An Invitation to the intersection of Quantum Computing & Programming Languages

Tutorial at POPL 2021

Xiaodi Wu
QuIGS & UMD
About this Tutorial:

Goal: An Invitation due to limited time
Cover Some Basic Quantum Computing & PL
Provide References / Pointers for further study
About this Tutorial:

Goal: An Invitation due to limited time
Cover Some Basic Quantum Computing & PL
Provide References / Pointers for further study

Format: Tutorial divided into 3 parts:

(1) Introduction to Quantum Computing and Potential Roles of Programming Languages (25 min + 5 Q & A)

(2) A Mini-Course of Quantum Hoare Logic on Quantum While Language (30 min + 5 Q & A)

(3) Discussion on existing and potential Programming Language research opportunities (20 min + 5 Q & A)
About the Speaker:

Wu: assistant professor at UMD working on quantum computing from CS perspective in general.
About the Speaker:

**Wu**: assistant professor at UMD working on quantum computing from CS perspective in general.

Teaching in Q. Computing

Past Courses
This is a collection of courses that I have taught in the past for your references. Please be cautious as these may be outdated.

**University of Maryland, College Park (2017 - present)**
- Complexity Theory (CMSC 652): graduate-level theory core course
  - Fall 2017
- Introduction to Quantum Computing (CMSC/PHYS 457): undergraduate-level introduction to quantum computing
  - Spring 2018, Spring 2020, Spring 2021
- Introduction to Quantum Information Processing (CMSC 657): graduate-level introduction to quantum computing
  - Fall 2018, Fall 2019

**University of Oregon (2015 - 2017)**
- Intermediate Data Structure (CIS 313): undergraduate CS major theory course.
- Introduction to Quantum Information Processing (CIS 410/510): senior undergraduate / graduate level

Mini-Library on Quantum Information and Computation
This page is meant to be a collection of representative and available references for the study and research of the theoretical and practical aspects of quantum computing. As possible and will be regularly maintained. Send me an email if you have any good suggestion.

**Expository Writings and Lecture Notes by myself**
- Tutorial at POPL 2021: An Invitation to the Intersection of Quantum Computing and Programming Languages
  - (Part I) A brief introduction to quantum computing and potential roles of programming languages
  - (Part II) A mini-course on the verification of quantum while languages based on quantum Hoare logic
  - (Part III) A discussion of existing and possible research directions at the intersection of quantum computing and programming languages
- **Lecture Notes (Fall 2019)**
  - Quantum Approximate Optimization Algorithm (QAOA)
  - Introduction to Quantum Hoare Logic (slides)
- **Lecture Notes (Fall 2018)**
  - Quantum Interactive Proofs and QIP=PSPACE
  - Quantum Algorithms for Linear Equation Systems
  - Quantum Algorithms for Semidefinite Programs

**Scientific Reports from Relevant Research Communities**
- More Reports at QuantumGov.

**General Study: Courses, Lecture Notes & Textbooks**
- Self-learning Materials for Beginners
  - Why now is the right time to study quantum computing by A. Harrow.
  - S. Aaronson: @UWaterloo Quantum Computing since Democritus
  - M. Nielsen’s Quantum Computing for the determined: 22 short (5-15 mins) youtube videos, each explaining a bit of quantum computing
  - 12th Canadian Summer School on Quantum Information Lecture Notes Youtube
About the Speaker:

**Wu**: assistant professor at UMD working on quantum computing from CS perspective in general.

---

Teaching in Q. Computing

**Past Courses**
This is a collection of courses that I have taught in the past for your references. Please be cautious as these are potential teaching materials.

**University of Maryland, College Park (2017 - present)**
- Complexity Theory (CMSC 652): graduate-level theory core course  
  - Fall 2017
- Introduction to Quantum Computing (CMSC/PHYS 457): undergraduate-level introduction to quantum  
  - Spring 2018, Spring 2020, Spring 2021
- Introduction to Quantum Information Processing (CMSC 657): graduate-level introduction to quantum  
  - Fall 2018, Fall 2019

**University of Oregon (2015 - 2017)**
- Intermediate Data Structure (CIS 313): undergraduate CS major theory course.  
- Introduction to Quantum Information Processing (CIS 410/510): senior undergraduate / graduate  

---

**Disclaimer:** perspectives and claims are potentially limited or biased by personal knowledge.
Outline

(1) Introduction to Quantum Computing and Potential Roles of Programming Languages (25 min + 5 Q & A)

(2) A Mini-Course of Quantum Hoare Logic on Quantum While Language (30 min + 5 Q & A)

(3) Discussion on existing and potential Programming Language research opportunities (20 min + 5 Q & A)

Reference: tutorial slides and some references are available at https://www.cs.umd.edu/~xwu/mini_lib.html
What Quantum Computing is **NOT**

It Isn’t Just Today’s Computers But Smaller or Faster

Credit: Scott Aaronson
What Quantum Computing is NOT

It Isn’t Just Today’s Computers But Smaller or Faster

It Isn’t A Magic Bullet That Solves All Problems Instantly

Credit: Scott Aaronson
What Quantum Computing is **NOT**

It Isn’t Just Today’s Computers But Smaller or Faster

It Isn’t A Magic Bullet That Solves All Problems Instantly

It Isn’t A Simple Matter of Trying All Possible Answers In Parallel
What Quantum Computing is NOT

It Isn’t Just Today’s Computers But Smaller or Faster

It Isn’t A Magic Bullet That Solves All Problems Instantly

It Isn’t A Simple Matter of Trying All Possible Answers In Parallel

But Nor Is It Science Fiction

Credit: Scott Aaronson
Experimental Comparison of Two Quantum Computing Architectures

N. M. Linke, D. Maslov, M. Roetteler, S. Debnath, C. Figgatt, K. A. Landsman, K. Wright, and C. Monroe

1 Joint Quantum Institute and Department of Physics, University of Maryland, College Park, MD 20742
2 National Science Foundation, Arlington, VA 22230
3 Joint Center for Quantum Information and Computer Science, University of Maryland, College Park, MD 20742
4 Microsoft Research, Redmond, WA 98052
5 IonQ, Inc., College Park, MD 20742

We run a selection of algorithms on two state-of-the-art 5-qubit quantum computers that are based on different technology platforms. One is a publicly accessible superconducting transmon device [1] with limited connectivity, and the other is a fully connected trapped-ion system [2]. Even though the two systems have different native quantum interactions, both can be programmed in a way that is blind to the underlying hardware, thus allowing the first comparison of identical quantum algorithms between different physical systems. We show that quantum algorithms and circuits that employ more connectivity clearly benefit from a better connected system of qubits. While the quantum systems here are not yet large enough to eclipse classical computers, this experiment exposes critical factors of scaling quantum computers, such as qubit connectivity and gate expressivity. In addition, the results suggest that co-designing particular quantum applications with the hardware itself will be paramount in successfully using quantum computers in the future.

Inspired by the vast computing power a universal quantum computer could offer, several candidate systems are being explored. They have allowed experimental demonstrations of quantum gates, operations, and algorithms of ever increasing sophistication. Recently, two architectures, superconducting transmon qubits [3–7] and trapped ions [2, 8], have reached a new level of maturity. They have become fully programmable multi-qubit machines that provide the user with the flexibility to implement arbitrary quantum circuits from a high-level interface. This makes it possible for the first time to test quantum computers irrespective of their particular physical implementation.

While the quantum computers considered here are still small scale and their capabilities do not currently reach beyond small demonstration algorithms, this line of inquiry can still provide useful insights into the performance of existing systems and the role of architecture in quantum computer design. These will be crucial for the realization of more advanced future incarnations of the present technologies.

The standard abstract model of quantum computation assumes that interactions between arbitrary pairs of qubits are available. However, physical architectures will in general have certain constraints on qubit connectivity, such as nearest-neighbor couplings only. These restrictions do not in principle limit the ability to perform arbitrary computations, since SWAP operations may be used to effect gates between arbitrary qubits using the connections available. For a general circuit, reducing a fully-connected system to the more sparse star-shaped or linear nearest-neighbor connectivity requires an increase in the number of gates of $O(n)$, where $n$ is the number of qubits [9]. How much overhead is incurred in practice depends on the connections used in a particular circuit and how efficiently they can be matched to the physical qubit-to-qubit interaction graph.

In this article, we make use of the public access recently granted by IBM to a 5-qubit superconducting device (illustrated in fig. 1(a)) via their "Quantum Experience" cloud service [1]. This allows us to repeat algorithms that we perform in our own ion trap experiment on an independent quantum computer of identical size and comparable capability but with a different physical implementation at its core.
Surge of Interests from Gov, Academia, & Industry

Gov: US (NSF, DOE + National Labs, DoD, NIST), China, Europe, ....
Industry: Google, IBM, Microsoft, Amazon, Alibaba, Tecent, Baidu, ....
Academia: #faculty in quantum computing ++

US GOV Policy & Efforts: (quantum|gov)
Quantum Computing: still too early to call!

The 2019 Gartner Hype Cycle for Artificial Intelligence, with quantum computing highlighted in yellow. Credit: Gartner
Scientific Reports from relevant research communities

Reference: links are available at https://www.cs.umd.edu/~xwu/mini_lib.html
What is Quantum Computing?

An Operation $O \rightarrow$ A Physical Evolution $P$

Computation:

Evolution of the Machine: $P_1, P_2, P_3, \ldots$

The accumulative evolution carries some computation!
What is Quantum Computing?

A Mechanical Computer

An Operation $O \rightarrow$ A Physical Evolution $P$

Computation:

Evolution of the Machine: $P_1, P_2, P_3, \ldots$

The accumulative evolution carries some computation!

A Quantum Computer

An Operation $O \rightarrow$ A Quantum Physical Evolution $Q$

Computation:

Evolution of the Machine: $Q_1, Q_2, Q_3, \ldots$

The accumulative evolution carries some computation!
What is Quantum Computing good at?

Assume a unit operation requires a unit time on respective machines.
What is Quantum Computing good at?

Assume a unit operation requires a unit time on respective machines.

**Classical Computing (T)**

Computation can be carried out by $P_1, P_2, \ldots, P_T$

**Quantum Computing (T)**

Computation can be carried out by $Q_1, Q_2, \ldots, Q_T$
What is Quantum Computing good at?

Assume a unit operation requires a unit time on respective machines.

Classical Computing (T)  
Computation can be carried out by  \( P_1, P_2, \ldots, P_T \)

Quantum Computing (T)  
Computation can be carried out by \( Q_1, Q_2, \ldots, Q_T \)
What is Quantum Computing good at?

Assume a *unit* operation requires a *unit* time on respective machines.

Classical Computing (T)

\[
P_1, P_2, \ldots, P_T
\]

Quantum Computing (T)

\[
Q_1, Q_2, \ldots, Q_T
\]

Quantum Simulation

Nature isn’t classical, and if you want to make a simulation of Nature, you’d better make it quantum mechanical, and by golly it’s a wonderful problem, because it doesn’t look so easy.

Richard Feynman, 1982

Simulating quantum systems is critical for the scientific discovery for natural science include physics, chemistry, biology, material science, and so on. And nowadays, it consumes a significant amount of our HPC computing power.
What is Quantum Computing good at?

- Linear systems
- Graph problems (minimum spanning tree, connectivity, shortest path, triangle finding, etc.)
- Formula evaluation
- Decomposing groups (abelian, dihedral, etc.)
- ......
What is Quantum Computing **good at**?

- Linear systems
- Graph problems (minimum spanning tree, connectivity, shortest path, triangle finding, etc.)
- Formula evaluation
- Decomposing groups (abelian, dihedral, etc.)
- ......

It was a **good surprise** that quantum physics can help solve classical problems that look nothing like quantum physics at all!

Any high-level intuition why?
What is Quantum Computing good at?

- Linear systems
- Graph problems (minimum spanning tree, connectivity, shortest path, triangle finding, etc.)
- Formula evaluation
- Decomposing groups (abelian, dihedral, etc.)
- ......

It was a good surprise that quantum physics can help solve classical problems that look nothing like quantum physics at all!

Any high-level intuition why?

Quantum Duality:

Particle + Wave
What is Quantum Computing good at?

- Linear systems
- Graph problems (minimum spanning tree, connectivity, shortest path, triangle finding, etc.)
- Formula evaluation
- Decomposing groups (abelian, dihedral, etc.)
- ......

It was a good surprise that quantum physics can help solve classical problems that look nothing like quantum physics at all!

Any high-level intuition why?

Quantum Duality: 

Particle + Wave

Interference of Waves:
Make Interference Work:

Waves of equal amplitude and opposite phase cancel out

Recording and inverting noise leaves you with your desired signal

Active Noise-Canceling!
Make Interference Work:

Waves of equal amplitude and opposite phase cancel out

Recording and inverting noise leaves you with your desired signal

Make Interference Work for Computation:

Quantum Computation: Get computational paths leading to incorrect answers to interfere destructively and cancel each other out.

Active Noise-Canceling!
Make Interference Work:

Waves of equal amplitude and opposite phase cancel out

\[ \begin{align*}
&> & = \\
\end{align*} \]

Recording and inverting noise leaves you with your desired signal

\[ \begin{align*}
+ & = \\
\end{align*} \]

Active Noise-Canceling!

Make Interference Work for Computation:

Quantum Computation: Get computational paths leading to *incorrect* answers to *interfere destructively* and cancel each other out.

Quantum vs Randomized:

Randomized Computation: Probabilities of computational paths leading to *incorrect* answers only *add up*, never cancel out.
A Rough Timeline of Quantum Applications

**NOW:** Quantum Supremacy

Computational tasks, *not necessarily useful*, which is feasible to implement with current q. machines, but hard to simulate by classical computation.

A *Milestone* Toward Useful Quantum Computation

Google: Random Circuit Sampling  
USTC: Boson Sampling
**A Rough Timeline of Quantum Applications**

**NOW: Quantum Supremacy**

Computational tasks, *not necessarily useful*, which is feasible to implement with current q. machines, but hard to simulate by classical computation.

