Linear dependent types for quantum circuit programming

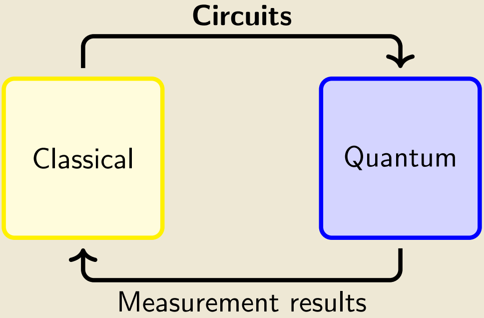
Frank Fu

Dalhousie University

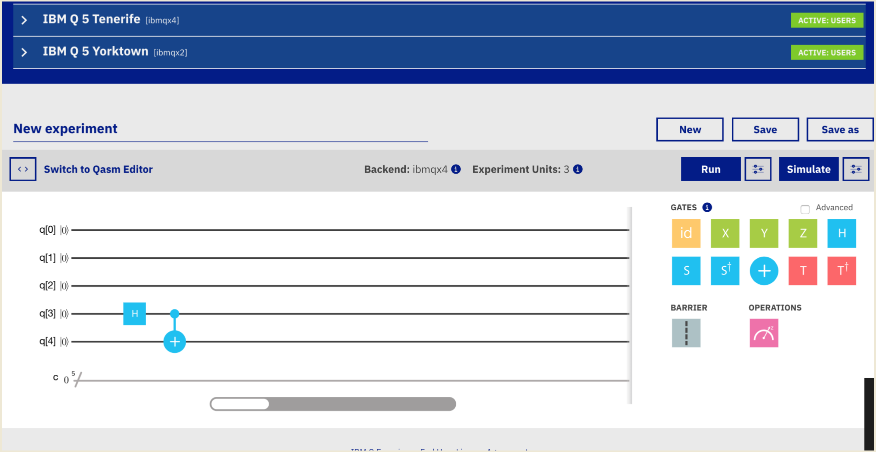
Joint work with: Kohei Kishida, Neil J. Ross, Peter Selinger

Feb 17, 2020, PL reading group, Maryland

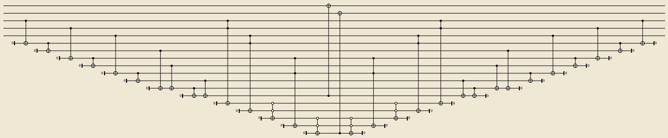
QRAM model



Quantum circuits



Quantum circuit programming languages

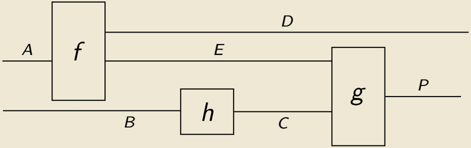


- QISKit, Q#, Cirq, ProjectQ.
- Quipper.

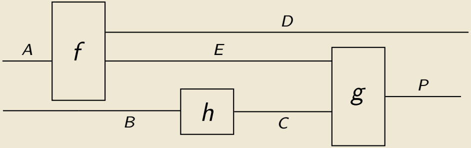
Linear types for quantum circuit programming

- No-cloning: $|\phi\rangle \mapsto |\phi\rangle \otimes |\phi\rangle$.
- Qwire.
- Proto-Quipper.

Linear types: $A \multimap B$, $A \otimes B$

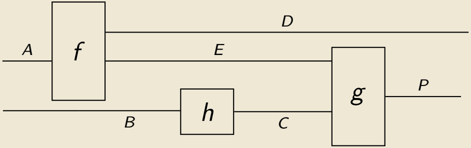


Linear types: $A \multimap B$, $A \otimes B$



$$(1_D \otimes g) \circ (1_D \otimes 1_E \otimes h) \circ (f \otimes 1_B)$$

Linear types: $A \multimap B$, $A \otimes B$



$$(1_D \otimes g) \circ (1_D \otimes 1_E \otimes h) \circ (f \otimes 1_B)$$

```
λ a b ->
  let (d, e) = f a
      c = h b
      p = g e c
  in (d, p)
```

Parameters and states

- Values that are known at circuit generation time (parameters).
- Values that are known at circuit execution time (states).
- For example, a *family* of circuits indexed by parameters.
- Qwire: parameters and states live in different name spaces.
- Proto-Quipper: parameters and states live in the same name space.

