Quantum Floyd-Hoare Verification and its Implementations

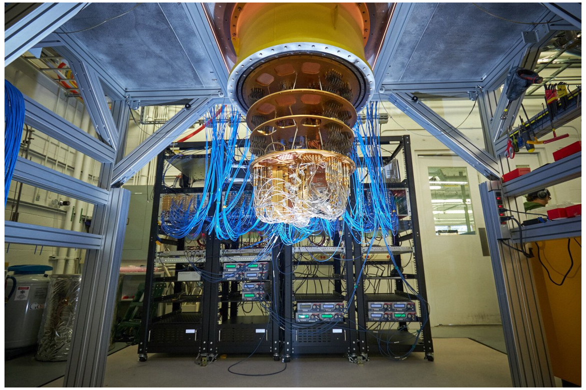
Samantha Norrie

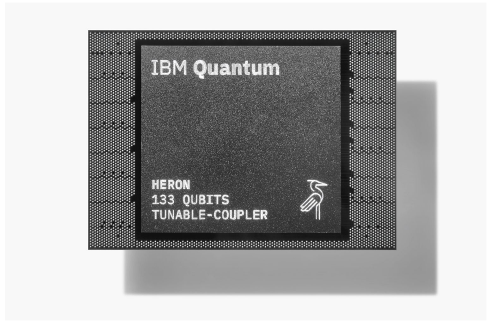
Quantum Computing Introduction

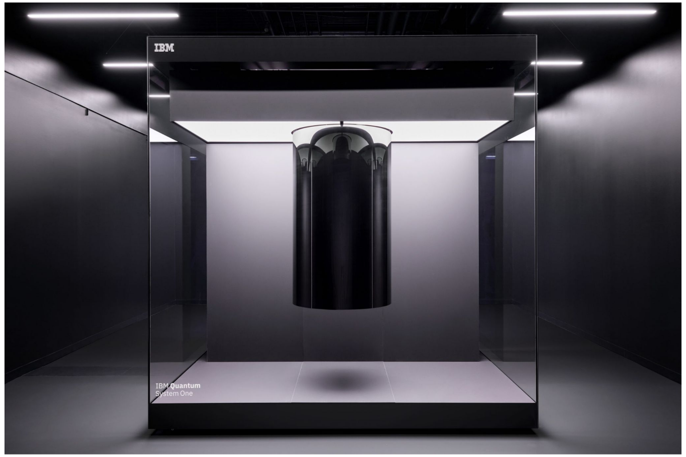
What is Quantum Computing?

Quantum Computing is a multidisciplinary field that focuses on harnessing the power of Quantum Mechanics to compute and solve problems that have been thought to be unsolvable []. Some of the fields involved in Quantum Computing include Physics, Biology, Chemistry, Computer Science, Software Engineering, Electrical Engineering, and Mechanical Engineering.

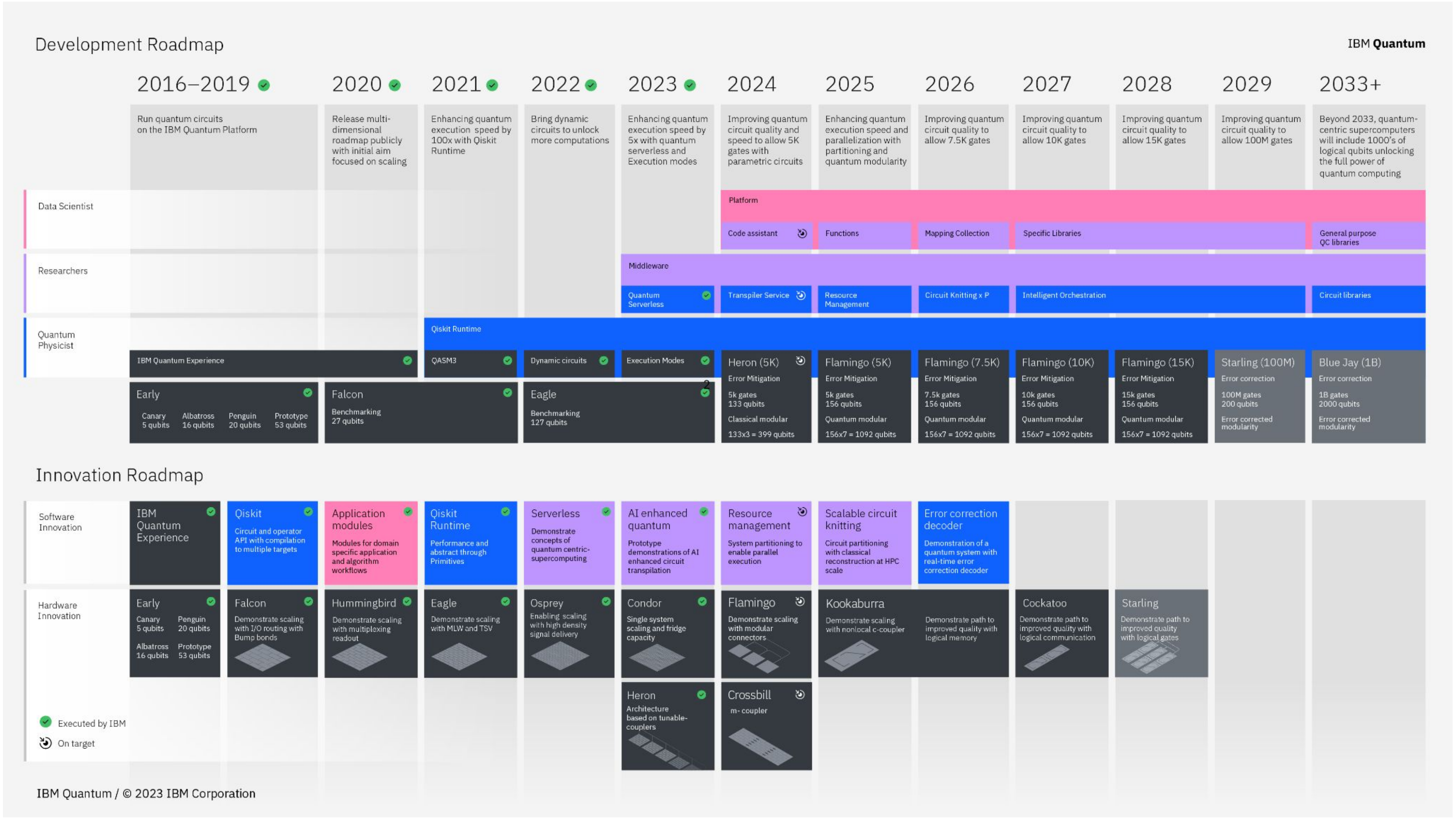
The Computer Science and Software Engineering research within Quantum Computing largely revolves around the creation and implementation of Quantum algorithms (which often are inspired by classical algorithms) as well as the creation of fault-tolerant Quantum Software.







