdot 0 - \frac{\sqrt{2}}{2} \cdot 1
\end{align*}
\]

When applied on the first qubit:

\[
\begin{align*}
\alpha_0 \cdot 01 & \mapsto \alpha_0 \cdot \left( \frac{\sqrt{2}}{2} \cdot 01 + \frac{\sqrt{2}}{2} \cdot 11 \right) \\
+ \alpha_1 \cdot 10 & \mapsto \alpha_1 \cdot \left( \frac{\sqrt{2}}{2} \cdot 00 - \frac{\sqrt{2}}{2} \cdot 10 \right)
\end{align*}
\]
Quantum memory

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

3. Unitary operations.

E.g. the Hadamard gate on one quantum bit. Sends

\[
\begin{align*}
0 & \mapsto \frac{\sqrt{2}}{2} \cdot 0 + \frac{\sqrt{2}}{2} \cdot 1 \\
1 & \mapsto \frac{\sqrt{2}}{2} \cdot 0 - \frac{\sqrt{2}}{2} \cdot 1
\end{align*}
\]

When applied on the first qubit:

\[
\begin{align*}
\alpha_0 \frac{\sqrt{2}}{2} \cdot 01 \\
\alpha_0 \cdot 01 & \mapsto + \alpha_0 \frac{\sqrt{2}}{2} \cdot 11 \\
+ \alpha_1 \cdot 10 & \mapsto + \alpha_1 \frac{\sqrt{2}}{2} \cdot 00 \\
+ \alpha_1 \frac{\sqrt{2}}{2} \cdot 10
\end{align*}
\]
Quantum memory

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

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

   \[ 1100 \longmapsto \frac{\sqrt{2}}{2} \cdot 1100 + \frac{\sqrt{2}}{2} \cdot 1110 \]

   . . . or remove it

   \[ \frac{\sqrt{2}}{2} \cdot 1100 + \frac{\sqrt{2}}{2} \cdot 1110 \longmapsto 1100 \]
Quantum memory

The operation 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:

\[
\begin{align*}
\alpha_0 \cdot 00 & \quad \alpha_0 \cdot 00 & \quad \alpha_0 \cdot 00 \\
+ \alpha_1 \cdot 01 & \quad + \alpha_1 \cdot 11 & \quad + \alpha_3 \cdot 01 \\
+ \alpha_2 \cdot 10 & \quad + \alpha_2 \cdot 10 & \quad + \alpha_2 \cdot 10 \\
+ \alpha_3 \cdot 11 & \quad + \alpha_3 \cdot 01 & \quad + \alpha_1 \cdot 11
\end{align*}
\]
Quantum memory

The co-processor has an internal (quantum) memory.

- Classical data can \textit{transparently flow in}.
- Internal operations are \textit{local}.
- Retrieval of quantum information is \textit{probabilistic} and modify the \textit{global state}.

In particular:

- The quantum memory has to be \textit{permanent}.
- To act on quantum memory, classical operations have to \textit{lifted}.
- This is \textit{potentially expensive}.
Quantum memory: hardware

Quantum data: encoded on the state of quantum particles.

- E.g. nucleus of an atom:

![Histidine Structure](image1.png)

The histidine as a 12-qubit memory.

![Perfluorobutadienyl Iron Complex](image2.png)

The perfluorobutadienyl iron complex as a 7-qubit memory.

- E.g. Photon polarization.

- E.g. Electrons (in superconducting devices)... 

Problems to overcome: Scalability, decoherence.

Nonetheless, we are already post-quantum...
Many experts predict a quantum computer capable of effectively breaking public key cryptography within [a few decades], and therefore NSA believes it is important to address that concern.

