Designing Quantum Programming Languages with Types

Frank Fu

Computer Science and Engineering Department, UofSC
Why quantum programming languages?

- Researchers have shown quantum algorithms can offer substantial speed-up for certain computing tasks.
- Advances in quantum hardware from companies like IBM and Google.
- Quantum algorithms are usually expressed using quantum circuits.
- Quantum algorithms are commonly expressed at a high level.
- Debugging quantum algorithms can be expensive.
My research interest

Build tools to facilitate programming quantum computers.

► How to design a high-level programming language for quantum circuits?
► How to verify quantum programs?
► How to run a high-level programming language on actual quantum computer?
► What algorithms to run on current quantum computer?
Why types?

- Lightweight specifications of programs.
- Allow compiler to enforce invariants via type checking.
- A well-typed program satisfies certain properties.
Background on types: an idealized programming language

- Programs $M, N := x | \lambda x. M | MN$.
- Types $A, B := C | A \rightarrow B$.
- Typing environment $\Gamma = x_1 : A_1, ..., x_n : A_n$.
- Typing judgment $\Gamma \vdash M : A$.
- Typing rules

\[
\begin{align*}
(x : A) \in \Gamma & \quad \Rightarrow \quad \Gamma \vdash x : A \\
\Gamma, x : A \vdash M : B & \quad \Rightarrow \quad \Gamma \vdash \lambda x. M : A \rightarrow B \\
\Gamma \vdash M : A \rightarrow B & \quad \& \quad \Gamma \vdash N : A \\
& \quad \Rightarrow \quad \Gamma \vdash MN : B
\end{align*}
\]
Type safety

- A *type checker* checks $\Gamma \vdash M : A$.
- An *evaluator* performs evaluation $M \downarrow V$.
- *Type safety*
  If $\Gamma \vdash M : A$ and $M \downarrow V$, then $\Gamma \vdash V : A$. 

Fancy types

- Linear types: $A \rightarrow B$.
- Dependent types: $(n : \text{Nat}) \rightarrow \text{Vec} A n \rightarrow \text{Vec} A n$.
- Types with modalities: $A \rightarrow_{\alpha} B$. 
Types for Quantum Computing

The basic types in Quantum Computing.

- **Bit**: $|0\rangle, |1\rangle$.

- **Qubit**: $|\phi\rangle = \alpha|0\rangle + \beta|1\rangle$, where $\alpha, \beta \in \mathbb{C}, |\alpha|^2 + |\beta|^2 = 1$.

- Multi-qubits are represented by a tensor product. Qubit $\otimes$ Qubit, Qubit $\otimes$ Qubit $\otimes$ Qubit, Qubit $\otimes$ Bit, etc.
Qubits are resource

▶ No cloning: *one can not duplicate a qubit.*

\[
\text{dup } x = (x, x)
\]

▶ Qubit does not exist in a vacuum.

\[\text{Init0} : \text{Unit} \to \text{Qubit}\]

\[
\text{let } x = \text{Init0} () \text{ in } ...
\]

▶ Qubit does not disappear into the ether.

\[\text{Discard} : \text{Qubit} \to \text{Unit}\]

\[
\text{let } x = \text{Init0} () \text{ in } ...
\]
\[
\text{let } _ = \text{Discard} x \text{ in } ...
\]
One way to update qubits is via *unitary operations*.

- Reversibility: $UU^\dagger = U^\dagger U = I$.