[1]



Development Roadmap

IBM **Quantum**

2016–2019 ✓	2020 ✓	2021 ✓	2022 ✓	2023 ✓	2024	2025	2026	2027	2028	2029	2033+
Run quantum circuits on the IBM Quantum Platform	Release multi-dimensional roadmap publicly with initial aim focused on scaling	Enhancing quantum execution speed by 100x with Qiskit Runtime	Bring dynamic circuits to unlock more computations	Enhancing quantum execution speed by 5x with quantum serverless and Execution modes	Improving quantum circuit quality and speed to allow 5K gates with parametric circuits	Enhancing quantum execution speed and parallelization with partitioning and quantum modularity	Improving quantum circuit quality to allow 7.5K gates	Improving quantum circuit quality to allow 10K gates	Improving quantum circuit quality to allow 15K gates	Improving quantum circuit quality to allow 100M gates	Beyond 2033, quantum-centric supercomputers will include 1000's of logical qubits unlocking the full power of quantum computing

Data Scientist

- Platform
  - Code assistant (On target)
  - Functions
  - Mapping Collection
  - Specific Libraries
  - General purpose QC libraries

Researchers

- Middleware
  - Quantum Serverless ✓
  - Transpiler Service (On target)
  - Resource Management
  - Circuit Knitting x P
  - Intelligent Orchestration
  - Circuit libraries

Quantum Physicist

- Qiskit Runtime
  - IBM Quantum Experience ✓
  - QASM3 ✓
  - Dynamic circuits ✓
  - Execution Modes ✓

Processor	Details
Early ✓	Canary 5 qubits; Albatross 16 qubits; Penguin 20 qubits; Prototype 53 qubits
Falcon ✓	Benchmarking 27 qubits
Eagle ✓	Benchmarking 127 qubits
Heron (5K) (On target)	Error Mitigation; 5k gates 133 qubits; Classical modular; 133x3 = 399 qubits
Flamingo (5K)	Error Mitigation; 5k gates 156 qubits; Quantum modular; 156x7 = 1092 qubits
Flamingo (7.5K)	Error Mitigation; 7.5k gates 156 qubits; Quantum modular; 156x7 = 1092 qubits
Flamingo (10K)	Error Mitigation; 10k gates 156 qubits; Quantum modular; 156x7 = 1092 qubits
Flamingo (15K)	Error Mitigation; 15k gates 156 qubits; Quantum modular; 156x7 = 1092 qubits
Starling (100M)	Error correction; 100M gates 200 qubits; Error corrected modularity
Blue Jay (1B)	Error correction; 1B gates 2000 qubits; Error corrected modularity

Innovation Roadmap

Software Innovation

- IBM Quantum Experience ✓
- Qiskit ✓ — Circuit and operator API with compilation to multiple targets
- Application modules ✓ — Modules for domain specific application and algorithm workflows
- Qiskit Runtime ✓ — Performance and abstract through Primitives
- Serverless ✓ — Demonstrate concepts of quantum centric-supercomputing
- AI enhanced quantum ✓ — Prototype demonstrations of AI enhanced circuit transpilation
- Resource management (On target) — System partitioning to enable parallel execution
- Scalable circuit knitting — Circuit partitioning with classical reconstruction at HPC scale
- Error correction decoder — Demonstration of a quantum system with real-time error correction decoder

Hardware Innovation

- Early ✓ — Canary 5 qubits; Penguin 20 qubits; Albatross 16 qubits; Prototype 53 qubits
- Falcon ✓ — Demonstrate scaling with I/O routing with Bump bonds
- Hummingbird ✓ — Demonstrate scaling with multiplexing readout
- Eagle ✓ — Demonstrate scaling with MLW and TSV
- Osprey ✓ — Enabling scaling with high density signal delivery
- Condor ✓ — Single system scaling and fridge capacity
- Heron ✓ — Architecture based on tunable-couplers
- Flamingo (On target) — Demonstrate scaling with modular connectors
- Crossbill (On target) — m- coupler
- Kookaburra — Demonstrate scaling with nonlocal c-coupler; Demonstrate path to improved quality with logical memory
- Cockatoo — Demonstrate path to improved quality with logical communication
- Starling — Demonstrate path to improved quality with logical gates

✓ Executed by IBM

On target

IBM Quantum / © 2023 IBM Corporation

[1]

Noisy Intermediate Scale Quantum Era = NISQ Era

Qiskit

Elements for building a quantum future

[2]

Qubits

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

The bit equivalent for Quantum Computers

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

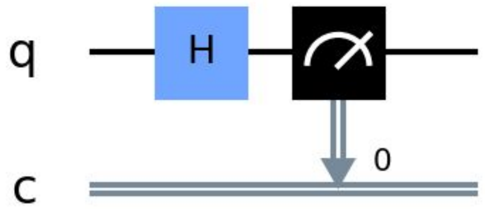
A qubit can exist as 0, 1, or as a state in between 0 and 1

[3]

Superposition

A qubit is in superposition when it is neither completely $|0\rangle$ or $|1\rangle$

Measuring a qubit takes it out of superposition and forces it to return a classical value (0 or q)

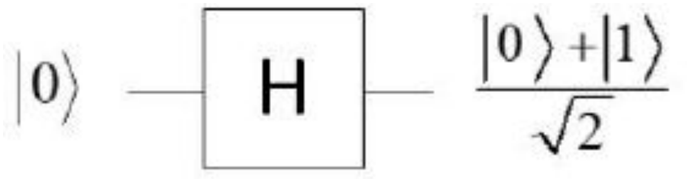


The application of the Hadamard gate to a single qubit, which is then measured to a classical bit

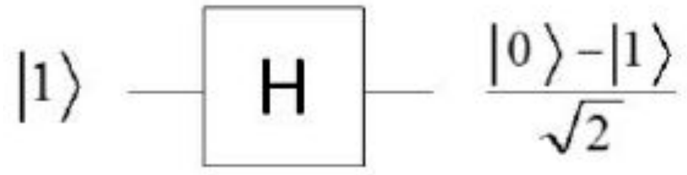
[3]

The Hadamard Gate: One of the stars of Quantum Computing

Puts a qubit into equal superposition



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

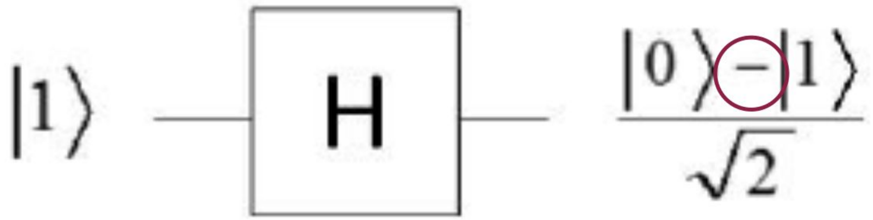


remember $|\alpha|^2 + |\beta|^2 = 1.$

[3]

Why the negative?

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



Denotes a negative **relative phase** (out of scope for today's presentation)!

