QCSG

The Quantum Computing Survival Guide

Written by Zack Schieberl

QCSG

Preface

This document is meant to be a modern equivalent to a textbook for quantum computing. Modern equivalent - don't textbooks still exist in today's world? Yes, but it is our belief that the current format of paper textbooks do not translate well to the new generation's style of learning.

- This document has links to other parts of itself for ease of access as well as links to outside resources. This format is better suited for reading on a computer than using a glossary or series of appendices.
- This document uses applets to produce animations and interactive figures. The PDF version does not support these applets, so static images are used as a replacement. Use the web version to interact with the figures.
- Every question has an answer. With the advent of online answer services, creating a list of exercises for the reader is synonymous with providing homework questions for teachers to use. This does nothing but discriminate against those who cannot afford the paid answer services. Instructors who decide to use this textbook in the setting of a classroom will need to create their own homework, or at least modify the existing questions we provide.
- This document is free. It will always be free. All future versions of this document will be free. Nearly all textbooks are pirated, scanned, and uploaded to the internet. Why not provide an encouraging environment for students rather than making them feel like criminals for learning?
- We accept corrections, suggestions, and new ideas from *anyone*. Simply post a new issue on the public GitHub repository for this textbook using the proper format. Having a public GitHub also allows any reader to view all past versions of the document and the authors of those changes.
- This document is covered by the CC BY 4.0 license, meaning the material may be redistributed, copied, or modified freely, by anyone, so long as proper attribution is given. The code associated with this document is covered by the Apache License 2.0, meaning the code may be redistributed, used in private or public projects, or modified freely, by anyone, so long as proper attribution is given. The code can not be patented except by the original author.
- We expect this document to eventually be adapted into a video lecture series, free and available to everyone, of course.

Quantum? What's that?

Quantum mechanics are the laws which govern *quantum physics*, the study of very small particles such as atoms, electrons, and photons. No previous knowledge of quantum mechanics or quantum physics is assumed or required to understand this document. However, we do make some assumptions about the reader's knowledge of mathematics and computer science:

- Precalculus: Complex Numbers, Trigonometry, Polar Coordinates, Radians
- Linear Algebra: Vectors, Matrices, Matrix Multiplication, 3 Dimensional Math
- Discrete Mathematics: Boolean Algebra, Sets, Turing Machines, Complexity Classes
- Computer Science: Introductory Programming, Data Structures, Algorithms, Computer Organization

- These topics are especially important: Arrays, Big-Oh Notation, Bitwise Manipulation

Some amount of review for these topics is provided, but we are assuming readers will already have been introduced. We expect this document will be understandable to a Computer Science undergraduate by the end of their second year.

Outline of the Document

The first chapter is a simple introduction to the history of quantum computing and why it is & will be useful. This chapter is not required, but can be useful for providing a bridge to the real world. The next three chapters form the foundations of quantum computing. After the chapter on entanglement, the document opens up, allowing instructors to focus on a particular subfield if they so choose.

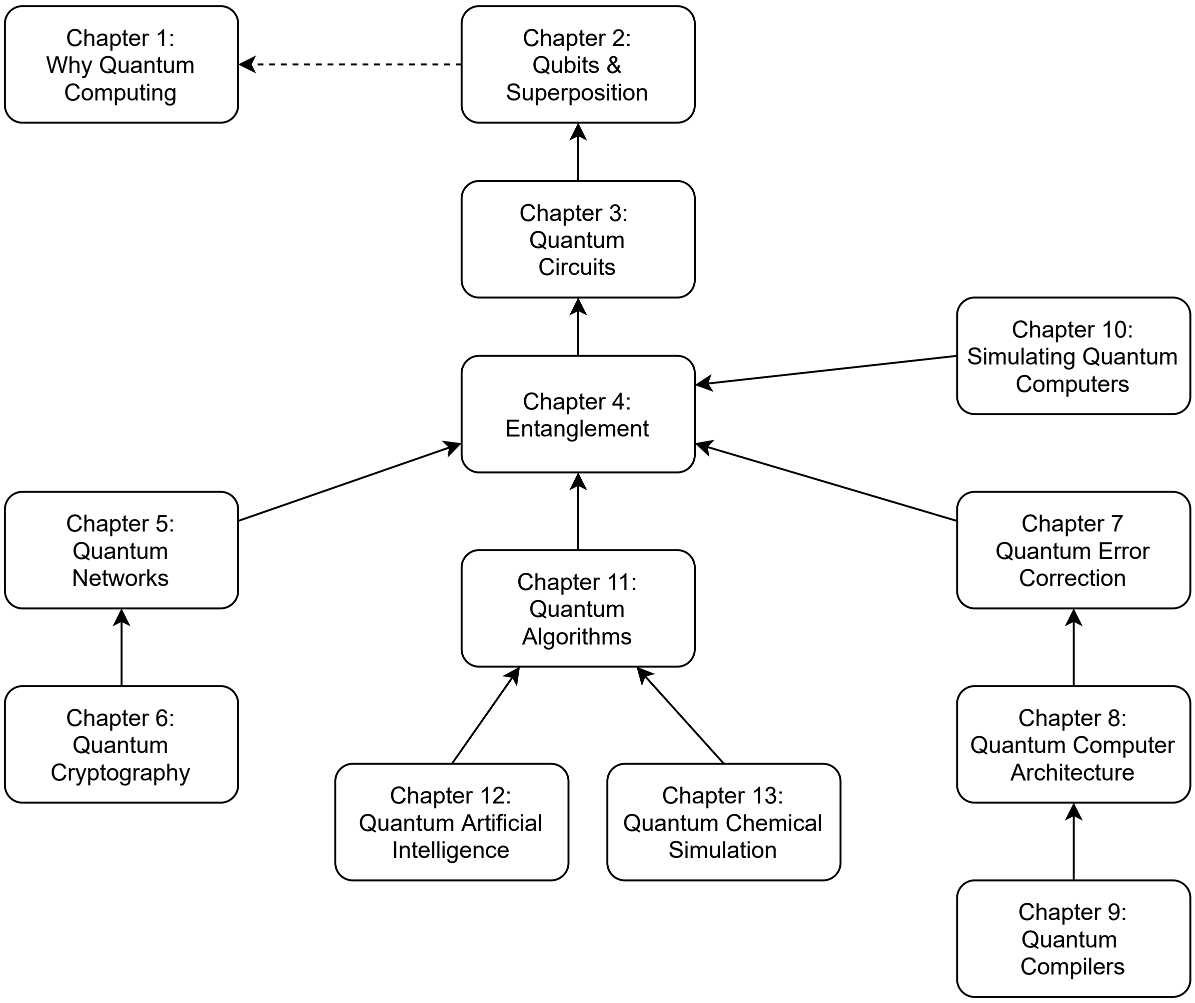

Chapter 1: Why Quantum Computing
Chapter 2: Qubits & Superposition
Chapter 3: Quantum Circuits
Chapter 10: Simulating Quantum Computers
Chapter 4: Entanglement
Chapter 5: Quantum Networks
Chapter 11: Quantum Algorithms
Chapter 7 Quantum Error Correction
Chapter 6: Quantum Cryptography
Chapter 12: Quantum Artificial Intelligence
Chapter 13: Quantum Chemical Simulation
Chapter 8: Quantum Computer Architecture
Chapter 9: Quantum Compilers

Figure 0.1.1 The Chapter Dependency Graph

Citations

MLA8

Schieberl, Zack. *Quantum Computing Survival Guide*. Github, 2020.

CSE Manual Scientific Style and Format

Schieberl Z. 2020. Quantum Computing Survival Guide. New York (NY): Github.

Chapter 1: Why Quantum Computing?

The entire document is used to describe what quantum computing is, but I think it is important to understand why quantum computers are important. What is their history and what can they do for society?

1.1 The New Age of Computing

Gordon Moore made an observation that the number of transistors in an integrated circuit doubles every 2 years, known as *Moore's Law*. There are only two ways to increase the number of transistors: make them smaller, or make the chip larger. So far we have continued to choose the first option, finding new methods to make the transistors smaller and smaller each year to follow this pattern. However, scientists can't continue to make them smaller forever. After all, we are bounded by the physical limits of our universe. The size of transistors can't breach the atomic barrier, becoming smaller than atoms. It is unclear when our computers will begin to diverge from Moore's Law.

Luckily, the end of Moore's Law does not mean the end of improvement for computers. There are certainly other methods to give computers more memory, make them faster, and improve power consumption. Once such method is the use of specialized hardware. Rather than doing every single operation on the *Central Processing Unit (CPU)*, computers can delegate specific tasks to other hardware which are designed to be efficient at certain types of jobs. One common example is the *Graphics Processing Unit (GPU)*, which is a separate device used to handle massively parallel programs. The architecture and style of programming for GPUs is different than CPUs. There are some problems that run incredibly fast on a GPU, however some problems actually run slower.

Every algorithm has a *time complexity*, a formula describing how long the algorithm takes to run based on the size of the input. If we built new hardware which could significantly lower the time complexity for certain problems, it would signal a new age for computers. Introducing *quantum computers*: computers which run off of different principals than classical computers, derived from the laws of quantum mechanics.

Do scientists just put the word 'quantum' in front of everything?

To answer your question: pretty much. Formally, the word *quantum*, and its plural *quanta*, can be used to describe any discrete chunk of something. In science, especially physics and chemistry, quantum is used to describe the study of very small particles: atoms, electrons, quarks, photons, etc. A quantum computer is any machine which uses a series of transformations, based off of quantum physics, to manipulate data into a desired result. These quantum computers are significantly different than classical computers. The methods of storing, transforming, and reading quantum data is unintuitive from a classical computer's perspective.

Researchers have shown that some problems such as integer factoring and searching unsorted data can be implemented more efficiently on a quantum computer than a classical one. Using a *Quantum Processing Unit (QPU)*, a CPU could send it specific jobs which are solved much faster on a quantum computer. There are many issues plaguing current quantum computers which make them difficult to offer to the general public, but the future of quantum computing looks very bright and there is still much to be discovered.

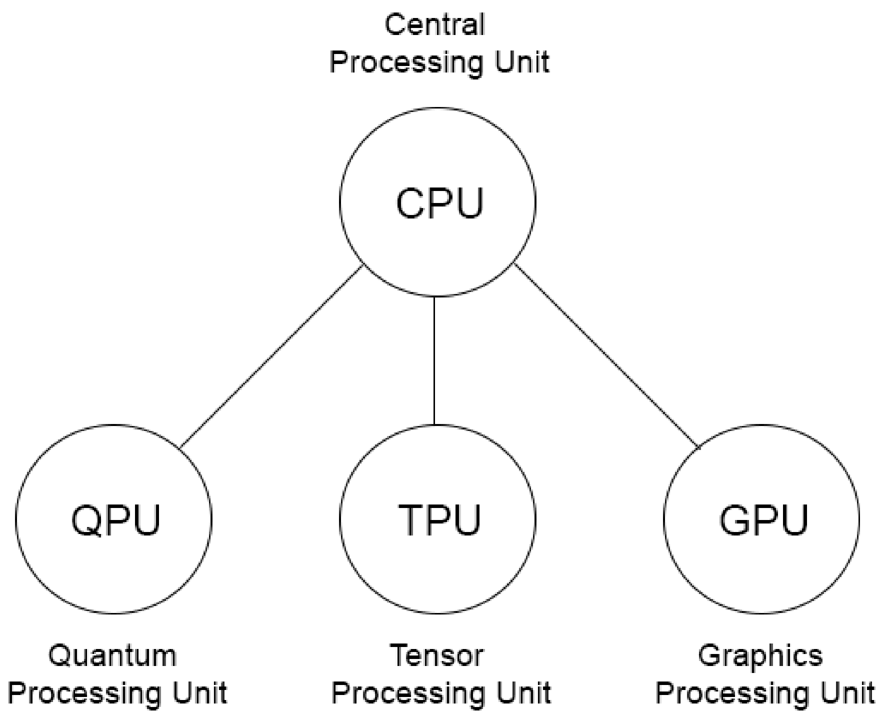


Figure 1.1.1 A Possible Future Computer Architecture

This figure shows a potential architecture of future computers. The central processing unit would act as the manager for the computer, running a majority of the operating system and delegating specific tasks to the devices suited for them.

Another possible architecture of future computers follows the trend of *cloud computing* in which users send tasks to a quantum computing cluster over the internet. The result of the computation would be sent back to the user when the program finishes. If our society begins to focus more on subscription services, this may end up becoming a more popular alternative to owning your own quantum computer.

1.2 A Brief History of Quantum Computing

The birth of quantum computing can be attributed to a man named Richard Feynman. Famous for a plethora of accomplishments in physics during the twentieth century, one of his biggest was the discovery of quantum electrodynamics. He was interested in simulating many different particles to verify his theory, but found that classical computing did not have the ability to effectively produce the results he was looking for. So, in the early 1980s, he proposed the idea of a quantum computer. Unfortunately, there was not a high demand for this type of computation and constructing a quantum computer is immensely difficult. For the next few decades, the field of quantum computing progressed slowly.

Then, in 1994, Peter Shor shocked the world with his quantum algorithm to factor integers in polynomial time. This not only revealed there was some use in quantum computing, but that there were probably many other useful quantum algorithms yet to be invented. Not too long after came Grover's algorithm along with other quantum search algorithms. By the late 1990s, the theoretical field of quantum computing was alive with activity.

The Field(s) of Quantum

The field of quantum computing has foundations in quantum physics, but ultimately, they have separated. New students looking to study quantum computing don't need previous knowledge from quantum physics similar to how classical computer scientists don't need knowledge from electrical engineering.

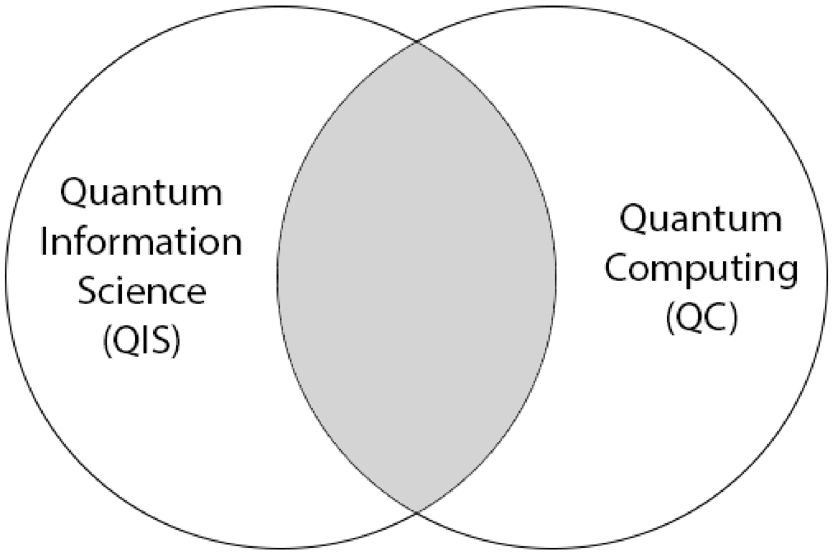


Figure 1.2.1 The Two Parallel Fields

Quantum information science is a parallel field of study to quantum computing, and the material often overlaps. Both fields are very new, less than half a century old, and have yet to be thoroughly defined. It is difficult to say if these fields are separate from one another or if these are both subsections of the same field. Typically, quantum information science covers the theory behind combining quantum mechanics with Turing machines, pioneered by Paul Benioff in the early 1980s. Quantum computing covers the implementation of a quantum computer and how it interacts with a classical computer. Both fields cover qubits, quantum circuits, entanglement, and teleportation which are discussed in this document. At this moment, it is unclear if the overlap between the fields is so large that they will be joined or if they will diverge. Only the time will tell.