Plan

1. Quantum memory
2. Quantum / Classical interaction
3. Internals of algorithms
4. Coding quantum algorithms
5. The language Quipper
6. A formalization : Proto-Quipper
7. Conclusion
Quantum / Classical interaction

Typical execution flow:

Program → Stream of instructions → Quantum Computation

Runtime → Feedback → Classical Unit → Quantum Unit
Quantum / Classical interaction

Stream of instructions

- Local actions on one (or two) qubit(s) at a time
- Limited moving of qubits
- No copying

(ion trap)
Quantum / Classical interaction

Stream of instructions

- **Local actions** on one (or two) qubit(s) at a time
- **Limited moving** of qubits
- **No copying**

\[ \text{dots} \equiv \text{ions} \equiv \text{qubits} \quad \text{action} \equiv \text{pulses through wires} \]
Quantum / Classical interaction

Stream of instructions

- Series of elementary actions applied on the quantum memory
- Summarized with a quantum circuit.
- $\text{wire} \equiv \text{qubit}$, $\text{box} \equiv \text{action}$, time flows left-to-right

No "quantum loop" or "conditional escape".

\[\begin{array}{c}
\text{Input} \\
\hline
\text{H} \\
\hline
\text{0} \\
\hline
\text{Output}
\end{array}\]
Quantum / Classical interaction

Input values → Parameters to the problem

Initializing quantum memory → Static circuit

Executing the circuit → Measure

Output values → Post-processing

New input values → Probabilitically

Answer to the problem
Quantum / Classical interaction

Some algorithms follow a simple scheme

![Simple scheme diagram]

Others are following a more adaptative scheme:

![Adaptative scheme diagram]

This is where quantum circuits differ from hardware design.

One cannot draw a quantum circuit once and for all.
Quantum / Classical interaction

A sound model of computation:
Interaction with the quantum memory seen as an I/O side effect

\[
\text{Circ } a := \text{Empty } a \\
\quad \mid \text{Write Gate (Circ } a) \\
\quad \mid \text{Read Wire (Bool } \rightarrow \text{ (Circ } a))
\]

- Output: emit gates to the co-processor
- Input: emit a read even to the co-processor, with a call-back function

Representing circuits

- static circuits: lists of gates
- dynamic circuits: trees of gates.
Quantum / Classical interaction

Moral

• Distinction parameter / input

• Circuits might be dynamically generated

• Parameters = govern the shape and size of the circuit

• Model of computation: specialized I/O side-effect
Plan

1. Quantum memory
2. Quantum / Classical interaction
3. Internals of algorithms
4. Coding quantum algorithms
5. The language Quipper
6. A formalization : Proto-Quipper
7. Conclusion
Internals of algorithms

The techniques used to described quantum algorithms are diverse.

1. Quantum primitives.
   - Phase estimation.
     Assuming $\omega = 0.x y$, we want
     \[
     \rho_0(e^{2\pi i x y})^0 \cdot 00 \\
     + \rho_1(e^{2\pi i x y})^1 \cdot 01 \\
     + \rho_2(e^{2\pi i x y})^2 \cdot 10 \\
     + \rho_3(e^{2\pi i x y})^3 \cdot 11
     \]
     \[\rightarrow 1 \cdot x y\]

Moving information from coefficients to basis vectors
The techniques used to described quantum algorithms are diverse.

1. Quantum primitives.
   - Phase estimation.
   - Amplitude amplification.

   Qubit 3 in state 1 means good.

\[
\begin{align*}
\rho_0 e^{i\phi_0} \cdot 000 & \quad \rightarrow \quad \rho_0 e^{i\phi_0} \cdot 000 \\
+ \rho_1 e^{i\phi_1} \cdot 011 & \quad + \quad \rho_1 e^{i\phi_1} \cdot 011 \\
+ \rho_2 e^{i\phi_2} \cdot 100 & \quad + \quad \rho_2 e^{i\phi_2} \cdot 100 \\
+ \rho_3 e^{i\phi_3} \cdot 110 & \quad + \quad \rho_3 e^{i\phi_3} \cdot 110
\end{align*}
\]

Increasing the probability of measuring the “good” states
Internals of algorithms

The techniques used to described quantum algorithms are diverse.

1. Quantum primitives.
   - Phase estimation.
   - Amplitude amplification.
   - Quantum walk.

```
0 1 2 3 4 5 6 7

15 14 13 12 11 10 9 8
```
Internals of algorithms

The techniques used to describe quantum algorithms are diverse.