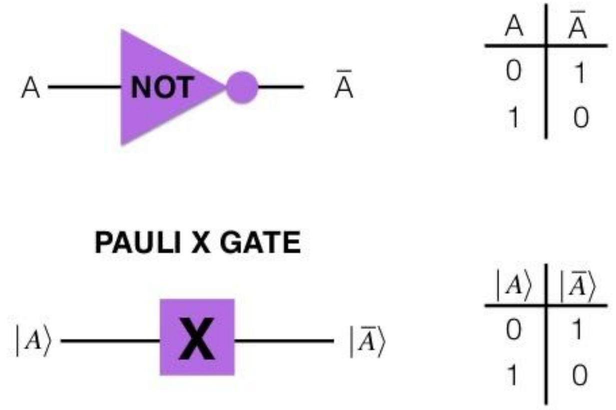
It is important to note that the negative relative phase does not impact measurement probabilities due to

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

[3]

The NOT (Pauli X) gate

Just like a classical NOT gate

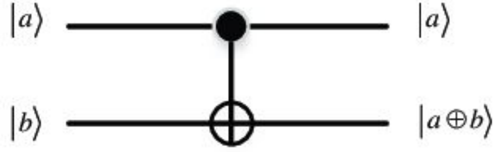


A	$\bar{A}$
0	1
1	0

$\lvert A\rangle$	$\lvert\bar{A}\rangle$
0	1
1	0

The CNOT (Controlled Not) Gate

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$



[3]

Entanglement and Bell States

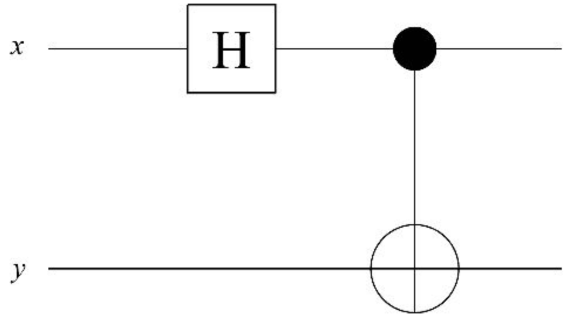
When qubits are entangled, the measurement of one instantly reveals the state of another

Bell States are simple two-qubit examples of entangled states

They only have two possible states

There are **four** two-qubit Bell states

Jupyter Notebook Exercise 1



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

One of the Bell States

[3]

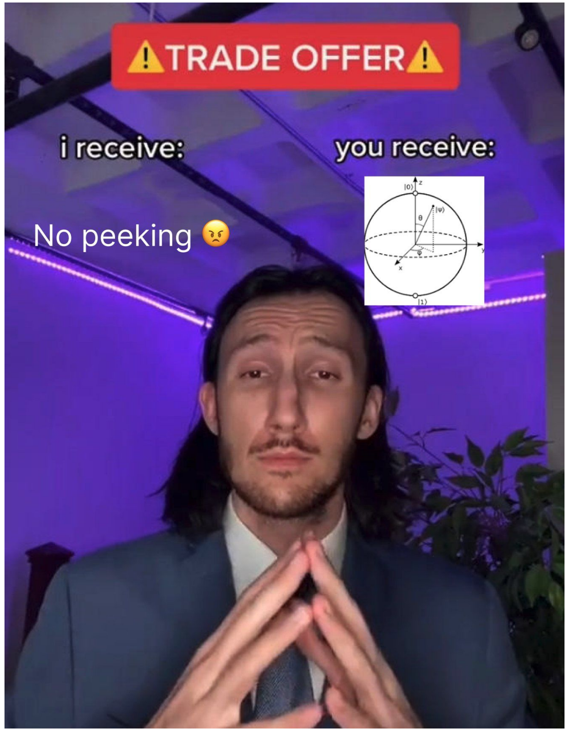
Entanglement

When qubits are entangled, the measurement of one instantly reveals the state of another

Quantum Computing Con: No Peeking

Measuring a qubit immediately returns a binary value, taking the qubit and any qubits entangled to it out of superposition

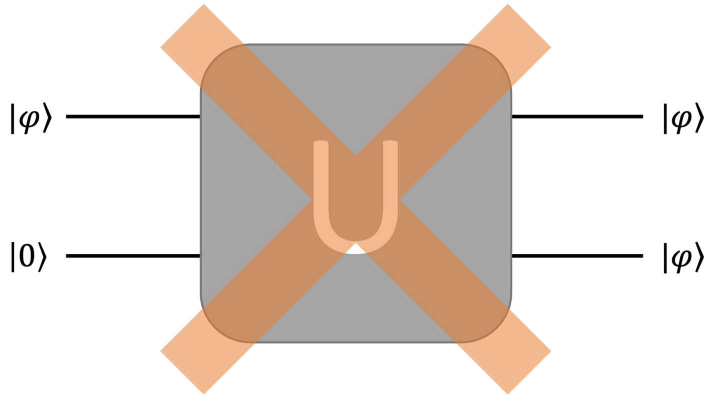
This can be referred to as the quantum measurement problem or the collapse of the wave function (useful when talking about quantum from a Quantum Mechanics perspective)



Quantum Computing Con: No-Cloning Theorem

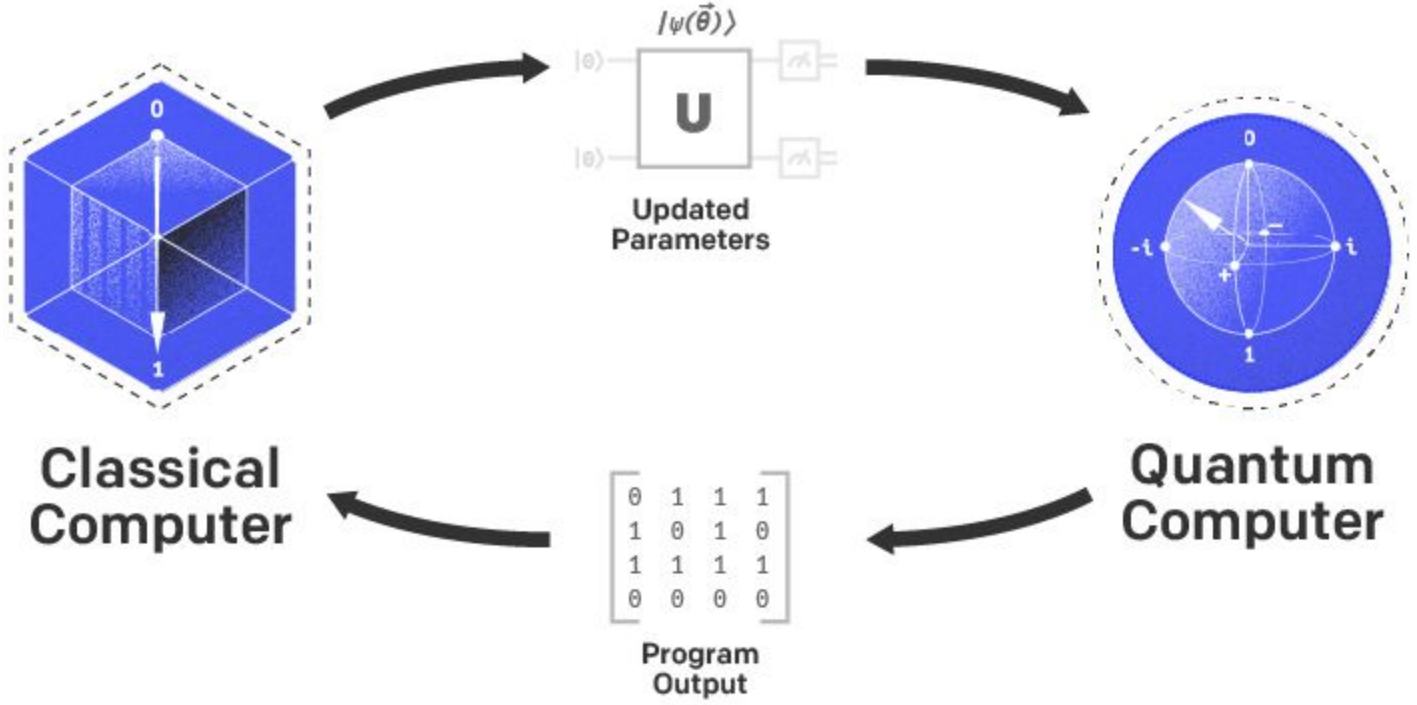
Unlike in classical computing, a value cannot be copied from one qubit to another. This is referred to as the *No-Cloning Theorem*