A **Milestone** Toward Useful Quantum Computation

- Google: Random Circuit Sampling
- USTC: Boson Sampling

**NISQ: Noise Intermediate-Scale Quantum machines ~ near future**

50 ~ 200, ~ 1000 controllable but noisy qubits, no fault-tolerant qubits

Or special-purpose quantum machines, like analog quantum simulators

**Quantum Simulation**

**Variational Q. Methods**
### NOW: Quantum Supremacy

Computational tasks, *not necessarily useful*, which is feasible to implement w/ current q. machines, but hard to simulate by classical computation.

**A Milestone** Toward Useful Quantum Computation

- **Google: Random Circuit Sampling**
- **USTC: Boson Sampling**

### NISQ: Noise Intermediate-Scale Quantum machines ~ near future

50 ~ 200, ~ 1000 controllable but noisy qubits, no fault-tolerant qubits

Or special-purpose quantum machines, like analog quantum simulators

Other quantum applications not in the computation domain: *quantum sensing, quantum communication*
A Rough Timeline of Quantum Applications

NOW: Quantum Supremacy

Computational tasks, *not necessarily useful*, which is feasible to implement w/ current q. machines, but hard to simulate by classical computation.

A Milestone Toward Useful Quantum Computation

Google: Random Circuit Sampling  USTC: Boson Sampling

NISQ: Noise Intermediate-Scale Quantum machines ~ near future

50 ~ 200, ~ 1000 controllable but noisy qubits, no fault-tolerant qubits

Or special-purpose quantum machines, like analog quantum simulators

Other quantum applications not in the computation domain: *quantum sensing, quantum communication*

Fault-Tolerant QC: ~ unknown future, a lot of uncertainty here

- Linear systems
- Graph problems (minimum spanning tree, connectivity, shortest path, triangle finding, etc.)
- Formula evaluation
- Decomposing groups (abelian, dihedral, etc.)
- ......
The Role of Programming Languages

Like the role of PL played for any other computing models, many similar first-principle questions can be asked in the context of quantum computing as well!
The Role of Programming Languages

Like the role of PL played for any other computing models, many similar first-principle questions can be asked in the context of quantum computing as well!

But of course, quantum computing model demonstrates some fundamental differences and unique needs, which requires new techniques to deal with.
The Role of Programming Languages

Like the role of PL played for any other computing models, many similar first-principle questions can be asked in the context of quantum computing as well!

But of course, quantum computing model demonstrates some fundamental differences and unique needs, which requires new techniques to deal with.

Disclaimer: perspectives and claims are potentially limited or biased by personal knowledge.

How to Program Q. Applications, Debug, and Verify Correctness?
How to Develop Software for Q. Computing, e.g., compiler, system?
How to Design and Implement Architecture for Quantum Computing?
How to Handle Quantum Security Issues in Design&Implementation?
How to Scale and Automate the Design of Quantum Hardware?
How to Program Q. Applications, Debug, and Verify Correctness?

The natural question with MOST investigation, but still a huge gap!
How to Program Q. Applications, Debug, and Verify Correctness?

The natural question with MOST investigation, but still a huge gap!

THEORY: quantum lambda-calculus, functional quantum PL, q. while language semantics in various pictures, q. Hoare logic and verification, ...
How to Program Q. Applications, Debug, and Verify Correctness?

The natural question with MOST investigation, but still a huge gap!

THEORY: quantum lambda-calculus, functional quantum PL, q. while language semantics in various pictures, q. Hoare logic and verification, ...

LANGUAGES: Quipper (embedded in Haskel), Scaffold (based on LLVM), Q# (based on F#, MSR), QWIRE/SQIR (embedded in Coq), SILQ, ...

python-lib Qiskit (IBM), Cirq (Google), Forrest (Rigetti), Braket (AWS), <- industry
How to Program Q. Applications, Debug, and Verify Correctness?

The natural question with MOST investigation, but still a huge gap!

THEORY: quantum lambda-calculus, functional quantum PL, q. while language semantics in various pictures, q. Hoare logic and verification, ...

LANGUAGES: Quipper (embedded in Haskel), Scaffold (based on LLVM), Q# (based on F#, MSR), QWIRE/SQIR (embedded in Coq), SILQ, ...

Gap: (1) too-low-level-abstraction: very hard to write complex programs
How to Program Q. Applications, Debug, and Verify Correctness?

The natural question with MOST investigation, but still a huge gap!

THEORY: quantum lambda-calculus, functional quantum PL, q. while language semantics in various pictures, q. Hoare logic and verification, ...

LANGUAGES: Quipper (embedded in Haskell), Scaffold (based on LLVM), Q# (based on F#, MSR), QWIRE/SQIR (embedded in Coq), SILQ, ... <- academia

python-lib Qiskit (IBM), Cirq (Google), Forrest (Rigetti), Braket (AWS), <- industry

Gap: (1) too-low-level-abstraction: very hard to write complex programs
(2) lack of scalable verification: very hard to write correct programs

Verifying the circuit by observation .... not scalable ...
How to Program Q. Applications, Debug, and Verify Correctness?

The natural question with MOST investigation, but still a huge gap!

THEORY: quantum lambda-calculus, functional quantum PL, q. while language semantics in various pictures, q. Hoare logic and verification, ...

LANGUAGES: Quipper (embedded in Haskell), Scaffold (based on LLVM), Q# (based on F#, MSR), QWIRE/SQIR (embedded in Coq), SILQ, ...

python-lib Qiskit (IBM), Cirq (Google), Forrest (Rigetti), Braket (AWS), <- industry

Gap: (1) too-low-level-abstraction: very hard to write complex programs
(2) lack of scalable verification: very hard to write correct programs
(3) lack of many desirable analyses, automation, & optimization: a lot of burdens on the programmers

Verifying the circuit by observation .... not scalable ...
How to Develop **Software** for Q. Computing, e.g., compiler, system?

Large Design Space for System Software for Quantum Computers.

F. Chong, D. Franklin, M. Martonosi, Nature 549, 180
How to Develop **Software** for Q. Computing, e.g., **compiler, system**?

Large Design Space for System Software for Quantum Computers.

**High-Assurance** Software Tool-chain both **desirable** and **challenging**.

- standard software assurance techniques, e.g., black-box / unit test, expensive in q.
- quantum mechanics prohibits certain testing, e.g., assertions

F. Chong, D. Franklin, M. Martonosi, Nature 549, 180
How to Develop Software for Q. Computing, e.g., compiler, system?

Large Design Space for System Software for Quantum Computers. High-Assurance Software Tool-chain both desirable and challenging.

- standard software assurance techniques, e.g., black-box / unit test, expensive in q.
- quantum mechanics prohibits certain testing, e.g., assertions

A possible solution: fully certified software, e.g., VOQC (POPL 2021)
How to Design and Implement Architecture for Quantum Computing?

Mapping, Error Mitigation, ...

approximate computing

ibmq_toronto
A lot of controlling operations need to be located close to quantum chips for small responsive time. 

*ISA + Fast Compilation*
How to Handle Quantum Security Issues in Design and Implementation?

Verification of Quantum Cryptography:

*Relational Quantum Hoare Logic* (Unruh; Barthe et al.)
Verification of Quantum Cryptography:
   Relational Quantum Hoare Logic (Unruh; Barthe et al.)

Quantum Cryptanalysis:
   Resource estimation of Complex Quantum Attack Programs
How to Handle Quantum Security Issues in Design and Implementation?

Verification of Quantum Cryptography:
Relational Quantum Hoare Logic (Unruh; Barthe et al.)

Quantum Cryptanalysis:
Resource estimation of Complex Quantum Attack Programs

Post-Quantum Cryptography:
Classical Cryptographic Systems Resilient to Quantum Attacks

For Classical Cryptographic Systems

(1) Identify their post-quantum security
(2) automate the procedure to upgrade its post-quantum security
(3) formal post-quantum security proofs

Formally generated security analysis will provide not only efficient and high assurance proofs that can replace the tedious and error-prone analysis for experts, but also independently verifiable proofs that can be used by security practitioners without much quantum knowledge.
How to Scale and Automate the Design of Quantum Hardware?
How to Scale and Automate the Design of Quantum Hardware?

Superconducting Credit: arXiv:1704.06208

How to Scale and Automate the Design of Quantum Hardware?

Superconducting Credit: arXiv:1704.06208

How to Scale and Automate the Design of Quantum Hardware?

Demonstrate A Lot of Design Choices
Hard to Scale without Automatic Tools
How to Scale and Automate the Design of Quantum Hardware?

Demonstrate **A Lot of Design Choices**
**Hard to Scale without Automatic Tools**

A Golden Age of Hardware Description Languages:
Applying Programming Language Techniques to Improve Design Productivity

**Lenny Truong**
Stanford University, USA
lenny@cs.stanford.edu

**Pat Hanrahan**
Stanford University, USA
hanrahan@cs.stanford.edu

SNAPL 2019
How to Scale and Automate the Design of Quantum Hardware?

Demonstrate A Lot of Design Choices
Hard to Scale without Automatic Tools

A Golden Age of Hardware Description Languages: Applying Programming Language Techniques to Improve Design Productivity

Lenny Truong
Stanford University, USA
lenny@cs.stanford.edu

Pat Hanrahan
Stanford University, USA
hanrahan@cs.stanford.edu

Appplies to Quantum Hardware too!
Summary

Quantum PLs: some

Software Tool-chain: a little

Architecture: a little

Security: a little

Hardware Design: almost none
Summary

Quantum PLs: some
Software Tool-chain: a little
Architecture: a little
Security: a little
Hardware Design: almost none

Satisfactory
Summary

Quantum PLs: some

Software Tool-chain: a little

Architecture: a little

Security: a little

Hardware Design: almost none

More questions could be asked!
Summary

Quantum PLs: some
Software Tool-chain: a little
Architecture: a little
Security: a little
Hardware Design: almost none

More questions could be asked!

More details will come back in Part III of the tutorial.
Further Readings: Thank You! Q & A

Reference: links are available at https://www.cs.umd.edu/~xwu/mini_lib.html
Outline

1. Introduction to Quantum Computing and Potential Roles of Programming Languages (25 min + 5 Q & A)

2. A Mini-Course of Quantum Hoare Logic on Quantum While Language (30 min + 5 Q & A)

3. Discussion on existing and potential Programming Language research opportunities (20 min + 5 Q & A)

Reference: tutorial slides and some references are available at https://www.cs.umd.edu/~xwu/mini_lib.html
What is Quantum Computing?

An Operation $O \rightarrow$ A Quantum Physical Evolution $Q$

Computation:
Evolution of the Machine: $Q_1, Q_2, Q_3, \ldots$

The accumulative evolution carries some computation!
What is Quantum Computing?

An Operation $O \rightarrow$ A Quantum Physical Evolution $Q$

Computation:

Evolution of the Machine: $Q_1, Q_2, Q_3, \ldots$

The accumulative evolution carries some computation!

Consider quantum machines of finite-dimension. Hilbert space $\rightarrow$ Euclidean space
What is Quantum Computing?

A Quantum Computer

An **Operation** $O \rightarrow A \ **Quantum** \ Physical \ Evolution \ Q$

**Computation:**

Evolution of the Machine: $Q_1, Q_2, Q_3, \ldots$

The accumulative evolution carries some computation!

Consider quantum machines of **finite-dimension**. Hilbert space $\rightarrow$ Euclidean space

The **Math Model** of Quantum Machines comes from the math model of $Q_i$s. (semantics)
What is Quantum Computing?

A Quantum Computer

An Operation $O \rightarrow$ A Quantum Physical Evolution $Q$

Computation:

Evolution of the Machine: $Q_1, Q_2, Q_3, \ldots$

The accumulative evolution carries some computation!

Consider quantum machines of finite-dimension. Hilbert space $\rightarrow$ Euclidean space

The Math Model of Quantum Machines comes from the math model of $Q_i$s. (semantics)

Four Postulates for Quantum Mechanics:

State Space postulate

Evolution postulate — No-Cloning theorem

Composite System postulate

Measurement postulate
State Space postulate: (pure) quantum state represented by unit complex vectors
**State Space postulate:** (pure) quantum state represented by unit complex vectors
State Space postulate: (pure) quantum state represented by unit complex vectors

A quantum bit (qubit) refers to a quantum system of dimension 2
State Space postulate: (pure) quantum state represented by unit complex vectors

A quantum bit (qubit) refers to a quantum system of dimension 2

Classical 0 and 1: \( |0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \) classical bits are special cases of quantum.
**State Space postulate:** (pure) quantum state represented by unit complex vectors

A *quantum bit* (**qubit**) refers to a quantum system of dimension 2.

Classical 0 and 1: $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$

Classical bits are special cases of quantum.
State Space postulate: (pure) quantum state represented by unit complex vectors

A quantum bit (qubit) refers to a quantum system of dimension 2

classical 0 and 1:

\[
|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}
\]

classical bits are special cases of quantum.

A general qubit:

\[
|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \quad \text{with} \quad |\alpha|^2 + |\beta|^2 = 1.
\]

\(\alpha, \beta\) are general complex numbers. Constraint due to Born’s probability amplitude interpretation.
**State Space postulate:** (pure) quantum state represented by unit complex vectors

A quantum bit (qubit) refers to a quantum system of dimension 2 classical 0 and 1: \[ |0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \]

A general qubit:

\[ |\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \quad \text{with} \quad |\alpha|^2 + |\beta|^2 = 1. \]

**Example:**

\[ |+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \]

\[ |--\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} \]

\( \alpha, \beta \) are general complex numbers. Constraint due to Born’s **probability amplitude** interpretation.
State Space postulate: (pure) quantum state represented by unit complex vectors

A quantum bit (qubit) refers to a quantum system of dimension 2 classical 0 and 1:
\[
|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}
\]
classical bits are special cases of quantum.

A general qubit:
\[
|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}
\]
with \(|\alpha|^2 + |\beta|^2 = 1\).