1. Quantum primitives.
   - Phase estimation.
   - Amplitude amplification.
   - Quantum walk.

   After 5 steps of a probabilistic walk:
Internals of algorithms

The techniques used to described quantum algorithms are diverse.

1. Quantum primitives.
   • Phase estimation.
   • Amplitude amplification.
   • Quantum walk.

After 5 steps of a quantum walk:
Internals of algorithms

The techniques used to described quantum algorithms are diverse.

2. Oracles.
   - Take a classical function \( f : \text{Bool}^n \rightarrow \text{Bool}^m \).
   - Construct
     \[
     \overline{f} : \text{Bool}^{n+m} \rightarrow \text{Bool}^{n+m}
     \]
     \[
     (x, y) \mapsto (x, y \oplus f(x))
     \]
   - Build the unitary \( U_f \) acting on \( n + m \) qubits computing \( \overline{f} \).
Internals of algorithms

The techniques used to described quantum algorithms are diverse.

2. Oracles, in real life

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) and (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) and (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

37
Internals of algorithms

The techniques used to described quantum algorithms are diverse.


- This is not a formal specification!
- Notion of “box”
- Size of the circuit depends on parameters
The techniques used to describe quantum algorithms are diverse.

4. **High-level** operations on circuit:
   - Circuit inversion.
     ![Circuit inversion diagram]
     (the circuit needs to be reversible…)
   - Repetition of the same circuit.
     ![Repetition diagram]
     (needs to have the same input and output arity…)
   - Controlling of circuits
Internals of 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).
Internals of algorithms

Summary

• Need of automation for oracle generation

• Distinction parameter / input

• Circuits as inputs to other circuits

• Regularity with respect to the size of the input

• Circuit construction:
  – Using circuit combinators: Inversion, repetition, control, etc
  – Procedural

• Lots of classical processing!
Plan

1. Quantum memory
2. Quantum / Classical interaction
3. Internals of algorithms
4. Coding quantum algorithms
5. The language Quipper
6. A formalization : Proto-Quipper
7. Conclusion
Coding algorithms

A very recent topic

• From complexity analysis to concrete circuits

• No machine yet, but
  – Resource analysis
  – Optimization
  – Emulation

• Scalable languages: in the last 5 years
  – Python’s libraries/DSL: Project-Q, QISKit, etc
  – Liqui⟩, Q# (Microsoft)
  – Quipper, QWIRE (academic)
Coding algorithms

Imperative programming and the quantum I/O

- Input/Output “as usual”: with commands
- Measurement returns a boolean (probabilistically)
- If well-behaved, provides high-level circuit operations
- Example with Project-Q:

```python
def circuit(q1, q2):
    H | q1
    with Control(q1):
        X | q2
    x = Measure | q1
    eng.flush()
    if x:
        Y | q2
    else:
        Z | q2
```
Coding algorithms

Functional programming and the quantum I/O

- **Monadic approach** to encapsulate I/O
- Inside the monad: quantum operations
- Outside the monad: classical operations and circuit manipulation
- Qubits only live inside the monad
Coding algorithms

Dealing with run-time errors

• Imperative-style: Quantum I/O is a memory mapping
  – → Type-systems based on separation logic should work
  – Hoare logic or Contracts

• Functional-style:
  – Non-duplicable quantum data: linear type system
  – Dependent-types
Plan

1. Quantum memory
2. Quantum / Classical interaction
3. Internals of algorithms
4. Coding quantum algorithms
5. The language Quipper
6. A formalization : Proto-Quipper
7. Conclusion
The Language Quipper

- **Embedded language** in Haskell
- Logical description of **hierarchical circuits**
- **Well-founded monadic semantics**. Allow to mix two paradigms
  - **Procedural** : describing low-level circuits
  - **Declarative** : describing high-level operation
- **Parameter/input distinction**
  - Parameter : determine the shape of the circuit
  - Input : determine what goes in the wires
- ...
The Language Quipper

A function in Quipper is a map

\[ A \rightarrow \text{Circ } B \]

- Input something of type \( A \)
- Output something of type \( B \)
- As a side effect, generate a circuit snippet

