Quantum Hoare Type Theory

Kartik Singhal

ks.cs.uchicago.edu

MS Presentation

December 9, 2020

EPiQC

THE UNIVERSITY OF CHICAGO

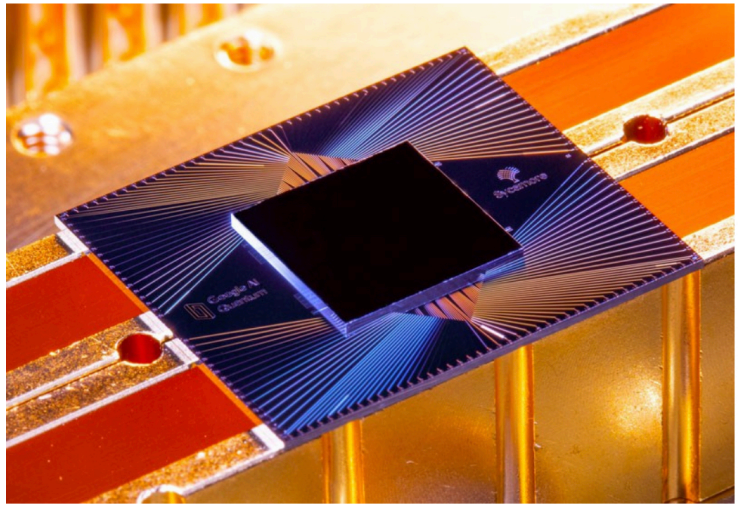
Quantum machines are here!

NEWS · 23 OCTOBER 2019

Hello quantum world! Google publishes landmark quantum supremacy claim

The company says that its quantum computer is the first to perform a calculation that would be practically impossible for a classical machine.

Elizabeth Gibney

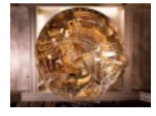


The Sycamore chip is composed of 54 qubits, each made of superconducting loops. Credit: Erik Lucero

PDF version

RELATED ARTICLES

Beyond quantum supremacy



Quantum gold rush: the private funding pouring into quantum start-ups



Physicists propose football-pitch-sized quantum computer



REPORT

Quantum computational advantage using photons

Han-Sen Zhong[1,2,*], Hui Wang[1,2,*], Yu-Hao Deng[1,2,*], Ming-Cheng Chen[1,2,*], Li-Chao Peng[1,2], Yi…

+ See all authors and affiliations

Science 03 Dec 2020:
eabe8770
DOI: 10.1126/science.abe8770

Article | Figures & Data | Info & Metrics | eLetters | PDF

Abstract

Quantum computers promises to perform certain tasks that are believed to be intractable to classical computers. Boson sampling is such a task and is considered as a strong candidate to demonstrate the quantum computational advantage. We perform Gaussian boson sampling by sending 50 indistinguishable single-mode squeezed states into a 100-mode ultralow-loss interferometer with full connectivity and random matrix—the whole optical setup is phase-locked—and sampling the output using 100 high-efficiency single-photon detectors. The obtained samples are validated against plausible hypotheses exploiting thermal states, distinguishable photons, and uniform distribution. The photonic quantum computer generates up to 76 output photon clicks, which yields an output state-space dimension of 10^{30} and a sampling rate that is ~10^{14} faster than using the state-of-the-art simulation strategy and supercomputers.

The Extended Church-Turing Thesis is a foundational tenet in computer science, which states that a probabilistic Turing machine can efficiently simulate any process on a realistic physical device (*1*). In the 1980s, Richard Feynman observed that many-body quantum problems seemed difficult for classical computers due to the exponentially growing size of the quantum state Hilbert space. He proposed that a quantum computer would be a natural solution.

A long-term vision for computing

“While a civil engineer as a professional provides specific guarantees about the efficacy and failure modes of a bridge, a software engineer for a software product, does not. Computing as a scientific discipline has come a long way, but as an engineering discipline, it has miles to go.”

We failed to achieve this goal with classical software over the last 70 years; quantum computing, as a fundamentally different way of computation, provides a unique opportunity to do things right in a principled manner.