Example:
\[
|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix}
\]

Evolution postulate: evolution of quantum systems is unitary

Unitary evolution is a simple consequence of being linear and preserving \(\ell_2\) norm
State Space postulate: (pure) quantum state represented by unit complex vectors

A quantum bit (qubit) refers to a quantum system of dimension 2 classical 0 and 1: \[ |0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \] classical bits are special cases of quantum.

A general qubit:

\[ |\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \] with \( |\alpha|^2 + |\beta|^2 = 1 \).

Example:

\[ |+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} \]

Evolution postulate: evolution of quantum systems is unitary

Unitary evolution is a simple consequence of being linear and preserving \( \ell_2 \) norm

Precisely, \[ |\psi\rangle \mapsto U|\psi\rangle \] since \( U|\psi\rangle \) is also a quantum state, so that

\[ \langle \psi | U^\dagger U |\psi\rangle = 1, \forall |\psi\rangle \implies U^\dagger U = I \] unitary (reversible)
State Space postulate: (pure) quantum state represented by unit complex vectors

A quantum bit (qubit) refers to a quantum system of dimension 2 classical 0 and 1: \(|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \ |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}\) classical bits are special cases of quantum.

A general qubit:

\[ |\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \text{ with } |\alpha|^2 + |\beta|^2 = 1. \]

Example: 

\[ +\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad -\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -1 \end{pmatrix} \]

Evolution postulate: evolution of quantum systems is unitary

Unitary evolution is a simple consequence of being linear and preserving \(\ell_2\) norm

Precisely, \( |\psi\rangle \mapsto U|\psi\rangle \) since \( U|\psi\rangle \) is also a quantum state, so that

\[ \langle\psi| U^\dagger U |\psi\rangle = 1, \forall |\psi\rangle \implies U^\dagger U = I \quad \text{unitary (reversible)} \]

Example: 
\[ H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad H|0\rangle = |+\rangle, H|1\rangle = |-\rangle \]
**Composite System postulate:** joint system \((A,B)\) in the tensor-product of \(A\) and \(B\).

The representation of two qubits lies in \(\mathbb{C}^2 \otimes \mathbb{C}^2\) (dim-4), where \(\mathbb{C}^2\) (dim-2) is for a qubit.
**Composite System postulate:** joint system \((A,B)\) in the *tensor-product* of \(A\) and \(B\)

The representation of two qubits lies in \(\mathbb{C}^2 \otimes \mathbb{C}^2\) (dim-4), where \(\mathbb{C}^2\) (dim-2) is for a qubit.

So \(|00\rangle = |0\rangle \otimes |0\rangle\)

\[
\begin{pmatrix}
1 \\
0
\end{pmatrix} \otimes \begin{pmatrix}
1 \\
0
\end{pmatrix} =
\begin{pmatrix}
1 \\
0 \\
0 \\
0
\end{pmatrix}
\]

\[
|01\rangle = \begin{pmatrix}
0 \\
1 \\
0 \\
0
\end{pmatrix}
\]

\[
|10\rangle = \begin{pmatrix}
0 \\
0 \\
1 \\
0
\end{pmatrix}
\]

\[
|11\rangle = \begin{pmatrix}
0 \\
0 \\
0 \\
1
\end{pmatrix}
\]
Composite System postulate: joint system (A,B) in the tensor-product of A and B.

The representation of two qubits lies in $\mathbb{C}^2 \otimes \mathbb{C}^2$ (dim-4), where $\mathbb{C}^2$ (dim-2) is for a qubit.

So $|00\rangle = |0\rangle \otimes |0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$

$|01\rangle = |0\rangle \otimes |1\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$

$|10\rangle = |1\rangle \otimes |0\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$

$|11\rangle = |1\rangle \otimes |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$

A $n$-qubit system requires $2^n$ dimensional space. Exponential cost in classical simulation!
**Composite System postulate:** joint system \((A,B)\) in the tensor-product of \(A\) and \(B\)

The representation of two qubits lies in \(\mathbb{C}^2 \otimes \mathbb{C}^2\) (dim-4), where \(\mathbb{C}^2\) (dim-2) is for a qubit.

So \(|00\rangle = |0\rangle \otimes |0\rangle\):

\[
\begin{pmatrix}
1 \\
0
\end{pmatrix} \otimes 
\begin{pmatrix}
1 \\
0
\end{pmatrix} = 
\begin{pmatrix}
1 \\
0 \\
0 \\
0
\end{pmatrix}
\]

\(|01\rangle, |10\rangle, |11\rangle\):

\[
\begin{pmatrix}
0 \\
1 \\
0 \\
0
\end{pmatrix}
\]

A \(n\)-qubit system requires \(2^n\) dimensional space. Exponential cost in classical simulation!

**Examples of Common Quantum Gates**

- **Pauli gates:** Single-qubit Gate
  
  \(X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \ Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \ Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\)

- **Hadamard gate:**

  \(H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}\)

- **Rotation about \(x\)-axis of the Bloch sphere:**

  \(R_x(\theta) = \begin{pmatrix}
\cos\frac{\theta}{2} & -i \sin\frac{\theta}{2} \\
-i \sin\frac{\theta}{2} & \cos\frac{\theta}{2}
\end{pmatrix}\)

- **The controlled-NOT (CNOT) gate:** Two-qubit Gate

  \(CNOT = \begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0
\end{pmatrix}\)
Composite System postulate: joint system \((A,B)\) in the tensor-product of \(A\) and \(B\).

The representation of two qubits lies in \(\mathbb{C}^2 \otimes \mathbb{C}^2\) (dim-4), where \(\mathbb{C}^2\) (dim-2) is for a qubit.

So \(|00\rangle = |0\rangle \otimes |0\rangle\)  
\[
\begin{pmatrix}1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix}1 \\ 0 \end{pmatrix} = \begin{pmatrix}1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \]

\(|01\rangle\), \(|10\rangle\), \(|11\rangle\)

\[
\begin{pmatrix}1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \]

\[
\begin{pmatrix}0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \]

\[
\begin{pmatrix}0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \]

\[
\begin{pmatrix}0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \]

A \(n\)-qubit system requires \(2^n\) dimensional space. Exponential cost in classical simulation!

Examples of Common Quantum Gates

- Pauli gates: Single-qubit Gate  
  
  \[X = \begin{pmatrix}0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix}0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix}1 & 0 \\ 0 & -1 \end{pmatrix}\]

- Hadarmard gate:
  \[H = \frac{1}{\sqrt{2}} \begin{pmatrix}1 & 1 \\ 1 & -1 \end{pmatrix}\]

- Rotation about \(x\)-axis of the Bloch sphere:
  \[R_x(\theta) = \begin{pmatrix}\cos \frac{\theta}{2} & -i \sin \frac{\theta}{2} \\ -i \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{pmatrix}\]

- The controlled-NOT (CNOT) gate: Two-qubit Gate
  \[CNOT = \begin{pmatrix}1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}\]

NO-CLONING Theorem

Assume a cloning procedure \(U\), then
\[U|0\rangle|0\rangle = |0\rangle|0\rangle \quad U|1\rangle|0\rangle = |1\rangle|1\rangle\]
**Composite System postulate:** joint system \((A,B)\) in the tensor-product of \(A\) and \(B\).

The representation of two qubits lies in \(\mathbb{C}^2 \otimes \mathbb{C}^2\) (dim-4), where \(\mathbb{C}^2\) (dim-2) is for a qubit.

So \(|00\rangle = |0\rangle \otimes |0\rangle\):

\[
\begin{pmatrix}
1 \\
0
\end{pmatrix} \otimes \begin{pmatrix}
1 \\
0
\end{pmatrix} = \begin{pmatrix}
1 \\
0 \\
0 \\
0
\end{pmatrix}
\]

\(|01\rangle, |10\rangle, |11\rangle\):

\[
\begin{pmatrix}
1 \\
0 \\
0 \\
1
\end{pmatrix}
\]

\[
\begin{pmatrix}
0 \\
1 \\
0 \\
0
\end{pmatrix}
\]

\[
\begin{pmatrix}
0 \\
0 \\
0 \\
1
\end{pmatrix}
\]

A \(n\)-qubit system requires \(2^n\) dimensional space. **Exponential cost in classical simulation!**

**Examples of Common Quantum Gates**

- **Pauli gates:** Single-qubit Gate
  
  \(X = \begin{pmatrix}
0 & 1 \\
1 & 0
\end{pmatrix}, \quad Y = \begin{pmatrix}
0 & -i \\
i & 0
\end{pmatrix}, \quad Z = \begin{pmatrix}
1 & 0 \\
0 & -1
\end{pmatrix}\)

- **Hadamard gate:**
  
  \(H = \frac{1}{\sqrt{2}} \begin{pmatrix}
1 & 1 \\
1 & -1
\end{pmatrix}\)

- **Rotation about \(x\)-axis of the Bloch sphere:**
  
  \(R_x(\theta) = \begin{pmatrix}
\cos \frac{\theta}{2} & -i \sin \frac{\theta}{2} \\
-i \sin \frac{\theta}{2} & \cos \frac{\theta}{2}
\end{pmatrix}\)

- **The controlled-NOT (CNOT) gate:** Two-qubit Gate
  
  \(CNOT = \begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0
\end{pmatrix}\)

---

**NO-CLONING Theorem**

Assume a cloning procedure \(U\), then

\(U |0\rangle |0\rangle = |0\rangle |0\rangle \quad U |1\rangle |0\rangle = |1\rangle |1\rangle\)

Consider an arbitrary state \(|\psi\rangle = \alpha |0\rangle + \beta |1\rangle\)

\(U |\psi\rangle |0\rangle = \alpha |0\rangle |0\rangle + \beta |1\rangle |1\rangle \neq |\psi\rangle |\psi\rangle\)

**CONTRADICTION!**
Measurement postulate: how to read classical info out of q. system?

This information reading procedure will distribute/collapse the underlying q. systems.
Measurement postulate: how to read classical info out of q. system?

This information reading procedure will distribute/collapse the underlying q. systems.

- A measurement is modelled as a set of operators $M = \{M_m\}$ with $\sum_m M_m^\dagger M_m = I$.
- If a quantum system was in pure state $|\psi\rangle$ before the measurement, then:
  - the probability that measurement outcome is $\lambda$:
    $$p(m) = ||M_m|\psi\rangle||^2$$
    where $|| \cdot ||$ is the length of vector.
  - the state of the system after the measurement:
    $$\frac{M_m|\psi\rangle}{\sqrt{p(m)}}$$
Measurement postulate: how to read classical info out of q. system?

This information reading procedure will distribute/collapse the underlying q. systems.

- A measurement is modelled as a set of operators $M = \{M_m\}$ with $\sum M_m^\dagger M_m = I$.

- If a quantum system was in pure state $|\psi\rangle$ before the measurement, then:
  - the probability that measurement outcome is $\lambda$:
    $$p(m) = ||M_m|\psi\rangle||^2$$

  where $|| \cdot ||$ is the length of vector.

  - the state of the system after the measurement:
    $$\frac{M_m|\psi\rangle}{\sqrt{p(m)}}$$

*Examples* Consider $|0\rangle$

Measured in $\{ |0\rangle\langle 0|, |1\rangle\langle 1| \}$

-> $|0\rangle$ w/ prob. 1 (recover classical)
**Measurement postulate:** how to read classical info out of q. system?

This information reading procedure will **distribute/collapse** the underlying q. systems.

- A *measurement* is modelled as a set of operators $M = \{M_m\}$ with $\sum_m M_m^\dagger M_m = I$.
- If a quantum system was in pure state $|\psi\rangle$ before the measurement, then:
  - the probability that measurement outcome is $\lambda$:
    $$p(m) = |<M_m|\psi>|^2$$
    where $|| \cdot ||$ is the length of vector.
  - the state of the system after the measurement:
    $$\frac{M_m|\psi\rangle}{\sqrt{p(m)}}$$

**Examples**

Consider $|0\rangle$

Measured in $\{|0\rangle\langle0|, |1\rangle\langle1|\}$

$\rightarrow |0\rangle$ w/ prob. 1 (recover classical)

Measured in $\{|+\rangle\langle+|, |−\rangle\langle−|\}$

$\rightarrow |+\rangle$ w/ prob. 0.5
$\rightarrow |−\rangle$ w/ prob. 0.5
**Measurement postulate:** how to read classical info out of q. system?

This information reading procedure will **distribute/collapse** the underlying q. systems.

- A *measurement* is modelled as a set of operators \( M = \{M_m\} \) with \( \sum_m M_m^\dagger M_m = I \).

- If a quantum system was in pure state \(|\psi\rangle\) before the measurement, then:
  - the probability that measurement outcome is \(\lambda\):
    \[
p(m) = ||M_m|\psi\rangle||^2
    \]
    where \(||\cdot||\) is the length of vector.
  - the state of the system after the measurement:
    \[
    M_m|\psi\rangle \over \sqrt{p(m)}
    \]

**Examples** Consider \(|0\rangle\)

- Measured in \{ |0\rangle\langle0|, |1\rangle\langle1| \} -> \(|0\rangle\) w/ prob. 1 (recover classical)

- Measured in \{ |+\rangle\langle+|, |−\rangle\langle−| \} -> |+\rangle w/ prob. 0.5

-> |−\rangle w/ prob. 0.5

More advanced math formulation of ensemble of quantum states

**Density matrices**

- In the \(n\)-dimensional Hilbert space \(\mathbb{C}^n\), an operator is represented by an \(n \times n\) complex matrix \(A\).

- The trace of an operator \(A\) is \(tr(A) = \sum_i A_{ii}\) (the sum of the entries on the main diagonal).

- A positive semidefinite matrix \(\rho\) is called a **partial density matrix** if \(tr(\rho) \leq 1\); in particular, a **density matrix** \(\rho\) is a partial density matrix with \(tr(\rho) = 1\).