Learning Approaches to Quantum Computing

Typically, there are three approaches to learning about quantum computing. These depend on which field the student is coming from. With classical computers, the fields in order of increasing abstractness are: electrical engineering, computer engineering, computer science, and mathematics. Electrical engineers deal with the physical implementations of computers, mathematicians deal with the proofs of logic, and the computer engineers & scientists fill in the gaps.

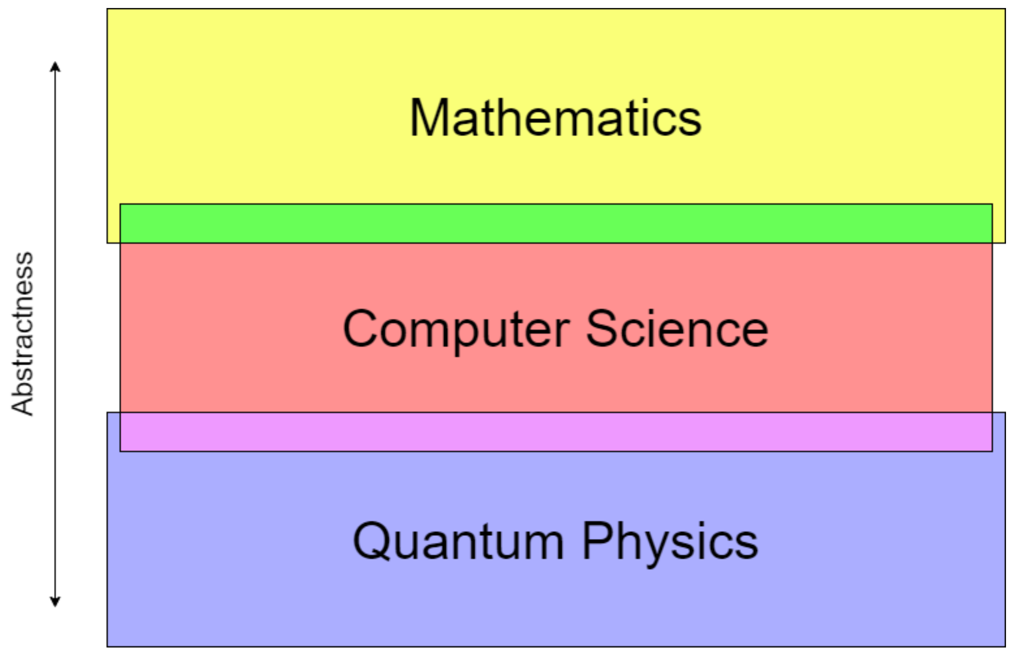


Figure 1.2.2 The Parent Fields

Quantum computing also has a similar structure of parent fields. The physical implementations involving working with quantum particles are done by physicists, the proofs for computation are handled by mathematicians, and the computer scientists bridge that gap. This document will primarily be focused on the computer scientist's perspective, although we will be exploring a little bit into the other sides. In the future we expect numerous subfields to appear, dealing with various aspects of quantum computing.

Historic Quantum Computers

Many different types of quantum computers have been designed over the past few decades. A few of the important milestones for the development of physical implementations are listed here:

- 1995 - First realization of a quantum logic gate
- 1998 - First implementation of Grover's Algorithm on a 2 qubit computer
- 2001 - First implementation of Shor's Algorithm factoring 15
- 2006 - First demonstration of teleportation
- 2011 - First commercially available quantum annealer (D-Wave One)
- 2016 - First quantum computer available on the cloud (IBM)
- 2019 - First instance of a claim to quantum supremacy (Google)

1.3 Modern Applications of Quantum Computers

Current quantum computers are room-sized machines which require dozens of skilled technicians to operate. They are owned by research teams at universities or companies and generally not available to the public. There are exceptions, such as IBM's small quantum computer which can be used over the cloud, but there are severe limits to the compute capabilities of such quantum computers.

For reasons discussed later, simulating quantum phenomena on classical computers is tricky. However, it stands to reason that simulating quantum phenomena on a quantum computer would be much easier. Modern quantum computers are not large enough to run many of the algorithms designed for them, but simulating other quantum particles requires relatively few resources. This makes quantum simulation a viable target for near-term applications.

1.4 Future Applications of Quantum Computers

Quantum computers hold enormous potential to solve some of the world's hardest problems in the future. Once the hardware for quantum computers has been developed and refined, our society will be able to tackle the problems which classical computers have struggled with. Here are a few examples of how quantum computers will be able to help other areas of science.

Cryptography and Network Security

When discussing the history of quantum computers, we have introduced Shor's algorithm. This quantum algorithm is able to factor integers efficiently which undermines the RSA protocol for key exchange. If a quantum computer was suddenly introduced today, it would overturn modern society because we are not yet prepared for switching from RSA to another protocol. Luckily, it seems quantum computers large enough to factor RSA numbers are more than a decade away.

With the bad, there comes some good as well. Even though quantum computers threaten to shake our cryptography, they also enable us to create quantum networks which are able to transfer data securely, guaranteed by the laws of physics.

Artificial Intelligence

The intersection between artificial intelligence and quantum computing is a new area of research in the academic community and has a promising future. Quantum algorithms have been developed for solving optimization problems, of which artificial intelligence is just a subset. Finding optimal solutions using quantum computers also has benefits to many other disciplines such as finance for portfolio management, transportation for delivery routes, and manufacturing for factory layout.

Pharmaceutical Development

As discussed in the previous chapter, it seems natural for quantum computers to simulate quantum interactions much better than classical computers. One of the most exciting applications of this is drug development in the pharmaceutical industry. Medicine compounds are made up of many molecules and perfectly simulating their interactions on a classical computer is infeasible. One day, chemical experiments might be reduced to running a quantum program and finding how all the molecules interact with each other.

Chapter 2: Qubits & Superposition

The smallest possible unit of data in a classical computer is a *bit*. It stands for binary digit. A single bit can either be zero or one. The term 'bit' is often used interchangeably with *boolean*, whose value is either true or false. Every piece of data which can be stored in a computer's memory is a sequence of these bits. Qubits are the smallest possible unit of data in a quantum computer and they have more properties than ordinary bits.

2.1 Quantum Particles

Although this document is focused towards computer scientists, it is important to learn about the nature of quantum particles to understand where qubits come from. This explanation will only scrape the surface of the wonderful world of quantum physics, but it will suffice for the purposes of this document.

Quantum particles have close ties to *waves*, a mathematical function which can be described by four attributes: the amplitude, frequency, and horizonal & vertical shifts. A wave function has the following form:

$$F(x) = A\sin(B(x+C)) + D$$

Applet 2.1.1 *The Sine Wave*

This figure shown above is called an *applet*. It is an interactable program embedded inside the document which allows readers to change values to view the topic in depth. For this applet, you can click and drag on the green sliders which change the 4 values to see their impact on the wave.

Several famous experiments, such as the double slit experiment, have shown that light and matter can have properties of both particles and waves. This is known as *wave/particle duality*. When an electron is released from a source, it acts as a wave which propagates through space until contacting a surface. Once the electron collides with a surface, it is only observable in one location.

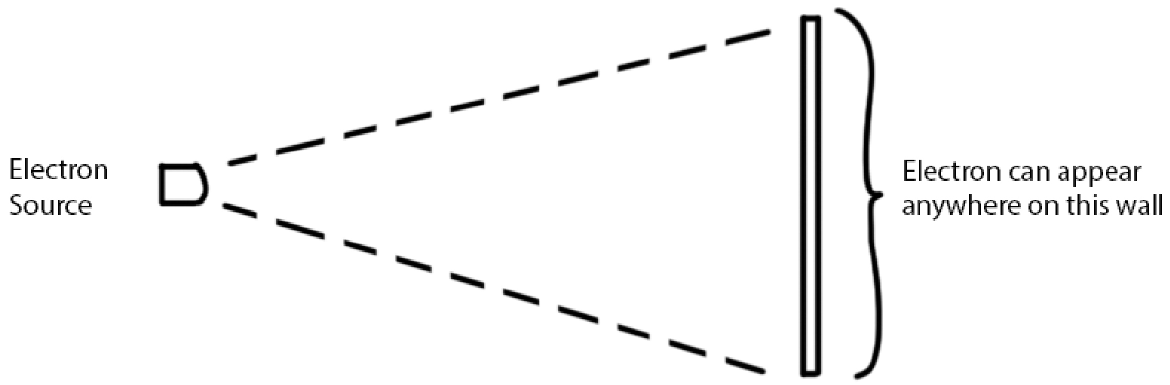


Figure 2.1.2 - Electron Colliding with a Wall in a Random Location

The location where it will be observable is random and cannot be determined ahead of time. However, the likelihood of the electron appearing in a specific location can be calculated using a probability function.

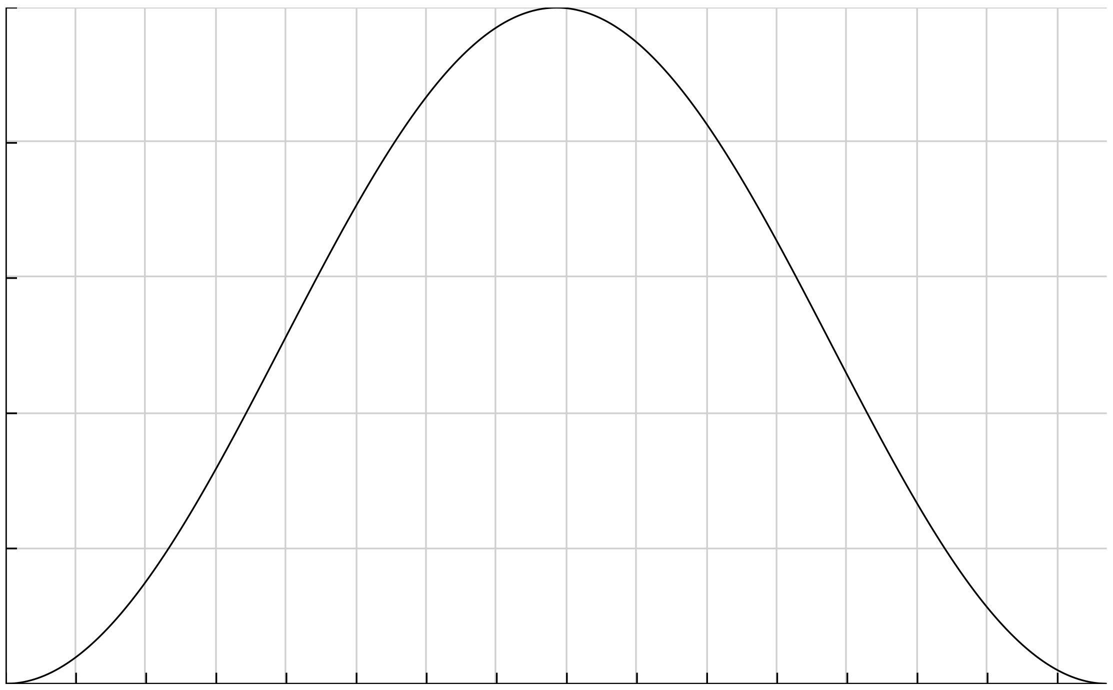


Figure 2.1.3 - A Probability Function

The function shown above contains information on how likely the electron is to be observed at a specified location. In this graph, the X axis is the location and the Y axis is how likely it is to appear at that location. The electron is very likely to appear somewhere in the middle and it is very unlikely to appear near the edges. In the bottom left corner of this graph, when $x = 0$ it shows that $y = 0$ as well. This means the electron can **never** appear at location 0. The same logic applies at the bottom right corner.

The rest of this document is dedicated towards explaining the rules governing qubits and their interactions, this is the only section which makes an attempt at showing where the rules are derived from. An inquiring mind may ask "why do qubits follow these rules specifically?" and the only answer is because they follow from the laws of quantum mechanics.

2.2 Modeling Waves with Vectors

The fundamental unit of information in classical computers are bits. They can either be on or off. There are only two possible states. Figure 2.1.2 shows a probability function which is continuous, there are an infinite number of locations, each with their own probability of being observed at said location. To make these probability functions palatable to computer scientists, the probability functions need to be transformed into binary.

There are already many decades worth of development in computer science which is in binary. Having two states allows quantum computer scientists to create algorithms, build architectures, and store data based on classical computers without factoring in a change of radix. Another reason for dividing the continuous function into two sections is because it makes many physical implementations of qubits easier.

To extend our example from the previous section, we can convert to binary by drawing a line across our wall and seeing if the electron is observed above or below the line. The electron has a probability p of being above the line where p is $0 \leq p \leq 1$. There is also a probability of $q = 1 - p$ that the electron is below the line where q is also $0 \leq q \leq 1$.

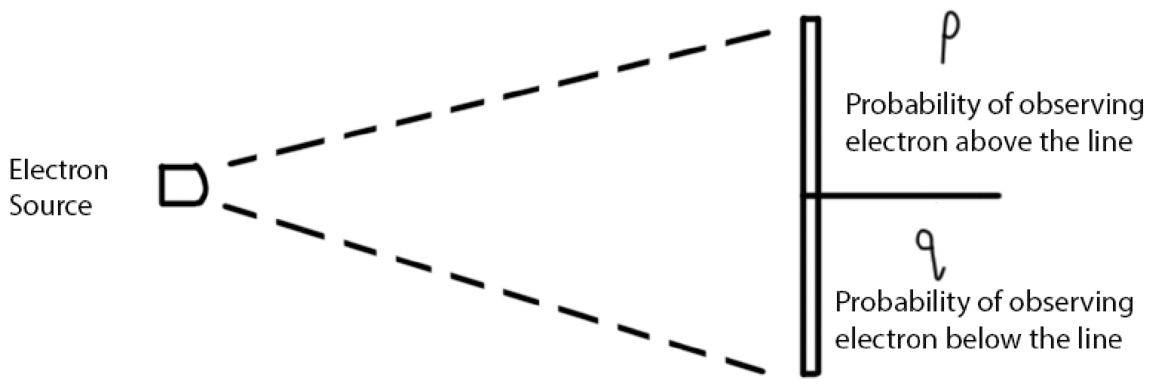

Electron
Source
p
Probability of observing electron above the line
q
Probability of observing electron below the line

Figure 2.2.1 - Electron Colliding with a Discretely Sectioned Wall

What we have just described is known as a *probabilistic bit*, or *pbit*. A pbit is a bit which has a certain probability or being off, which we have denoted above as p. It also has a certain probability of being on, which we have denoted above as q. As both these values are probabilities, they are constrained to $p + q = 1$. The concept of a pbit is not exclusive to quantum, they also are used in binary classifiers from machine learning, random number generators, etc.

Our goal is to convert a wave equation like the sine function into a pbit. One of the first things a precalculus course teaches is the relationship between trigonometric functions and circles. Taking a look at the unit circle, we can see the X and Y coordinates of the circle with radius 1 are the outputs of the cosine and sine functions.

Applet 2.2.2 *The Unit Circle*

A *vector* is a mathematical object which has both magnitude and direction. We have a vector here whose magnitude is 1 and direction is determined by the variable θ, the angle from the positive X axis to the vector. θ is called a *polar coordinate* which is the input to the trigonometric functions that produce the cartesian coordinates x and y. The vector $\mathbf{v}$ is described as:

$$\mathbf{v} = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix}$$

This can be used to fit our description of a pbit by remembering the Pythagorean trigonometric identity:

$$\begin{aligned} x^2 + y^2 &= 1 \\ \cos^2\theta + \sin^2\theta &= 1 \\ p + q &= 1 \end{aligned}$$

From this we can see our probabilistic bit relate to the quantum wave functions as:

$$\begin{aligned} p &= \cos^2\theta \\ q &= \sin^2\theta \end{aligned}$$

Since p and q are always positive, we can apply the constraint $0 \leq \theta \leq \frac{\pi}{2}$.