Approaches to Software Reliability

- Social
  - Code reviews
  - Extreme/Pair programming
- Methodological
  - Design patterns
  - Test-driven development
  - Version control
  - Bug tracking
- Technological
  - "lint" tools, static analysis
  - Fuzzers, random testing
- Mathematical
  - Sound type systems
  - Formal verification



Less "formal": Lightweight, inexpensive techniques (that may miss problems)

This isn't a tradeoff… all of these methods should be used.

Even the most "formal" argument can still have holes:

- Did you prove the right thing?
- Do your assumptions match reality?
- Knuth: *"Beware of bugs in the above code; I have only proved it correct, not tried it."*

More "formal": eliminate *with certainty* as many problems as possible.

Slide from Benjamin Pierce's CIS500: Software Foundations, Lecture 1

The need for reliable quantum software

Reasoning techniques from classical computing do not transfer:

State collapse on observation of state.

Simulation beyond tens of qubits is expensive or impossible.

The cost of running a program on a physical quantum computer is high; it gets worse if we run *incorrect* programs.

Hence, it is important to ensure correctness statically.

Our approach:

Adapt the best of classical static typing techniques to the quantum realm.

Bring reasoning techniques as close to programming as possible.

Kartik Singhal and John Reppy (2020). *Quantum Hoare Type Theory: Extended Abstract*. QPL

A map of this talk

Motivation

Background

- Floyd-Hoare Logic
- Hoare Type Theory (HTT)

} Classical

- Quantum Computation
- Quantum Hoare Logic

} Quantum

Quantum Hoare Type Theory (QHTT)

- Primitive Commands
- Teleportation

Recap

Future Directions

Floyd-Hoare logic

Established technique in classical programming
to reason about imperative effectful programs.

Hoare triple:

$$\{P\}\ c\ \{Q\}$$

c is the program (command) to be executed
P and *Q* are pre- and postconditions on the program variables (state)

"If *P* is true before execution of *c*, then *Q* will be true on its completion."

Example:

$$\{x = 1\}\ x := 5\ \{x > 0\}$$

Reasoning in Hoare logic

Axioms and rules of inference

AXIOM OF ASSIGNMENT

$$\{P[x \to e]\}\ x := e\ \{P\}$$

RULE OF CONSEQUENCE

$$\frac{P \Rightarrow Q \qquad \{Q\}\ c\ \{R\} \qquad R \Rightarrow S}{\{P\}\ c\ \{S\}}$$

Precondition Strengthening

Postcondition Weakening

RULE OF COMPOSITION

$$\frac{\{P\}\ c\ \{Q\} \qquad \{Q\}\ d\ \{R\}}{\{P\}\ c\ ;\ d\ \{R\}}$$

Example proof in Hoare logic

Swapping two assignable variables:

$$\{x = m \wedge y = n\}\ t := x;\ x := y;\ y := t\ \{y = m \wedge x = n\}$$

Using assignment axiom and composition rule, we get:

$$\{x = m \wedge y = n\}\ t := x;\ x := y\ \{t = m \wedge x = n\}$$

And again:

$$\{x = m \wedge y = n\}\ t := x\ \{t = m \wedge y = n\}$$

Ghost variables

Hoare Type Theory (HTT) combines the power of Hoare logic and dependent types

Hoare types specify pre- and postconditions and are very expressive

$$\Delta.\{P\}\ x : A\ \{Q\}$$

P, *Q* are pre- and postconditions (as before)
x is the return value of type *A*
Δ is an optional context to include ghost variables

For example, here is the type of the swap function in HTT:

$$\forall\alpha.\forall\beta.\Pi x{:}\,nat.\Pi y{:}\,nat.$$
$$m{:}\,\alpha, n{:}\,\beta.\{x =_\alpha m\ *\ y =_\beta n\}\ r{:}\,unit\ \{x =_\alpha n\ *\ y =_\beta m\}$$

which is a polymorphic function that takes as input two locations of type *nat*, swaps their contents and returns a result of type *unit.*

Nanevski *et al* (2008). *Hoare Type Theory, Polymorphism and Separation*. J. Funct. Program.

F*-inspired syntax for HTT

Our initial example of a Hoare triple written as a Hoare type:

$$x := 5 : \{x = 1\}\, r{:}\, unit\, \{x > 0\}$$

In F*, each effect is annotated by its name such as *ST*.