- For any mixed state \(\{(p_1, |\psi_1\rangle), \ldots, (p_k, |\psi_k\rangle)\}\),
  \[
  \rho = \sum_i p_i |\psi_i\rangle\langle\psi_i|
  \]
**Measurement postulate:** how to read classical info out of q. system?

This information reading procedure will *distribute/collapse* the underlying q. systems.

- A *measurement* is modelled as a set of operators $M = \{M_m\}$ with $\sum_m M_m^\dagger M_m = I$.
- If a quantum system was in pure state $|\psi\rangle$ before the measurement, then:
  - the probability that measurement outcome is $\lambda$:
    $$p(m) = ||M_m|\psi\rangle||^2$$
    where $|| \cdot ||$ is the length of vector.
  - the state of the system after the measurement:
    $$\frac{M_m|\psi\rangle}{\sqrt{p(m)}}$$

**Examples**

Consider $|0\rangle$

**Measured** in $\{ |0\rangle\langle 0|, |1\rangle\langle 1| \}$

- $\rightarrow |0\rangle$ w/ prob. 1 (recover classical)

**Measured** in $\{ |+\rangle\langle +|, |-\rangle\langle -| \}$

- $\rightarrow |+\rangle$ w/ prob. 0.5
- $\rightarrow |-\rangle$ w/ prob. 0.5

**More advanced math formulation of ensemble of quantum states**

**Density matrices**

- In the $n$-dimensional Hilbert space $\mathbb{C}^n$, an operator is represented by an $n \times n$ complex matrix $A$.
- The trace of an operator $A$ is $\text{tr}(A) = \sum_i A_{ii}$ (the sum of the entries on the main diagonal).
- A positive semidefinite matrix $\rho$ is called a *partial density matrix* if $\text{tr}(\rho) \leq 1$; in particular, a *density matrix* $\rho$ is a partial density matrix with $\text{tr}(\rho) = 1$.

**Example:**

$$\{(\frac{2}{3}, |0\rangle), (\frac{1}{3}, |-\rangle)\} \rightarrow \rho = \frac{2}{3} |0\rangle\langle 0| + \frac{1}{3} |-\rangle\langle -| = \frac{1}{6} \begin{pmatrix} 5 & -1 \\ -1 & 1 \end{pmatrix}$$

- For any mixed state $\{(p_1, |\psi_1\rangle), ..., (p_k, |\psi_k\rangle)\}$,
  $$\rho = \sum_i p_i |\psi_i\rangle\langle \psi_i|$$
Quantum While-Language

Syntax

A *core* language for imperative quantum programming

\[
S ::= \text{skip} \mid q := |0\rangle \\
S_1; S_2 \\
\overline{q} := U[q] \\
\text{if } (\Box m \cdot M[\overline{q}] = m \rightarrow S_m) \text{ fi} \\
\text{while } M[\overline{q}] = 1 \text{ do } S \text{ od}
\]
Quantum While-Language

Syntax

A core language for imperative quantum programming

\[
S ::= \text{skip} | q := \left|0\right\rangle | S_1; S_2 | \overline{q} := U[\overline{q}] |
\text{if} (\Box m \cdot M[\overline{q}] = m \rightarrow S_m) \text{ fi} |
\text{while} M[\overline{q}] = 1 \text{ do } S \text{ od}
\]

Classically, one has
\[ u := t \quad t \sim \text{expression.} \]

However, due to no-cloning,
1) initialization
2) unitary operation
Quantum While-Language

Syntax

A core language for imperative quantum programming

\[
S ::= \text{skip} \quad \quad q := |0\rangle \\
S_1 ; S_2 \\
\overline{q} := U[\overline{q}] \\
\text{if } (\Box m \cdot M[\overline{q}] = m \rightarrow S_m) \text{ fi} \\
\text{while } M[\overline{q}] = 1 \text{ do } S \text{ od}
\]

Classically, one has

\[ u := t \quad t \sim \text{expression}. \]

However, due to no-cloning,

1) initialization
2) unitary operation
Quantum While-Language

Syntax

A core language for imperative quantum programming

\[ S ::= \textbf{skip} \mid q := |0\rangle \]
\[ \mid S_1; S_2 \]
\[ \mid \overline{q} := U[\overline{q}] \]
\[ \mid \textbf{if} \ (m \cdot M[\overline{q}] = m \rightarrow S_m) \ \textbf{fi} \]
\[ \mid \textbf{while} M[\overline{q}] = 1 \ \textbf{do} S \ \textbf{od} \]

Quantum Data, Classical Control

Classically, one has
\[ u := t \quad t \sim \text{expression}. \]

However, due to no-cloning,
1) initialization
2) unitary operation

Classical control requires reading information out of quantum systems.

However, by measuring the guard, it leads to

1) a probabilistic choice of branches
2) a collapse of the guard state before entering each branch
Quantum 1-D Loop Walk

\[ \text{QW} \equiv c := \left| L \right>; \]
\[ p := \left| 0 \right>; \]
\[ \textbf{while } M[p] = \text{no do} \]
\[ c := H[c]; \]
\[ c, p := S[c, p] \]
\[ \textbf{od} \]

Operator Definition

\[ S = \sum_{i=0}^{n-1} \left| L \right> \left< L \right| \otimes \left| i \oplus 1 \right> \left< i \right| + \sum_{i=0}^{n-1} \left| R \right> \left< R \right| \otimes \left| i \oplus 1 \right> \left< i \right|. \]
Quantum 1-D Loop Walk

\[ QW \equiv c := |L\rangle; \quad \text{coin space} = \{L, R\} \]
\[ p := |0\rangle; \quad \text{position space} = \{0, ..., n-1\} \]
\[ \text{while } M[p] = \text{no do} \]
\[ c := H[c]; \]
\[ c, p := S[c, p] \]
\[ \text{od} \]

Operator Definition

\[ S = \sum_{i=0}^{n-1} |L\rangle\langle L| \otimes |i \oplus 1\rangle\langle i| + \sum_{i=0}^{n-1} |R\rangle\langle R| \otimes |i \oplus 1\rangle\langle i|. \]
Quantum 1-D Loop Walk

\[ QW \equiv c := |L\rangle; \quad \text{coin space} = \{L, R\} \]
\[ p := |0\rangle; \quad \text{position space} = \{0, \ldots, n-1\} \]

\[ \textbf{while } M[p] = \text{no } \textbf{do} \]
\[ c := H[c]; \quad \text{Create a new coin in superposition!} \]
\[ c, p := S[c, p] \quad \text{Random walk based on that coin!} \]
\[ \textbf{od} \]

Operator Definition

\[ S = \sum_{i=0}^{n-1} |L\rangle\langle L| \otimes |i \oplus 1\rangle\langle i| + \sum_{i=0}^{n-1} |R\rangle\langle R| \otimes |i \oplus 1\rangle\langle i|. \]
Quantum 1-D Loop Walk

\[ QW \equiv c := |L\rangle; \quad \text{coin space} = \{L, R\} \]
\[ p := |0\rangle; \quad \text{position space} = \{0, \ldots, n-1\} \]

\textbf{while} \( M[p] = \text{no} \) \textbf{do} \quad \text{Terminal of loop: position 1}
\[ c := H[c]; \quad \text{Create a new coin in superposition!} \]
\[ c, p := S[c, p] \quad \text{Random walk based on that coin!} \]
\textbf{od}

Operator Definition

\[ S = \sum_{i=0}^{n-1} |L\rangle\langle L| \otimes |i \ominus 1\rangle\langle i| + \sum_{i=0}^{n-1} |R\rangle\langle R| \otimes |i \oplus 1\rangle\langle i|. \]
Quantum 1-D Loop Walk

Goal: reason about this program

\[ QW \equiv c := |L\rangle; \quad \text{coin space} = \{L, R\} \]
\[ p := |0\rangle; \quad \text{position space} = \{0, \ldots, n-1\} \]

\[ \text{while } M[p] = \text{no do} \quad \text{Terminal of loop: position 1} \]
\[ c := H[c]; \quad \text{Create a new coin in superposition!} \]
\[ c, p := S[c, p] \quad \text{Random walk based on that coin!} \]
\[ \text{od} \]

Operator Definition

\[
S = \sum_{i=0}^{n-1} |L\rangle \langle L| \otimes |i \oplus 1\rangle \langle i| + \sum_{i=0}^{n-1} |R\rangle \langle R| \otimes |i \oplus 1\rangle \langle i|. 
\]
Operational Semantics

A configuration: \( \langle S, \rho \rangle \)
- \( S \) is a quantum program or \( E \) (the empty program)
- \( \rho \) is a partial density operator in \( \mathcal{H}_{\text{all}} = \bigotimes_q \mathcal{H}_q \)

\( (Sk) \quad \langle \text{skip}, \rho \rangle \rightarrow \langle E, \rho \rangle \)

\( (Ini) \quad \langle q := \ket{0}, \rho \rangle \rightarrow \langle E, \rho_0^q \rangle \)

- \text{type}(q) = \text{Boolean}:
  \[
  \rho_0^q = |0\rangle_q\langle 0| \rho |0\rangle_q\langle 0| + |0\rangle_q\langle 1| \rho |1\rangle_q\langle 0| 
  \]

- \text{type}(q) = \text{integer}:
  \[
  \rho_0^q = \sum_{n=-\infty}^{\infty} |0\rangle_q\langle n| \rho |n\rangle_q\langle 0| 
  \]
Semantics of Quantum While-Language

Operational Semantics

A configuration: $\langle S, \rho \rangle$

- $S$ is a quantum program or $E$ (the empty program)
- $\rho$ is a partial density operator in

$$\mathcal{H}_{\text{all}} = \bigotimes_q \mathcal{H}_q$$

for all $q$

\begin{align*}
(\text{Sk}) & \quad \langle \text{skip}, \rho \rangle \rightarrow \langle E, \rho \rangle \\
(\text{Ini}) & \quad \langle q := \ket{0}, \rho \rangle \rightarrow \langle E, \rho_0^q \rangle \\
\end{align*}

- $\text{type}(q) = \text{Boolean}$:

$$\rho_0^q = |0\rangle_q \langle 0|_q \rho_q |0\rangle_q \langle 0| + |0\rangle_q \langle 1|_q |1\rangle_q \langle 0|$$

- $\text{type}(q) = \text{integer}$:

$$\rho_0^q = \sum_{n=-\infty}^{\infty} |0\rangle_q \langle n|_q |\rho|_q |n\rangle_q \langle 0|$$

\begin{align*}
(\text{Uni}) & \quad \langle \overline{q} := U[\overline{q}], \rho \rangle \rightarrow \langle E, U\rho U^\dagger \rangle \\
(\text{Seq}) & \quad \langle S_1, \rho \rangle \rightarrow \langle S_1', \rho' \rangle \\
& \quad \langle S_1; S_2, \rho \rangle \rightarrow \langle S_1'; S_2, \rho' \rangle \\
\text{Convention:} & \quad E; S_2 = S_2. \\
(\text{IF}) & \quad \langle \text{if } (\square m \cdot M[\overline{q}] = m \rightarrow S_m) \text{ fi}, \rho \rangle \rightarrow \langle S_{m_0}.M_{m_0} \rho M_{m_0}^\dagger \rangle \\
\end{align*}

for each outcome $m$
Semantics of Quantum While-Language

Operational Semantics

A configuration: \( \langle S, \rho \rangle \)

- \( S \) is a quantum program or \( E \) (the empty program)
- \( \rho \) is a partial density operator in

\[
\mathcal{H}_{\text{all}} = \bigotimes_{\text{all } q} \mathcal{H}_q
\]

Operational Semantics

(Sk)

\[ \langle \text{skip}, \rho \rangle \rightarrow \langle E, \rho \rangle \]

(Ini)

\[ \langle q := \ket{0}, \rho \rangle \rightarrow \langle E, \rho^q_0 \rangle \]

- type(q) = Boolean:

\[
\rho^q_0 = \ket{0}_q\bra{0}_q\rho_{\ket{0}_q\bra{0}_q} + \ket{1}_q\bra{1}_q\rho_{\ket{1}_q\bra{1}_q}\ket{0}_q
\]

- type(q) = integer:

\[
\rho^q_0 = \sum_{n=-\infty}^{\infty} \ket{0}_q\bra{n}_q\rho_{\ket{n}_q\bra{n}_q}\ket{0}_q
\]

(Uni)

\[ \langle \overline{q} := U[\overline{q}], \rho \rangle \rightarrow \langle E, U\rho U^\dagger \rangle \]

(Seq)

\[ \langle S_1, \rho \rangle \rightarrow \langle S'_1, \rho' \rangle \]

\[ \langle S_1; S_2, \rho \rangle \rightarrow \langle S'_1; S_2, \rho' \rangle \]

Convention: \( E; S_2 = S_2. \)

(IF)

\[ \langle \text{if } (\square m \cdot M[\overline{q}] = m \rightarrow S_m) \text{ fi}, \rho \rangle \rightarrow \langle S_m; M_m\rho M^+_m \rangle \]

for each outcome \( m \)

Loop:

(L0)

\[ \langle \text{while } M[\overline{q}] = 1 \text{ do } S \text{ od}, \rho \rangle \rightarrow \langle E; M_0\rho M^+_0 \rangle \]

(L1)

\[ \langle \text{while } M[\overline{q}] = 1 \text{ do } S, \rho \rangle \rightarrow \langle S; \text{while } M[\overline{q}] = 1 \text{ do } S; M_1\rho M^+_1 \rangle \]
Operational Semantics

A configuration: \( \langle S, \rho \rangle \)

- \( S \) is a quantum program or \( E \) (the empty program)
- \( \rho \) is a partial density operator in

\[
H_{\text{all}} = \bigotimes_{q} H_{q}
\]

\((Sk)\)  \( \langle \text{skip}, \rho \rangle \to \langle E, \rho \rangle \)

\((Ini)\)  \( \langle q := \ket{0}, \rho \rangle \to \langle E, \rho_{0}^{q} \rangle \)

- \( \text{type}(q) = \text{Boolean}: \)

\[
\rho_{0}^{q} = |0\rangle_{q}\langle 0|\rho\langle 0|_{q} + |0\rangle_{q}\langle 1|\rho\langle 1|_{q} \langle 0|
\]

- \( \text{type}(q) = \text{integer}: \)

\[
\rho_{0}^{q} = \sum_{n=-\infty}^{\infty} |0\rangle_{q}\langle n|\rho\langle n|_{q} \langle 0|
\]

\((Uni)\)  \( \langle \overline{q} := U[q], \rho \rangle \to \langle E, U\rho U^{\dagger} \rangle \)

\((Seq)\)  \( \langle S_{1}, \rho \rangle \to \langle S'_{1}, \rho' \rangle \)

\( \langle S_{1}; S_{2}, \rho \rangle \to \langle S'_{1}; S'_{2}, \rho' \rangle \)

Convention: \( E; S_{2} = S_{2} \).