Boxing and simple types

- We use simple types to describe (quantum) states. E.g., $\mathbf{Unit}, \mathbf{Qubit}, \mathbf{Qubit} \otimes \mathbf{Qubit}$.
- Box/unbox isomorphism.

$$\begin{aligned} \mathrm{box} &: (a \multimap b) \to \mathbf{Circ}(a, b) \\ \mathrm{unbox} &: \mathbf{Circ}(a, b) \to (a \multimap b) \end{aligned}$$

  Note that a, b must be simple types.
- List of qubits is not a simple type because

$$\mathbf{List}\,\mathbf{Qubit} \cong \mathbf{Unit} + \mathbf{Qubit} + \mathbf{Qubit} \otimes \mathbf{Qubit} + \ldots$$

A problem with quantum data types

- Can't box qft : $\mathbf{List\,Qubit} \multimap \mathbf{List\,Qubit}$.
- Can't reverse qft : $\mathbf{List\,Qubit} \multimap \mathbf{List\,Qubit}$.

A problem with quantum data types

- Can't box $\mathit{qft} : \mathbf{List\,Qubit} \multimap \mathbf{List\,Qubit}$.
- Can't reverse $\mathit{qft} : \mathbf{List\,Qubit} \multimap \mathbf{List\,Qubit}$.
- Dependent types to the rescue.
  $\mathit{qft} : (n : \mathbf{Nat}) \multimap (\mathbf{Vec\,Qubit}\ n \multimap \mathbf{Vec\,Qubit}\ n)$.
- Note that $\mathbf{Vec\,Qubit}\ n$ is a simple type.

$$\mathbf{Vec\,Qubit}\ n \cong \underbrace{\mathbf{Qubit} \otimes \mathbf{Qubit} \otimes \ldots \otimes \mathbf{Qubit}}_{n}$$

Incorporating linear and dependent types

Let $f : (x : A) \multimap B[x]$ and $a : A$, what is the meaning of $f\ a : B[a]$?

- Cervesato and Pfenning 1996. The term a is used twice, therefore it should be a parameter.
- McBride 2016. The term a in the type $B[a]$ is not considered used, it is only “comtemplated”.
- Our proposal (Fu, Kishida and Selinger 2020). The term a is used once, and the type B depends only on the shape of a.

Shape operation

$$
\begin{array}{lcl}
\textit{Linear} & \supseteq & \textit{Parameters} \\
\hline
\\
\mathrm{Sh}(\textbf{Qubit}) & = & \textbf{Unit} \\
\mathrm{Sh}(\textbf{List}\, A) & = & \textbf{List}\, \mathrm{Sh}(A) \\
\mathrm{Sh}(A \otimes B) & = & \mathrm{Sh}\, A \otimes \mathrm{Sh}\, B \\
\mathrm{Sh}((x : A) \multimap B[x]) & = & (x : \mathrm{Sh}(A)) \to \mathrm{Sh}(B[x])
\end{array}
$$

Types depend on shapes

- Kinding linear dependent types:

$$\frac{\Phi, x : \mathrm{Sh}(A) \vdash B[x] : *}{\Phi \vdash (x : A) \multimap B[x] : *}$$

Types depend on shapes

- Kinding linear dependent types:

$$\frac{\Phi, x : \mathrm{Sh}(A) \vdash B[x] : *}{\Phi \vdash (x : A) \multimap B[x] : *}$$

- Typing rule for application:

$$\frac{\Gamma_1 \vdash M : (x : A) \multimap B[x] \quad \Gamma_2 \vdash N : A}{\Gamma_1 + \Gamma_2 \vdash MN : B[\mathrm{Sh}(N)]}$$

For $\Gamma_2 \vdash N : A$, we have:

$$\mathrm{Sh}(\Gamma_2) \vdash \mathrm{Sh}(N) : \mathrm{Sh}(A)$$

Proto-Quipper-D

- A linear dependent type system for programming quantum circuits.
- A categorical semantics based on *state-parameter fibration*.
- A prototype implementation of Proto-Quipper-D.

More details can be found in our recent drafts.

- "Linear dependent type theory for quantum programming panguages", Fu, Kishida and Selinger, 2020.
- "An introduction to quantum circuit programming in dependently typed Proto-Quipper". Fu, Kishida, Ross and Selinger, 2020.

Demo. An implementation is available at
`https://gitlab.com/frank-peng-fu/dpq-remake`.

Thank you for your attention!