Or

- Input a \textit{value} of type \( A \)
- Output a \textit{“computation”} of type \( \text{Circ } B \)

Families of circuits

- represented with lists, e.g. \([\text{Qubit}] \rightarrow \text{Circ } [\text{Qubit}]\)
The Language Quipper

New base type: Qubit \equiv wire

Building blocks

- \text{qinit} :: \text{Bool} \rightarrow \text{Circ Qubit}
- \text{qdiscard} :: \text{Qubit} \rightarrow \text{Circ ()}
- \text{hadamard} :: \text{Qubit} \rightarrow \text{Circ Qubit}
- \text{hadamard\_at} :: \text{Qubit} \rightarrow \text{Circ ()}

Composition of functions \equiv\ composition of circuits

\[
\begin{array}{c}
\text{Bool} \xrightarrow{\text{qinit}} \text{Circ Qubit} \\
\text{Qubit} \xrightarrow{\text{hadamard}} \text{Circ Qubit}
\end{array}
\]

High-level circuit combinators

- \text{controlled} :: \text{Circ a} \rightarrow \text{Qubit} \rightarrow \text{Circ a}
- \text{inverse} :: (\text{a} \rightarrow \text{Circ b}) \rightarrow \text{b} \rightarrow \text{Circ a}
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
Coding quantum algorithms: Quipper

Result (3 wires):
Coding quantum algorithms: Quipper

Result (30 wires):
Built on Haskell’s static type system, but

- **unchecked linearity**

  \[ \text{controlled \ (qnot \ x) \ x} \]

- **uncaught shape mismatches**
  - Consider \( f :: \ldbrack \text{Qubit} \rceil \to \text{Circ} \ldbrack \text{Qubit} \rceil \)
  - Assume that \( \text{length} \ (f \ l) = 2 \times \text{length} \ l \)
  - Then `reverse f` cannot be applied on lists of odd lengths
Towards tools for program analysis

One cannot “read” the quantum memory

- Testing / debugging expensive
- Probabilistic model
- What does it mean to have a “correct” implementation?

Emulation of circuits

- Only for “small” instances
- Taming the testing problem
- For experimentation of error models

Formal methods

- **Type systems**: capture errors at compile-times
- **Static analysis tools**: analyze quantum programs
- **Proof assistants**: verify code transformation and optimization
Towards a quantum compiler

Current quantum programming languages maps to quantum circuits

• native representation structures of quantum algorithms
• Good enough for visualization, numerical emulation
• But very rigid:
  – accounts for one computational model...
  – ...but misses other models
  – occults intrinsic parallelism of computation
  – fails to capture geometrical properties of backends
    → Grid-like physical layout, graph-states, etc.
  – Ad-hoc graphical notation
Towards a quantum compiler

A missing piece in a compilation stack

High-level → Quipper, Liqui⟩, Project-Q...

Circuits
QASM

Missing IR

Error-corrected qubits
Physical, noisy qubits
Plan

1. Quantum memory
2. Quantum / Classical interaction
3. Internals of algorithms
4. Coding quantum algorithms
5. The language Quipper
6. A formalization : Proto-Quipper
7. Conclusion
Proto-Quipper

A core subset of Quipper [Ross 2015]: A lambda-calculus

• Focused on the circuit-description part of the language
  → no measurement

• Simple linear type system

\[
A, B ::= \text{qubit} | 1 | A \otimes B | \text{bool} | A \rightarrow B | !A | \text{Circ}(T, U)
\]

\[
T, U ::= \text{qubit} | 1 | T \otimes U
\]

• A special class of values for representing circuits
  → Built on an algebra of circuits

• Built-in circuit operators

\[
\begin{align*}
\text{box} & : \! (T \rightarrow U) \rightarrow !\text{Circ}(T, U) \\
\text{unbox} & : \text{Circ}(T, U) \rightarrow ! (T \rightarrow U) \\
\text{rev} & : \text{Circ}(U, T) \rightarrow !\text{Circ}(T, U)
\end{align*}
\]
Proto-Quipper