It is important to note that a qubit's value can be teleported, that is the value can be moved to another qubit, but the value at the original qubit will be destroyed.



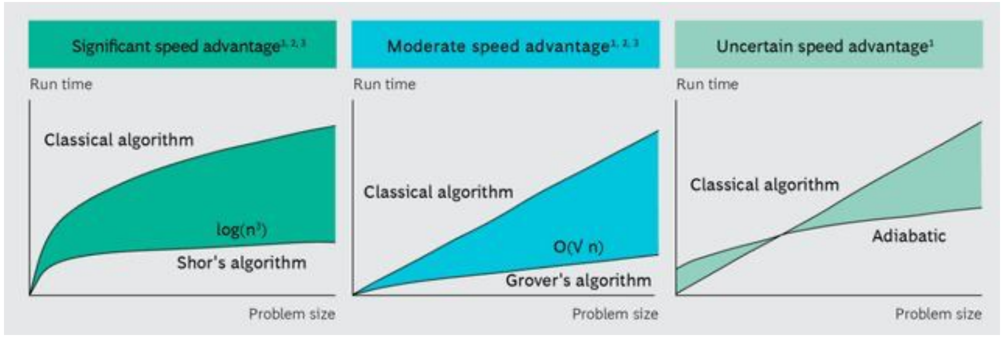
No-Cloning Theorem

[4]


$|\psi(\vec{\theta})\rangle$
$|0\rangle$
$|0\rangle$
U
Updated Parameters
0
1
-i
i
+
-
Classical Computer
Quantum Computer
0 1 1 1
1 0 1 0
1 1 1 1
0 0 0 0
Program Output

[5]

Quantum Computing Pros



Significant algorithmic speedup, leading to advancements in cryptography, machine learning, and chemistry.

[6]

Classical Floyd–Hoare Verification: A Review

Floyd-Hoare Verification aims to prove correctness by using different rules

1. The assignment axiom **{P[x/e]} x = e {P}**
2. The sequential composition rule **{P} C1 {Q}, {Q} C2 {R} => {P} C1; C2 {R}**
3. The conditional rule **{P ∧ B} C1 {Q}, {P ∧ ¬B} C2 {Q} => {P} if B then C1 else C2 {Q}**
4. The while loop rule {P ∧ B} C {P} => {P} while B do C {P ∧ ¬B}
5. The consequence rule P' ⇒ P, {P} C {Q}, Q ⇒ Q' => {P'} C {Q'}

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

If the precondition (P) is true before the execution of program ©, then postcondition (Q) will be met []

So, how does Quantum Floyd-Hoare Verification differ from Classical Floyd-Hoare Verification?

Quantum Floyd-Hoare is new and evolving!

Many new definitions have been being created as Quantum Computing has started becoming more and more popular

Quantum Floyd–Hoare Verification Version 1

Proposed by Mingsheng Ying

Notation

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

Notation

New letter! S is used to denote quantum program

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

[7]

Notation

New letter! S is used to denote quantum program

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

In order for classical Floyd-Hoare to work for quantum programs, the notions of weakest precondition and weakest liberal precondition must be used in order to ensure relative completeness.

Notation

New letter! S is used to denote quantum program

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

In order for classical Floyd-Hoare to work for quantum programs, the notions of weakest precondition and weakest liberal precondition must be used in order to ensure relative completeness.

Weakest precondition: defines what must be true in order for postcondition to be satisfied using a *minimal* number of requirements
Weakest liberal precondition: similar to the weakest precondition, only it accounts for non-deterministic behaviour

[7]

Rules

$(Skip)$ $$\frac{}{\langle \textbf{skip}, \rho\rangle \to \langle E, \rho\rangle}$$

$(Initialization)$ $$\frac{}{\langle q := 0, \rho\rangle \to \langle E, \rho_0^q\rangle}$$

$(Sequential\ Composition)$ $$\frac{\langle S_1, \rho\rangle \to \langle S_1', \rho'\rangle}{\langle S_1; S_2, \rho\rangle \to \langle S_1'; S_2, \rho\rangle}$$

$(Measurement)$ $$\frac{}{\langle \textbf{measure}\ M[\overline{q}] : \overline{S}, \rho\rangle \to \langle S_m, M_m \rho M_m^\dagger\rangle}$$

$(Loop\ 0)$ $$\frac{}{\langle \textbf{while}\ M[\overline{q}] = 1\ \textbf{do}\ S, \rho\rangle \to \langle E, M_0 \rho M_0^\dagger\rangle}$$

$(Loop\ 1)$ $$\frac{}{\langle \textbf{while}\ M[\overline{q}] = 1\ \textbf{do}\ S, \rho\rangle \to \langle S; \textbf{while}\ M[\overline{q}] = 1\ \textbf{do}\ S, M_1 \rho M_1^\dagger\rangle}$$

[7]

Rules

(*Skip*)
$$\frac{}{\langle \mathbf{skip}, \rho\rangle \rightarrow \langle E, \rho\rangle}$$

Qubits are always initialized to zero

(*Initialization*)
$$\frac{}{\langle q := 0, \rho\rangle \rightarrow \langle E, \rho_0^q\rangle}$$

(*Sequential Composition*)
$$\frac{\langle S_1, \rho\rangle \rightarrow \langle S_1', \rho'\rangle}{\langle S_1; S_2, \rho\rangle \rightarrow \langle S_1'; S_2, \rho\rangle}$$

Classical measurement

(*Measurement*)
$$\frac{}{\langle \mathbf{measure}\ M[\overline{q}] : \overline{S}, \rho\rangle \rightarrow \langle S_m, M_m \rho M_m^\dagger\rangle}$$

Classical measurement

(*Loop 0*)
$$\frac{}{\langle \mathbf{while}\ M[\overline{q}] = 1\ \mathbf{do}\ S, \rho\rangle \rightarrow \langle E, M_0 \rho M_0^\dagger\rangle}$$