- Linearity: $U(\alpha|0\rangle + \beta|1\rangle) = \alpha U|0\rangle + \beta U|1\rangle$. 
Common quantum gates

▶ Hadamard gate.

\[
H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)
\]
\[
H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)
\]

▶ Phase gate.

\[
S|0\rangle = |0\rangle
\]
\[
S|1\rangle = i|1\rangle
\]

▶ T gate.

\[
T|0\rangle = |0\rangle
\]
\[
T|1\rangle = \omega |1\rangle, \text{ where } \omega^2 = i
\]

▶ CNOT gate.

\[
\text{CNOT}|00\rangle = |00\rangle \quad \text{CNOT}|01\rangle = |01\rangle
\]
\[
\text{CNOT}|10\rangle = |11\rangle \quad \text{CNOT}|11\rangle = |10\rangle
\]
Types for quantum gates

- Hadamard gate.

\[ H : \text{Qubit} \rightarrow \text{Qubit} \]

- Phase gate.

\[ S : \text{Qubit} \rightarrow \text{Qubit} \]

- T gate.

\[ T : \text{Qubit} \rightarrow \text{Qubit} \]

- CNOT gate.

\[ \text{CNOT} : \text{Qubit} \otimes \text{Qubit} \rightarrow \text{Qubit} \otimes \text{Qubit} \]
Measurement

Measurement is needed to readout the bit information from qubit.

\[ \text{Meas} : \text{Qubit} \rightarrow \text{Bit} \]

\[ M(\alpha|0\rangle + \beta|1\rangle) = |0\rangle \text{ with probability } |\alpha|^2. \]

\[ M(\alpha|0\rangle + \beta|1\rangle) = |1\rangle \text{ with probability } |\beta|^2. \]
bell00 : !(Unit -> Qubit * Qubit)
bell00 u =
  let a = Init0 ()
  b = Init0 ()
  in CNot b (H a)
Programming quantum circuits in Proto-Quipper

alice : !(Qubit -> Qubit -> Bit * Bit)
alice a q =
  let (a, q) = CNot a q
    q = H q
  in (Meas a, Meas q)
Programming quantum circuits in Proto-Quipper

```
bob : !(Qubit -> Bit -> Bit -> Qubit)
bob q x y =
  let (q, x) = C_X q x
    (q, y) = C_Z q y
    _ = Discard x
    _ = Discard y
  in q
```
tele : !(Qubit -> Qubit)

```haskell
tele q =
    let (b, a) = bell00 ()
        (x, y) = alice a q
        z = bob b x y
    in z
```

```
H
Meas
X  Z

0
H
Meas
X  Z

0
```
Interleaving circuit generation time and circuit execution via dynamic lifting

```
alice2 : !(Qubit -> Qubit -> Bool * Bool)
alice2 a q =
    let (a, q) = CNot a q
    q = H q
    in (dylift (Meas a), dylift (Meas q))

bob2 : !(Qubit -> Bool -> Bool -> Qubit)
bob2 q x y =
    let q = if x then QNot q else q
    q = if y then ZGate q else q
    in q
```

Nat, Bool

Qubit, Bit

Quantum computer
Types for dynamic lifting

- $\Gamma \vdash_\alpha M : A$, where $\alpha = 0 \mid 1$.

- Dynamic lifting.

  \[
  \Gamma \vdash_\alpha M : \text{Bit} \\
  \Gamma \vdash_0 \text{dynlift } M : \text{Bool}
  \]

- Type system distinguishes computation that uses dynamic lifting vs computation that corresponds to quantum circuits.
v3 : !(Qubit -> Qubit)
v3 q =
  let a1 = tgate_inv (H (Init0 ()))
    a2 = H (Init0 ())
    (a1, a2) = CNot a1 a2
    a1 = H (TGate a1)
  in if dynlift (Meas a1)
    then
      let _ = Discard (Meas a2)
      in v3 q
    else let q = ZGate (TGate q)
      (a2, q) = CNot a2 q
      a2 = H (TGate a2)
      in if dynlift (Meas a2)
        then v3 (ZGate q)
        else q
Future research

▶ How do we verify the correctness of a quantum program?
  ▶ How to prove two quantum circuits are equal?
  ▶ How to develop tests to ensure the programs perform correctly?

▶ How do we compile a high-level quantum programs to lower level languages (e.g., QIR, OpenQasm)?

▶ Suppose we have a 127 Qubits machine, what algorithms should we run on it?
Thank you!