There are two unit vectors which are special enough to be named: $\hat{\imath}$ and $\hat{\jmath}$. These vectors are both of length 1 and are orthogonal to each other.

$$\begin{aligned} \hat{\imath} &= \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ \hat{\jmath} &= \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{aligned}$$

These *basis vectors* represent our X and Y axes since $\hat{\imath}$ is directly along the X axis and $\hat{\jmath}$ is directly along the Y axis. The vector $\mathbf{v}$ can be rewritten using vector addition:

$$\mathbf{v} = \begin{bmatrix} \sqrt{p} \\ \sqrt{q} \end{bmatrix} = \sqrt{p}\begin{bmatrix} 1 \\ 0 \end{bmatrix} + \sqrt{q}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \sqrt{p}\hat{\imath} + \sqrt{q}\hat{\jmath}$$

After viewing this relationship, we see:

- $\hat{\imath}$ conceptually represents the off state, or 0, or the electron being observed above the line on the wall
- $\hat{\jmath}$ conceptually represents the on state, or 1, or the electron being observed below the line on the wall
- p and q still represent the probabilities of those states occurring
- $\mathbf{v}$ represents the entire state of the pbit, or the electron & wall system

This model of a pbit is not a completely new innovation, it is simply a transformation from the quantum wave equation which is easier to digest for computer scientists.

However, our transformation is not complete. We have translated the probabilistic nature of quantum particles but there is still another feature of those particles which we have not translated: phase. Without including phase, we cannot say our model acts as a true quantum bit, also known as qubit.

Before extending our vector based model to the true qubit model, a new style of notation needs to be introduced. The notation of quantum physicists: the Dirac notation.

2.3 Dirac Notation

Paul Dirac invented a notation system for discussing quantum mechanics and it has been adopted into quantum computing. The notation is called "bra-ket" since it uses the angle brackets $\langle\rangle$. Truly the peak of twentieth century physics humor.

Ket Notation

Any vector can be expressed by placing it inside a vertical line $|$ and right angle bracket $\rangle$. By placing it inside, $|\mathbf{v}\rangle$, we pronounce it "ket-v". This object conceptually represents a quantum state, however it is still mathematically equivalent to a vector. Specifically, a quantum state is a member of a *complex Hilbert space*, but thinking of them as vectors is acceptable in quantum computing.

Since a ket represents a quantum state, we can put our own quantum states inside the ket like: $|\text{above-line}\rangle$ and $|\text{below-line}\rangle$ to represent the electron being observed above or below the line on the wall. Some more examples of kets are:

- $|\text{spin-up}\rangle$ and $|\text{spin-down}\rangle$ to represent the spin of a photon
- $|\text{ground}\rangle$ and $|\text{excited}\rangle$ to represent the energy level of an electron
- $|/\rangle$ and $|\backslash\rangle$ to represent diagonal and anti-diagonal polarization of light
- $|\text{clockwise}\rangle$ and $|\text{counterclockwise}\rangle$ to represent direction of current flow in a circuit

However by far the most common kets used in this document and in the quantum computing community are $|0\rangle$ and $|1\rangle$. We can abstract away the physics used for the quantum state and focus on the value. Similar to how computer scientists don't care if their bit is representing a magnetic field in their hard drive or amplitude of current in a wire, they just care about the value. $|0\rangle$ and $|1\rangle$ can be any two orthogonal basis states in quantum mechanics.

The quantum state representing an entire system is denoted by the special Greek letter ψ, written "Psi", and pronounced "Sai". Using all of this information, we can rewrite our pbit model using Dirac notation:

$$\begin{array}{rl} \text{Vector Notation} & \mathbf{v} = \sqrt{p}\hat{i} + \sqrt{q}\hat{j} \\ \text{Dirac Notation} & |\psi\rangle = \alpha|0\rangle + \beta|1\rangle \end{array}$$

Even though we have not extended the pbit to the full qubit model, the above equation still holds for qubits. This equation is called **The General Form of the Qubit** and is one of the most important concepts in all of quantum computing.

It is important to understand that all of the symbols in the Dirac notation are the same mathematical objects. $|0\rangle$ and $|1\rangle$ are the exact same unit vectors previously represented by $\hat{i}$ and $\hat{j}$. α and β are still scalars which determine the probabilities when squared.

Using our new form of notation, let's reason about the process of projecting an electron at a wall. If the electron is detected above the line, then we know $|\psi\rangle = |0\rangle$. This is the same as saying "the state of the electron is the state of being above the line". On the other hand, if the electron is detected below the line, we know $|\psi\rangle = |1\rangle$. This is the same as saying "the state of the electron is the state of being below the line".

However, what do we call the state before the electron contacts the wall? The electron is still a wave moving through space. We do not know if the electron will be observed above or below the line since it is random, but we know the likelihood of those events occurring: $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$. Until the electron makes contact with the wall, it is reasonable to say "the state of the electron is both the state of being above and below the line". In other words, the electron's state is a combination of the two outcome states. When a quantum state can only be described as a combination of two or more basis states, we say the quantum state is in *superposition*.

Superposition

Superposition is one of those words which people think is more magical than it really is. Flipping a coin can be described with superposition too. When the coin is in the air, before it lands, the state can be modeled as: $|coin\rangle = {}^{1}/_{\sqrt{2}}|heads\rangle + {}^{1}/_{\sqrt{2}}|tails\rangle$. When this coin is twirling around in the air, you have no idea which side it will land on. At this moment in time, it is reasonable to say the coin is a combination of heads and tails. The coin is in superposition. To a human, the outcome of a coin flip is a random event. The only difference between the coin and the electron is if you have a good enough computer, it could simulate the physics of the coin toss and determine the outcome before the coin landed whereas the outcome of the electron cannot be determined before it lands.

When the electron collides with the wall, it is observed. The moment the electron is observed, the state goes from being in superposition to a basis state. This process is called *collapsing* and is instantaneous. The probabilities α^2 and β^2 and destroyed and only the state $|0\rangle$ or $|1\rangle$ is left. Colliding with a wall is not the only way to collapse a quantum state. Making any sort of measurement or observation on the system will cause the superposition to be destroyed and the system will choose one of the basis states to collapse to using the assigned probabilities.

Erwin Schrödinger is a famous physicist who was instrumental in developing the quantum wave equation. He also created a hypothetical scenario to help explain the concept of superposition: *Schrödinger's Cat*. This thought experiment describes putting a cat inside a soundproof box with a bomb which has a 50% chance to explode after 1 minute passes. After a minute passes, there are two possible outcomes of the experiment:

- The bomb explodes, and the cat is dead
- The bomb does not explode, and the cat is alive

However, to see if the cat is alive or not, the box must be opened. Let's begin the experiment. Put the cat inside the box, close it, and wait a minute. After the minute passes, what is the state of the cat? Is it alive or dead? We don't know. Nobody does, until the box is opened and the state is observed. Before the box is opened, we can say the state of the cat is in superposition. The cat is a combination of dead and alive at the same time. We can also model the state of the cat as: $|cat\rangle = {}^{1}/_{\sqrt{2}}|alive\rangle + {}^{1}/_{\sqrt{2}}|dead\rangle$.

It is important to note that no cats were harmed in this theoretical scenario.

Bra Notation

The ket notation is used to describe column vectors; there is also a partner notation to describe row vectors: bra. The vector placed between a left angle bracket $\langle$ and a vertical line | is pronounced "bra-v": $\langle\mathbf{v}|$. The bra is just the *conjugate transpose* of ket, also called the *Hermitian transpose*.

$$\langle\mathbf{v}| = |\mathbf{v}\rangle^\dagger$$

Now that we have defined our mathematical notation for vectors, we can describe common operations seen in linear algebra.

The notation for matrix-vector multiplication is identical to the standard notation:

$$BA|\mathbf{v}\rangle$$
$$|BA\mathbf{v}\rangle$$

Matrices can be inside or outside of the ket, but the order is right-to-left. A is applied first followed by B.

Dot products, also called *inner products*, between two vectors can be described by a combined bra-ket notation:

$$\langle\mathbf{u}|\mathbf{v}\rangle = \mathbf{u}^\top\mathbf{v} = \mathbf{u}\cdot\mathbf{v}$$

In Dirac notation, this is called the "overlap" between $\mathbf{u}$ and $\mathbf{v}$ since the dot product is related to how far one vector can project onto another. A vector will always overlap completely with itself, so $\langle\psi|\psi\rangle = 1$. Any two vectors which are mutually orthogonal will have no overlap, so $\langle 0|1\rangle = 0$.

Outer products, which are not commonly used in this document, can still be described in Dirac notation:

$$|\mathbf{u}\rangle\langle\mathbf{v}| = \mathbf{u}\mathbf{v}^\top$$

This product will produce an $m \times n$ matrix when $\mathbf{u}$ is an $m \times 1$ vector and $\mathbf{v}$ is an $n \times 1$ vector.

2.4 Bloch Sphere Model

Now that we have a proper system of notation to discuss quantum states, we will extend our model of a pbit to the full model of a quantum bit, also called a *qubit*. Quantum particles have the ability to randomly collapse to a basis state, but they also have another key attribute called *phase*.

Phase is another variable we need to account for in the model for the qubit, which means we will change the system of coordinates we have been using up until now.

Spherical Coordinates

Polar coordinates is a system of coordinates in 2 dimensions which uses a distance from the origin and an angle to describe every possible point: (r, θ). Our pbit model used polar coordinates but restricted r to always be equal to 1. To introduce phase, we need to add another variable to our model and add another dimension.

Spherical coordinates is a system of coordinates in 3 dimensions which uses a distance from the origin, an angle from the positive Z axis, and an angle from the positive X axis to describe every possible point: (r, θ, ϕ). In this model, θ is the angle from the positive Z axis and is called the *inclination angle*. ϕ is the angle from the positive X axis and is called the *azimuth angle*.

This notation may differ from a multivariable calculus class, where θ & ϕ's roles are switched and r is denoted ρ. We will be using the notation commonly used in physics and defined by the ISO standard.

Applet 2.4.1 The Bloch Sphere

This above applet allows the changing the values of θ and ϕ & moving the 3D sphere around by clicking and dragging.

The **Bloch sphere**, named after Felix Bloch, is the full model of the qubit. It uses spherical coordinates to describe every possible state a quantum particle can occupy. A *pure state* is any quantum state which lies on the surface of the unit sphere with radius 1. A quantum state could also be a *mixed state* which has radius less than 1, but will not be discussed here. Here are some facts about our new Bloch sphere model:

- The aquamarine point indicates the current quantum state, $|\psi\rangle$.
- The $|0\rangle$ and $|1\rangle$ states are vertical and orthogonal to each other.
- The inclination angle, θ, has the bounds: $0 \leq \theta \leq \pi$
- The azimuth angle, ϕ, has the bounds: $0 \leq \phi < 2\pi$

The fact that $|0\rangle$ and $|1\rangle$ are orthogonal to each other may be surprising and perhaps confusing. They certainly do not seem to be perpendicular, in fact they look parallel! Let us look at a static image of the Bloch sphere:

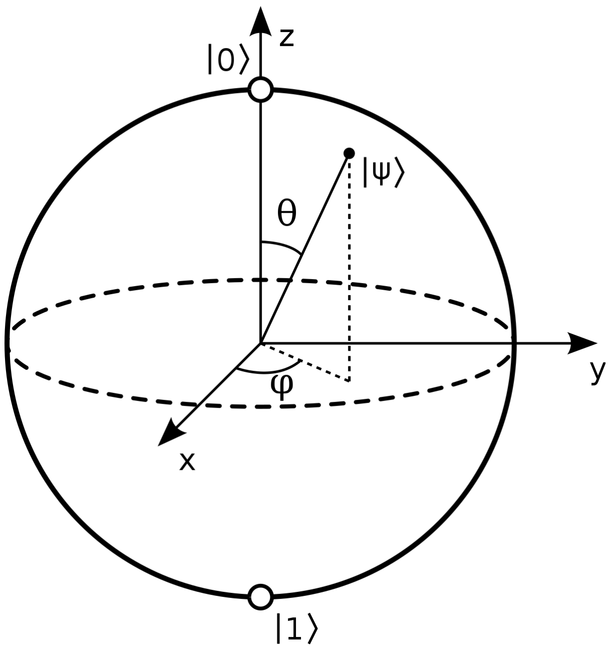


Figure 2.4.2 - A Static Bloch Sphere

A quantum state with radius 1 on the positive Z axis is $|0\rangle$ and one on the negative Z axis is $|1\rangle$. The reason why our orthogonal basis states are allowed to exist on the same axis is because the Bloch sphere technically exists inside *Hilbert space* instead of the typical *Euclidean space* we are used to. Don't sweat the details too much since Hilbert space is just a generalization of Euclidean space.

The basis coefficients α and β can now be defined in terms of the inclination and azimuth angles:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \cos(\frac{\theta}{2})|0\rangle + e^{i\phi}\sin(\frac{\theta}{2})|1\rangle$$

When describing qubits, α and β are complex numbers. Since they are complex, we need to make a minor adjustment to the formula describing the probability of collapse:

$$P(|\psi\rangle = |0\rangle) = |\alpha|^2$$
$$P(|\psi\rangle = |1\rangle) = |\beta|^2$$
$$|\beta|^2 = 1 - |\alpha|^2$$

The absolute value is necessary since we are now dealing with complex numbers. Not using it would mean some values of α and β would yield negative probabilities, which are not allowed.

The basis coefficients can also be described in Dirac notation with overlap:

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

Let's Talk Phase

Any description of phase would be incomplete without talking about *global phase*. Technically, the phase we have been working with in this section is formally defined as *relative phase*. When defining the basis coefficients α and β, we specifically stated that they are both complex numbers. However, looking at their definitions, α appears completely real. This is because the actual definition of the basis coefficients is:

$$|\psi\rangle = e^{ia}(e^{ib}\cos(\frac{\theta}{2})|0\rangle + e^{ic}\sin(\frac{\theta}{2})|1\rangle)$$

In this definition, a is the global phase. The interesting fact about global phase is that physicists have concluded global phase does not have any impact on the qubit. Not even indirectly. We can arbitrarily set $a = -b$ so the complex part of α cancels out and leaves us with just a real part. Continuing this example, $\phi = c - b$ which is the relative phase. In this document, phase is always talking about relative phase. Global phase will always be explicitly labelled.

Phase can be somewhat unintuitive at first glance. Why would we introduce (relative) phase to our model which does not directly affect the probability? The variable which represents phase, ϕ, only appears in the $e^{i\phi}$ term in β. Furthermore, when determining the probability, $|e^{i\phi}|^2$ will always equal 1 for any value of ϕ. It seems that θ is the only variable which can influence the probability of collapse. However, phase can indirectly influence the probability of a qubit through the transformations described in the next section.

2.5 Transforming a Qubit

Using qubits as data is only useful if we can edit their state. We used the Bloch sphere to geometrically describe the state of a single qubit, however we can't edit the angles θ and ϕ directly. The way we transform a quantum state is to rotate around the X, Y, or Z axis.