(*Loop 1*)
$$\frac{}{\langle \mathbf{while}\ M[\overline{q}] = 1\ \mathbf{do}\ S, \rho\rangle \rightarrow \langle S; \mathbf{while}\ M[\overline{q}] = 1\ \mathbf{do}\ S, M_1 \rho M_1^\dagger\rangle}$$

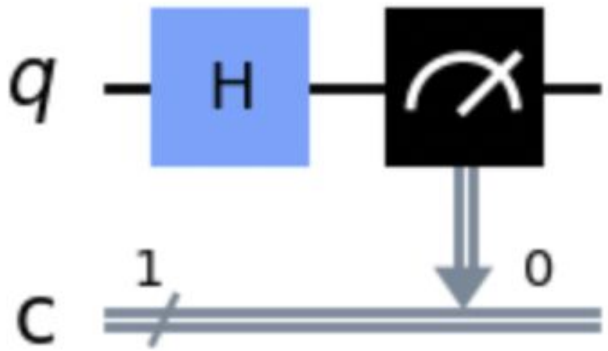
[7]

A Quick note on types

In Ying’s Quantum-Floyd Hoare Verification, variables can be of type Boolean or Integer

I won’t be using integers in today’s examples, but they will be briefly mentioned later

An example using a Hadamard gate



$$S \equiv q := 0; q := Hq; \textbf{measure}\ M[q] : \overline{S},$$

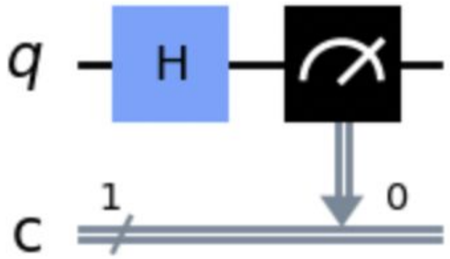
Good precondition and postcondition?

Assert that the quantum state is valid *before* and *after* the quantum program (S) has executed.

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

[7]

An example using a Hadamard gate



$$S \equiv q := 0; q := Hq; \mathbf{measure}\ M[q] : \overline{S},$$

$$\begin{aligned}
\langle S, \rho\rangle &\to \langle q := Hq; \mathbf{measure}, \rho\rangle \\
&\to \left\langle \mathbf{measure}, |+\rangle_q \left\langle +| \otimes \bigotimes_{q'\neq q} |0\right\rangle_{q'} \langle 0|\right\rangle \\
&\to \begin{cases} \left\langle S_0, \frac{1}{2}|0\right\rangle_q \langle 0| \otimes \bigotimes_{q'\neq q} |0\rangle_{q'}\langle 0|\rangle \to \left\langle E, \frac{1}{2}\rho\right\rangle, \\ \left\langle S_1, \frac{1}{2}|1\right\rangle_q \langle 1| \otimes \bigotimes_{q'\neq q} |0\rangle_{q'}\langle 0|\rangle \to \left\langle E, \frac{1}{2}\rho\right\rangle, \end{cases}
\end{aligned}$$

S = our quantum state (our quantum program)
p = the current state(s) of our quantum state

[7]

An example using a Hadamard gate

$$\langle S, \rho \rangle \rightarrow \langle q := Hq; \textbf{measure}, \rho \rangle$$

$$\rightarrow \langle \textbf{measure}, |+\rangle_q \left\langle +| \otimes \bigotimes_{q' \neq q} |0 \right\rangle_{q'} \langle 0| \rangle$$

$$\rightarrow \begin{cases} \left\langle S_0, \tfrac{1}{2}|0\right\rangle_q \langle 0| \otimes \bigotimes_{q' \neq q} |0\rangle_{q'} \langle 0| \rangle \rightarrow \left\langle \boldsymbol{E}, \tfrac{1}{2}\rho \right\rangle, \\ \left\langle S_1, \tfrac{1}{2}|1\right\rangle_q \langle 1| \otimes \bigotimes_{q' \neq q} |0\rangle_{q'} \langle 0| \rangle \rightarrow \left\langle \boldsymbol{E}, \tfrac{1}{2}\rho \right\rangle, \end{cases}$$

S = our quantum state (our quantum program)
p = the current state(s) of our quantum state

[7]

An example using a Hadamard gate

$$\langle S, \rho\rangle \rightarrow \langle q := Hq; \mathbf{measure}, \rho\rangle$$

|0>H

$$\rightarrow \langle \mathbf{measure}, |+\rangle_q \qquad \rangle$$

$|\alpha|^2$

$$\rightarrow \begin{cases} \left\langle S_0, \frac{1}{2}|0\rangle_q \qquad \right\rangle \rightarrow \left\langle \boldsymbol{E}, \frac{1}{2}\rho\right\rangle, \\ \left\langle S_1, \frac{1}{2}|1\rangle_q \qquad \right\rangle \rightarrow \left\langle \boldsymbol{E}, \frac{1}{2}\rho\right\rangle, \end{cases}$$

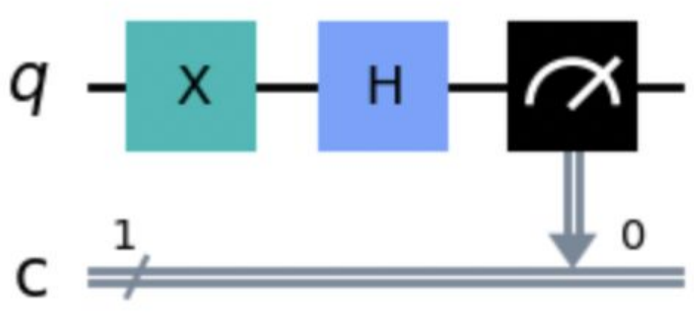
$|\beta|^2$

S = our quantum state (our quantum program)
p = the current state(s) of our quantum state