\((IF)\)  \( \langle \text{if } (\Box m \cdot M[q] = m \to S_{m}) \text{ fi}, \rho \rangle \to \langle S_{m}; M_{m} \rho M_{m}^{\dagger} \rangle \)

for each outcome \( m \)

Loop:

\((L0)\)  \( \langle \text{while } M[q] = 1 \; \text{do } S \; \text{ od}, \rho \rangle \to \langle E_{2}, M_{0} \rho M_{0}^{\dagger} \rangle \)

\((L1)\)  \( \langle \text{while } M[q] = 1 \; \text{do } S, \rho \rangle \to \langle S; \text{while } M[q] = 1 \; \text{do } S_{1}, M_{1} \rho M_{1}^{\dagger} \rangle \)

Capture the Collapse of the Guard state.
Denotational Semantics

Semantic function of quantum program $S$:

$$\llbracket S \rrbracket : \mathcal{D}(\mathcal{H}_{\text{all}}) \rightarrow \mathcal{D}(\mathcal{H}_{\text{all}})$$

$$\llbracket S \rrbracket (\rho) = \sum \{|\rho' : \langle S, \rho \rangle \rightarrow^* \langle E, \rho' \rangle|\} \text{ for all } \rho \in \mathcal{D}(\mathcal{H}_{\text{all}})$$
Semantics of Quantum While-Language

Denotational Semantics

Semantic function of quantum program $S$:

$$\llbracket S \rrbracket : \mathcal{D}(\mathcal{H}_{\text{all}}) \rightarrow \mathcal{D}(\mathcal{H}_{\text{all}})$$

$$\llbracket S \rrbracket (\rho) = \sum \{|\rho' : \langle S, \rho \rangle \rightarrow^* \langle E, \rho' \rangle|\} \text{ for all } \rho \in \mathcal{D}(\mathcal{H}_{\text{all}})$$

Observation:

$$\text{tr}(\llbracket S \rrbracket (\rho)) \leq \text{tr}(\rho)$$

for any quantum program $S$ and all $\rho \in \mathcal{D}(\mathcal{H}_{\text{all}})$.

- $\text{tr}(\rho) - \text{tr}(\llbracket S \rrbracket (\rho))$ is the probability that program $S$ diverges from input state $\rho$.  

Quantum Predicate & Hoare Triple

- A *quantum predicate* is a Hermitian operator (observable) $P$ such that $0 \subseteq P \subseteq I$.

A quantum predicate is a Hermitian operator (observable) $P$ such that $0 \subseteq P \subseteq I$.

A quantum predicate is a Hermitian operator (observable) $P$ such that $0 \leq P \leq I$.


A correctness formula is a statement of the form:

$$\{P\} S \{Q\}$$

where:
- $S$ is a quantum program
- $P$ and $Q$ are quantum predicates.
- Operator $P$ is called the precondition and $Q$ the postcondition.
A quantum predicate is a Hermitian operator (observable) $P$ such that $0 \subseteq P \subseteq I$.


A correctness formula is a statement of the form:

$$\{P\} S \{Q\}$$

where:
- $S$ is a quantum program
- $P$ and $Q$ are quantum predicates.
- Operator $P$ is called the precondition and $Q$ the postcondition.
Quantum Predicate & Hoare Triple

- A **quantum predicate** is a Hermitian operator (observable) \( P \) such that \( 0 \subseteq P \subseteq I \).


- A **correctness formula** is a statement of the form:

  \[
  \{P\} S \{Q\}
  \]

  where:
  - \( S \) is a quantum program
  - \( P \) and \( Q \) are quantum predicates.
  - Operator \( P \) is called the **precondition** and \( Q \) the **postcondition**.

1. \( \{P\} S \{Q\} \) is true in the sense of **total correctness**:

   \[
   \models_{\text{tot}} \{P\} S \{Q\}
   \]

   if

   \[
   tr(P\rho) \leq tr(Q[S](\rho)) \text{ for all } \rho.
   \]
Quantum Predicate & Hoare Triple

- A **quantum predicate** is a Hermitian operator (observable) $P$ such that $0 \leq P \leq I$.


- A **correctness formula** is a statement of the form:

\[
\{P\} S \{Q\}
\]

where:
- $S$ is a quantum program
- $P$ and $Q$ are quantum predicates.
- Operator $P$ is called the **precondition** and $Q$ the **postcondition**.

1. $\{P\} S \{Q\}$ is true in the sense of **total correctness**:

\[
\models_{\text{tot}} \{P\} S \{Q\}
\]

if

\[
\text{tr}(P \rho) \leq \text{tr}(Q[S](\rho)) \text{ for all } \rho.
\]
Quantum Predicate & Hoare Triple

- A quantum predicate is a Hermitian operator (observable) $P$ such that $0 \subseteq P \subseteq I$.


- A correctness formula is a statement of the form:

$$\{P\}S\{Q\}$$

where:

- $S$ is a quantum program
- $P$ and $Q$ are quantum predicates.
- Operator $P$ is called the precondition and $Q$ the postcondition.

1. $\{P\}S\{Q\}$ is true in the sense of total correctness:

$$\models_{\text{tot}} \{P\}S\{Q\}$$

if $\text{tr}(P\rho) \leq \text{tr}(Q[S](\rho))$ for all $\rho$.
Quantum Predicate & Hoare Triple

- A *quantum predicate* is a Hermitian operator (observable) $P$ such that $0 \subseteq P \subseteq I$.


- A *correctness formula* is a statement of the form:

$$\{P\}S\{Q\}$$

where:
- $S$ is a quantum program
- $P$ and $Q$ are quantum predicates.
- Operator $P$ is called the *precondition* and $Q$ the *postcondition*.

1. $\{P\}S\{Q\}$ is true in the sense of *total correctness*:

$$\models_{\text{tot}} \{P\}S\{Q\}$$

if $\text{tr}(P\rho) \leq \text{tr}(Q[S](\rho))$ for all $\rho$.

2. $\{P\}S\{Q\}$ is true in the sense of *partial correctness*:

$$\models_{\text{par}} \{P\}S\{Q\},$$

if $\text{tr}(P\rho) \leq \text{tr}(Q[S](\rho)) + [\text{tr}(\rho) - \text{tr}(S(\rho))]$.
Quantum Predicate & Hoare Triple

- A quantum predicate is a Hermitian operator (observable) \( P \) such that \( 0 \leq P \leq I \).


- A correctness formula is a statement of the form:
  \[
  \{ P \} S \{ Q \}
  \]
  where:
  - \( S \) is a quantum program
  - \( P \) and \( Q \) are quantum predicates.
  - Operator \( P \) is called the precondition and \( Q \) the postcondition.

1. \( \{ P \} S \{ Q \} \) is true in the sense of total correctness:
   \[
   \models_{\text{tot}} \{ P \} S \{ Q \}
   \]
   if \( \text{Pre-S State} \quad Post-S State \)
   \[
   tr(P\rho) \leq tr(Q[S](\rho)) \quad \text{for all } \rho.
   \]

2. \( \{ P \} S \{ Q \} \) is true in the sense of partial correctness:
   \[
   \models_{\text{par}} \{ P \} S \{ Q \},
   \]
   if
   \[
   tr(P\rho) \leq tr(Q[S](\rho)) + [tr(\rho) - tr([S](\rho))]
   \]

---

Continuous logic
[0, 1]
Matrix Upgrade

Similar as Classical Hoare triple w/ different semantics

Pre-S State  Post-S State

Semantics
Divergence
(Axiom Sk) \[ \{P\} \text{Skip}\{P\} \]

(Axiom Ini)

\[ \text{type}(q) = \text{Boolean} : \]
\[ \{0\}_q\langle 0\| P\rangle_0\langle 0| + |1\}_q\langle 0\| P\rangle_0\langle 1| \}q := |0\}_q\{P\} \]

\[ \text{type}(q) = \text{integer} : \]
\[ \{ \sum_{n=-\infty}^{\infty} |n\}_q\langle 0\| P\rangle_0\langle n| \}q := |0\}_q\{P\} \]

(Axiom Uni) \[ \{U'^{\dagger}PU\}q := U[q]\{P\} \]

(Rule Seq) \[ \{P\} S_1\{Q\} \quad \{Q\} S_2\{R\} \]
\[ \{P\} S_1; S_2\{R\} \]

(Rule IF) \[ \{P_m\} S_m\{Q\} \quad \text{for all } m \]
\[ \{\sum_m M_m^\dagger P_m M_m\} \quad \text{if } (\square m \cdot M[q] = m \rightarrow S_m) \quad \text{fi} \{Q\} \]

(Rule LP) \[ \{Q\} S\{M_0^\dagger P M_0 + M_1^\dagger Q M_1\} \]
\[ \{M_0^\dagger P M_0 + M_1^\dagger Q M_1\} \quad \text{while } M[q] = 1 \quad \text{do } S\{P\} \]

(Rule Ord) \[ P \subseteq P' \quad \{P'\} S\{Q'\} \quad Q' \subseteq Q \]
\[ \{P\} S\{Q\} \]
Quantum Hoare logic for Partial Correctness

(Axiom Sk) \( \{P\} \text{Skip} \{P\} \)

(Axiom Ini)

\[ \text{type}(q) = \text{Boolean} : \]
\[ \{ |0\rangle_q \langle 0|P|0\rangle_q \langle 0| + |1\rangle_q \langle 0|P|0\rangle_q \langle 1| \} q := |0\} \{P\} \]

\[ \text{type}(q) = \text{integer} : \]
\[ \{ \sum_{n=-\infty}^{\infty} |n\rangle_q \langle 0|P|0\rangle_q \langle n| \} q := |0\} \{P\} \]

(Axiom Uni) \( \{U^\dagger PU \} \bar{q} := U[\bar{q} \} \{P\} \)

(Rule Seq) \( \{P\} S_1 \{Q\} \quad \{Q\} S_2 \{R\} \quad \{P\} S_1; S_2 \{R\} \)

(Rule IF) \( \{P_m\} S_m \{Q\} \) for all \( m \)
\( \{\sum_m M^\dagger_m P M_m \} \text{if} (\Box m \cdot M[\bar{q}] = m \rightarrow S_m) \text{fi} \{Q\} \)

(Rule LP)
\( \{Q\} S \{M^\dagger_0 P M_0 + M^\dagger_1 Q M_1 \} \)
\( \{M^\dagger_0 P M_0 + M^\dagger_1 Q M_1 \} \text{while} M[\bar{q}] = 1 \text{ do } S \{P\} \)

(Rule Ord)
\( P \subseteq P' \quad \{P'\} S \{Q'\} \quad Q' \subseteq Q \quad \{P\} S \{Q\} \)

Parts of Classical Hoare Logic

AXIOM 2: ASSIGNMENT

\( \{p[u := t]\} \quad u := t \quad \{p\} \)
Quantum Hoare logic for Partial Correctness

\begin{align*}
(Axiom \ Sk) & \quad \{P\} \text{Skip}\{P\} \\
(Axiom \ Ini) & \quad \text{type}(q) = \text{Boolean :} \\
& \quad \{0\}_{q}\langle 0\rangle_{q}|0\rangle_{q} + |1\rangle_{q}\langle 0\rangle_{q}|0\rangle_{q}\langle 1\rangle_{q} := |0\rangle \{P\} \\
& \quad \text{type}(q) = \text{integer :} \\
& \quad \{ \sum_{n=-\infty}^{\infty} |n\rangle_{q}\langle 0|0\rangle_{q}\langle n\rangle_{q} := |0\rangle \{P\} \} \\
(Axiom \ Uni) & \quad \{U^{\dagger}PU\} q := U [\bar{q}] \{P\} \\
\end{align*}

\begin{align*}
\text{(Rule Seq)} & \quad \{P\} S_1 \{Q\} \quad \{Q\} S_2 \{R\} \\
& \quad \{P\} S_1 ; S_2 \{R\} \\
\text{(Rule IF)} & \quad \{P_m\} S_m \{Q\} \quad \text{for all } m \\
& \quad \sum_m M_m^* P_m M_m \text{ if } (\square m \cdot M[\bar{q}] = m \rightarrow S_m) \text{ fi } \{Q\} \\
\text{(Rule LP)} & \quad \{Q\} S \{M_0^* P M_0 + M_1^* Q M_1\} \\
& \quad \{M_0^* P M_0 + M_1^* Q M_1\} \text{ while } M[\bar{q}] = 1 \text{ do } \{P\} \\
\text{(Rule Ord)} & \quad \mathbf{P} \sqsubseteq \mathbf{P}' \quad \{P'\} S \{Q'\} \quad \mathbf{Q'} \sqsubseteq \mathbf{Q} \\
& \quad \{P\} S \{Q\} \\
\end{align*}

Parts of Classical Hoare Logic

**AXIOM 2: ASSIGNMENT**
\[
\{p[u := t]\} u := t \{p\}
\]

**RULE 4: CONDITIONAL**
\[
\{p \land B\} S_1 \{q\}, \{p \land \neg B\} S_2 \{q\} \\
\{p\} \text{ if } B \text{ then } S_1 \text{ else } S_2 \text{ fi } \{q\}
\]