There are many different systems of notation used for describing rotations around the Bloch sphere. The one we will use is exponentiation: X^{Δ}. This says rotate counterclockwise around the X axis by $\pi\Delta$ radians. The Δ describes how many half-revolutions to rotate. If Δ is negative then rotate clockwise instead. Replace X with Y or Z to rotate around another axis.

$$X^0 = 0 = 0^\circ$$
$$X^{0.25} = \frac{\pi}{4} = 45^\circ$$
$$X^{0.5} = \frac{\pi}{2} = 90^\circ$$
$$X^1 = \pi = 180^\circ$$
$$X^2 = 2\pi = 360^\circ$$

Just like any angle, Δ is a cyclic number. For any number n there exists a $0 \leq \Delta < 2$ which performs the same rotation.

$$\Delta = n\ mod\ 2$$

It is also common to replace $X^{0.5}$ with $\sqrt{X}$. Do not change $X^{-0.7}$ into $\frac{1}{X^{0.7}}$.

Applet 2.5.1 Rotations on the Bloch Sphere

The above applet shows how rotations impact the current state of the qubit on the Bloch sphere. The sliders on the left directly change θ and ϕ and the slider on the right changes Δ. The buttons at the bottom change which axis the rotation is being applied to. Since Δ controls how far to rotate around the selected axis, it also changes θ and ϕ.

There is also a blue ring around the sphere which moves as the qubit state changes. It shows every location the qubit could be at after a rotation around the selected axis. This ring is the intersection of the surface of the Bloch sphere with a plane which is parallel to the other two axes. For example, when rotating around the X axis, the blue slice is parallel to the YZ plane and the X coordinate of the ring is the same as the qubit state.

Looking at the blue ring introduces us to the concept of *state space*. Formally, state space is a set of states a qubit is allowed to occupy. We are currently talking about pure states, so the *pure state space* of a qubit is the surface of the Bloch sphere. The *accessible state space* of a rotation is the blue ring and it changes as the state being rotated changes. The accessible state space of a rotation is a proper subset of the pure state space.

Collapse as a Transformation

Rotations are just one way of transforming a qubit's state. We previously talked about the process of measuring the qubit which collapses the superposition of the quantum state into a basis state. This operation is a transformation on the qubit because it manipulates the state.

When thinking about the theory of quantum computation it can be helpful to separate the process of collapsing the quantum state into two halves: the *phase collapse* and the *probability collapse*. The phase collapse, as the name implies, destroys the phase of the qubit and turns it into a probability. The probability collapse destroys the probability and turns it into a basis state.

```
phase_collapse($|\psi\rangle$) :
    return $|\alpha|^2$

probability_collapse($p$) :
    if (random() $> p$) :
        return $|1\rangle$
    else:
        return $|0\rangle$
```

When working with qubits in the real world, the two processes of collapse happen sequentially and instantaneously. There is no way to stop the collapse halfway through and observe the internal workings. One reason why it can be helpful to think about the phase and probability collapsing as separate processes is the task of state reconstruction. Consider a black box which can produce an unknown quantum state; once we observe the state, it collapses. Find the quantum state.

We are given an infinite number of identical quantum states, but they are hidden to us, so we need to glean information about what the unknown state is. Finding the probability of the qubit is easy, just measure n of them and count how many times it collapsed to $|0\rangle$:

$$P(|\psi\rangle = |0\rangle) \approx \frac{\text{count}(|0\rangle)}{n}$$

However finding the phase of this unknown state takes more effort. Every value of ϕ maps to the same probability during the phase collapse, so just looking at the probability we acquired does not directly find phase. The good news is that the probability does narrow down the number of states it could be considerably.

When we didn't know anything about the black box's quantum state, the state space of what it could have been was the entire Bloch sphere. Now that we know what the probability is, it narrows down the state space to a ring around the Z axis. Sound familiar? The set of possible states our box produces is the very same set of states produced by the accessible state space of a Z rotation. As we know from the above applet, the accessible state space changes when we move the quantum state.

The key to finding the phase for our unknown state is to apply our own rotations and find the new probability after the rotation has been applied. When we apply a $X^{0.5}$ rotation, we are rotating the entire ring of possible states around the X axis. Now, the possible state space after the rotation is a ring around the Y axis. Once we observe the new probability, we can narrow down the set of possible states to just two. This is because every value of ϕ will produce a different probability after a rotation around the X axis. The only phase which maps to the same probability after a rotation is the opposite side of the ring. Since there are only two possible states that our black box can produce, we can apply a $Y^{0.5}$ rotation to our unknown state. This will produce different probabilities for the two different possible points which narrows it down to our unknown state.

We have successfully found our unknown state using $3n$ qubits produced from our black box! The variable n controls how good the approximation is compared to the real state since $\lim_{n \to \infty}$ would be required to get a perfect representation. This process grows more complicated when there is more qubits involved, but it gives a nice introduction to the concepts of state space and phase versus probability collapse.

When does this idea of a black box producing an infinite number of qubits occur in reality? Exceedingly often! When we run quantum programs on a quantum computer, we are asking the computer to perform a sequence of operations and then it produces a quantum state. We don't know what this state is; if we did, why would we need to run the quantum program? However, simply running the program once and then measuring the outcome isn't useful because of the randomness involved with collapsing superposition. To see the result of our computation, we need to run the program many times and use a similar method to the one outlined above to find the quantum state. Using the above method requires running the program $3n$ times.

Thinking about the process of measurement and collapse in theoretical terms is straightforward, but the actual physics concerning collapse is not well understood and is called the *measurement problem*. Since we are unable to directly observe the collapse process, it is difficult to study. Perhaps one day physicists will have a better understanding of how and why superposition collapses which will allow us to avoid using approximations to find unknown states.

2.6 Multiple Rotations

A *quantum state* describes the current state of our system. A *state space* is a set of states that are allowed to exist under a certain condition. The collection of quantum states inside a given state space may be finite or infinite. The state space P^1 is the set of all 1 qubit pure quantum states. In terms of the Bloch sphere, this is every point on the surface of the sphere. We can also describe the state space which is created by a rotation around a given axis on a given state:

$$\text{span}(X) = \{\Delta : X^{\Delta}|\psi\rangle\}$$

Since we've defined Δ to exist $0 \leq \Delta < 2$, this set-builder creates a set of every accessible state by rotating around the X axis. Geometrically, this set describes the blue ring around the X axis, and the set changes as $|\psi\rangle$ changes. If $|\psi\rangle \in P^1$ then $\text{span}(X) \subset P^1$; this is true for any axis. The span function takes in an axis to rotate around and a state to act on and returns the set of all states which can be produced by rotating around the given axis.

As we saw when discussing collapse as a transformation, state space can be rotated too. We saw a rotation of $X^{0.5}$ being applied to $\text{span}(Z)$, which rotated the entire circle around the X axis. This is because it rotated every state inside of $\text{span}(Z)$ and produced the set:

$$\{|\psi\rangle \in \text{span}(Z) : X^{0.5}|\psi\rangle\}$$

The next logical step is to consider multiple rotations. What happens when we are allowed to rotate around 1 axis, but it can be any of the three? The set describing this operation is $\text{span}(X) \cup \text{span}(Y) \cup \text{span}(Z)$. Geometrically, this describes the overlay of three separate rings around each of the axes.

Sequential Rotations

What happens when we are allowed to rotate around 2 axes sequentially? Consider rotating around the X axis and then rotating around the Z axis:

$$\text{span}(XZ) = \{\Delta_1, \Delta_2 : Z^{\Delta_2} X^{\Delta_1}|\psi\rangle\}$$

The above set-builder describes the state space of every accessible state by rotating some amount around X and then rotating some amount around Z. Notice how the direction of application is different inside the span versus inside the set-builder. Recall that our ket object $|\psi\rangle$ is simply a vector representing the quantum state. The X^{Δ_1} rotation is actually a matrix acting on the state, which is why the notation is right-to-left instead of left-to-right like how the axes are ordered.

We have described the accessible state space for an XZ sequential rotation using set-builder notation, but what does it look like geometrically? There is another method of describing the state space which is easier to visualize:

$$\text{span}(XZ) = \bigcup_{|\psi\rangle \in \text{span}(X)} \text{span(Z)}$$

In English, this says to first create the set $\text{span}(X)$ of our state, and then union the set $\text{span}(Z)$ for every state inside of $\text{span}(X)$. Geometrically, this describes stacking an infinite number of rings on top of each other, one for every point on the ring created by $\text{span}(X)$. This creates a band around the Bloch sphere.

Applet 2.6.1 State Space of Two Sequential Rotations

This applet shows the blue state space created by sequential rotations around two different axes. In the case of XZ, the red ring is $\text{span}(X)$ and the blue surface is $\text{span}(XZ)$. The boundaries of the blue surface are rings around the Z axis. They are parallel to each other while also tangent and orthogonal to the red ring around the X axis.

The qubit slider in the bottom right is a representation of the sphere into 2 dimensions. It is the same representation of our spherical earth projected onto a 2 dimensional map.

The X and Z rotations are actually matrices and matrices are non-communitive, so it makes sense that:

$$Z^{\Delta_2} X^{\Delta_1}|\psi\rangle \neq X^{\Delta_1} Z^{\Delta_2}|\psi\rangle$$

Since we are able to see the accessible state space of XZ, we can actually take this one step further. Suppose there exists some function f which is able to switch the axes and change the amount of each rotation:

$$f : Z^{\Delta_2} X^{\Delta_1}|\psi\rangle \rightarrow X^{\Delta_a} Z^{\Delta_b}|\psi\rangle$$
$$f \text{ cannot exist}$$

However, we are able to deduce that the function f cannot exist because there are some states in $\text{span}(XZ)$ which do not exist inside $\text{span}(ZX)$, that is to say $\text{span}(XZ) \nsubseteq \text{span}(ZX)$. So the function f may be able to work for limited values of Δ_1 and Δ_2, but not for every rotation amount. To generalize this idea, we see:

$$f : U_1|\psi\rangle \to U_2|\psi\rangle$$
$$f \text{ exists when } \text{span}(U_1) \subseteq \text{span}(U_2)$$

If the accessible state space of some transformation U_1 is is a subset of the accessible state space of a different transformation U_2, then there exists some function which is able to change U_1 to U_2 for every state $|\psi\rangle$. This idea of transforming transformations can be found in various areas of quantum computer science, but particularly in the area of quantum compilers. We would like our quantum programs to run efficiently, and programmers do not necessarily write the fastest code. It can be necessary to transform the user's operations into a different, equivalent sequence which runs faster.

Complete State Space

Some set of states S is said to be complete with respect to another set of states P if $S = P$. Since we are currently talking about the surface of the Bloch sphere, all sets are complete if they are equal to P^1. For example, $\text{span}(XYZ)$ is complete because it is possible to go from any point to any other point through a sequence of X, Y, and then Z rotations.

Interestingly enough, $\text{span}(XZ) \cup \text{span}(ZX)$ is also complete, which means we could completely remove the ability to do Y rotations and still access every state. This is true for any axis. This approach may or may not be used in practice depending on the physical implementation of the qubit.

When we are discussing transformations being complete with respect to P^1, a transformation is complete if it has at least 3 degrees of freedom. So $\text{span}(XYZ)$ has three degrees of freedom, thus it is complete. The ability to rotate around a given axis by any amount counts as one degree of freedom. The space $\text{span}(XZ) \cup \text{span}(ZX)$ is also complete because the ability to switch the order of the axes counts as one degree of freedom.

The span of any three major axes is complete if there aren't two of the same axis sequentially. So $\text{span}(ZYZ)$ is complete, but $\text{span}(ZZY)$ is not complete. This is because sequential rotations around the same axis can be reduced to one rotation:

$$X^{\Delta_2} X^{\Delta_1} |\psi\rangle = X^{\Delta_1 + \Delta_2} |\psi\rangle$$

Note that the concept of 3 degrees of freedom only applies to mutually orthogonal axes. Consider an axis $\widetilde{Z}$ which is nearby, but not equal to the axis Z. The state space $\text{span}(Z\widetilde{Z}X)$ has 3 degrees of freedom, but is not complete. To confirm this, think about the state $|0\rangle$. Rotating $|0\rangle$ around the Z axis doesn't do anything because the state is already on the Z axis. The state space $\text{span}(\widetilde{Z})$ when the state is $|0\rangle$ is a small ring near the top of the sphere. Then the state space $\text{span}(\widetilde{Z}X)$ is a small band on the surface of the sphere around the X axis, which is not complete.

It is pleasant to visualize state space using parts of the Bloch sphere, but unfortunately when we increase the number of qubits, the space can no longer be described using geometric interpretations because there are not enough dimensions to work with. Luckily, math continues to work no matter the number of dimensions we are using!

2.7 Qubit Implementations

The math we have talked about in this chapter involves quantum states, superposition, the Bloch sphere, and rotations, but what are the connections to the real world? We have used the term *qubit* which refers to a quantum system that has 2 basis states: 0 and 1. This qubit is an abstract concept and can by implemented in the real world by using any two level quantum object.

Superconductors

Currently, many American companies involved with quantum computers are using superconductors to implement their qubits. Google, IBM, and Rigetti are some of the larger companies working with them. IBM has even enabled users to send quantum programs to one of their quantum computers via the cloud.

Although there is a lot of excitement about superconducting qubits, they are unstable and must be kept very cold. The basis states of a superconducting qubit are the directions of current flow in the superconducting loops: clockwise and counter-clockwise. The current direction can be put into superposition and measured when the wires are cooled to near absolute zero. Any disturbance to the computer causes errors in the output.

Ion Traps

Another favorite of corporate and research groups alike is the ion trap. IonQ is a company working on building trapped ion qubits in America.

Trapped ions are atomic particles which have been confined using electromagnetic fields. The basis states are the grounded and excited levels of the ion. These qubits are very stable, but since they use lasers to manipulate the quantum state, they are also slow.

Photons

The basis states of a photon are a pair of orthogonal linear polarizations. Photons can be used to transmit quantum states over long distances. A research group successfully sent a quantum state from China to Austria.

Others

There are many different methods of implementing qubits, and each of them come with their own advantages and disadvantages. There are even some theoretical methods of quantum computing which have yet to be realized in our world.

Chapter 3: Quantum Circuits

We have begun discussing the process of transforming a qubit, but we do not have a concise notation to describe many sequential transformations. This chapter will introduce the concept of *quantum circuits*, a quantum analogue to *classical circuits*. Quantum circuits are a widely used system of notation used to describe operations on the quantum state.

3.1 Definition of a Gate

Classical computers perform all of their operations through logic *gates*. What are gates? An in abstract sense, a gate is simply a boolean function which maps n input boolean variables to m output boolean variables $f : \{0, 1\}^n \to \{0, 1\}^m$.

In computer science, we can represent any boolean function with a *truth table* which contains 2^n rows, one for each possible combination of inputs.

x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

AND gate

x_1	x_2	$x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1

OR gate

x_1	x_2	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

XOR gate

x_1	$\neg x_1$
0	1
1	0

NOT gate

The above truth tables describe four common classical logic gates seen in computer science. A set of gates is said to be *universal* if every boolean function can be implemented using a sequence of gates only inside the set. One such set of universal gates is {AND, OR, NOT}, but there are many universal sets. The set {NAND} is a universal set which only contains one gate inside, which is the AND gate followed immediately by the NOT gate.

