A formalization of the Quipper quantum programming language

Henri Chataing, Neil J. Ross & Peter Selinger

Dalhousie University & École Polytechnique

2014 TYPES Meeting
Quantum computing is computing based on the laws of quantum physics.

The standard model of quantum computing is Knill’s Qram model, in which a classical computer is connected to a quantum device.
The instructions for the quantum device are arranged in a quantum circuit.

The gates that compose quantum circuits can be *unitaries*, which are reversible operations, or *measurements*, which are probabilistic operations.
Quipper is a programming language for quantum computing, implemented as an embedded language within Haskell.

Several non-trivial algorithms from the quantum computing literature have been implemented in Quipper.

Quipper is a circuit description language.
Quipper’s circuit as data paradigm.

```
circuit :: [Qubit] -> Circ ([Qubit], [Qubit])
circuit qs = do
  y <- with_computed subcircuit $ \subcircuit -> do
    qc_copy subcircuit
  return (qs, y)
```
Quipper’s type system does not guarantee that quantum programs are physically meaningful.

```plaintext
self_control :: Qubit -> Circ Qubit
self_control q = do
    qnot_at q 'controlled' q
    return q
```
Goals:

▶ Define a type-safe language, Proto-Quipper, that will serve as a basis for the development of Quipper as a stand-alone language.

Chosen features for Proto-Quipper:

▶ Have a type system to enforce the physics (draw inspiration from the quantum lambda calculus).
▶ Capture Quipper's circuits as data paradigm.

Simplifying assumption:

▶ No measurements (all circuits are therefore reversible).
The Proto-Quipper language:

Type \( A, B ::= 1 \mid \text{bool} \mid A \otimes B \mid A \multimap B \mid !A \mid \) 

\( \text{qubit} \mid \text{Circ}(T, U) \)

QDataType \( T, U ::= \text{qubit} \mid 1 \mid T \otimes U \)

Term \( a, b, c ::= \ldots \mid q \mid (t, C, a) \mid \text{box}^T \mid \text{unbox} \mid \text{rev} \)

QDataTerm \( t, u ::= q \mid * \mid \langle t, u \rangle \)
Some basic built-in gates:

- HAD := \text{unbox}(q, \text{HAD}, q)
- CNOT := \text{unbox}(\langle q_1, q_2 \rangle, \text{CNOT}, \langle q_1, q_2 \rangle)
- INIT0 := \text{unbox}(\ast, 0, q)

A Proto-Quipper term (not quite) for subcircuit:

\[
\text{subcircuit} := \text{box}^{\text{qubit}}(\lambda x.\text{CNOT}(\text{HAD } x, \text{INIT0 } \ast))
\]

\[
\begin{array}{c}
\text{H} \\
(0) \\
\langle q, q' \rangle
\end{array}
\]
Proto-Quipper’s operational semantics supposes a circuit constructor.

The circuit constructor is assumed to be able to perform some basic operations: appending gates, reversing circuits, . . .

The reduction will be defined on closures $[C, t]$ consisting of a term $t$ of the language and a circuit state $C$ representing the circuit currently being built.
The operational semantics of Proto-Quipper (a selection):

\[
\text{Spec}_{\text{FQ}(v)}(T) = t \quad \text{new}(\text{FQ}(t)) = D
\]

\[
[C, \text{box}^T(v)] \rightarrow [C, (t, D, vt)]
\]

\[
[D, a] \rightarrow [D', a']
\]

\[
[C, (t, D, a)] \rightarrow [C, (t, D', a')]
\]

\[
\text{bind}(v, u) = b \quad \text{Append}(C, D, b) = (C', b') \quad \text{FQ}(u') \subseteq \text{dom}(b')
\]

\[
[C, (\text{unbox}(u, D, u')) v] \rightarrow [C', b'(u')]
\]
subcircuit := $box^{qubit}(\lambda x.\text{CNOT}(\text{INIT0} *, \text{HAD } x))$

$[\cdot, \text{subcircuit}] \rightarrow [\quad, \text{CNOT}(\text{INIT0} *, \text{HAD } q))]$

$\rightarrow [\quad, \text{CNOT}(\text{INIT0} *, q))]$

$\rightarrow [\quad, \text{CNOT}(q', q))]$

$\rightarrow [\quad, \langle q', q \rangle]$

$\rightarrow [\cdot, (q, C, \langle q', q \rangle)]$
For each of the constants $\text{box}^T$, $\text{unbox}$, and $\text{rev}$, we introduce a type:

- $A_{\text{box}}(T, U) = !((T \rightarrow U) \circ ! \text{Circ}(T, U))$,
- $A_{\text{unbox}}(T, U) = \text{Circ}(T, U) \circ !(T \rightarrow U)$, and
- $A_{\text{rev}}(T, U) = \text{Circ}(T, U) \circ ! \text{Circ}(U, T)$.

And a typing rule, for $c \in \{\text{box}^T, \text{unbox}, \text{rev}\}$:

$$
\frac{!A_c(T, U) <: B}{!\Delta; \emptyset \vdash c : B}
$$
The type system of Proto-Quipper (a selection):

\[
\begin{align*}
A <: B & \quad \frac{!\Delta, x : A; \emptyset \vdash x : B}{(ax_c)} \\
!\Delta; \{q\} \vdash q : \text{qubit} & \quad \frac{!\Delta; \emptyset \vdash b : B}{(ax_q)} \\
\Gamma, x : A; Q \vdash b : B & \quad \frac{\Gamma; Q \vdash \lambda x. b : A \rightarrow B}{(\lambda_1)} \\
!\Delta; \emptyset \vdash \lambda x. b : !^{n+1}(A \rightarrow B) & \quad \frac{!\Delta, x : A; \emptyset \vdash b : B}{(\lambda_2)} \\
\Gamma_1, !\Delta; Q_1 \vdash a : !^n A & \quad \frac{\Gamma_2, !\Delta; Q_2 \vdash b : !^n B}{(\otimes-i)} \\
\Gamma_1, \Gamma_2, !\Delta; Q_1, Q_2 \vdash \langle a, b \rangle : !^n(A \otimes B) & \\
Q_1 \vdash t : T & \quad \frac{!\Delta; Q_2 \vdash a : U}{(\circ)} \\
\text{In}(C) = Q_1 & \quad \text{Out}(C) = Q_2 \\
Q_1 \vdash (t, C, a) : !^n \text{Circ}(T, U) & \quad \frac{!\Delta; \emptyset \vdash (t, C, a) : !^n \text{Circ}(T, U)}{(circ)}
\end{align*}
\]
Proto-Quipper is a type-safe language. It enjoys subject reduction and progress.

*Subject reduction:* If \( \Gamma; FQ(a) \vdash [C, a] : A, (Q'\mid Q'') \) is a valid typed closure and \( [C, a] \rightarrow [C', a'] \), then \( \Gamma; FQ(a') \vdash [C', a'] : A, (Q'\mid Q'') \) is a valid typed closure.
References:

- P. Selinger and B. Valiron. *Quantum lambda calculus*. 