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

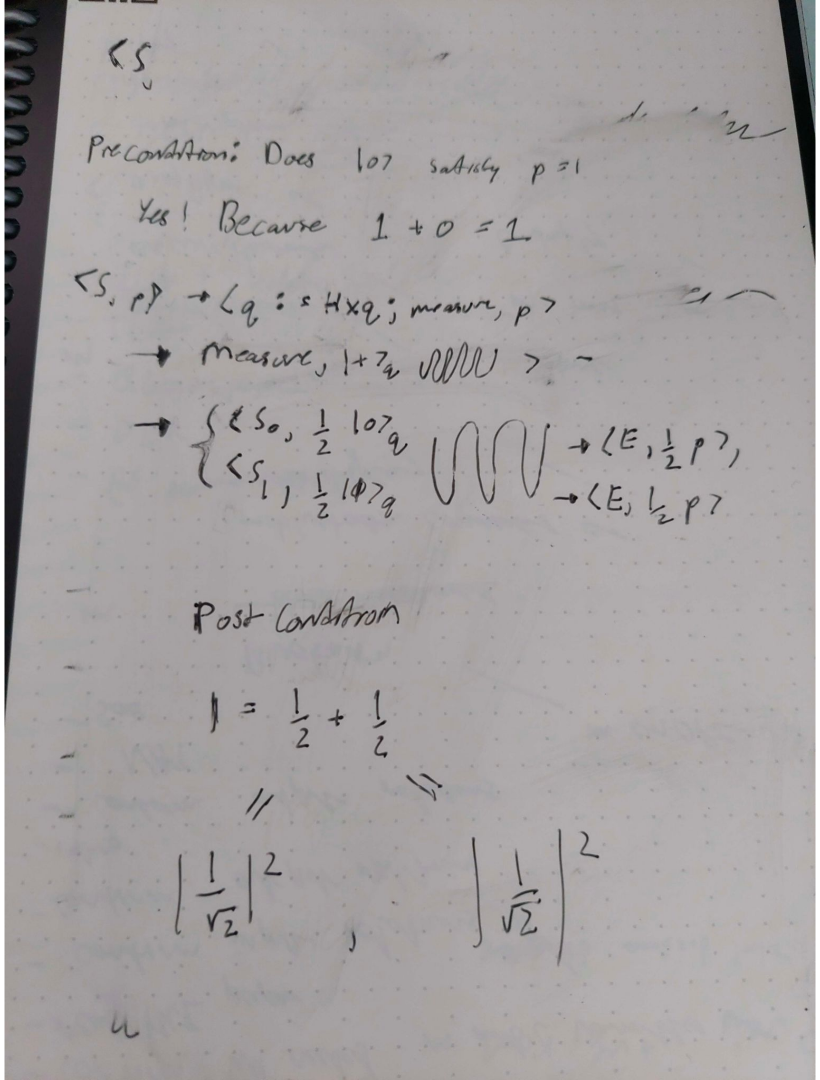
[7]

Exercise Two



Hint: | - > = | 1 > H

My solution



$\langle S$

Precondition: Does $|0\rangle$ satisfy $p = 1$

Yes! Because $1 + 0 = 1$

$\langle S, p\rangle \rightarrow \langle q :\!= H \times q;\ \text{measure}, p\rangle$

$\rightarrow \langle \text{measure}, |+\rangle_q \rangle$

$\rightarrow \begin{cases} \langle S_0, \frac{1}{2} |0\rangle_q \rightarrow \langle E, \frac{1}{2} p\rangle, \\ \langle S_1, \frac{1}{2} |\phi\rangle_q \rightarrow \langle E, \frac{1}{2} p\rangle \end{cases}$

Post Condition

$1 = \frac{1}{2} + \frac{1}{2}$

$\left|\frac{1}{\sqrt{2}}\right|^2, \quad \left|\frac{1}{\sqrt{2}}\right|^2$

Is Ying’s Quantum Floyd-Hoare Verification correct?

It can be both partially correct and totally correct.

Any quantum program using Ying’s Quantum Floyd-Hoare Verification will be partially correct (complete and sound).

What distinguishes whether it is partially correct or totally correct is the while loop rule used.

In order for the formal method to be totally correct, the while loop rule must be bounded by the number of iterations used.

[7]

Mini example

Using the Born rule!

Quantum Floyd–Hoare Verification with Ghost variables

Proposed by Dominique Unruh

$$\mathfrak{c},\mathfrak{d} ::= \underline{\textbf{apply}}\ U\ \underline{\textbf{to}}\ \mathbf{X} \mid \underline{\textbf{init}}\ \mathrm{x} \mid \underline{\textbf{if}}\ \mathrm{y}\ \underline{\textbf{then}}\ \mathfrak{c}\ \underline{\textbf{else}}\ \mathfrak{d} \mid \underline{\textbf{while}}\ \mathrm{y}\ \underline{\textbf{do}}\ \mathfrak{c} \mid \mathfrak{c};\mathfrak{d} \mid \underline{\textbf{skip}}$$

But first, the syntax being used

What are ghost variables

Ghost variables are variables that are only visible in the precondition and the postcondition. They are often referred to as ‘ghosts’.

Program variables are variables that are visible in the precondition, program, and postcondition

X = y^3

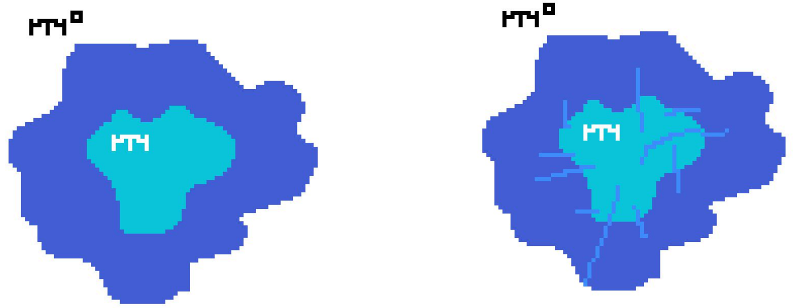
With classical Floyd-Hoare verification, we say that the postcondition above is only true if x and y satisfy it *and* that they are both **program variables**

With ghost variables, the postcondition is satisfied if it is possible to assign any value (ex. Of natural numbers, integers, .etc).

Isn’t this too vague?

In the classical case, yes...

But ghosts can be used to help describe quantum system



M = programming variable set

M˙ = programming and ghost variable set (called ‘mixed memory’)

Programming and ghost variables can be entangled, but the decision to allow them to be entangled or not changes the evaluation process.

[8]

Quantum Floyd–Hoare with Hoare Type Theory

Proposed by Kartik Singhal

What is Hoare Type Theory?

Even the Hoare Triple has a type. It is referred to as a *constructor.* x is returned from the system. A is x's type.

Hoare Type Theory asserts that each term in a system has a type

$$\begin{tikzcd}
\{P\}\ c\ \{Q\} \arrow[d] \\
\Delta.\{P\}\ x : A\ \{Q\}
\end{tikzcd}$$

[9]

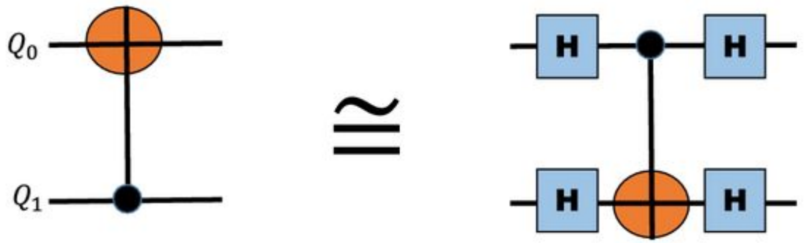
A few rules before we go any further

- We must assume that we are working in a universe with countably infinite qubits
- We must assume that only one-qubit and two-qubit gates exist in our world

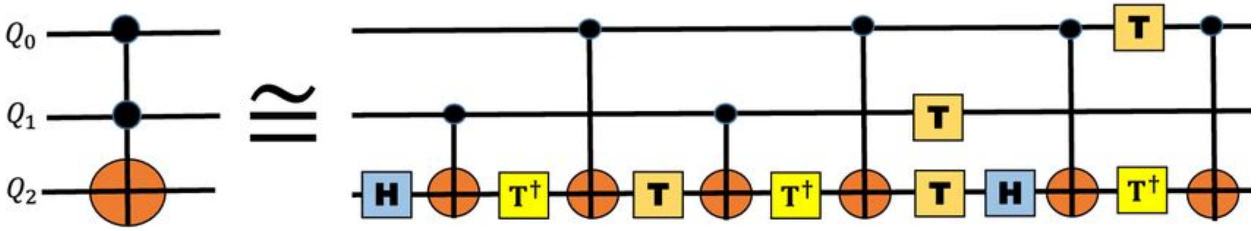
Is the latter a major issue? Not particularly! The current quantum space is very much like this.

