Formalization of a Programming Language for Quantum Circuits with Measurement and Classical Control

Dong-Ho Lee[1,3] Sébastien Bardin[1] Valentin Perrelle[1]
Benoît Valiron[2,3]

[1]CEA LIST, Université Paris-Saclay, France
[2]CentraleSupélec, Université Paris-Saclay, France
[3]LRI, Université Paris-Saclay, France

29/Nov/2019

Outline

QRAM Model

QRAM is a model of quantum computation which consists of two parts:

- classical computer does classical computation with the ability to construct and send quantum process.
- quantum co-processor simulates quantum process and informs the classical computer with the observed values.

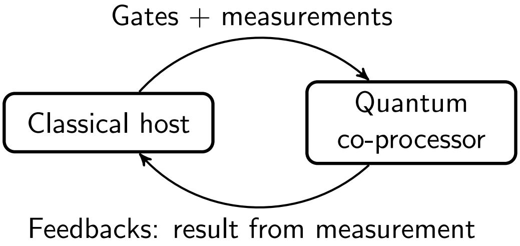


Figure 1: QRAM model

Quantum computation can be statistical set of quantum computation

Necessity of a programming language

Some difficulties in quantum programming:

- non-intuitive design and reason
- difficulty of testing induced by probabilistic nature
- side effect regarding quantum state when debugging

Programming language helps:

- represent large circuits efficiently
- construct circuits in a structured way

Expressive programs including:

- circuit level operators
- representation of families of circuits
- utilisation of the host/quantum co-processor interaction
- high-order data types and recursion

Necessity of formal methods

We want to analyze quantum programs without execution:

- correctness and safety of the program
- analysis of resources like memory and time

How to apply formal methods:

- formalization of operational semantics
- type systems for guaranteeing the safety of program

Several remarks on formalization:

- probabilistic side-effects from the host/quantum co-processor interaction
- duplicable versus non-duplicable data
- formalization of families of circuits
- circuit-level operations (control, reverse, etc) are not applicable to all quantum circuits

Quipper and QWire

Examples of programming languages based on QRAM model:

- Quipper [1] is an expressive, high-order programming language, embedded into Haskell
  It has been used to various existing quantum algorithms
  However, it lacks its own type checker, depending on Haskell
- QWire [2] is a general circuit language embedded into Coq
  It is expressive and analyzes program in Coq's logic
  Proofs can be automatized using tactics
  But still not scalable easily

Programming language allows one to construct circuits using classical computation and run them by simulation
Classical data by measurement of quantum state is transferred to the classical host

Semantics of measurement and classical control

Dynamic lifting: information transferred from the quantum co-processor to the classical host used in circuit construction
Quipper and QWire implement dynamic lifting without an explicit formalization of its semantics

- In QWire, the operational semantics performs normalization for composition and unbox operations but the classical control by dynamic lifting is hidden in the host term within the unbox.
- In Quipper, the operational semantics is encoded in Haskell's monadic type system and capture a notion of dynamic circuit including measurements. However, this semantics has never been fully formalized.

Quantum circuit construction in the classical host can be dependent on the result of a measurement

Semantics of circuit description

- ProtoQuipper [3]
  - Subset of Quipper regarding circuit construction and transfer is formalized semantics with an abstract machine
  - Description of host/quantum co-processor interaction
    - $\text{box} : (A \multimap B) \multimap \text{Circ}(A, B)$
    - $\text{unbox} : \text{Circ}(A, B) \multimap (A \multimap B)$
  - Linear/non-linear type system by subtyping
  - Missing: measurement, lists, etc.
- Monoidal Symmetric Categories
  - Models of graphical languages consist of boxes (morphisms) and strings (objects) [4]
  - Provide the categorical semantics of a quantum circuit language [5]
  - CPO-enrichment gives interpretations of recursion [6]
  - The evaluation of quantum circuits is not formalized, hence the feedback of quantum co-processor in circuit construction

Semantics of quantum computation

- C^*-algebra
  - Interprets quantum computation as completely positive unital (CPU) maps between C^*-algebras
  - A state is a map from a C^*-algebra to $\mathbb{C}$
- Equational Theory of Quantum Computation [7]
  - Considers quantum computation as algebraic structure with linear parameters (e.g. $\mathit{measurement} : (1 \mid 0, 0)$)
  - Provides a set of axioms for an equation theory of quantum computation
  - Proves the full completeness of the theory with respect to the model of algebraic theories which is based on C^*-algebra
- Quantum Domain Theory [8]
  - Provides the semantics of the evaluation of quantum circuits
  - Uses probabilistic distribution monad

Outline of research

Objective: formalize a semantics for interleaved quantum circuits and dynamic lifting

