Programming Quantum Circuits in Proto-Quipper

Frank Fu

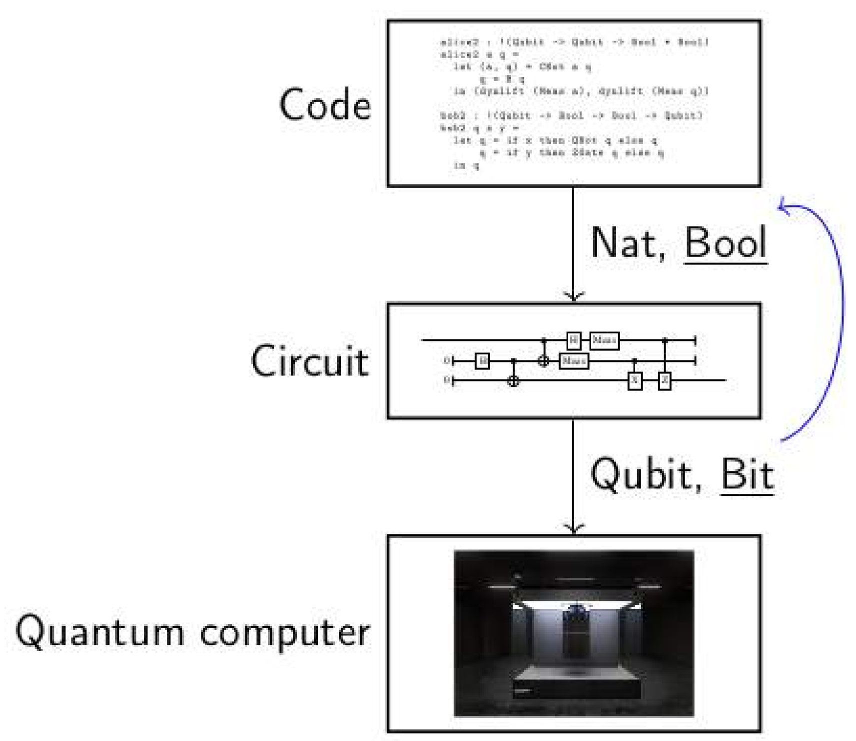
Computer Science and Engineering Department, University of South Carolina

Objectives of Proto-Quipper

- Provide formal foundations for high-level quantum programming languages.
- Identify high-level programming constructs for quantum circuits.
- Support programming a wide range of quantum algorithms.

Introduction

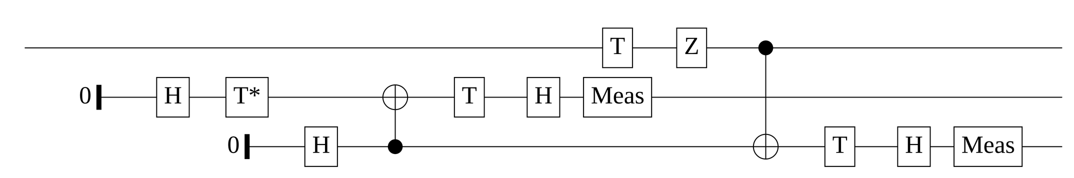
Quipper is a functional programming language for quantum computing. Proto-Quipper is a family of languages aiming to provide a formal foundation for Quipper. We added a notion of dependent types to Proto-Quipper in 2020, which enables Proto-Quipper to express family of quantum circuits indexed by parameters. Recently, we have extended Proto-Quipper with a construct called dynamic lifting. Dynamic lifting is an operation that enables a state, such as the result of a measurement, to be lifted to a parameter, where it can influence the generation of the next portion of the circuit. As a result, dynamic lifting enables Proto-Quipper programs to interleave classical and quantum computation.



A Repeat-Until-Success Algorithm

The repeat-until-success paradigm provides a technique to apply a unitary that cannot be implemented exactly, at the cost of potentially running the same circuit multiple times. In order to apply a non-Clifford+T gate N to a target qubit $|\phi\rangle$, one first initializes several ancillary qubits before applying a well-chosen Clifford+T circuit C to the target and the ancillas and measuring the ancillas. If all of the measurement results are 0, the target qubit is guaranteed to be in the state $N\,|\phi\rangle$. Otherwise, a correction is applied to the target to return it to its initial state and the process is repeated.

The following circuit gives an illustration of how to implement the gate $V_3 = \frac{I+2iZ}{\sqrt{5}}$ [Paetznick and Svore 2014].