**RULE 5: LOOP**
\[
\{p \land B\} S \{p\} \\
\{p\} \text{ while } B \text{ do } S \text{ od } \{p \land \neg B\}
\]
Quantum Hoare logic for Partial Correctness

(Axiom Sk) \( \{P\} \text{Skip}\{P\} \)

(Axiom Ini)

\begin{align*}
type(q) & = \text{Boolean} : \\
\{0\}_q\langle 0|P|0\rangle_q\langle 0 | + |1\rangle_q\langle 0|P|0\rangle_q\langle 1 \rangle q := |0\rangle \{P\}
\end{align*}

\begin{align*}
type(q) & = \text{integer} : \\
\{ \sum_{n=-\infty}^{\infty} \{n\}_q\langle 0|P|0\rangle_q\langle n | \} q := |0\rangle \{P\}
\end{align*}

(Axiom Uni) \( U^\dagger PU\bar{q} := U[\bar{q}]\{P\} \)

(Rule Seq) \( \frac{\{P\} S_1\{Q\} \quad \{Q\} S_2\{R\}}{\{P\} S_1; S_2\{R\}} \)

(Rule IF) \( \frac{\sum_m M^\dagger_m P_m M_m}{\text{if } (\square m \cdot M[q] = m \rightarrow S_m) \text{ fi } \{Q\}} \)

(Rule LP) \( \frac{\{Q\} S\{M^\dagger_0 P M_0 + M^\dagger_1 Q M_1\}}{\text{while } M[q] = 1 \text{ do } \{P\}} \)

(Rule Ord) \( \frac{P \sqsubseteq P' \quad \{P'\} S\{Q'\} \quad Q' \sqsubseteq Q}{\{P\} S\{Q\}} \)

Parts of Classical Hoare Logic

**AXIOM 2: ASSIGNMENT**

\( \{p[u := t]\} u := t \{p\} \)

**RULE 4: CONDITIONAL**

\( \{p \land B\} S_1 \{q\}, \{p \land \neg B\} S_2 \{q\} \quad \{p\} \text{ if } B \text{ then } S_1 \text{ else } S_2 \text{ fi } \{q\} \)

**RULE 5: LOOP**

\( \frac{\{p \land B\} S \{p\}}{\{p\} \text{ while } B \text{ do } S \text{ od } \{p \land \neg B\}} \)

Theorem (Soundness and Completeness)

For any quantum program \( S \) and quantum predicates \( P, Q, \)

\( \models_{\text{par}} \{P\} S\{Q\} \) if and only if \( \vdash_{PD} \{P\} S\{Q\} \).

Ying. TOPLAS, 2011.
Quantum Hoare logic for Total Correctness

Proof System for Total Correctness

Let $P$ be a quantum predicate and $\epsilon > 0$. A function

$$t: \mathcal{D}(\mathcal{H}_{\text{all}}) \ (\text{density operators}) \rightarrow \mathbb{N}$$

is called a $(P, \epsilon)$-\textit{ranking function} of quantum loop:

$$\textbf{while } M[\overline{q}] = 1 \textbf{ do } S \textbf{ od}$$

if for all $\rho$:

1. $t(\mathbb{S}(M_1 \rho M_1^*)) \leq t(\rho)$;  

2. $\text{tr}(P\rho) \geq \epsilon$ implies $t(\mathbb{S}(M_1 \rho M_1^*)) < t(\rho)$

\textbf{Theorem (Soundness and Completeness)}

For any quantum program $S$ and quantum predicates $P \ Q$,

$$\models_{\text{tot}} \{P\}S\{Q\} \text{ if and only if } \vdash_{TD} \{P\}S\{Q\}.$$ 

Quantum Hoare logic and Invariants: POPL17

\[
QW \equiv c := |L\rangle; \quad \text{coin space} = \{L, R\}
\]

\[
p := |0\rangle; \quad \text{position space} = \{0, \ldots, n-1\}
\]

while \( M[p] = \text{no} \) do

\[
c := H[c]; \quad \text{Terminal of loop: position 1}
\]

\[
c, p := S[c, p] \quad \text{Create a new coin in superposition!}
\]

od

\[
c, p := S[c, p] \quad \text{Random walk based on that coin!}
\]
Quantum Hoare logic and Invariants: POPL17

\[ QW \equiv c := |L\rangle; \quad \text{coin space} = \{L, R\} \]
\[ p := |0\rangle; \quad \text{position space} = \{0, \ldots, n-1\} \]
\[ \textbf{while } M[p] = no \textbf{ do } \quad \text{Terminal of loop: position 1} \]
\[ c := H[c]; \quad \text{Create a new coin in superposition!} \]
\[ c, p := S[c, p] \quad \text{Random walk based on that coin!} \]
\[ \textbf{od} \]

Control - Flow - Graph
Quantum Hoare logic and Invariants: POPL17

\[ QW \equiv c := |L\rangle; \quad \text{coin space} = \{L, R\} \]
\[ p := |0\rangle; \quad \text{position space} = \{0, \ldots, n-1\} \]
\[ \textbf{while } M[p] = \text{no } \textbf{do } \]
\[ c := H[c]; \quad \text{Create a new coin in superposition!} \]
\[ c, p := S[c, p] \quad \text{Random walk based on that coin!} \]
\[ \textbf{od} \]

Control - Flow - Graph

Invariants

- A set \( \Pi \) of paths is \textit{prime} if for each
  \[ \pi = l_1 \xrightarrow{\epsilon_1} \ldots \xrightarrow{\epsilon_{n-1}} l_n \in \Pi \]
  its proper initial segments \( l_1 \xrightarrow{\epsilon_1} \ldots \xrightarrow{\epsilon_{k-1}} l_k \notin \Pi \) for all \( k < n \).

- Let \( G = \langle \mathcal{H}, L, l_0, \rightarrow \rangle \), \( \Theta \) a quantum predicate (initial condition), \( l \in L \). An \textit{invariant} at \( l \) is a quantum predicate \( O \) such that for any density operator \( \rho \), any prime set \( \Pi \) of paths from \( l_0 \) to \( l \):
  \[ \text{tr}(\Theta \rho) \leq 1 - \text{tr} (\mathcal{E}_\Pi(\rho)) + \text{tr} (O \mathcal{E}_\Pi(\rho)) \]
  where \( \mathcal{E}_\Pi = \sum \{|\mathcal{E}_\pi : \pi \in \Pi|\} \).
Quantum Hoare logic and Invariants: POPL17

\[ QW \equiv c := |L\rangle; \quad \text{coin space} = \{L, R\} \]
\[ p := |0\rangle; \quad \text{position space} = \{0, \ldots, n-1\} \]
\[ \textbf{while } M[p] = no \textbf{ do} \quad \text{Terminal of loop: position 1} \]
\[ c := H[c]; \quad \text{Create a new coin in superposition!} \]
\[ c, p := S[c, p] \quad \text{Random walk based on that coin!} \]
\[ \textbf{od} \]

Invariants

- A set \( \Pi \) of paths is \textit{prime} if for each
  \[ \pi = l_1 \xrightarrow{\epsilon_1} \cdots \xrightarrow{\epsilon_{n-1}} l_n \in \Pi \]
  its proper initial segments \( l_1 \xrightarrow{\epsilon_1} \cdots \xrightarrow{\epsilon_{k-1}} l_k \notin \Pi \) for all \( k < n \).

- Let \( G = \langle \mathcal{H}, l_0, \rightarrow \rangle \), \( \Theta \) a quantum predicate (initial condition), \( l \in L \). An \textit{invariant} at \( l \) is a quantum predicate \( O \) such that for any density operator \( \rho \), any prime set \( \Pi \) of paths from \( l_0 \) to \( l \):
  \[ tr(\Theta \rho) \leq 1 - tr(\mathcal{E}_\Pi(\rho)) + tr(\Theta \mathcal{E}_\Pi(\rho)) \]
  where \( \mathcal{E}_\Pi = \sum \{|\mathcal{E}_\pi : \pi \in \Pi|\} \).
Finding Quantum Invariants

Theorem (Partial Correctness)

Let $P$ be a quantum program. If $O$ is an invariant at $l_{out}^P$ in $S_P$, then

$$\models_{par} \{\Theta\} P\{O\}$$
Finding Quantum Invariants

Theorem (Partial Correctness)

Let $P$ be a quantum program. If $O$ is an invariant at $l_{out}^P$ in $S_P$, then

$$\models_{par} \{\Theta\} P\{O\}$$

Inductive Assertion Maps

- Given $G = \langle \mathcal{H}, L, l_0, \rightarrow \rangle$ with a cutset $C$ and initial condition $\Theta$.
- An assertion map $\eta$ is a mapping $\eta$ from each cutpoint $l \in C$ to a quantum predicate $\eta(l)$.
- $\Pi_l$: the set of all basic paths from $l$ to some cutpoint.
- $l_{\pi}$: the last location in a path $\pi$.

An assertion map $\eta$ is inductive if:

- **Initiation**: for any density operator $\rho$:

  $$\text{tr}(\Theta \rho) \leq 1 - \text{tr}\left(\mathcal{E}_{\Pi_{l_0}}(\rho)\right) + \sum_{\pi \in \Pi_{l_0}} \text{tr}\left(\eta(l_{\pi})\mathcal{E}_{\pi}(\rho)\right);$$

- **Consecution**: for any density operator $\rho$, each cutpoint $l \in C$:

  $$\text{tr}(\eta(l)\rho) \leq 1 - \text{tr}\left(\mathcal{E}_{\Pi_l}(\rho)\right) + \sum_{\pi \in \Pi_l} \text{tr}\left(\eta(l_{\pi})\mathcal{E}_{\pi}(\rho)\right).$$
Finding Quantum Invariants

Theorem (Partial Correctness)

Let $P$ be a quantum program. If $O$ is an invariant at $l^P_{out}$ in $S_P$, then

$$\models_{par} \{\Theta\} P\{O\}$$

Inductive Assertion Maps

- Given $\mathcal{G} = \langle \mathcal{H}, L, l_0, \rightarrow \rangle$ with a cutset $C$ and initial condition $\Theta$.
- An assertion map is a mapping $\eta$ from each cutpoint $l \in C$ to a quantum predicate $\eta(l)$.
- $\Pi_l$: the set of all basic paths from $l$ to some cutpoint.
- $l_{\pi}$: the last location in a path $\pi$.
- An assertion map $\eta$ is inductive if:
  - **Initiation**: for any density operator $\rho$:
    $$\text{tr}(\Theta \rho) \leq 1 - \text{tr} \left( \mathcal{E}_{\Pi l_0} (\rho) \right) + \sum_{\pi \in \Pi l_0} \text{tr} \left( \eta(l_{\pi}) \mathcal{E}_{\pi} (\rho) \right);$$
  - **Consecution**: for any density operator $\rho$, each cutpoint $l \in C$:
    $$\text{tr}(\eta(l) \rho) \leq 1 - \text{tr} \left( \mathcal{E}_{\Pi l} (\rho) \right) + \sum_{\pi \in \Pi l} \text{tr} \left( \eta(l_{\pi}) \mathcal{E}_{\pi} (\rho) \right).$$

Reducing Global Constraints Into Local Ones

Reduce to a SDP (Semi-Definite Programming) Problem

- Assume $C = \{l_0, l_1, ..., l_m\}$.
- Write $O_i = \eta(l_i)$ for $i = 0, 1, ..., m$.
- $\mathcal{E}_{ij}^\pi = \sum \{|E_{\pi}^\ast : \text{basic path } l_i \rightarrow l_j \}$ for $i, j = 0, 1, ..., m$. 

$$\nabla$$
Theorem

Invariant Generation Problem is equivalent to find complex matrices $O_0, O_1, ..., O_m$ satisfying the constraints:

\[ 0 \sqsubseteq \sum_j \mathcal{E}_{0j}(O_j) + A, \]

\[ 0 \sqsubseteq \sum_{j \neq i} \mathcal{E}_{ij}(O_j) + (\mathcal{E}_{ii} - I)(O_i) + A_i \quad (i = 0, 1, ..., m), \]

\[ 0 \sqsubseteq O_i \sqsubseteq I \quad (i = 0, 1, ..., m), \]

where:

\[
\begin{cases}
A = I - \sum_j \mathcal{E}_{0j}^*(I) - \Theta, \\
A_i = I - \sum_j \mathcal{E}_{ij}^*(I) \quad (i = 0, 1, ..., m).
\end{cases}
\]
\( QW \equiv c := |L\rangle; \)
\( p := |0\rangle; \)
\[
\textbf{while} \ M[p] = no \ \textbf{do} \quad c := H[c]; \quad c, p := S[c, p] \\
\textbf{od}
\]
\[ QW \equiv c := |L\rangle; \]
\[ p := |0\rangle; \]
\[ \textbf{while } M[p] = \text{no do} \]
\[ c := H[c]; \]
\[ c, p := S[c, p] \]
\[ \textbf{od} \]

**Invariant SDPs for Quantum 1-D Loop Walk**

Choose cut-set \( C = \{l_0, l_3\} \) with \( l_3 = l_{out} \). \( \Theta = I \). Invariants \( O_0 \) and \( O_3 \) satisfy the following constraints:

\[
0 \subseteq \mathcal{E}^*_0(O_0) + \mathcal{E}^*_3(O_3) - \Theta, \tag{1}
\]
\[
0 \subseteq (\mathcal{E}^*_0 - \mathcal{I})(O_0) + \mathcal{E}^*_3(O_3), \tag{2}
\]
\[
0 \subseteq (\mathcal{E}^*_3 - \mathcal{I})(O_3) - (I - \mathcal{E}^*_3(I)), \tag{3}
\]
\[
0 \subseteq O_0, O_3 \subseteq I \tag{4}
\]

\( E_{00} = E_{00} \circ E_{00}^+, E_{03} = E_{03} \circ E_{03}^+, E_{33} = \mathcal{I} \),

\( E_{00} = S(H \otimes I_p)(I_c \otimes M_{no}), E_{03} = I_c \otimes M_{yes}, \) and \( I_c, I_p \) identities.
\[ QW \equiv c := |L\rangle; \]
\[ p := |0\rangle; \]
\[ \text{while } M[p] = \text{no do} \]
\[ c := H[c]; \]
\[ c, p := S[c, p] \]
\[ \text{od} \]

Invariant SDPs for Quantum 1-D Loop Walk

Choose cut-set \( C = \{l_0, l_3\} \) with \( l_3 = l_{\text{out}} \). \( \Theta = I \). Invariants \( O_0 \) and \( O_3 \) satisfy the following constraints:

\[ 0 \subseteq \mathcal{E}_{00}^*(O_0) + \mathcal{E}_{03}^*(O_3) - \Theta, \quad (1) \]
\[ 0 \subseteq (\mathcal{E}_{00}^* - \mathcal{I})(O_0) + \mathcal{E}_{03}^*(O_3), \quad (2) \]
\[ 0 \subseteq (\mathcal{E}_{33}^* - \mathcal{I})(O_3) - (I - \mathcal{E}_{33}^*(I)), \quad (3) \]
\[ 0 \subseteq O_0, O_3 \subseteq I \quad (4) \]

\( \mathcal{E}_{00} = E_{00} \circ E_{00}^\dagger, \mathcal{E}_{03} = E_{03} \circ E_{03}^\dagger, \mathcal{E}_{33} = \mathcal{I}, \)
\( E_{00} = S(H \otimes I_p)(I_c \otimes M_{\text{no}}), E_{03} = I_c \otimes M_{\text{yes}}, \) and \( I_c, I_p \) identities.