The example becomes:

$$x := 5 : ST\ (r{:}\, unit)(requires\ \{x = 1\})\ (ensures\ \{x > 0\})$$

We will use this syntax and *QST* effect for Quantum Hoare types.

Swamy et al (2016). *Dependent Types and Multi-Monadic Effects in F**. POPL

Questions so far?

- Motivation
- Background
  - Floyd-Hoare Logic
  - Hoare Type Theory (HTT)

  } Classical
  - Quantum Computation
  - Quantum Hoare Logic

  } Quantum
- Quantum Hoare Type Theory (QHTT)
  - Primitive Commands
  - Teleportation
- Recap
- Future Directions

Quantum Computation

The four postulates of quantum mechanics:

1. States

   Unit vectors in a complex vector space, $\mathbb{C}^n$
   aka, Hilbert space, $\mathcal{H}$

2. Dynamics

   Unitary matrices acting on state space

3. Measurement

   Getting information out of a quantum system

4. Combining systems

   State spaces *A* and *B* can be combined by taking their tensor product, $A \otimes B$

Computational basis

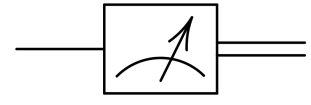
$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}; \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$\psi = \alpha|0\rangle + \beta|1\rangle$$

$$|\alpha|^2 + |\beta|^2 = 1$$

Superposition

$$X.\psi = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.\psi = \alpha|1\rangle + \beta|0\rangle$$



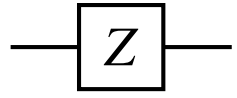
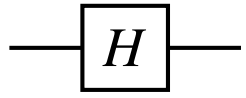
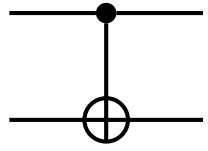
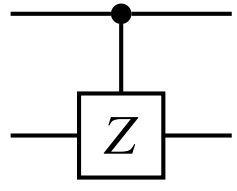
Primitive commands

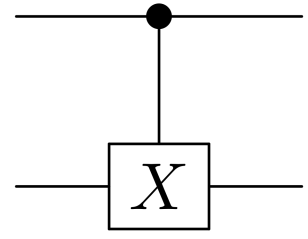
`init` *M*

`apply` *U* `to` *M*

`meas` *M*

Commonly used unitary matrices and their gate notation

Z	Pauli Z		$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
H	Hadamard		$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
CX	controlled-NOT		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
CZ	controlled-Z		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$



Quantum Hoare logic (QHL)

Closed subspaces of a state space can be thought of as logical propositions (Birkhoff and von Neumann [1936]).

Unruh's QHL defines interesting predicates based on this idea:

- $\top$ is logically equivalent to always true, complete space
- $P \wedge Q$ is the set intersection of the two subspaces
- $U \cdot P$ is equal to the subspace $\{U.\psi : \psi \in P\}$
- $X =_q \psi$ says that qubits *X* are in a specific state ψ
- $X \equiv_q Y$ denotes quantum equality between two variables
- $X \equiv_{cl} Y$ denotes classical equality between two variables

Dominique Unruh (2019). *Quantum Hoare Logic with Ghost Variables*. LICS

QHL and ghost variables

Unruh further introduces quantum-specific ghost variables.

e denotes a fresh entangled ghost variable

u denotes a fresh unentangled ghost variable

Then he defines some predicates using these ghosts:

separable(X), which is sugar for $X \equiv_q u$

class(X), which is sugar for $X \equiv_{cl} e$

Two axioms from Unruh's QHL with ghost variables:

UNITARY

$\{P\}$ **apply** U **to** x $\{(U$ **on** $x) \cdot P\}$

INITIALIZATION

$\{P\}$ **init** x $\{P[x \to e],\ x =_q |0\rangle\}$

Dominique Unruh (2019). *Quantum Hoare Logic with Ghost Variables*. LICS

Questions so far?

Motivation

Background

- Floyd-Hoare Logic
- Hoare Type Theory (HTT)

} Classical

- Quantum Computation
- Quantum Hoare Logic

} Quantum

Quantum Hoare Type Theory (QHTT)

- Primitive Commands
- Teleportation

Recap

Future Directions

Quantum Hoare Type Theory (QHTT)

Like classical effects, we can index the result of a quantum computation with its return type and pre- and postconditions to obtain a *Quantum Hoare type*.