[9]

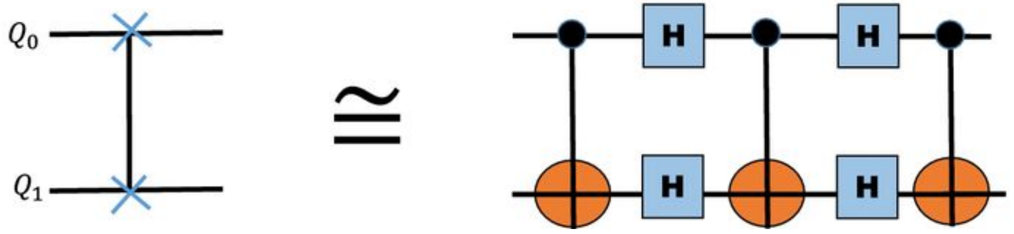
(a)


Q_0
Q_1
$\cong$
H
H
H
H

(b)


Q_0
Q_1
Q_2
$\cong$
H
$T^\dagger$
T
$T^\dagger$
T
T
H
T
$T^\dagger$

(c)


Q_0
Q_1
$\cong$
H
H
H
H

[10]

Notation for Floyd–Hoare Verification with Hoare Type Theory

(Inspired by Unruh and F*)

Some functions and symbols

- class(X) returns true if X holds a classical state
- separable(X) returns true if X is separable (i.e. that it is not entangled with another system)
- T corresponds to the complete state space (when used as a boolean, it always evaluates to true)

Initialization Notation

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

[9]

Initialization Notation

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

Evaluates to true. We can always initialize a qubit in our universe of countably infinite qubits

A qubit with the classical value of b will always be created

Measurement Notation

a ghost variable

$$\begin{array}{l}
\texttt{meas : (q: qbit)} \rightarrow \texttt{QST (x: bit)} \\
\quad \{\psi\texttt{: vector, e}_x\texttt{: qbit}\} \\
\quad \texttt{(requires } \{(\texttt{q} \otimes \mathscr{H}\texttt{[V\textbackslash q]}) =_q \psi\}) \\
\quad \texttt{(ensures } \{\texttt{class(q)} \wedge (\texttt{e}_x \otimes \mathscr{H}\texttt{[V\textbackslash q]}) =_q \psi\})
\end{array}$$

Asserts that q exists in our quantum space

q is measured to a classic value

Ghost variable is used to check that

Measurement Notation

```
meas : (q: qbit) $\to$ QST (x: bit)
                           {$\psi$: vector, e$_x$: qbit}
                           (requires {(q $\otimes \mathscr{H}$[V\q]) $=_q$ $\psi$})
                           (ensures  {class(q) $\wedge$ (e$_x$ $\otimes \mathscr{H}$[V\q]) $=_q$ $\psi$})
```

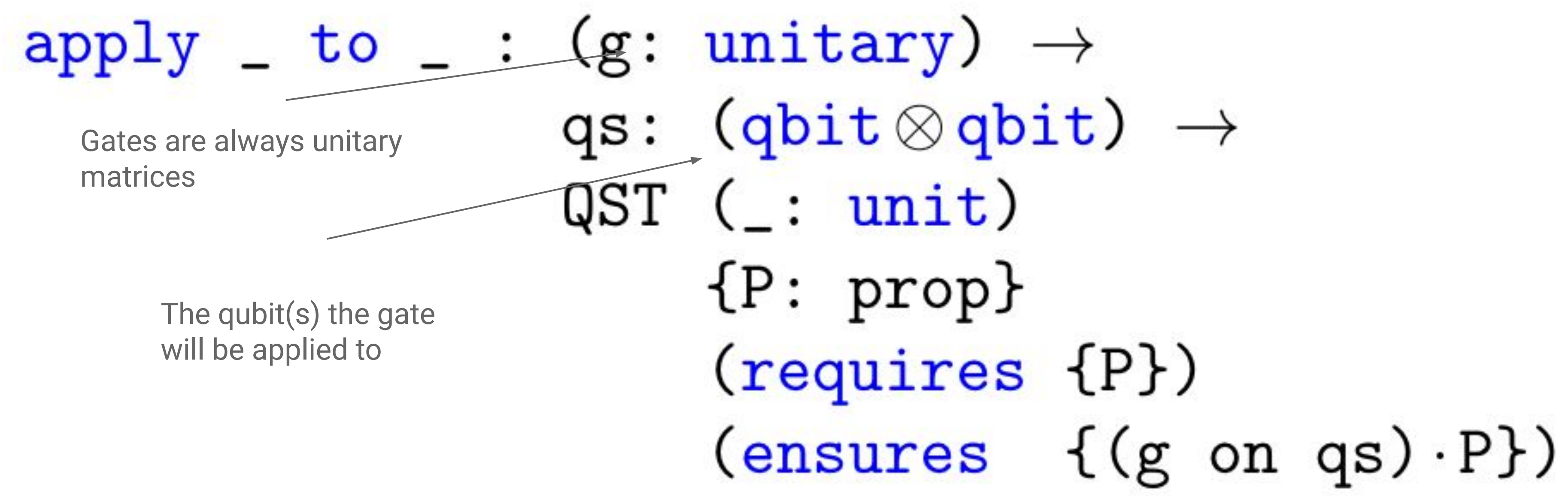
[9]

Unitary Gate Application Notation

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

[9]

Unitary Gate Application Notation



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

Gates are always unitary matrices

The qubit(s) the gate will be applied to

A Great Example From the Paper

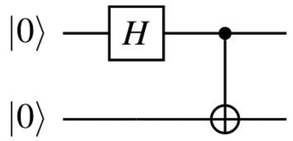


Figure 1: Circuit to prepare the first Bell state

```
bell00 : unit → QST (a, b): (qbit ⊗ qbit)
                   (requires {⊤})
                   (ensures  {(a,b) =_q |β_00⟩})
bell00 = λx.do
           a ← init 0
           apply H to a
           b ← init 0
           apply CX to (a, b)
           return (a, b)
```

[9]

Exercise Three: Try making the Bell State you found earlier using this notation

Want to try something a bit more difficult? Try making a circuit with three qubits.