Using SDP Solver

\[ O_3 = I_c \otimes |1\rangle \langle 1| \]
\[ \text{tr}(O_3 \rho_{\text{out}}) \geq \text{tr}(\Theta \rho_{\text{in}}) = 1. \]

Namely, QW always terminates at the position \( |1\rangle \) regardless of the input state \( \rho_0 \).
Invariant SDPs for Quantum 1-D Loop Walk

QW ≡ c := |L⟩;
p := |0⟩;
while M[p] = no do
c := H[c];
c, p := S[c, p]
od

Invariant SDPs for Quantum 1-D Loop Walk

Choose cut-set C = \{l_0, l_3\} with l_3 = l_{out}. Θ = I. Invariants O_0 and O_3 satisfy the following constraints:

1. \(0 \subseteq \mathcal{E}^*_0(O_0) + \mathcal{E}^*_3(O_3) - \Theta\),
2. \(0 \subseteq (\mathcal{E}^*_0 - \mathcal{I})(O_0) + \mathcal{E}^*_3(O_3)\),
3. \(0 \subseteq (\mathcal{E}^*_3 - \mathcal{I})(O_3) - (I - \mathcal{E}^*_3(I))\),
4. \(0 \subseteq O_0, O_3 \subseteq I\)

\(\mathbb{E}_{00} = E_{00} \circ E^\dagger_{00}, \mathbb{E}_{03} = E_{03} \circ E^\dagger_{03}, \mathbb{E}_{33} = \mathcal{I} , E_{00} = S(H \otimes I_p)(I_c \otimes M_{no}), E_{03} = I_c \otimes M_{yes}, \text{and } I_c, I_p \text{ identities.}\)

Using SDP Solver

\(O_3 = I_c \otimes |1⟩⟨1|\)
\(\text{tr}(O_3 \rho_{out}) \geq \text{tr}(\Theta \rho_{in}) = 1\).

Namely, QW always terminates at the position |1⟩ regardless of the input state \(ρ_0\).

Drawback: all these matrices are exponentially large.
Further Readings: Thank You! Q & A

Applications

- Quantum walk on an $n$-circle.
- Quantum Metropolis sampling on $n$-qubits.
- Repeat-Until-Success.
- Quantum Search.
- Quantum Bernoulli Factory.
- Recursively written Quantum Fourier Transformation.

References

Outline

(1) Introduction to Quantum Computing and Potential Roles of Programming Languages (25 min + 5 Q & A)

(2) A Mini-Course of Quantum Hoare Logic on Quantum While Language (30 min + 5 Q & A)

(3) Discussion on existing and potential Programming Language research opportunities (20 min + 5 Q & A)

Reference: tutorial slides and some references are available at https://www.cs.umd.edu/~xwu/mini_lib.html
Summary from Part I

Quantum PLs: some

Software Tool-chain: a little

Architecture: a little

Security: a little

Hardware Design: almost none

From the implementation perspective

Highlight some concrete problems! (Not a survey)
Design of Quantum Programming Languages

Gap: (1) too-low-level-abstraction: very hard to write complex programs
     (2) lack of scalable verification: very hard to write correct programs
     (3) lack of many desirable analyses, automation, & optimization: a lot of burdens on the programmers

Verifying the circuit by observation .... not scalable ...
Design of Quantum Programming Languages

**Gap:**
1. **too-low-level-abstraction:** very hard to write complex programs
2. **lack of scalable verification:** very hard to write correct programs
3. **lack of many desirable analyses, automation, & optimization:** a lot of burdens on the programmers

Existing work on type enforced **correctness** in QPLs

**No-Cloning:** use linear types for quantum variables (Quipper, QWIRE)

Verifying the circuit by observation …. not scalable …
Design of Quantum Programming Languages

**Gap:** (1) **too-low-level-abstraction:** very hard to write **complex** programs
(2) **lack of scalable verification:** very hard to write **correct** programs

(3) **lack of many desirable analyses, automation, & optimization:** a lot of burdens on the programmers

---

Verifying the circuit by observation .... not scalable ...

---

**Existing work on type enforced correctness** in QPLs

**No-Cloning:** use **linear** types for quantum variables (Quipper, QWIRE)

**Ancilla:** keep track of the scope of ancilla qubits (Quipper)
Design of QPLs: the level of abstraction

**GAP:** in the past discussion, we focus on circuit-level-abstraction on bits

Hard to code even *real numbers* and basic *arithmetic* operations common as part of quantum algorithm design
Design of QPLs: the level of abstraction

GAP: in the past discussion, we focus on *circuit-level-abstraction* on *bits*

Hard to code even *real numbers* and basic *arithmetic* operations

*common as part of quantum algorithm design*

**Question 1:** high-level DSLs for classical computation in superposition?

Need to compile classical computation into *reversible computation*

Handle the *ancilla qubits* and potentially simpler *error-correction* issues.
Design of QPLs: the level of abstraction

**GAP:** in the past discussion, we focus on circuit-level-abstraction on *bits*

Hard to code even *real numbers* and basic *arithmetic* operations

*common as part of quantum algorithm design*

**Question 1:** high-level DSLs for classical computation in superposition?

Need to compile classical computation into *reversible computation*

Handle the *ancilla qubits* and potentially simpler *error-correction* issues.

**Question 2:** high-level abstractions for quantum applications?

Circuits pass little *structural information* of the target applications.

*e.g.*, encoding, structural freedom or so for *automation* and *optimization*
Design of QPLs: the level of abstraction

GAP: in the past discussion, we focus on circuit-level-abstraction on bits

Hard to code even real numbers and basic arithmetic operations common as part of quantum algorithm design

Question 1: high-level DSLs for classical computation in superposition?

Need to compile classical computation into reversible computation

Handle the ancilla qubits and potentially simpler error-correction issues.

Question 2: high-level abstractions for quantum applications?

Circuits pass little structural information of the target applications.

- e.g., encoding, structural freedom or so for automation and optimization

Candidate applications: Quantum Simulation
Quantum Variational Methods
Design of QPLs: the level of abstraction

GAP: in the past discussion, we focus on circuit-level-abstraction on bits

Hard to code even real numbers and basic arithmetic operations common as part of quantum algorithm design

Question 1: high-level DSLs for classical computation in superposition?

Need to compile classical computation into reversible computation

Handle the ancilla qubits and potentially simpler error-correction issues.

Question 2: high-level abstractions for quantum applications?

Circuits pass little structural information of the target applications.

- e.g., encoding, structural freedom or so for automation and optimization

Candidate applications: Quantum Simulation
Quantum Variational Methods

Question 3: allow program analysis w/ high-level abstractions?
GAP: existing QPLs focus on describing circuits, while not using other common high-level abstractions, e.g., objects, data structures.
Design of QPLs: the support of high-level objects

**GAP:** existing QPLs focus on describing circuits, while not using other common high-level abstractions, e.g., *objects, data structures*.

**Question 4:** allow programmers to use (classical) data structures?

Growing need to use **complicated** DS. (e.g. Ambainis’s element distinctness)
Design of QPLs: the support of high-level objects

**GAP:** existing QPLs focus on describing circuits, while not using other common high-level abstractions, e.g., *objects, data structures*.

**Question 4:** allow programmers to use (classical) data structures?

Growing need to use *complicated* DS. (e.g. Ambainis’s element distinctness)

But using classical DS in quantum faces many issues:

- e.g., data manipulation is generally non-reversible, even if computation can be made so.
- **Reversibility** alone does not guarantee correct quantum interference b/c workspace.
- **Efficiency** issues about reimplementing DS w/ above constraints.
Design of QPLs: the support of high-level objects

**GAP:** existing QPLs focus on describing circuits, while not using other common high-level abstractions, e.g., *objects, data structures.*

**Question 4:** allow programmers to use (classical) data structures?

Growing need to use complicated DS. (e.g. Ambainis’s element distinctness)

But using classical DS in quantum faces many issues:

e.g., data manipulation is generally non-reversible, even if computation can be made so. **Reversibility** alone does not guarantee correct quantum interference b/c workspace. **Efficiency** issues about reimplementing DS w/ above constraints.

However, well-defined classical problems that PL might help with.
Design of QPLs: the support of high-level objects

**GAP:** existing QPLs focus on describing circuits, while not using other common high-level abstractions, e.g., *objects, data structures*.

**Question 4:** allow programmers to use (classical) data structures?

Growing need to use *complicated* DS. (e.g. Ambainis’s element distinctness)

But using classical DS in quantum faces many issues:

- e.g., data manipulation is generally *non-reversible*, even if computation can be made so.
  - *Reversibility* alone does not guarantee correct quantum interference b/c workspace.
  - *Efficiency* issues about reimplementing DS w/ above constraints.

However, well-defined *classical problems* that PL might help with.

**Question 5:** allow programmers to define quantum object/DS?
Design of QPLs: the support of high-level objects

**GAP:** existing QPLs focus on describing circuits, while not using other common high-level abstractions, e.g., *objects, data structures*.

**Question 4:** allow programmers to use (classical) data structures?

Growing need to use **complicated** DS. (e.g. Ambainis’s element distinctness)

But using classical DS in quantum faces many issues:

- e.g., data manipulation is generally **non-reversible**, even if computation can be made so.
  - **Reversibility** alone does not guarantee correct quantum interference b/c workspace.
  - **Efficiency** issues about reimplementing DS w/ above constraints.

However, well-defined **classical problems** that PL might help with.

**Question 5:** allow programmers to define quantum object/DS?

Allow direct modeling of **quantum hardware components** (QRAM, Sensors)
Design of QPLs: the support of high-level objects

GAP: existing QPLs focus on describing circuits, while not using other common high-level abstractions, e.g., objects, data structures.

**Question 4:** allow programmers to use (classical) data structures?

Growing need to use complicated DS. (e.g. Ambainis’s element distinctness)

But using classical DS in quantum faces many issues:

- e.g., data manipulation is generally non-reversible, even if computation can be made so.
  - Reversibility alone does not guarantee correct quantum interference b/c workspace.
  - Efficiency issues about reimplementing DS w/ above constraints.

However, well-defined classical problems that PL might help with.

**Question 5:** allow programmers to define quantum object/DS?

Allow direct modeling of quantum hardware components (QRAM, Sensors)

Consider quantum stack ~ truly quantum recursion ~ quantum apps
Verifying Quantum Programs: Scalability & Settings

**GAP:** the drawback of q. Hoare logic make existing verification schemes not **scalable**. Moreover, how about verification in more general settings?
Verifying Quantum Programs: Scalability & Settings

**GAP:** the drawback of q. Hoare logic make existing verification schemes not scalable. Moreover, how about verification in more general settings?

**Question 1:** how to make verification of quantum programs scalable?

Hard questions also for classical programs. Solutions for special cases.
Verifying Quantum Programs: Scalability & Settings

**GAP:** the drawback of q. Hoare logic make existing verification schemes not **scalable**. Moreover, how about verification in more general settings?

**Question 1:** how to make verification of quantum programs scalable?

Hard questions also for classical programs. Solutions for special cases.

Verification w/ **classical** machines:

*symbolic, abstract interpretation, or so, but certainly nontrivial!*
Verifying Quantum Programs: Scalability & Settings

**GAP:** the drawback of q. Hoare logic make existing verification schemes not **scalable**. Moreover, how about verification in more general settings?

**Question 1:** how to make verification of quantum programs scalable?

Hard questions also for classical programs. Solutions for special cases.

Verification w/ **classical** machines:

- *symbolic, abstract interpretation*, or so, but certainly nontrivial!

Verification w/ **quantum** machines:

- Largely unexplored! *Run-time verification* or other possibility?
Verifying Quantum Programs: Scalability & Settings

**GAP:** the drawback of q. Hoare logic make existing verification schemes not **scalable**. Moreover, how about verification in more general settings?

**Question 1:** how to make verification of quantum programs scalable?

Hard questions also for classical programs. Solutions for special cases.

**Verification w/ classical machines:**

*symbolic, abstract interpretation*, or so, but certainly nontrivial!

**Verification w/ quantum machines:**

*Largely unexplored!* **Run-time verification** or other possibility?

**Question 2:** how to do verification of quantum internet applications?

Quantum Internet/Communication is another recent interest
Verifying Quantum Programs: Scalability & Settings

**GAP:** the drawback of q. Hoare logic make existing verification schemes not **scalable**. Moreover, how about verification in more general settings?