Main differences from HTT are the modeling of quantum state and the primitive commands specific to quantum computation.

Compared to Unruh's QHL, our rules are simpler and presented in a type-theoretical setting.

Primitive Commands: initialization

We can initialize one qubit at a time in either of the two computational basis states, $|0\rangle$ or $|1\rangle$.

```
init M : (b: bit) → QST (q: qbit)
                        (requires {⊤})
                        (ensures  {q =_q |b⟩})
```

For example, the initialization command `x ← init 0` allocates a fresh qubit from the global pool of qubits, initializes it to $|0\rangle$ and returns a reference to the qubit and binds it to *x*.

Primitive Commands: measurement

Measurements are in the computation basis.

```
meas M : (q: qbit) → QST (x: bit)
                         {ψ: vector, e_x : qbit}
                         (requires {(q⊗ℋ[V\q]) =_q ψ})
                         (ensures  {class(q) ∧ (e_x ⊗ℋ[V\q]) =_q ψ})
```

On measurement, a qubit can be assumed to be discarded,
even though we can still refer to it in the logical specification.

The outcome distribution of the qubit is maintained by
replacing the measured qubit with an *entangled ghost variable*
in the *local state* under consideration.

Primitive Commands: unitary application

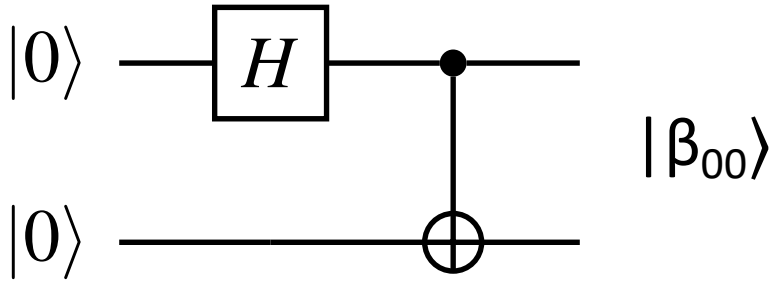
Unitary application is the same as Unruh's rule:

```
apply G to M : (g: unitary) →
               qs: (qbit ⊗ qbit) →
               QST unit
                   {P: prop}
                   (requires {P})
                   (ensures  {(g on qs)·P})
```

It helps to think of this rule in terms of its two variants, single-qubit and two-qubit unitary applications:

```
apply₁ G₁ to M₁ : unitary → qbit → QST unit
apply₂ G₂ to M₂ : unitary → (qbit ⊗ qbit) → QST unit
```

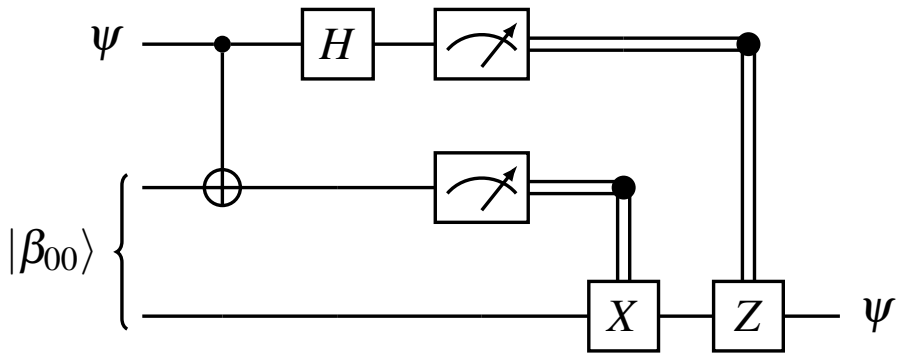
Bell state preparation



$$|\beta_{00}\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

```
bell00 : unit → QST (a, b): (qbit⊗qbit)
                    (requires {⊤})
                    (ensures  {(a,b) =q |β₀₀⟩})
bell00 = λx.do
           -- ⊤
           a ← init 0
           -- a =q |0⟩
           apply H to a
           -- (H on a)·(a =q |0⟩) ⇔ a =q |+⟩
           b ← init 0
           -- a =q |+⟩, b =q |0⟩
           apply CX to (a, b)
           -- (CX on (a, b))·(a =q |+⟩, b =q |0⟩)
           -- ⇔ (a, b) =q |β₀₀⟩
           return (a, b)
```