Quantum Gates

Gates are some operation which acts on the input to produce an output. Quantum gates follow that same definition by operating on a quantum state to produce a different quantum state. However there are some characteristics which are required for a quantum gate to be valid. First, recall our notation to describe a quantum state of 1 qubit:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \cos(\frac{\theta}{2})|0\rangle + e^{i\phi}\sin(\frac{\theta}{2})|1\rangle = \begin{bmatrix} \cos(\frac{\theta}{2}) \\ e^{i\phi}\sin(\frac{\theta}{2}) \end{bmatrix}$$

Since our quantum state is just a vector, a quantum gate is a matrix which acts on our quantum state through matrix-vector multiplication. Specifically, the gate is a square matrix whose size is equal to the quantum state. A gate acting on our 1 qubit system is a 2x2 matrix with complex entries. Another requirement for quantum gates is that their matrix needs to be *unitary*. A unitary matrix has the property that its adjoint (conjugate-transpose) is the same as its inverse.

$$\begin{aligned} U^\dagger &= U^{-1} \\ UU^\dagger &= U^\dagger U = I \\ |\det(U)| &= 1 \end{aligned}$$

The conjugate transpose was introduced as a method of converting between kets and bras in Dirac notation. Another property of these gate matrices is that all of the columns and rows are orthonormal. This means the absolute value of the determinant is 1.

Since every quantum gate is unitary, that means it is also *reversible*. Consider some sequence of quantum gates $U_1, U_2, \ldots, U_n$. The result of applying the sequence can be converted back into the original state by applying the adjoint of each in reverse order:

$$\begin{aligned} U_n \cdots U_2 U_1 |\psi\rangle &= |\psi'\rangle \\ U_1^\dagger U_2^\dagger \cdots U_n^\dagger |\psi'\rangle &= |\psi\rangle \end{aligned}$$

The transformations to the Bloch sphere discussed in the previous chapter can be described with quantum gates, except for collapse. The process of measuring a qubit cannot be expressed as a 2x2 unitary matrix since it collapses the quantum state into classical data. Since collapse is not reversible, we need to measure an infinite number of qubits to perfectly reconstruct the quantum state.

Recovering Angles from Vector Representation

In the following sections, we will be discussing quantum states in their vector forms since the ket forms are notationally verbose. Converting between the coordinates of the Bloch sphere and the vector representation of the qubit is common when working with computer programs. It can valuable to learn the different data types used for containing a qubit's data and how to convert between them. Starting with complex numbers, they can either be stored in Euclidean or polar form:

$$\begin{aligned} a + bi &= re^{i\varphi} \\ re^{i\varphi} &\rightarrow r(\cos\varphi + i\sin\varphi) \\ a + bi &\rightarrow \sqrt{a^2 + b^2}e^{i\operatorname{atan2}(a,b)} \end{aligned}$$

The function $\operatorname{atan2}(x, y)$ is a perfect inverse trigonometric function which returns the angle from the position X axis to the point defined by x and y. Traditional inverse trigonometric functions have domain restrictions which make them difficult to use in programs. Our function $\operatorname{atan2}$ can be defined a number of ways, but we will use:

$$\operatorname{atan2}(x, y) = \begin{cases} \operatorname{acos}(\frac{x}{\sqrt{x^2+y^2}}) & y \geq 0 \\ -\operatorname{acos}(\frac{x}{\sqrt{x^2+y^2}}) & y < 0 \end{cases}$$

The denominator inside the acos function can be disregarded if it is assumed that x and y are distance 1 away from the origin.

Typically our quantum state vector will hold Euclidean complex numbers since they are the easiest to work with in programs. Our first step to convert the vector contents to polar complex numbers:

$$\begin{bmatrix} a_1 + b_1 i \\ a_2 + b_2 i \end{bmatrix} \rightarrow \begin{bmatrix} r_1 e^{i\varphi_1} \\ r_2 e^{i\varphi_2} \end{bmatrix} = \begin{bmatrix} \cos(\frac{\theta}{2}) \\ e^{i\phi}\sin(\frac{\theta}{2}) \end{bmatrix}$$

Now we can easily calculate our spherical coordinates using our polar complex numbers:

$$\begin{aligned} \theta &= 2\operatorname{atan2}(r_1, r_2) \\ \phi &= \varphi_2 - \varphi_1 \end{aligned}$$

The phase needs to be calculated as the difference between the polar angles because of the qubit's global phase. Recall that every amplitude of a quantum state can be multiplied by e^{ia} and there will be no observable changes. We choose to set $a = -\varphi_1$ so the first amplitude is real. Many quantum gates change a quantum state's global phase, so we cannot assume $|0\rangle$ will always stay real without manually setting it.

3.2 Rotation Gates

Quantum gates can be described by unitary matrices acting on our vector representation of a quantum state. The rotations around the Bloch sphere also have matrix forms, as shown below.

$$\begin{aligned} X^\Delta &= \begin{bmatrix} \cos(\frac{\Delta\pi}{2}) & -i\sin(\frac{\Delta\pi}{2}) \\ -i\sin(\frac{\Delta\pi}{2}) & \cos(\frac{\Delta\pi}{2}) \end{bmatrix} \\ Y^\Delta &= \begin{bmatrix} \cos(\frac{\Delta\pi}{2}) & -\sin(\frac{\Delta\pi}{2}) \\ \sin(\frac{\Delta\pi}{2}) & \cos(\frac{\Delta\pi}{2}) \end{bmatrix} \\ Z^\Delta &= \begin{bmatrix} 1 & 0 \\ 0 & e^{i\Delta\pi} \end{bmatrix} \end{aligned}$$

These gates are known as the **Rotation Operators** and form the foundation for all single qubit transformations. Every other single qubit gate can be expressed as a combination of these three gates. Technically, only 2 of the 3 are necessary because one can be expressed from the other two. For example, an X rotation can be decomposed into:

$$\begin{aligned}
X^\Delta &= Z^{-0.5}Y^\Delta Z^{0.5} \\
&= \begin{bmatrix} 1 & 0 \\ 0 & e^{-i0.5\pi} \end{bmatrix} \begin{bmatrix} \cos(\frac{\Delta\pi}{2}) & -\sin(\frac{\Delta\pi}{2}) \\ \sin(\frac{\Delta\pi}{2}) & \cos(\frac{\Delta\pi}{2}) \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & e^{i0.5\pi} \end{bmatrix} \\
&= \begin{bmatrix} \cos(\frac{\Delta\pi}{2}) & -e^{i0.5\pi}\sin(\frac{\Delta\pi}{2}) \\ e^{-i0.5\pi}\sin(\frac{\Delta\pi}{2}) & e^{i0.5\pi}e^{-i0.5\pi}\cos(\frac{\Delta\pi}{2}) \end{bmatrix} \\
&= \begin{bmatrix} \cos(\frac{\Delta\pi}{2}) & -i\sin(\frac{\Delta\pi}{2}) \\ -i\sin(\frac{\Delta\pi}{2}) & \cos(\frac{\Delta\pi}{2}) \end{bmatrix}
\end{aligned}$$

The special case when $\Delta = 1$ produces the set of **Pauli Gates** shown below. Another common system of notation for the Pauli gates is $\sigma_x, \sigma_y, \sigma_z$.

$$\begin{aligned}
X &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \\
Y &= \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \\
Z &= \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}
\end{aligned}$$

All of the Pauli gates are their own inverse, which means:

$$X^2 = Y^2 = Z^2 = -iXYZ = I$$

When all three of the Pauli matrices are multiplied together, it produces the identity matrix, except for the $-i$ scalar attached in front. The $-i$ is a global phase! This global phase is the same reason that the Pauli X and Y gate differ from what the rotation gate produces when $\Delta = 1$:

$$X^1 = \begin{bmatrix} \cos(\frac{\pi}{2}) & -i\sin(\frac{\pi}{2}) \\ -i\sin(\frac{\pi}{2}) & \cos(\frac{\pi}{2}) \end{bmatrix} = \begin{bmatrix} 0 & -i \\ -i & 0 \end{bmatrix} = -i\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Global phase can apply to a quantum gate as well, this means different matrices can perform the same operation. The gates U_1 and U_2 are equivalent if they can be expressed as $U_1 = e^{i\gamma}U_2$ for some value γ.

Common Gates

Here are some common gates and their adjoints which appear in quantum computing literature:

$$
\begin{aligned}
S = \sqrt{Z} &= \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \\
S^\dagger = Z\sqrt{Z} &= \begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix} \\
T = \sqrt{S} = \sqrt[4]{Z} &= \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{\sqrt{2}}(1+i) \end{bmatrix} \\
T^\dagger = ZS\sqrt{S} &= \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{\sqrt{2}}(1-i) \end{bmatrix} \\
\sqrt{NOT} = \sqrt{\neg} = \sqrt{X} &= \frac{1}{2}\begin{bmatrix} 1+i & 1-i \\ 1-i & 1+i \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & -i \\ -i & 1 \end{bmatrix} \\
\sqrt{NOT}^\dagger = \sqrt{\neg}^\dagger = X\sqrt{X} &= \frac{1}{2}\begin{bmatrix} 1-i & 1+i \\ 1+i & 1-i \end{bmatrix} = \frac{-1}{\sqrt{2}}\begin{bmatrix} 1 & i \\ i & 1 \end{bmatrix}
\end{aligned}
$$

A rotation matrix's adjoint is equivalent to rotating clockwise instead of counterclockwise. That is to say, ${X^\Delta}^\dagger = X^{-\Delta} = X^{2-\Delta}$, since rotating one way by Δ and then immediately rotating the other way by Δ will produce an identity operation.

Floating Point Rotations

Classical computers don't always store precise values. For a value like π, it would require an infinite number of bits to perfectly represent. Instead, classical computers store estimations using a fixed number of bits which are called *floating point numbers*. It may not be possible to rotate around the Bloch sphere by an arbitrary amount with perfect precision, so we will have to use approximations.

The **Solovay-Kitaev Theorem** implies any single qubit gate can be approximated using $O(\log^c(\frac{1}{\epsilon}))$ gates from a finite set. The value c is a small constant, roughly 2. The minimum possible value of c has not been proven. The value $0 < \epsilon < 1$ represents how accurate the approximation is. Since our rotation gates take the Δ parameter, any set containing the rotation gate will be infinite. An example of a finite set would be $\{\sqrt{X}, \sqrt{Y}, T\}$ where the Δ parameter is constant for every gate in the set. The only requirement for the finite set is that it must *fill* P^1.

If a finite set of gates S fills P^1, then for every pair of quantum states, there must exist some sequence of gates $U_1, U_2, \ldots, U_n \in S$ which links them.

$$S \text{ fills } P^1 \leftrightarrow \forall|\psi\rangle\ \forall|\psi'\rangle\ \exists(U_1, U_2, \ldots, U_n) \in S \quad U_n \cdots U_2 U_1 |\psi\rangle = |\psi'\rangle$$

The elements of the sequence do not need to be unique. This requirement of a filling set is not satisfied by all sets of gates. For example the set $\{X^{0.1}\}$ does not fill P^1. In fact, no value of Δ on a single axis will produce a filling set. This is because some quantum state $|\psi\rangle$ will never be able to leave the ring generated by $\operatorname{span}(X)$, so there are states which cannot be accessed.

Another example of a non-filling set is $\{\sqrt{X}, \sqrt{Y}, \sqrt{Z}\}$. This one is slightly deceptive because for some generic state $|\psi\rangle$, it appears to fill P^1. However, consider the state $|0\rangle$. There are only 6 states which are accessible: two states for each axis where the axis intersects the Bloch sphere. Conversely, it is also impossible to transform a generic state $|\psi\rangle$ into $|0\rangle$. Since there exists a state which cannot access every other state, this set does not fill P^1.

Checking if a set fills P^1 can be reduced to a graph problem. Suppose there is an infinite graph with one vertex per quantum state, and each vertex as n edges, one for each gate in the set. A vertex may have a loop, where an edge connects the vertex to itself. If the graph has two or more *isolated subgraphs*, then the set does not fill P^1. Isolated subgraphs, also called *components*, are a collection of vertices which cannot be connected to the rest of the graph.

This theorem is very important because it allows us to create arbitrary rotations out of a finite set of gates, and do it efficiently. Physical implementations of quantum computers may only be able to use certain fixed gates due to the immense engineering challenges present in building the hardware. Certain methods of error correction do not work for the continuous rotation gates, so approximating them from a finite set is required.

3.3 Alternate Axes

We have discussed rotating around the three major axes: X, Y, and Z. However, there is no reason why we can't rotate around other axes. A *minor axis* is a line defined by the origin and a single point. Technically, the data representing an axis is the same as the quantum state, since it is just a vector. For example, consider the W axis which is defined to be $(X+Y)/\sqrt{2}$. This is a unit vector which lies in the XY plane, it represents the line where $X=Y$. Just like the major axis rotations, there is also a matrix for rotating around the W axis:

$$W^{\Delta}=\begin{bmatrix}\cos(\frac{\Delta\pi}{2}) & -\frac{1}{\sqrt{2}}\sin(\frac{\Delta\pi}{2})(1+i)\\ \frac{1}{\sqrt{2}}\sin(\frac{\Delta\pi}{2})(1-i) & \cos(\frac{\Delta\pi}{2})\end{bmatrix}$$

The matrix for when $\Delta=0.5$ is:

$$\sqrt{W}=\begin{bmatrix}\frac{1}{\sqrt{2}} & -\frac{1}{2}-\frac{1}{2}i\\ \frac{1}{2}-\frac{1}{2}i & \frac{1}{\sqrt{2}}\end{bmatrix}=\frac{1}{\sqrt{2}}\begin{bmatrix}1 & -\sqrt{i}\\ \sqrt{-i} & 1\end{bmatrix}$$

Although the W axis is not frequently seen in quantum computing literature, it provides a nice example of a minor axis rotation matrix.

The Hadamard Axis

The **Hadamard Gate**, denoted H, is one of the most common gates used in quantum computing, and it performs the following operation:

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

The Hadamard gate is synonymous with generating superposition because it transforms both basis states into equal superposition. The state $|0\rangle$ is mapped to the positive X axis and the state $|1\rangle$ is mapped to the negative X axis. This operation is exactly what is performed by *beam splitters*. A beam splitter is a half-silvered mirror which will split an incoming stream of photons, reflecting one stream and letting the other stream pass through.

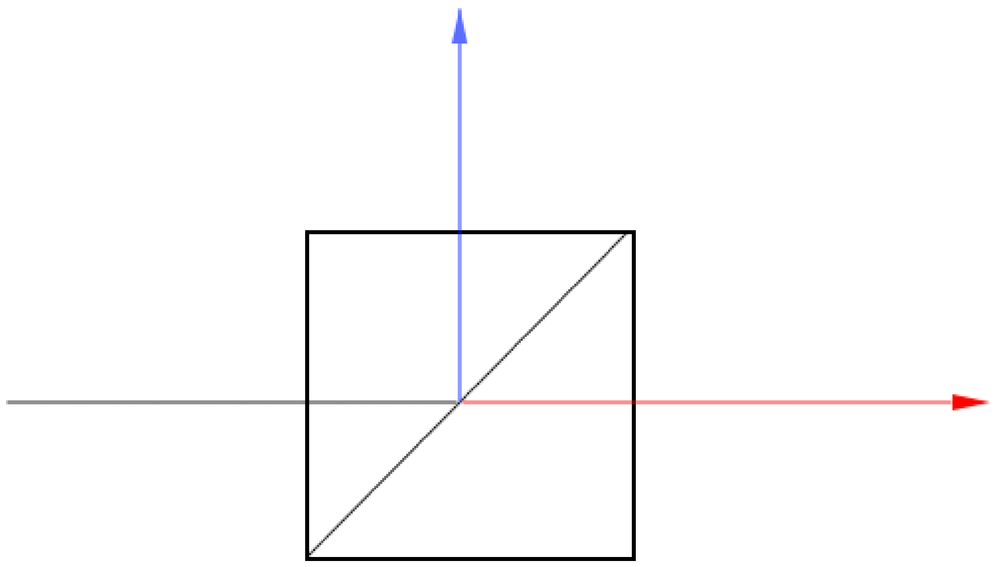


Figure 3.3.1 - Beam Splitter Diagram

In the above diagram, the beam of light is coming from the left and splits into two separate beams. The half-silvered mirror is placed diagonally in the box. We can place two beam splitters and two mirrors in a configuration called a **Mach-Zender Interferometer** and add detectors for each of the projected output beams.