**Question 1:** how to make verification of quantum programs scalable?

Hard questions also for classical programs. Solutions for special cases.

Verification w/ classical machines:

*symbolic, abstract interpretation, or so, but certainly nontrivial!*

Verification w/ quantum machines:

*Largely unexplored! Run-time verification or other possibility?*

**Question 2:** how to do verification of quantum internet applications?

Quantum Internet/Communication is another recent interest

Develop Q Hoare logic for **parallel, concurrent, distributed** programs.

*Some preliminary results exist. Essential difficulty exists due to quantum correlations.*
GAP: assertion-based debugging might in general distribute q. systems. Li et al. (OOPSLA 2020) provides projection-based assertion scheme, which in principle resolves the issue for capable quantum computers. How about NISQ?
Debugging Quantum Programs for NISQ

**GAP:** assertion-based debugging might in general distribute q. systems. Li et al. (OOPSLA 2020) provides *projection-based* assertion scheme, which in principle resolves the issue for capable quantum computers. How about NISQ?

**Question 3:** how to verify and debug NISQ applications?
Debugging Quantum Programs for NISQ

**GAP:** assertion-based debugging might in general distribute q. systems. Li et al. (OOPSLA 2020) provides *projection-based* assertion scheme, which in principle resolves the issue for capable quantum computers. How about NISQ?

**Question 3:** how to verify and debug NISQ applications?

Need to develop new frameworks as program features are simple

*e.g., only contains simple conditional and loops*
Debugging Quantum Programs for NISQ

**GAP:** assertion-based debugging might in general distribute q. systems. Li et al. (OOPSLA 2020) provides projection-based assertion scheme, which in principle resolves the issue for capable quantum computers. How about NISQ?

**Question 3:** how to verify and debug NISQ applications?

Need to develop new frameworks as program features are simple 

* e.g., only contains simple conditional and loops

Need to be very resilient to hardware errors

*For NISQ machines, all operations could be erroneous*
Debugging Quantum Programs for NISQ

**GAP:** assertion-based debugging might in general distribute q. systems. Li et al. (OOPSLA 2020) provides projection-based assertion scheme, which in principle resolves the issue for capable quantum computers. How about NISQ?

**Question 3:** how to verify and debug NISQ applications?

Need to develop new frameworks as program features are simple
e.g., only contains simple conditional and loops

Need to be very resilient to hardware errors

*For NISQ machines, all operations could be erroneous*

Need also to be scalable

*Classical simulation hard to scale; large q operations might contain more errors*
Debugging Quantum Programs for NISQ

**GAP:** assertion-based debugging might in general distribute q. systems. Li et al. (OOPSLA 2020) provides **projection-based** assertion scheme, which in principle resolves the issue for capable quantum computers. How about NISQ?

**Question 3:** how to verify and debug NISQ applications?

Need to develop new frameworks as program features are simple  
"e.g., only contains simple conditional and loops"

Need to be very resilient to hardware errors  
"For NISQ machines, all operations could be erroneous"

Need also to be scalable  
"Classical simulation hard to scale; large q operations might contain more errors"

Likely to be application-specific
Compilation of Quantum Application: Analog Machines

**GAP:** most of existing tool-chains compile to circuits with non-native gates on the hardware. Lead to very inefficient use of NISQ machines.
Compilation of Quantum Application: Analog Machines

**GAP:** most of existing tool-chains compile to circuits with non-native gates on the hardware. Lead to very inefficient use of NISQ machines.

**Question 1:** develop hardware-aware compilation?

Recent study suggests: compilation to *control pulses, qutrits*, or so
**Compilation of Quantum Application: Analog Machines**

**GAP:** most of existing tool-chains compile to circuits with non-native gates on the hardware. Lead to very inefficient use of NISQ machines.

**Question 1:** develop hardware-aware compilation?

Recent study suggests: compilation to *control pulses, qutrits*, or so examples identified, but no systematic study for e.g., *efficiency, and verification*

*Shi et al. Proceedings of the IEEE, Jun 2020*
Compilation of Quantum Application: Analog Machines

**GAP:** most of existing tool-chains compile to circuits with non-native gates on the hardware. Lead to very inefficient use of NISQ machines.

**Question 1:** develop hardware-aware compilation?

Recent study suggests: compilation to *control pulses, qutrits*, or so examples identified, but no systematic study for e.g., *efficiency*, and *verification*

*Shi et al. Proceedings of the IEEE, Jun 2020*

**Question 2:** direct compilation to analog / special purpose q machines?

Unexplored yet. But would be of great interests!
Compilation of Quantum Application: Analog Machines

**GAP:** most of existing tool-chains compile to circuits with non-native gates on the hardware. Lead to very inefficient use of NISQ machines.

**Question 1:** develop hardware-aware compilation?

Recent study suggests: compilation to *control pulses, qutrits*, or so examples identified, but no systematic study for e.g., *efficiency*, and *verification*  

*Shi et al. Proceedings of the IEEE, Jun 2020*

**Question 2:** direct compilation to analog / special purpose q machines?

Unexplored yet. But would be of great interests!

Classical Examples:  
Achour et al. (PLDI16)  
Achour & Rinard (ASPLOS 20)
ERROR
Nature

Quantum Error Correction
Fight Quantum Decoherence

ERROR
Approximate Computing & Quantum Computing

- General-purpose fault-tolerant quantum computers are *impractical* in the near term.
- *Near-term* practical quantum applications must focus on Noisy and Intermediate-Scale Quantum (NISQ) computers, where precisely controllable qubits are *expensive, error-prone, and scarce.*
Approximate Computing & Quantum Computing

• General-purpose fault-tolerant quantum computers are *impractical* in the near term.

• *Near-term* practical quantum applications must focus on Noisy and Intermediate-Scale Quantum (*NISQ*) computers, where precisely controllable quits are *expensive*, *error-prone*, and *scarce*.

**Goal:** reliable quantum programs with resource optimization!
Approximate Computing & Quantum Computing

- General-purpose fault-tolerant quantum computers are *impractical* in the near term.
- Near-term practical quantum applications must focus on Noisy and Intermediate-Scale Quantum (NISQ) computers, where precisely controllable quits are *expensive, error-prone, and scarce*.

**Goal: reliable quantum programs with resource optimization!**

- Quantitive guarantee on the reliability/accuracy of quantum programs based on specific hardware information.
Approximate Computing & Quantum Computing

• General-purpose fault-tolerant quantum computers are *impractical* in the near term.

• *Near-term* practical quantum applications must focus on Noisy and Intermediate-Scale Quantum (NISQ) computers, where precisely controllable quits are *expensive, error-prone, and scarce*.

**Goal:** reliable quantum programs with resource optimization!

• Quantitative guarantee on the reliability/accuracy of quantum programs based on specific hardware information.

• High-level abstraction of error-handling primitives in quantum programs.
Approximate Computing & Quantum Computing

- General-purpose fault-tolerant quantum computers are *impractical* in the near term.
- *Near-term* practical quantum applications must focus on Noisy and Intermediate-Scale Quantum (NISQ) computers, where precisely controllable qubits are *expensive*, *error-prone*, and *scarce*.

**Goal: reliable quantum programs with resource optimization!**

- Quantitative guarantee on the reliability/accuracy of quantum programs based on specific hardware information.
- High-level abstraction of error-handling primitives in quantum programs.
- Automatic error-resource-optimization on a per-program basis!
Methodology
Methodology

• Elevate the handling of errors to the level of programming language.
Methodology

• Elevate the handling of errors to the level of programming language.
• Reason *reliability/accuracy* of quantum programs via *static* analysis.
Methodology

• Elevate the handling of errors to the level of programming language.
• Reason reliability/accuracy of quantum programs via static analysis.
• Conduct resource optimization via code synthesis of quantum programs.
Methodology

• Elevate the handling of errors to the level of programming language.
• Reason reliability/accuracy of quantum programs via static analysis.
• Conduct resource optimization via code synthesis of quantum programs.

An important classical tool: approximate computing!
Methodology

• Elevate the handling of errors to the level of programming language.
• Reason reliability/accuracy of quantum programs via static analysis.
• Conduct resource optimization via code synthesis of quantum programs.

An important classical tool: approximate computing!

• Return possibly inaccurate/approximate results!
Methodology

- Elevate the handling of errors to the level of programming language.
- Reason reliability/accuracy of quantum programs via static analysis.
- Conduct resource optimization via code synthesis of quantum programs.

An important classical tool: approximate computing!

- Return possibly inaccurate/approximate results!
  - unreliable hardware
Methodology

• Elevate the handling of errors to the level of programming language.
• Reason reliability/accuracy of quantum programs via static analysis.
• Conduct resource optimization via code synthesis of quantum programs.

An important classical tool: approximate computing!

• Return possibly inaccurate/approximate results!
  - unreliable hardware
  - limited computational resource
Methodology

- Elevate the handling of errors to the level of programming language.
- Reason reliability/accuracy of quantum programs via static analysis.
- Conduct resource optimization via code synthesis of quantum programs.

An important classical tool: approximate computing!

- Return possibly inaccurate/approximate results!
  - unreliable hardware
  - limited computational resource
- Good when approximate results are sufficient for applications!
Methodology

• Elevate the handling of errors to the level of programming language.
• Reason reliability/accuracy of quantum programs via static analysis.
• Conduct resource optimization via code synthesis of quantum programs.

An important classical tool: approximate computing!

• Return possibly inaccurate/approximate results!
  - unreliable hardware
  - limited computational resource
• Good when approximate results are sufficient for applications!
  - vision, machine learning; also with guarantees for critical data
Methodology

• Elevate the handling of errors to the level of programming language.
• Reason reliability/accuracy of quantum programs via static analysis.
• Conduct resource optimization via code synthesis of quantum programs.

An important classical tool: approximate computing!

• Return possibly inaccurate/approximate results!
  - unreliable hardware
  - limited computational resource
• Good when approximate results are sufficient for applications!
  - vision, machine learning; also with guarantees for critical data
• Various techniques developed in classical PL literature.
Overview

Software Developers

- Exact Program
- Reliability/Accuracy Specification

Hardware Designer

- Approximate Hardware Specification

Reliability/Accuracy Constraint Generator

error handling primitives

Resource Optimization Objective Generator

Back-end Optimizer

Neural-based Code Synthesizer

Reliable Quantum Programs with Optimal Resources
Overview

Software Developers

- Exact Program
- Reliability/Accuracy
- Constraint Generator

Hardware Designer

- Approximate Hardware
- Specification

Reliability/Accuracy
Constraint Generator

error handling primitives

Resource Optimization
Objective Generator

Back-end Optimizer

Neural-based Code Synthesizer

Reliable Quantum Programs with Optimal Resources

a basic framework in POPL 19
Nature

Quantum Error Correction
Fight
Quantum Decoherence

Human

Intel Pentium FPU error

Ariane 5

MCAS safety system engages

Horizontal tail
Nose down
Being careful cannot solve the human error problem in either classical or quantum.

**Quantum case**: Significantly More **CHALLENGING** than Classical
- standard software assurance techniques, e.g., black-box / unit test, expensive in q.
- quantum mechanics prohibits certain testing, e.g., assertions
Human Errors in Quantum Software Engineering

Being careful cannot solve the human error problem in either classical or quantum.

Quantum case: Significantly More CHALLENGING than Classical
- standard software assurance techniques, e.g., black-box / unit test, expensive in q.
- quantum mechanics prohibits certain testing, e.g., assertions

Reality: testing in quantum today

confirming the circuit by observation…. not scalable…
Human Errors in Quantum Software Engineering

Being careful cannot solve the human error problem in either classical or quantum.

Quantum case: Significantly More CHALLENGING than Classical
- standard software assurance techniques, e.g., black-box / unit test, expensive in q.
- quantum mechanics prohibits certain testing, e.g., assertions

Reality: testing in quantum today

QISKIT Compiler ERRORs
Much HARDER to detect!
Serious Consequences!

confirming the circuit by observation.... not scalable...
Human Errors in Quantum Software Engineering

Being careful cannot solve the human error problem in either classical or quantum.

**Quantum case**: Significantly More **CHALLENGING** than Classical

- standard software assurance techniques, e.g., black-box / unit test, expensive in q.
- quantum mechanics prohibits certain testing, e.g., assertions

**Reality**: testing in quantum today

confirming the circuit by observation…. not scalable…

**QISKIT Compiler ERRORs**

Much **HARDER** to detect!

Serious Consequences!

**WARNING**

Similar Concerns in classical !

More **SERIOUS** in quantum !
The Verifying Compiler: A Grand Challenge for Computing Research

TONY HOARE

Microsoft Research Ltd., Cambridge, UK

Certified software: a solution to validation of q. software

The Verifying Compiler: A Grand Challenge for Computing Research

TONY HOARE

Microsoft Research Ltd., Cambridge, UK

The Verifying Compiler: A Grand Challenge for Computing Research

TONY HOARE

Microsoft Research Ltd., Cambridge, UK


Certified software: a solution to validation of q. software

GCC : many bugs in software testing
CompCert: a certified “GCC”, bug-free
Certified software: a solution to validation of q. software

(1) Ensure correctness of code by construction.
(2) Scalability for quantum based on symbolic proofs.
**VOQC**: a first step towards a fully certified quantum compiler.

**SQIRE**: a simple quantum intermediate-representation embedded in Coq.

Our infrastructure powerful enough:

an end-to-end implementation of **Shor's algorithm** & its correctness proof.
About Today’s Tutorial:

Goal: Some Basic Quantum Computing & PL + References

(1) Introduction to Quantum Computing and Potential Roles of Programming Languages (25 min + 5 Q & A)

(2) A Mini-Course of Quantum Hoare Logic on Quantum While Language (30 min + 5 Q & A)

(3) Discussion on existing and potential Programming Language research opportunities (20 min + 5 Q & A)

Reference: tutorial slides and some references are available at https://www.cs.umd.edu/~xwu/mini_lib.html