Quantum teleportation




```
teleport : q: qbit -> QST b: qbit
                              {ψ: vector}
                              (requires {q =q ψ})
                              (ensures  {class(q), b =q ψ})

teleport = λx.do
           (a, b) ← bell00 ()
           apply CX to (q, a)
           apply H to q
           x ← meas q
           y ← meas a
           apply CX to (y, b)
           apply CZ to (x, b)
           return b
```

Verifying teleportation (1)

```
teleport = λq.do
              -- q =_q ψ
              -- ⇔ q =_q α|0⟩+β|1⟩
              (a, b) ← bell00 ()
              -- q =_q ψ, (a, b) = |β00⟩
              -- ⇔ (q, a, b) =_q 1/√2(α|0⟩(|00⟩+|11⟩)+β|1⟩(|00⟩+|11⟩))
              apply CX to (q, a)
              -- (CX on (q, a))·(q =_q ψ, (a, b) = |β00⟩)
              -- ⇔ (q, a, b) =_q 1/√2(α|0⟩(|00⟩+|11⟩)+β|1⟩(|10⟩+|01⟩))
              apply H to q
              -- (H on q)·((q, a, b) =_q 1/√2(α|0⟩(|00⟩+|11⟩)+β|1⟩(|10⟩+|01⟩)))
              -- ⇔ (q, a, b) =_q 1/2(α(|0⟩+|1⟩)(|00⟩+|11⟩) + β(|0⟩-|1⟩)(|10⟩+|01⟩))
              -- ⇔ (q, a, b) =_q 1/2(|00⟩(α|0⟩+β|1⟩) + |01⟩(α|1⟩+β|0⟩)
              --                   + |10⟩(α|0⟩-β|1⟩) + |11⟩(α|1⟩-β|0⟩))
              x ← meas q
              -- class(q) ∧
              -- (e_x, a, b) =_q 1/2(|00⟩(α|0⟩+β|1⟩) + |01⟩(α|1⟩+β|0⟩)
              --                   + |10⟩(α|0⟩-β|1⟩) + |11⟩(α|1⟩-β|0⟩))
              y ← meas a
              -- class(q) ∧ class(a) ∧
              -- (e_x, e_y, b) =_q 1/2(|00⟩(α|0⟩+β|1⟩) + |01⟩(α|1⟩+β|0⟩)
              --                     + |10⟩(α|0⟩-β|1⟩) + |11⟩(α|1⟩-β|0⟩))
```

Verifying teleportation (2)

```
19  x ← meas q
20  -- class(q) ∧
21  -- (e_x, a, b) =_q 1/2(|00⟩(α|0⟩+β|1⟩) + |01⟩(α|1⟩+β|0⟩)
22  --                  + |10⟩(α|0⟩-β|1⟩) + |11⟩(α|1⟩-β|0⟩))
23  y ← meas a
24  -- class(q) ∧ class(a) ∧
25  -- (e_x, e_y, b) =_q 1/2(|00⟩(α|0⟩+β|1⟩) + |01⟩(α|1⟩+β|0⟩)
26  --                    + |10⟩(α|0⟩-β|1⟩) + |11⟩(α|1⟩-β|0⟩))
27  apply CX to (y, b)
28  -- class(q) ∧ class(a) ∧
29  -- (e_x, e_y, b) =_q 1/2(|00⟩(α|0⟩+β|1⟩) + |01⟩(α|0⟩+β|1⟩)
30  --                    + |10⟩(α|0⟩-β|1⟩) + |11⟩(α|0⟩-β|1⟩))
31  apply CZ to (x, b)
32  -- class(q) ∧ class(a) ∧
33  -- (e_x, e_y, b) =_q 1/2(|00⟩(α|0⟩+β|1⟩) + |01⟩(α|0⟩+β|1⟩)
34  --                    + |10⟩(α|0⟩+β|1⟩) + |11⟩(α|0⟩+β|1⟩))
35  -- ⇔ (e_x, e_y) =_q 1/2(|00⟩+|01⟩+|10⟩+|11⟩), b =_q α|0⟩+β|1⟩
36  return b
37  -- class(q) ∧ b =_q α|0⟩+β|1⟩
38  -- ⇔ class(q), b =_q ψ
```