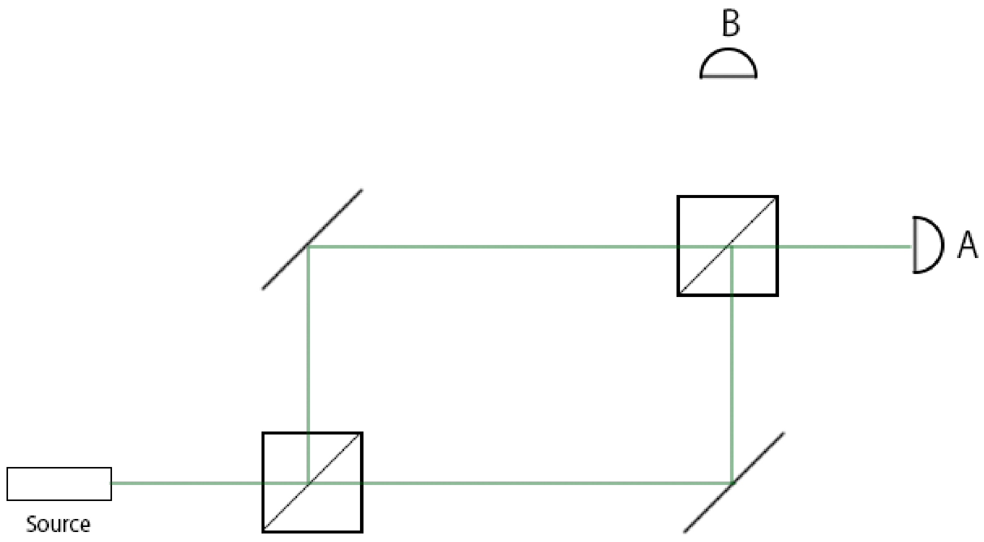


Figure 3.3.2 - Mach-Zender Interferometer

Using the configuration of beam splitters and mirrors described above, say we produce 100 photons. How many of those photons will be detected by B?

The answer is zero. All of the photons will be detected by A, the detector at B will never see a photon. This result seems unintuitive, if a beam splitter splits a stream of photons, why doesn't the second half-silvered mirror split both streams again? This is because $H^2 = I$. In this experiment, the photon acted as a qubit, initialized to the state $|0\rangle$. After passing through the first beam splitter, the qubit entered superposition, but came back out of superposition after the second beam splitter. The Hadamard gate's matrix representation is:

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

$$HH|0\rangle = |0\rangle$$

The H gate can be decomposed into $H = \sqrt{Y}Z = X\sqrt{Y}$. Specifically, the Hadamard gate is a rotation of $\Delta = 1$ around the Hadamard axis, $H = (X + Z)/\sqrt{2}$. The H axis is a line in the XZ plane where $X = Z$. Just like the W axis, there is a matrix which generalizes a rotation of any amount:

$$H^{\Delta} = \begin{bmatrix} \cos(\frac{\Delta\pi}{2}) - \frac{i}{\sqrt{2}}\sin(\frac{\Delta\pi}{2}) & -\frac{i}{\sqrt{2}}\sin(\frac{\Delta\pi}{2}) \\ -\frac{i}{\sqrt{2}}\sin(\frac{\Delta\pi}{2}) & \cos(\frac{\Delta\pi}{2}) + \frac{i}{\sqrt{2}}\sin(\frac{\Delta\pi}{2}) \end{bmatrix}$$

These beam splitters are more than neat party tricks. Using the **KLM Protocol**, it is possible to create universal quantum computers using just mirrors, half-silvered mirrors, and phase shifters.

Arbitrary Axes

The next step is to rotate our quantum state around any arbitrary axis. This is an applet which shows the accessible state space of rotating the state around any axis. The sliders on the left change the θ and ϕ of the quantum state and the sliders on the right change the θ and ϕ of the axis. The axis is set to the Hadamard axis by default.

Applet 3.3.3 Arbitrary Axis Rotations Applet

The matrix for rotating around any axis $\hat{n}$ can be written as the sum of the identity matrix and the Pauli matrices:

$$\hat{n}^{\Delta} = \cos(\frac{\Delta\pi}{2})I - i\sin(\frac{\Delta\pi}{2})(n_x X + n_y Y + n_z Z)$$

This is the equation used to derive the rotation around the W and H axes described above. Alternatively, the same matrix can be written as the product of the Y and Z rotation matrices:

$$a = \frac{n_\phi}{\pi}$$
$$b = \frac{n_\theta}{\pi}$$
$$\hat{n}^{\Delta} = Z^a Y^b Z^{\Delta} Y^{-b} Z^{-a}$$

Note how the sum equation uses the axis' Euclidean coordinates and the product equation uses the axis' spherical coordinates.

Since a quantum state is mathematically equivalent to a vector, that implies we could use a quantum state as an axis to rotate around. For example, rotating around itself is always an identity operation:

$$\psi^{\Delta}|\psi\rangle = I$$

However, we don't actually know the θ and ϕ of some generic quantum state. To rotate around an axis, the coordinates of the axis must be known, otherwise how would we know which gates to apply?

Single Qubit Gate Decomposition

Decomposing a rotation around an arbitrary axis into 5 major axis rotations is decent, but we can take it one step further. In the previous chapter, we stated that $\mathrm{span}(ZYZ)$ was complete with respect to P^1. This means any generic single qubit gate U can be decomposed into three gates and a global phase: $e^{ia}Z^{\Delta_1}Y^{\Delta_2}Z^{\Delta_3}$

$$\begin{aligned}
U &= \begin{bmatrix} r_1 e^{i\varphi_1} & r_2 e^{i\varphi_2} \\ r_3 e^{i\varphi_3} & r_4 e^{i\varphi_4} \end{bmatrix} \\
&= \begin{bmatrix} \cos(\frac{\Delta_2\pi}{2})e^{ia} & -\sin(\frac{\Delta_2\pi}{2})e^{ia+i\Delta_3\pi} \\ \sin(\frac{\Delta_2\pi}{2})e^{ia+i\Delta_1\pi} & \cos(\frac{\Delta_2\pi}{2})e^{ia+i\Delta_1\pi+i\Delta_3\pi} \end{bmatrix} \\
&= e^{ia}\begin{bmatrix} 1 & 0 \\ 0 & e^{i\Delta_1\pi} \end{bmatrix}\begin{bmatrix} \cos(\frac{\Delta_2\pi}{2}) & -\sin(\frac{\Delta_2\pi}{2}) \\ \sin(\frac{\Delta_2\pi}{2}) & \cos(\frac{\Delta_2\pi}{2}) \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & e^{i\Delta_3\pi} \end{bmatrix} \\
&= e^{ia}Z^{\Delta_1}Y^{\Delta_2}Z^{\Delta_3}
\end{aligned}$$

We can find the rotation parameters using the following equations:

$$\begin{aligned}
a &= \varphi_1 \\
\Delta_1 &= \frac{\varphi_3 - \varphi_1}{\pi} \\
\Delta_2 &= \frac{2}{\pi}\mathrm{atan2}(r_1, r_3) \\
\Delta_3 &= \frac{\varphi_2 - \varphi_1}{\pi}
\end{aligned}$$

Applying this method of gate decomposition to our rotation around an arbitrary axis, we can shorten it to 3 gates instead of 5. The first step is to get the arbitrary axis formula into one matrix so we can create our r and φ variables.

$$\begin{aligned}
\hat{n}^{\Delta} &= \cos(\frac{\Delta\pi}{2})I - i\sin(\frac{\Delta\pi}{2})(n_xX + n_yY + n_zZ) \\
&= \begin{bmatrix} \cos(\frac{\Delta\pi}{2}) - in_z\sin(\frac{\Delta\pi}{2}) & -n_y\sin(\frac{\Delta\pi}{2}) - in_x\sin(\frac{\Delta\pi}{2}) \\ n_y\sin(\frac{\Delta\pi}{2}) - in_x\sin(\frac{\Delta\pi}{2}) & \cos(\frac{\Delta\pi}{2}) + in_z\sin(\frac{\Delta\pi}{2}) \end{bmatrix}
\end{aligned}$$

The next step is to create the r and φ variables:

$$\begin{aligned}
r_1 &= \sqrt{\cos^2(\frac{\Delta\pi}{2}) + n_z^2\sin^2(\frac{\Delta\pi}{2})} \\
r_3 &= \sin^2(\frac{\Delta\pi}{2})\sqrt{n_x^2 + n_y^2} \\
\varphi_1 &= \mathrm{atan2}(\cos(\frac{\Delta\pi}{2}), -n_z\sin(\frac{\Delta\pi}{2})) \\
\varphi_2 &= \mathrm{atan2}(-n_y\sin(\frac{\Delta\pi}{2}), -n_x\sin(\frac{\Delta\pi}{2})) \\
\varphi_3 &= \mathrm{atan2}(n_y\sin(\frac{\Delta\pi}{2}), -n_x\sin(\frac{\Delta\pi}{2}))
\end{aligned}$$

These variables can simply be substituted into the equations for the Δ variables to find how much to rotate around the ZYZ axes. The formulas for rotation with 3 gates are significantly less clean than the 5 gate version. Since modern quantum computers are slower and less accurate compared to classical computers, it is worth the extra steps on the classical side to reduce the number of gates the quantum computer needs to apply.

In the previous chapter, we also stated that $\mathrm{span}(YZ) \cup \mathrm{span}(ZY)$ is complete with respect to P^1 , which implies any single qubit gate could be reduced even further to 2 gates, if the order could be swapped. However, this is not done because the order of the gates is state-dependent, and the current state to be rotated is not known at compile time, when the substitution is deployed.

Furthermore, the actual state of the qubit is not known until measurement, so we can't know how to order the gates until after they have already been applied.

3.4 Quantum Registers

Up to this point, we have used the term *quantum system* interchangeably with the term *qubit* because we have only focused on a single qubit. However, a quantum system is a more general concept, it refers to all quantum objects of interest to us in a particular setting. When we have collection of qubits, it is called a *quantum register* and they are grouped together under one quantum system.

$$|\psi\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$$

Above is an example of a two qubit quantum system in equal superposition. The $|\psi\rangle$ is still used to describe the entire quantum register, although now there are more *basis states*. Rather than just having $|0\rangle$ and $|1\rangle$, there are now four basis states to describe every possible combination of states the qubits could collapse into. Each basis state also has a complex coefficient assigned to it, the probability of the quantum system collapsing into a particular basis state i can be found: $P(|\psi\rangle = |i\rangle) = |c_i|^2$. In general, a quantum register of n qubits can be described in ket notation:

$$\begin{gathered} |\psi\rangle = \sum_{i=0}^{2^n} c_i |i\rangle \\ \text{where } \sum_{i=0}^{2^n} |c_i|^2 = 1 \end{gathered}$$

Suppose we are given two separate qubits, how do we create our quantum system to describe both of them?

$$\begin{aligned} |\psi_0\rangle &= \alpha_0|0\rangle + \beta_0|1\rangle \\ |\psi_1\rangle &= \alpha_1|0\rangle + \beta_1|1\rangle \\ |\psi\rangle = |\psi_1\rangle|\psi_0\rangle &= (\alpha_1|0\rangle + \beta_1|1\rangle) \otimes (\alpha_0|0\rangle + \beta_0|1\rangle) \\ &= \alpha_1\alpha_0|0\rangle|0\rangle + \alpha_1\beta_0|0\rangle|1\rangle + \beta_1\alpha_0|1\rangle|0\rangle + \beta_1\beta_0|1\rangle|1\rangle \\ &= \alpha_1\alpha_0|00\rangle + \alpha_1\beta_0|01\rangle + \beta_1\alpha_0|10\rangle + \beta_1\beta_0|11\rangle \end{aligned}$$

The key to merging two separate qubits into one quantum system is the *tensor product*. This operation is denoted by the $\otimes$ symbol and it looks similar to distributing two binomials. There are different notations for the tensor product, but they all mean the same thing:

$$|\psi_1\rangle \otimes |\psi_0\rangle = |\psi_1\rangle|\psi_0\rangle = |\psi_1\psi_0\rangle$$

We have described how the tensor product works using the Dirac notation, but quantum states can also be described using vectors:

$$|\psi_1\rangle \otimes |\psi_0\rangle = \begin{bmatrix} \alpha_1 \\ \beta_1 \end{bmatrix} \otimes \begin{bmatrix} \alpha_0 \\ \beta_0 \end{bmatrix} = \begin{bmatrix} \alpha_1 \begin{bmatrix} \alpha_0 \\ \beta_0 \end{bmatrix} \\ \beta_1 \begin{bmatrix} \alpha_0 \\ \beta_0 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \alpha_1\alpha_0 \\ \alpha_1\beta_0 \\ \beta_1\alpha_0 \\ \beta_1\beta_0 \end{bmatrix}$$

In general, a tensor product between a vector with n rows and another vector with m rows produces a vector with nm rows. The tensor product is associative, but it is not communitive, so $|\psi_0\psi_1\rangle \neq |\psi_1\psi_0\rangle$. For this reason, we need to be clear about the ordering of the qubits. The standard convention is to have the rightmost qubit be the least significant bit. This means the state $|100\rangle$ will represent the number 4 instead of the number 1. Since the quantum state is being created from an array of qubits, the indexes follow zero-based numbering. The least significant qubit is described as the zeroth qubit in the register.

Here is an algorithm which will create the amplitude at index row after the tensor product of n qubits:

```
def create_amplitude(row_index, qubits):
    amplitude = 1
    for i in n:
        if ((row_index & 1 << i) == 0):
            amplitude *= qubits[i].alpha
        else:
            amplitude *= qubits[i].beta
    return amplitude
```

Consider the index 01001 in a 5 qubit register. The amplitude at this position is $\alpha_4\beta_3\alpha_2\alpha_1\beta_0$. Each 0 in the index corresponds to the amplitude being multiplied by that qubit's α component and each 1 means the amplitude is multiplied by that qubit's β component.

The algorithm above iterates through each bit in the index and multiplies the amplitude by the corresponding component depending if the bit is 0 or 1. This algorithm uses *bitwise operators* to determine the state of a bit inside the index. The following bitwise operators are frequently used:

- `&` - Bitwise AND
- `|` - Bitwise OR
- `^` - Bitwise XOR
- `<<` - Bitwise left shift
- `>>` - Bitwise right shift

Quantum Circuits

When describing which gates are being applied to which qubits, using the Dirac notation can become cluttered. *Quantum circuits* are a visual way to show the ordering of the gates. Below is an applet to show a simple quantum circuit with 2 qubits and an X, Y, and Z rotation gate on each.

Applet 3.4.1 Simple Quantum Circuit Applet

Starting from the left, first there are the inputs to the quantum circuit. Typically, a quantum circuit is initialized with $|0\rangle$ for each qubit, but sometimes a generic state $|\psi\rangle$ is the input when comparing multiple circuits. The top qubit is the least significant qubit which corresponds to the rightmost qubit in the Dirac notation.