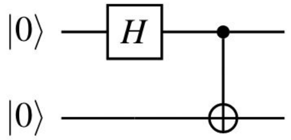


Figure 1: Circuit to prepare the first Bell state

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

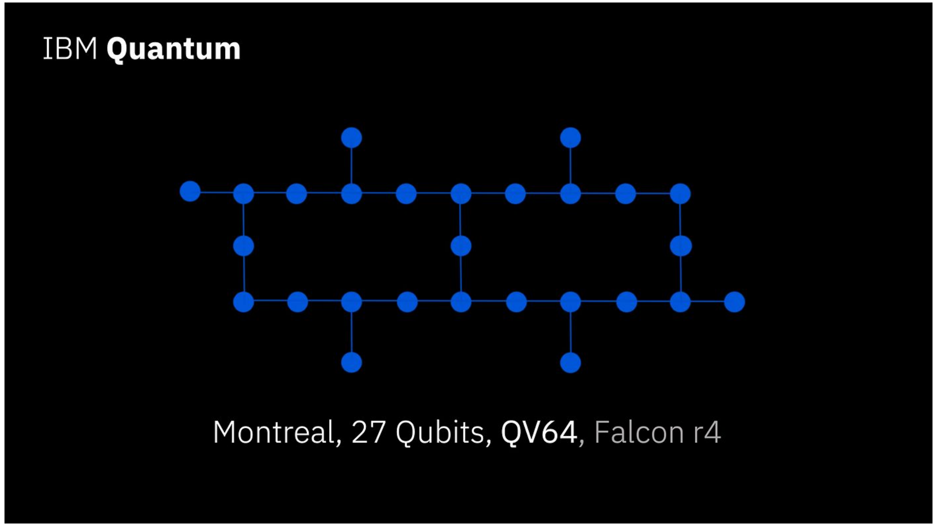
Formal Methods in Quantum Software Engineering

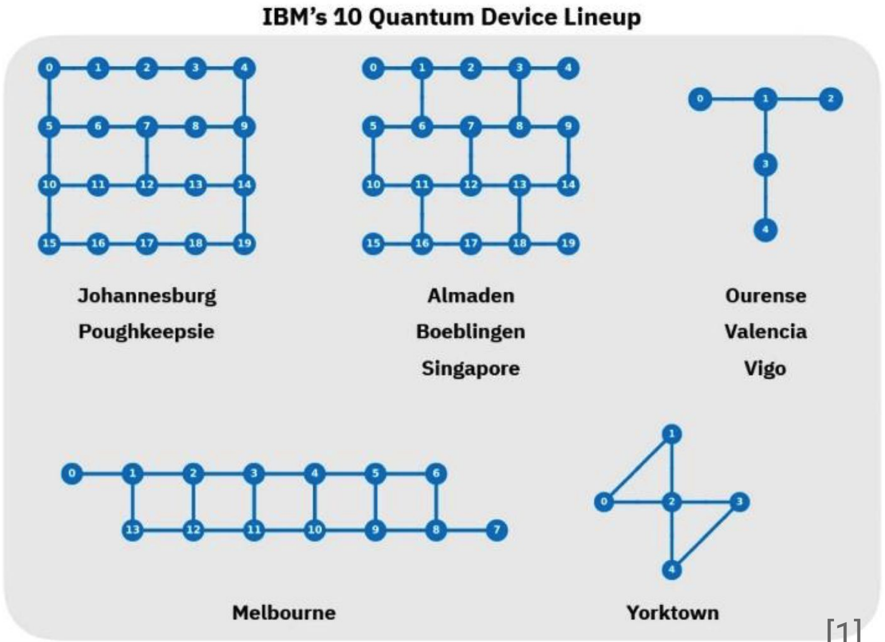
- Multiple formal methods (and multiple versions of Quantum Floyd-Hoare) have been used to describe quantum programs on paper, but not many have been used within Quantum Software Engineering
- My first hypothesis: many formal methods may not be suitable for NISQ era quantum computers (even if mathematically they are suitable)
  - Likely a scalability issue
- My second hypothesis: the application of formal methods in Quantum Computing is done within *Quantum Error Correction*

Quantum Error Correction

An algorithmic approach to analysing and preventing unexpected behaviour that may occur while a quantum program is running.

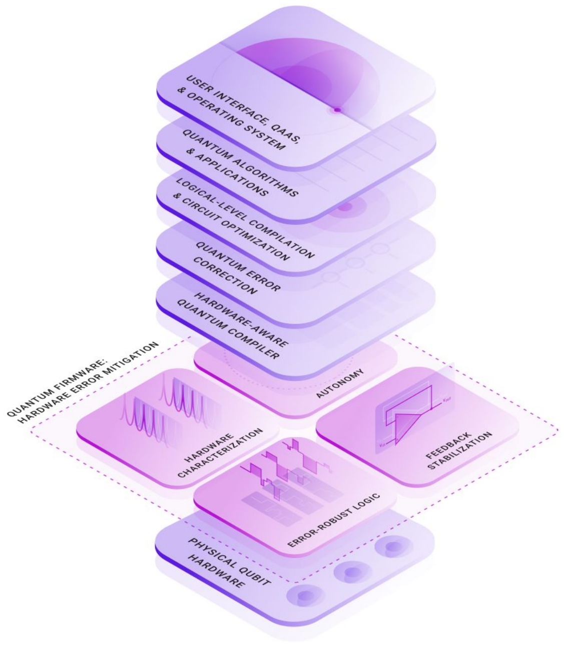
Predictive Quantum Error Correction

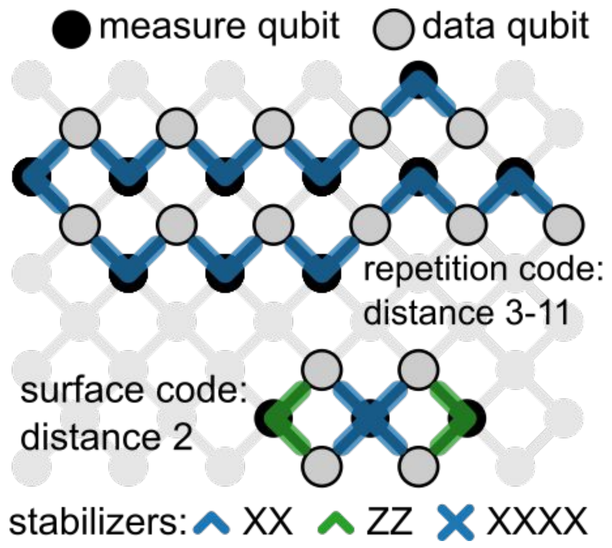




[1]

Quantum Error Correction Within the Program


USER INTERFACE, QAAS, & OPERATING SYSTEM
QUANTUM ALGORITHMS & APPLICATIONS
LOGICAL-LEVEL COMPILATION & CIRCUIT OPTIMIZATION
QUANTUM ERROR CORRECTION
HARDWARE-AWARE QUANTUM COMPILER
QUANTUM FIRMWARE: HARDWARE ERROR MITIGATION
AUTONOMY
HARDWARE CHARACTERIZATION
FEEDBACK STABILIZATION
ERROR-ROBUST LOGIC
PHYSICAL QUBIT HARDWARE



[11], [12]

Thank you for listening!

All references have been included in the Jupyter Notebook!