Proto-Quipper Program for V_3

The following is a Proto-Quipper program that implements the repeat-until-success algorithm for V_3 gate outlined above.

```
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
```

Quantum Fourier Transform

The quantum Fourier transform is the map defined by

$$|a_1,\dots,a_n\rangle \mapsto \frac{(|0\rangle + e^{2\pi i 0.a_1a_2\dots a_n}|1\rangle)\dots(|0\rangle + e^{2\pi i 0.a_{n-1}a_n}|1\rangle)(|0\rangle + e^{2\pi i 0.a_n}|1\rangle)}{2^{n/2}}.$$

Let us define the controlled rotation gates $R(k)$ by

$$R(k) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{2\pi i/2^k} \end{pmatrix}.$$

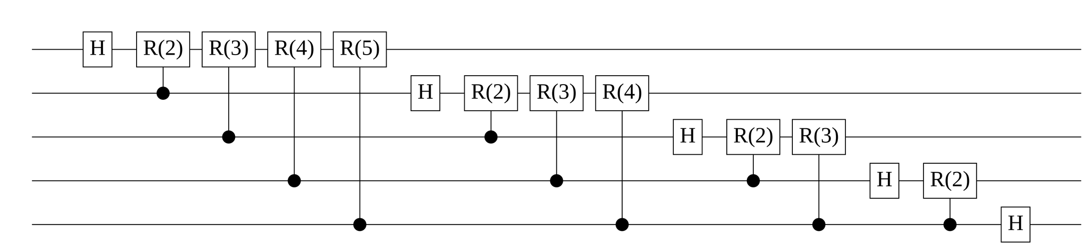
Applying the Hadamard gate to the first qubit produces the following state

$$H_1|a_1,\dots,a_n\rangle = \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 0.a_1}|1\rangle) \otimes |a_2,\dots,a_n\rangle,$$

where the subscript on the gate indicates the qubit on which the gate acts. We then apply a sequence of controlled rotations using the the first qubit as the target. This yields

$$R(n)_{1,n}\dots R(2)_{1,2}H_1|a_1,\dots,a_n\rangle = \frac{1}{2^{1/2}}(|0\rangle + e^{2\pi i 0.a_1a_2\dots a_n}|1\rangle)\otimes|a_2,\dots,a_n\rangle,$$

where the subscripts i and j in $R(k)_{i,j}$ indicate the target and control qubit, respectively. The following circuit corresponds to QFT on 5 qubits.



QFT in Proto-Quipper

The following is a Proto-Quipper program describes QFT as a family of circuits parameterized by the number of qubits.

```
qft : ! forall (n : Nat) -> Vec Qubit n -> Vec Qubit n
qft v =
  case v of
    VNil -> VNil
    VCons q qs ->
      let q' = H q
          (q'', qs') = rotate 2 q' qs
          qs'' = qft qs'
      in VCons q'' qs''

qftBox : ! (n : Nat) -> Circ(Vec Qubit n, Vec Qubit n)
qftBox n = box (Vec Qubit n) qft
```

Summary and Future Work

We show Proto-Quipper programs for a repeat-until-success algorithm and Quantum Fourier Transform. They showcase the use of dynamic lifting and dependent types. For future work, on the aspect of language design, we are investigating the support for controlling and reversing quantum circuits in Proto-Quipper. On the aspect of implementation and compilation, we plan to explore the possibility of compiling Proto-Quipper to lower level languages such as OpenQASM and QIR.

References

[1] Peng Fu, Kohei Kishida, Neil J. Ross, and Peter Selinger. Proto-quipper with dynamic lifting. *Proc. ACM Program. Lang.*, 7(POPL), jan 2023.

[2] Peng Fu, Kohei Kishida, and Peter Selinger. Linear Dependent Type Theory for Quantum Programming Languages. *Logical Methods in Computer Science*, Volume 18, Issue 3, September 2022.

[3] Peng Fu, Kohei Kishida, Neil J. Ross, and Peter Selinger. A tutorial introduction to quantum circuit programming in dependently typed Proto-Quipper. In *Proceedings of the 12th International Conference on Reversible Computation, RC 2020, Oslo, Norway*, volume 12227 of *Lecture Notes in Computer Science*, pages 153–168. Springer, 2020.

Acknowledgements

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

Contact Information

- Gitlab repository: https://gitlab.com/frank-peng-fu/dpq-remake
- Web: https://cse.sc.edu/~pfu
- Email: pfu@cse.sc.edu