Next are all of the gates in the circuit. Each wire runs horizontally and contains a sequence of gates which are applied to the qubit. Unlike the matrix form of gates, the gates in a circuit are read left-to-right. In our simple circuit, the X gate is applied first, followed by the Y gate and then the Z gate. For this example, each gate has a slider which controls the Δ parameter of that gate.

The square on the right side is a visual representation of the quantum state. Each sub-box represents a basis state, the size of the circle represents the magnitude and the orientation of the circle represents the phase. The global phase is automatically adjusted so the $|0\rangle$ state remains real.

This example is just a simple quantum circuit for demonstration purposes. A more advanced quantum circuit builder is available through Quirk.

Matrix Tensor Product

A 2-qubit quantum system can be created by using the tensor product on two vectors. How do we apply a gate to this larger vector? Our 2×2 matrices cannot be directly applied to a vector with 4 rows, there would be a dimension mismatch. The key is to use the tensor product on the gate as well!

$$|\psi_1\rangle \otimes H|\psi_0\rangle = (I \otimes H)|\psi_1\psi_0\rangle$$

If we would like to apply a Hadamard gate on the right qubit and nothing on the left qubit, it is the tensor product between the H and I matrices. In general, a tensor product between two matrices takes the form:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \otimes \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a\begin{bmatrix} e & f \\ g & h \end{bmatrix} & b\begin{bmatrix} e & f \\ g & h \end{bmatrix} \\ c\begin{bmatrix} e & f \\ g & h \end{bmatrix} & d\begin{bmatrix} e & f \\ g & h \end{bmatrix} \end{bmatrix} = \begin{bmatrix} ae & af & be & bf \\ ag & ah & bg & bh \\ ce & cf & de & df \\ cg & ch & dg & dh \end{bmatrix}$$

The tensor product for matrices is still non-communitive. In order to apply a 2×2 gate to the ith qubit, there must be n matrices tensored together:

$$|\psi_{n-1}\cdots\psi_{i+1}\rangle \otimes U|\psi_i\rangle \otimes |\psi_{i-1}\cdots\psi_0\rangle = \left(\bigotimes_{j=i+1}^{n} I\right) \otimes U \otimes \left(\bigotimes_{j=0}^{i} I\right)|\psi\rangle$$

When applying just one 2×2 gate to a quantum register, it gets tensored with $n-1$ identity matrices. This fact highlights an interesting characteristic of the tensor product. Say we would like to apply two 2×2 gates on a 2 qubit system, U_0 and U_1. We could tensor the gates together like shown with the general form of the tensor product, but we could also tensor each gate with the identity matrix like shown with the ith qubit form and then multiply:

$$U_1 \otimes U_0 = (U_1 \otimes I)(I \otimes U_0) = (I \otimes U_0)(U_1 \otimes I)$$

Tensoring each matrix with the identity matrix and them multiplying them together is equivalent to tensoring the matrices together directly. Furthermore, the matrix multiplication of $U_1 \otimes I$ and $I \otimes U_0$ is communitive! This can be generalized to any number of gates acting on a quantum register, the matrix multiplication between all of these identity-tensored gates is communitive.

3.5 Controlled Gates

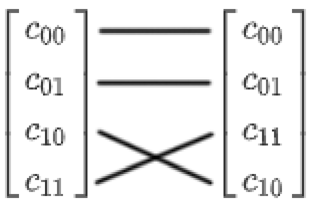
The goal of this section is to address how qubits interact with each other. So far, we have treated qubits as separate entities which can be operated on independently. To create a powerful model of computation, qubit states need to be able to impact each other. In quantum computing, we use transformations called *controlled operations* which are able to connect qubits.

Controlled Not Gate

The most fundamental controlled gate is called the *controlled not* and it takes in 2 qubits: a control qubit and a target qubit. In English, the transformation a controlled not (CNOT) gate performs is described as: *if the control qubit is* $|1\rangle$, *then flip the target qubit*. We can create a truth table which describes the operation if the left qubit is the control and the right qubit is the target

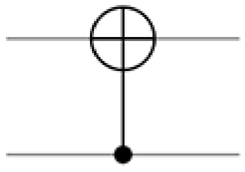
$$\begin{aligned} &|c\,t\rangle \\ &|00\rangle \rightarrow |00\rangle \\ &|01\rangle \rightarrow |01\rangle \\ &|10\rangle \rightarrow |11\rangle \\ &|11\rangle \rightarrow |10\rangle \end{aligned}$$

The above ket notation shows a truth table for every basis state. If the control qubit is $|0\rangle$, the CNOT acts as an identity operation and nothing happens. If the control qubit is $|1\rangle$, the target qubit is flipped. The same operation can be visualized in the tensored vector form:

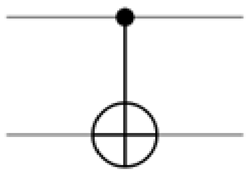


The CNOT transformation switches the placement of the $|10\rangle$ and $|11\rangle$ coefficients. The CNOT gate is also called the CX gate since it is essentially a controlled X rotation. The matrix form of the CNOT gate is written as:

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



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



Notice how the transformation the CNOT applies changes depending which qubit is acting as the control. In circuit form, the small black dot denotes the control qubit and the large $\oplus$ shows the target qubit. Remember that we are using the standard where the topmost qubit line is the least significant qubit. The truth table and the tensored vector descriptions are referring to the top circuit where the control qubit is below the target qubit. Some sources use a different standard where the bottom-most qubit is the least significant qubit, which would switch the matrix-circuit representations.

The control and target qubits cannot be the same qubit in a CNOT gate. Such an operation would create the following truth table, which is not reversible:

$$\begin{gathered} |0\rangle \rightarrow |0\rangle \\ |1\rangle \rightarrow |0\rangle \\ \text{Not a reversible operation!} \end{gathered}$$

One question students often ask is "why use the $\oplus$ symbol for the CNOT gate?" In computational logic, the $\oplus$ symbol refers to the exclusive or (XOR) of two bits. The controlled-not gate is a reversible version of XOR, the control qubit is unaffected and the target qubit becomes the XOR of the control and the target:

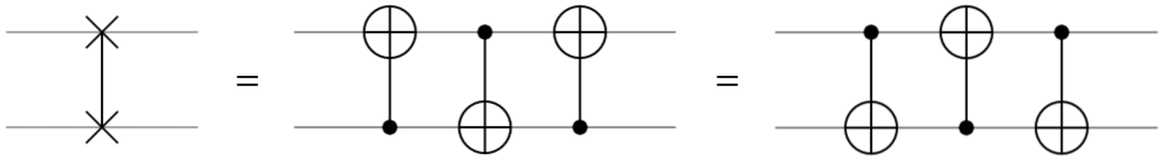
$$|c\rangle|t\rangle \rightarrow |c\rangle|c \oplus t\rangle$$

SWAP Gate

An interesting property of the XOR operation is that it can swap the values of two variables without using a temporary variable. Consider the binary variables X and Y and perform the following algorithm on them:

$$\begin{aligned} X &:= X \oplus Y \\ Y &:= Y \oplus X \\ X &:= X \oplus Y \end{aligned}$$

After 3 XORs, the variable X will contain the original value of Y and vice versa. As we have already discussed, the CNOT gate is the reversible equivalent to the XOR gate, which means we can create a SWAP gate out of 3 CNOT gates:



This gate switches the incoming qubit states so the bottom qubit now contains the original state of the top qubit and vice versa. The SWAP gate is symmetric with respect to the qubits it is swapping. That is to say, $\text{SWAP}(|\psi\rangle, |\varphi\rangle) = \text{SWAP}(|\varphi\rangle, |\psi\rangle)$. As expected, the SWAP gate is its own inverse, so two sequential SWAP gates is an identity operation. The matrix of the SWAP gate is:

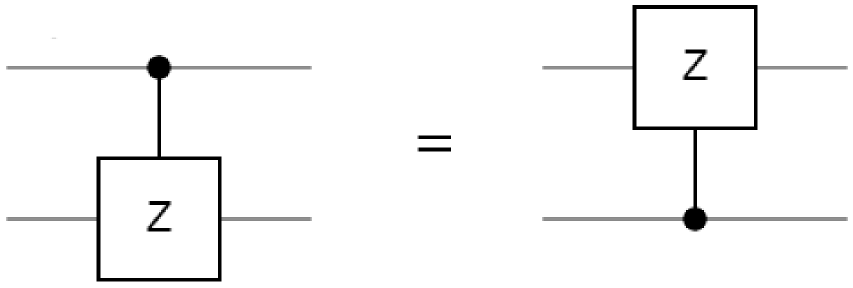
$$\text{SWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Controlled Z Gate

Similar to how the CNOT gate is a controlled X rotation of $\Delta = 1$, we can also create the CZ gate for a controlled Z rotation. The CZ gate performs the following operation:

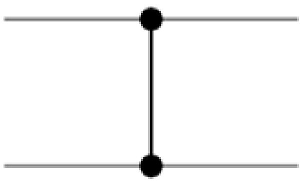
$$\begin{aligned} &|c\,t\rangle \\ &|00\rangle \rightarrow \quad |00\rangle \\ &|01\rangle \rightarrow \quad |01\rangle \\ &|10\rangle \rightarrow \quad |10\rangle \\ &|11\rangle \rightarrow -|11\rangle \end{aligned}$$

The controlled Z gate applies a phase shift only when both qubits are in the state $|11\rangle$. This operation is symmetric with respect to the input qubits:

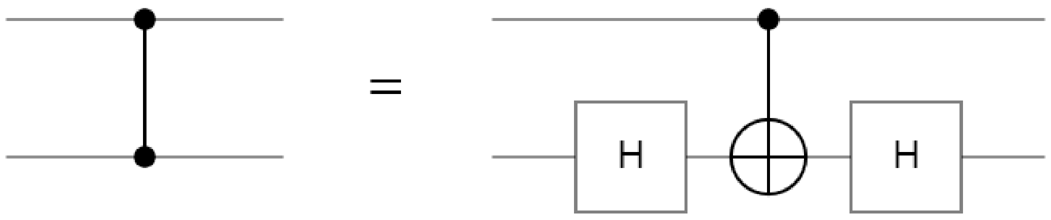


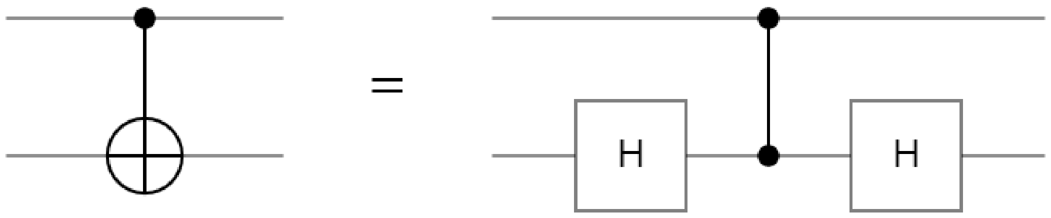
Since the orientation of the CZ gate doesn't matter, another common notation for the gate is to have control dots on both qubits. The matrix representation of the gate is:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

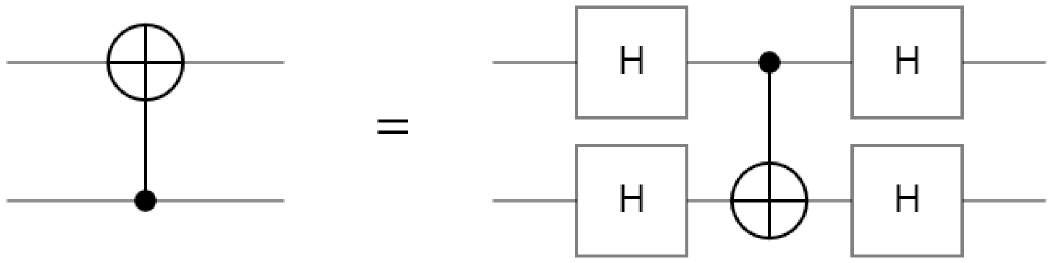


A CZ gate can be constructed using a CNOT gate and 2 Hadamard gates like so:





Using the above relationship, we can flip the CZ gate since it is symmetric and create a method to flip the CNOT gate using Hadamard gates.



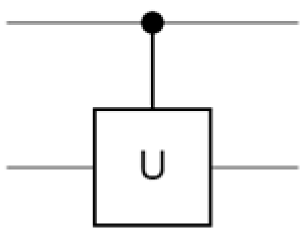
The controlled Z gate has applications in simulating quantum circuits on quantum computers. The runtime of simulating a CZ gate is faster than a CNOT gate, so if a circuit can be transformed into an equivalent circuit using approximately the same number of gates, the simulation will be noticeably faster.

Controlled U Gate

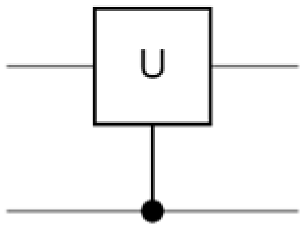
The controlled X and controlled Z gates are specific instances of the general controlled U gate. Any single qubit gate U can be transformed into a CU gate:

$$U = \begin{bmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{bmatrix}$$

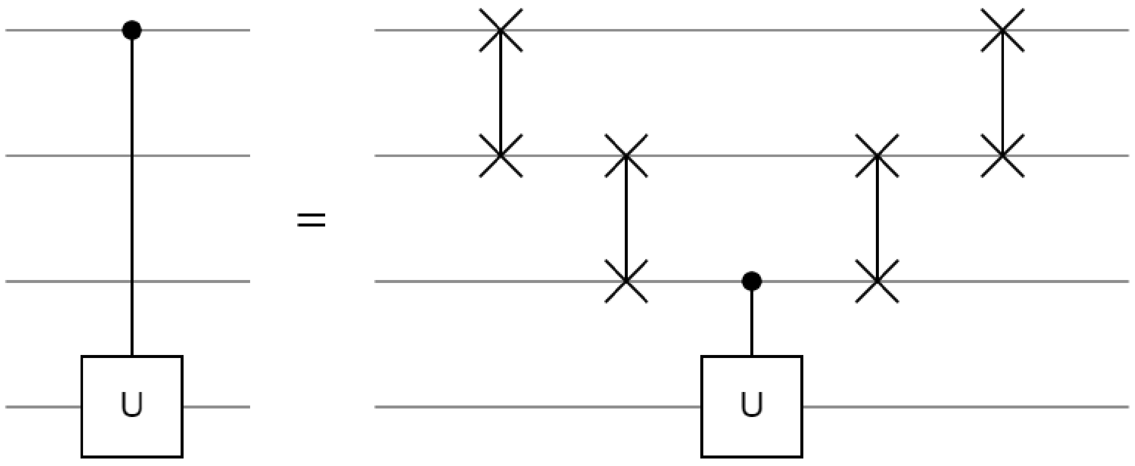
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & u_{00} & 0 & u_{01} \\ 0 & 0 & 1 & 0 \\ 0 & u_{10} & 0 & u_{11} \end{bmatrix}$$



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & u_{00} & u_{01} \\ 0 & 0 & u_{10} & u_{11} \end{bmatrix}$$



Every 2 qubit gate described above has acted on adjacent qubits inside the circuit. It is also possible to have the control further away and "jump over" qubits to reach the target. The CU gate does not impact the two qubits which do not have a control or gate placed on them. However, what if we have a constraint on our circuit that all 2 qubit gates must act on adjacent qubits? This can be overcome by simply using SWAP gates to pull the control closer:



This idea can be extended for any number of intermediate qubits by increasing the staircase of SWAP gates. Engineering physical quantum computers is already a challenging task, creating controlled gates between any pair of arbitrary qubits in the circuit may be infeasible. As a result, this trick can be used so engineers only need to implement controlled operations between adjacent qubits in the circuit.