Modular teleportation

```
teleport : q: qbit → QST b: qbit
                          {ψ: vector}
                          (requires {q =_q ψ})
                          (ensures  {class(q), b =_q ψ})
teleport = λq.do
              (a, b) ← bell00 ()
              (x, y) ← alice (q, a)
              b ← bob (x, y, b)
              return b
```

```
alice : (q, a): (qbit⊗qbit) →
        QST (x, y): (bit⊗bit)
            {e: qbit, e_x: qbit, e_y: qbit, α: complex, β: complex}
            (requires {q =_q α|0⟩+β|1⟩, (a, e) = |β_00⟩})
            (ensures  {class(q), class(a),
                       (e_x, e_y, e) =_q 1/2(|00⟩(α|0⟩+β|1⟩) + |01⟩(α|1⟩+β|0⟩)
                                          + |10⟩(α|0⟩-β|1⟩) + |11⟩(α|1⟩-β|0⟩))})
```

```
bob : (x, y, b): (bit⊗bit⊗qbit) →
      QST q: qbit
          {c: complex, d: complex}
          (requires {b =_q c|0⟩+d|1⟩})
          (ensures  {y =_c 0 ⇒ q =_q (c|0⟩+ (-1)^x d|1⟩) ∧
                     y =_c 1 ⇒ q =_q (c|1⟩+ (-1)^x d|0⟩)})
```

Recap

Classical	Quantum
Hoare logic(s)	Quantum Hoare logic(s)
HTT	★ **QHTT** ★

A unified system for programming, specifying, and reasoning about quantum computation.

- A dependent type theory in which types include specifications (pre- and postconditions) for quantum programs.
- Successful type checking implies correctness of the program to those specifications.
- A rich and expressive predicate language based on Unruh's ghost variables for semantic properties.
- Proofs of teleportation and Deutsch algorithm.

arXiv:2012.02154

Future directions

Mechanization in an expressive dependent type theory: Coq or F*. Extraction into a lower-level quantum language.

Support more constructs in the language such as iteration and/or recursion, composing pure unitary operators.

Integrate:

- Linear types proposed by Peter Selinger's group.
- Gottesman types based on Heisenberg representation.

Other richer types that QHTT may enable: probabilistic? Those based on quantum resource theories?

Scalability of verification parameterized over *n* qubits (families of circuits).

Can we avoid the trap of simulation to reason about quantum state?

Fu, Kishida, Selinger (2020). *Linear Dependent Type Theory for Quantum Programming Languages: Extended Abstract*. LICS

Rand, Sundaram, **Singhal**, Lackey (2020). *Gottesman Types for Quantum Programs.* QPL

Rand, Sundaram, **Singhal**, Lackey (2019). *A Type System for Quantum Resources.* Draft

Questions?

Acknowledgments

Advisors:

John Reppy and Robert Rand

Committee:

Bill Fefferman *+ advisors*

Collaborators:

Aarthi Sundaram and Brad Lackey
Microsoft Quantum

Bettina Heim and Sarah Marshall
Q# team

Fellow PhD students who hang out at our Virtual Teatime for their feedback.

Standing on the shoulders of giants

Nielsen and Chuang (2010). *Quantum Computation and Quantum Information*.

D'hondt and Panangaden (2006). *Quantum Weakest Preconditions*. Math. Struct. Comput. Sci.

Ying (2012). *Floyd–Hoare Logic for Quantum Programs*. ACM Trans. Program. Lang. Syst.

Altenkirch and Green (2009). *The Quantum IO Monad.*

Green *et al* (2013). *Quipper: A Scalable Quantum Programming Language*. PLDI

Paykin, Rand, Zdancewic (2017). *QWIRE: a core language for quantum circuits*. POPL

Svore *et al* (2018). *Q#: Enabling Scalable Quantum Computing and Development with a High-level DSL*. RWDLS