Circuits in Proto-Quipper

Formalized as a pair \((S, Q)\) of enumerable sets

- \(S\) : set of circuit states
- \(Q\) : set of wire identifiers
- Operators relating them :
  - \(\text{New} : \mathcal{P}_f(Q) \to S\)
  - \(\text{In} : S \to \mathcal{P}_f(Q)\)
  - \(\text{Rev} : S \to S\)
  - \(\text{Out} : S \to \mathcal{P}_f(Q)\)
  - \(\text{Append} : S \times S \times \text{Bij}_f(Q) \hookrightarrow S \times \text{Bij}_f(Q)\)

- Various equations, such as
  - \(\text{In} \circ \text{Rev} = \text{Out}\)
  - \(\text{In} \circ \text{New} = \text{Out} \circ \text{New} = \text{id}\)
Proto-Quipper

Circuits versus functions

• A value of type $U \rightarrow T$ is a suspended computation
• A value of type $\text{Circ}(T, U)$ is a circuit and corresponds to an element of $S$.

In particular

• One can access the “content” of a circuit
• The term operator $\text{rev} = \text{algebra operator Rev}$
• Unboxing a circuit = “running it” = using $\text{Append}$

In a sense

• $\text{Circ}(T, U)$ is the type for precomputed, first-order functions on quantum data
• whereas $T \rightarrow U$ could contain e.g. non-terminating functions
Proto-Quipper

Linear type system

- quantum data is non-duplicable
- Subtyping relation: “A duplicable element can be used only once”

\[ \neg A <: A \]

An opaque type for qubits

- no constructors
- only accessible through circuit combinators
- or as variables

Absence of inductive types

- Only one possible shape of value for a given first-order type

\[ \text{qubit} \otimes \text{bool} \quad \text{qubit} \otimes (\text{qubit} \otimes \text{qubit}) \]
Limitations of Proto-Quipper

Absence of lists or other inductive types

- Good: unboxing sends $\text{Circ}(T, U)$ to total functions $T \to U$
- Bad: An element of type $\text{Circ}(T, U)$ is one circuit
  - No representation of families of circuits, as in Quipper

Adding lists

- Makes $\text{[qubit]} \to \text{[qubit]}$ represent families of circuits
  (Note: not monadic...)
- But $\text{Circ}([\text{qubit}], [\text{qubit}])$
  - is still one circuit of $S$ with a fixed number of wires
  - ruins the totality of unboxing and reversing
  - makes boxing not ill-defined: which circuit from the family?
Mitigating Limitations of Proto-Quipper

To mitigate problems with lists: Two main solutions

1 (not ours) – Use of dependent types
- Types to correctly specify box, unbox and rev
- Burden of proof of correctness on the programmer
- Require a full first-order linear logic

2 (ours) – Only extend type system with a notion of shape
- Captures the structure of a value of first-order type
- Boxing now takes as arguments
  - A function !(\(T \to U\))
  - A shape for \(T\)
- Does not solve the run-time error with unbox and rev
  - Allow run-time errors related to shapes (and only those)
  - Leave proof of correctness to auxiliary tool
- Joint work on this topic between LRI and CEA/Nano-Innov
Plan

1. Quantum memory
2. Quantum / Classical interaction
3. Internals of algorithms
4. Coding quantum algorithms
5. The language Quipper
6. A formalization : Proto-Quipper
7. Conclusion
Quantum @ LRI

Thematics

- **Formal methods** (Benoît Valiron, Chantal Keller, Thibault Balabonski)
- **Scientific computing and HPC** (Benoît Valiron, Marc Baboulin)

Students

- **Timothée Goubault de Brugière** : Thèse CIFRE/Atos
  → **Synthesis of unitaries** : Householder decomposition, BFGS
- **Dong-Ho Lee** : Thèse CEA (just starting)
  → **Formalization of Quipper-like languages**

Projects

- **ANR SoftQPro, European project Quantex**
- **Partnership with CEA-Nano-Innov, Atos/Bull, LORIA (Nancy)**

Postdocs

- **We have funding for at least 2 one-year postdocs!**
We have funding for postdocs!