Difficulty: analysis of the structure of computation without simulation of measurement

Solution: make circuits not only lists but trees branching over the result of measurements (*quantum channels*)
Dynamic lift: generation of multiple control flows in classical computation each of which is interpreted as a quantum channel

Contribution:

- a typed language extending quantum lambda calculus [9] with box and unbox operators and quantum channel constants
- a small step operational semantics formalizing dynamic lift and quantum channel construction
  The formalization extends the one of Proto-Quipper [3]

Algebraic structure of quantum channel

QCAlg: abstract structure of quantum channels which represent quantum circuits with tree-shaped control flow with measurement

$$\epsilon(W) \ \mid\ U(W)\ Q \ \mid\ \text{init } b\ w\ Q \ \mid\ \text{meas } w\ Q_1\ Q_2 \ \mid\ \text{free } w\ Q$$

where w, b, and W respectively refer to wires, booleans, and finite sets of wires.

Accessible quantum channel: a pair of a quantum channel object and a tree of binding function (Q, b) satisfying

- the structure of Q corresponds to the structure of b meaning each terminal of Q and b can be matched
- each binding function of the terminals of b links variables to the wire names being used in the corresponding branch of Q

Quantum channel constant: a tuple $(p, (Q, b), m)$ where $\mathbf{supp}(p) = \mathbf{in}(Q)$ and free variables of each term of m are linked by the corresponding binding function of b to the wires of the corresponding branch of Q.

Type $QChan(-, -)$ is for first class objects of quantum channel

Branching term and linear type system

Quantum computation containing dynamic lift may reduce to different values depending on the results of measurements

Branching term: represents probabilistic distributions of computations

Language:

$$\begin{aligned}\mathrm{Term}(M) \ &::= x \mid * \mid \mathrm{tt} \mid \mathrm{ff} \mid (p,(Q,b),m) \mid \lambda x.M \mid M_a M_b \mid \langle M_a, M_b\rangle \mid \\ &\quad \textbf{let}\ \langle x,y\rangle = M_a\ \textbf{in}\ M_b \mid \textbf{if}\ M\ \textbf{then}\ M_a\ \textbf{else}\ M_b \mid \mathrm{box}(M) \mid \mathrm{unbox} \\ \text{Branching term}(m) \ &::= M \mid [m_a, m_b]\end{aligned}$$

Linear type system ensures variables are used exactly once in each branch of control flow

Type rules ensure that all terms of a branching term share the same type

Type rules for circuit construction operators:

$$\frac{\emptyset \vdash M : P \multimap A}{\emptyset \vdash \mathrm{box}(M) : \mathrm{QChan}(P,A)} \qquad \frac{}{\emptyset \vdash \mathrm{unbox} : (\mathrm{QChan}(P,A) \otimes P) \multimap A}$$

Circuit-buffering operational semantics

Circuit-buffering abstract machine operates on quantum channel while reducing the term

Configuration is represented by a pair $((Q, b), m)$ consisting of an accessible quantum channel (Q, b) and a branching term m

We use a graphical representation of configuration where the green box represents the accessible quantum channel and each square-boxed term is linked to a leaf.

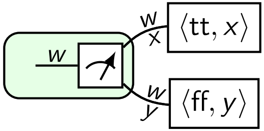

w
w
x
$\langle \mathrm{tt}, x \rangle$
w
y
$\langle \mathrm{ff}, y \rangle$

Figure 2: Graphical representation of a configuration with measurement

The labels above the edge show the bundle of wires while the labels for the variables are below the edge.

Reduction can take place in each branch of branching term.

QRAM-based operational semantics

QRAM-based operational semantics simulates quantum circuit by applying the corresponding operation on quantum register

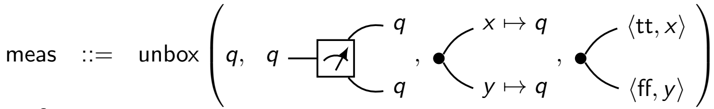
Configuration is $(\phi, L, (Q, b), m)$ where (ϕ, L) is the state of a quantum memory with linking function and $((Q, b), m)$ is a circuit-buffering configuration

The reduction $\overset{p}{\Rightarrow}$ is probabilistic relation over QRAM-based configurations with probability p

Accessible quantum channel is generated according to the circuit-buffering operational semantics

Semantics of measurement

A constant meas of type qubit $\multimap$ bool $\otimes$ qubit is defined as:



Behavior of measurement:

- Classical computation: creates states for each outcome
- Quantum channel: add a measurement gate to the circuit

Formally by circuit-buffering operational semantics:

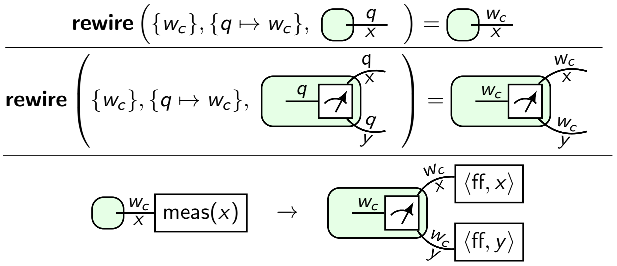


Figure 3: Application of the reduction rule for meas

Example of non-trivial circuit construction

The program in Eq. (1) measures the qubit v_c and construct circuits depending on the measurement.

$$\begin{aligned} \text{exp} &::= \textbf{let } \langle b, v_c \rangle = \text{meas}(v_c) \textbf{ in } T \\ T &::= \textbf{if } b \textbf{ then } \langle \text{init}(\text{tt}), \text{free}(v_c) \rangle \textbf{ else } \langle v_c, * \rangle \end{aligned} \tag{1}$$

Despite simple structure, the program does not correspond to a circuit because of the classical control.

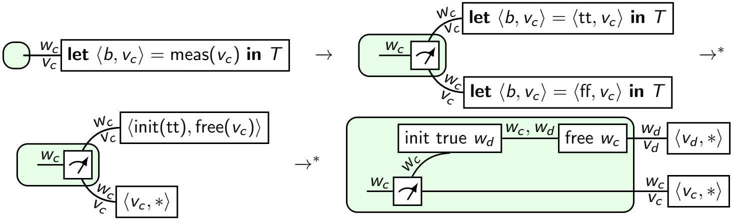

w_c v_c **let** $\langle b, v_c \rangle = \text{meas}(v_c)$ **in** T
$\rightarrow$
w_c
w_c v_c **let** $\langle b, v_c \rangle = \langle \text{tt}, v_c \rangle$ **in** T
w_c v_c **let** $\langle b, v_c \rangle = \langle \text{ff}, v_c \rangle$ **in** T
$\rightarrow^*$
w_c
w_c v_c $\langle \text{init}(\text{tt}), \text{free}(v_c) \rangle$
w_c v_c $\langle v_c, * \rangle$
$\rightarrow^*$
init true w_d
w_c, w_d
free w_c
w_d v_d $\langle v_d, * \rangle$
w_c
w_c
w_c v_c $\langle v_c, * \rangle$

Figure 4: Reduction of a term creating a branching term and a quantum channel with measurement

Conclusion

Summary: we have:

- defined a simple language extending both the quantum lambda calculus and ProtoQuipper equipped with a strictly linear type system
- formalized a semantics of dynamic lifting and non-trivial classical control flows generated by measurements in quantum circuit construction
- shown a subject reduction and a progress lemma for both circuit-buffering and QRAM-based operational semantics

Future goal:

- extend the formalization of the language to important features such as recursion, inductive data types
- linearity and non-linearity type system

Thank You!

References

[1] A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, and B. Valiron, "Quipper: A scalable quantum programming language", in *ACM SIGPLAN Notices*, ACM, vol. 48, 2013, pp. 333–342.

[2] J. Paykin, R. Rand, and S. Zdancewic, "Qwire: A core language for quantum circuits", in *ACM SIGPLAN Notices*, ACM, vol. 52, 2017, pp. 846–858.

[3] N. J. Ross, "Algebraic and logical methods in quantum computation", *PhD thesis, Dalhousie University*, 2015.

[4] P. Selinger, "A survey of graphical languages for monoidal categories", in *New structures for physics*, Springer, 2010, pp. 289–355.

[5] F. Rios and P. Selinger, "A categorical model for a quantum circuit description language", *arXiv preprint arXiv:1706.02630*, 2017.

[6] B. Lindenhovius, M. Mislove, and V. Zamdzhiev, "Enriching a linear/non-linear lambda calculus: A programming language for string diagrams", in *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, ACM, 2018, pp. 659–668.

[7] S. Staton, "Algebraic effects, linearity, and quantum programming languages", *SIGPLAN Not.*, vol. 50, no. 1, pp. 395–406, Jan. 2015, ISSN: 0362-1340. DOI: 10.1145/2775051.2676999. [Online]. Available: http://doi.acm.org/10.1145/2775051.2676999.

[8] M. Rennela, "Enrichment in categorical quantum foundations", PhD thesis, Nov. 2018. DOI: 10.13140/RG.2.2.10344.52485.

[9] P. Selinger and B. Valiron, "A lambda calculus for quantum computation with classical control", *Mathematical Structures in Computer Science*, vol. 16, no. 3, pp. 527–552, 2006.