Circuit Specifications

In the above CU circuit, originally there weren't any gates on the middle two qubits. However, there are several gates acting on them after the transformation. In this circuit, the middle two qubits are acting as *ancilla qubits*. An ancilla qubit is a qubit which has operations performed on it that do not directly contribute to the result, but help indirectly. They typically come in two flavors: identity ancilla qubits and garbage ancilla qubits.

The ancilla qubits in the CU circuit are acting as identity ancilla qubits because even though there are gates being applied to them, the end result is equivalent to the identity operation. These identity ancilla qubits were borrowed to accomplish a more complicated goal, but had the affects undone after the goal was accomplished. Garbage ancilla qubits do not have such restrictions, the results after measuring a garbage ancilla qubit are discarded. Garbage ancilla qubits are particularly useful in quantum computation for reducing the *depth* of a circuit.

The depth of a circuit refers to how long it takes to run. This is measured by the number of *timesteps* a circuit has. In one timestep, several gates can be applied to the quantum register, but two gates cannot be applied to the same qubit in a single timestep. Many quantum circuits are extremely parallelizable since multiple gates are applied in sync. The depth of a circuit can change after a quantum complier changes the circuit to fit the architecture of the computer. For example, if our quantum computer can only apply two qubit gates to adjacent qubits, the CU circuit above transforms from having a depth of 1 to a depth of 5. This is similar to how classical programs change length after being compiled for a target machine. Typically, the actual depth of a quantum circuit is not used, just the asymptotic gate depth. It is expressed using big-oh notation as a function of the input. For example, linear gate depth would be $O(n)$.

3.6 Reversible Computing

As discussed in previous sections, all quantum gates are reversible. However, the concept of reversible computation is not exclusive to quantum computers. *Reversible computing* has been studied extensively in the field of classical computer architectures in an attempt to improve the energy consumption of computers. Irreversible bit operations like AND, OR, and XOR dissipate energy and create entropy, but reversible bit operations like NOT and CNOT preserve the entropy of the system. Therefore, it is very important to know if the AND, OR, and XOR gates can be expressed in terms of NOT and CNOT operations.

Scrambled State Problem

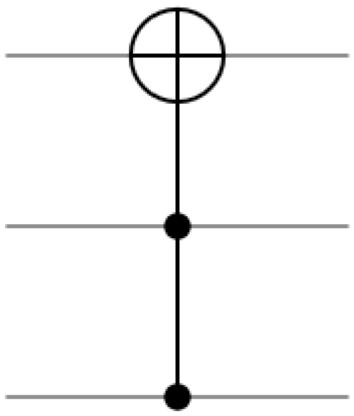
We know the AND, OR, and XOR gates are universal, which means every boolean function $f:\{0,1\}^n \to \{0,1\}$ can be implemented as a sequence of those gates. Equivalently, we can check if every bijective boolean function $f:\{0,1\}^n \to \{0,1\}^n$ can be implemented as a sequence of NOT and CNOT gates. Another way of viewing bijective boolean functions is the *scrambled state problem*. Consider a vector of length 2^n, if each element of the vector is the index the element is at, the vector is said to be unscrambled. Here is an example for $n=3$:

$$\begin{aligned} \text{unscrambled state} &= \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{bmatrix} \\ \text{scrambled state} &= \begin{bmatrix} 4 & 1 & 6 & 0 & 3 & 7 & 2 & 5 \end{bmatrix} \end{aligned}$$

If the state is scrambled, it means the order of the elements have been shuffled. Each vector has 2^n entries, which means each entry can be expressed in binary using n bits. Our goal is to take in a scrambled state and use reversible operations to unscramble it. The question remains: can we can unscramble every scrambled state using just NOT and CNOT gates? For $n=1$ and $n=2$, yes! However, not every state can be unscrambled when $n=3$. Out of the $2^3! = 40,320$ states, only $1,344$ of them can be unscrambled using NOT and CNOT gates.

How can we unscramble the remaining states? For this, we'll introduce a new gate, the **Toffoli** gate. The Toffoli gate is a controlled-controlled NOT operation, denoted as CCNOT or $C^2(X)$. If both of the control bits are on, the target bit gets flipped.

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



a

Toffoli, Fredkin

Universal set of reversible computing

Proving a quantum computer can simulate a classical computer in P time, will quantum computers completely replace classical ones?

Arbitrary num controlled U gate

3.7 Quantum Programming

After discussing the operations a quantum computer can perform to its qubits, how would we program a quantum computer? To date, a high-level quantum programming language has not yet been developed so modern quantum computers are coded with individual gates, similar to assembly for classical computers. There aren't even any independent quantum languages, most of the "languages" are just libraries accessed from classical languages.

Modern quantum programming workflow first creates a desired circuit in a quantum programming library on a classical computer. Then that circuit is sent to either a quantum computer or a quantum simulator to run. After, the result of the computation is returned to the user, typically as a histogram of measurement results. This histogram is now considered classical data and the user can use it as they please.

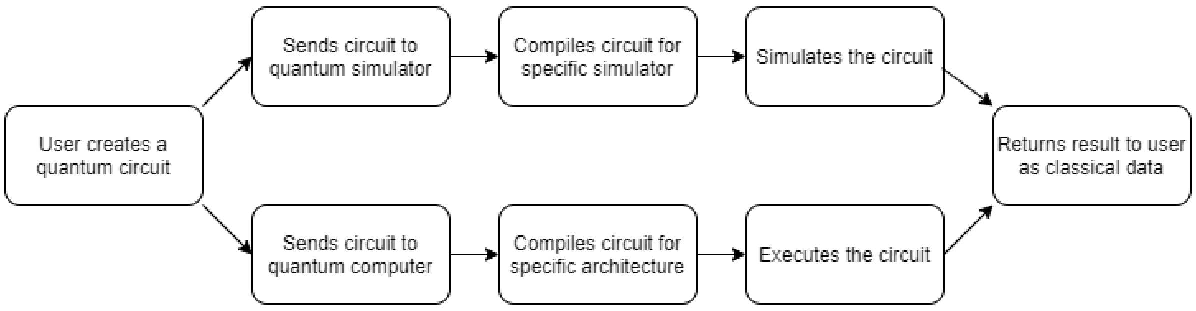

User creates a quantum circuit
Sends circuit to quantum simulator
Compiles circuit for specific simulator
Simulates the circuit
Sends circuit to quantum computer
Compiles circuit for specific architecture
Executes the circuit
Returns result to user as classical data

Figure 3.7.1 - Modern Quantum Programming Workflow

The compilers are a very important step in this process which often get overlooked. Just like classical compilers, a quantum compiler takes in some general circuit and changes the gates to run most effectively on the target platform. The platform can be a specific architecture or simulator type which handles certain types of circuits better than others.

There are many libraries used for creating quantum circuits, most companies have developed their own so it is likely there won't be a standard one until quantum computers become more ubiquitous.

Qiskit

Released in March of 2017, Qiskit is IBM's quantum computing framework for Python. Users can create circuits and use a range of tools examine them. Qiskit comes with four components: Terra, Aer, Aqua, and Ignis. Terra is element used to build circuits and send them to a real quantum computer over the cloud. Aer is a quantum simulator which can model a perfect or imperfect quantum computer. Aqua is a package of premade algorithms usable without understanding of quantum computing. Ignis is a collection of tools for understanding noise. Qiskit is the most used framework among the quantum computing community and is the one used in this document for describing circuits with code.

Q#

Released in December of 2017, Qsharp is Microsoft's quantum computing framework for C#. Currently, Q# is just a quantum simulator capable of processing up to 32 qubits on a local machine and up to 40 on Microsoft Azure, their cloud computing service. The community surrounding Q# expects Microsoft to eventually offer a hardware service similar to Qiskit using *topological qubits*. Topological qubits are a currently theoretical implementation of a qubit which is still under development.

Cirq

Released in July of 2018, Cirq is Google's quantum computing framework for Python. Similar to Q#, Cirq is just a quantum simulator with no currently announced plans of offering a physical backend to run circuits. Cirq has close ties with Google's OpenFermion project which uses quantum algorithms to simulate fermionic systems like quantum chemistry.

Chapter 4: Entanglement

swoosh

4.1 Generating Entanglement

mixed states, r<1

undistributable

EPR pair

information instantly

due to entanglement, 2 quantum computers with 50 qubits is FAR less powerful than 1 computer with 100 qubits

4.2 Bell States

Hadamard then CNOT

4 bell states

translate the bell states into each other

read 1 then know the other instantly (Maximally Entangled) but not always! (Partially Entangled)

4.3 Entangled State Space

Q state space compared to P state space

Applet

How many layers to access all states?

4.4 Gate Decomposition

Decomposing a gate into minimum universal set

CAN gate and Weyl chamber

Full state reconstruction

4.5 State Mapping

mapping a scrambled state to an unscrambled state

4.6 Case Study: Google's Quantum Supremacy

In late 2019, Google announced they had achieved *quantum supremacy* on their 53 qubit quantum processor named Sycamore, published in Nature. Quantum supremacy is the term for when a quantum program can beat a classical program at a task. What kind of task? It doesn't matter. There is no official benchmark, so the task can be any type of algorithm or can simply have a useless objective. As long as the QPU is significantly faster than a classical computer at **any** objective, quantum supremacy is declared.

What task did Google use to have their quantum computer compete in? Simulating a quantum computer. Sycamore is a quantum computer, so the simulation is just running a quantum circuit. So as long as the quantum computer can run faster than a classical computer can simulate it, the quantum computer wins. This may appear rather silly since there isn't really any objective to the computation, but that is actually why it is clever. If there was an objective to compute then there would be alternative methods for a classical programmer or patterns to exploit to make the classical program faster.

Specifically, Google called this task *random circuit sampling*. Every quantum circuit creates a probability distribution, so the objective was to find the probability distribution of a randomly generated circuit. A random circuit does not have any patterns for a classical programmer to use for time or memory exploitations. However, not all quantum circuits take the same amount of time to simulate. The hardest circuits for a classical simulator are random and all the qubits are highly entangled (but not maximally entangled)!

There are limitations on the quantum computer as well. The Sycamore processor is noisy, every gate applied causes small errors to build up and can lead to the wrong answers if too many gates are applied. So the ideal circuit should have the following properties:

- No observable patterns
- Highly entangled
- Small depth

To create circuits that have all three properties, the Google quantum team used a pseudorandom generator which uses a seed to create circuits in this format:

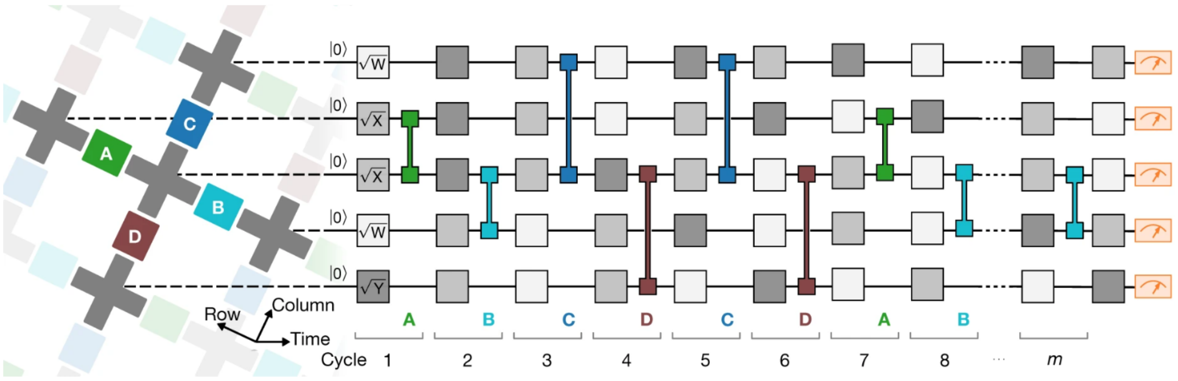


Figure 4.5.1 - Pseudorandom Circuit Architecture

The circuit is divided into *m* cycles in which every cycle has 1 single qubit gate applied per qubit followed a double qubit gate. The single qubit gates are chosen pseudo randomly and can be $\sqrt{X}$, $\sqrt{Y}$, or $\sqrt{W}$. Two gates of the same axis cannot be chosen sequentially. The $\sqrt{W}$ gate is a rotation of $\frac{\pi}{2}$ radians around the axis $(X+Y)/\sqrt{2}$. Similar to the Hadamard axis, this axis is formed by the line $X=Y$ when $Z=0$. The double qubit gates are chosen from a set sequence: repeat $ABCDCDAB$. The chosen letter represents the gate being applied to every pair of neighbor qubits connected by a colored coupler.

Applet 4.5.2 Sycamore's QPU Architecture

To understand the double qubit gates better, we can look at Sycamore's qubit layout. In figure 4.5.2, the gray crosses represent qubits and the colored boxes are couplers which connect the qubits together. The outlined cross on the top row is a qubit which does not work which is why this QPU is only 53 qubits instead of the original 54. When a double qubit gate is chosen from the set, for example A, all of the green couplers activate at the same time. So the double qubit layer in the circuit does not just refer to one pair of qubits; it refers to all pairs of qubits joined by a green coupler. Not every qubit is affected by the A gate however, since the bottom row does not have any green couplers connecting to them.

Quantum vs. Classical

Every quantum circuit essentially creates a probability distribution, but the way a quantum computer finds this distribution is different than a classical computer. A quantum computer approximates the distribution by running the circuit many times and recording the result of each measurement. After taking a large number of samples, the sampled distribution is roughly equal to the real distribution. Google's team sampled their largest circuit 30 million times, taking around 100 minutes.

Google's team used a Schrödinger/Feynman hybrid algorithm to simulate a 43 qubit circuit on a classical computer. The Schrödinger algorithm stores all 2^n coefficients in memory and the Feynman algorithm calculates each coefficient independently which only requires a polynomial amount of memory, but requires exponentially more time proportional to the depth of the circuit.

In their paper, the team made the claim that using this hybrid algorithm to simulate the largest 53 qubit circuit would take 10,000 years on the world's best supercomputers, thus proving quantum supremacy. This time estimate was created by assuming the memory was constrained to Random Access Memory (RAM). Other experts did not agree with this assumption.

IBM's Rebuttal

The quantum computing team at IBM responded to Google's claim to quantum supremacy by using a pure Schrödinger algorithm. They showed that it was possible to rotate the memory for the quantum state out to disk, only keeping the active parts in RAM. Every coefficient is 8 bytes since they are complex numbers, stored as two single precision floating point numbers. The quantum state is 8×2^{53} = 64 petabytes. That size is certainly large, but not unachievable by modern computers. IBM's team used a host of performance enhancing techniques to perform the simulation in 2.5 days.

By definition, Google's team still technically proved quantum supremacy since the quantum computer ran faster than the classical simulation (100 minutes < 2.5 days), even if it wasn't by their predicted margin of 10,000 years. However, due to the exponential nature of simulating quantum computers, a 100 qubit quantum state would take $8 \times 2^{100} \approx$ 9,000,000,000,000,000 (9 quadrillion) petabytes to store. This amount of memory is simply unachievable by modern or near future computers. Even if a 53 qubit processor strikes controversy over if it deserves quantum supremacy or not, 100 qubit processors will be developed in the near future and the outcome will be clear.

One incredibly important point brought up in IBM's response is the entire concept of quantum "supremacy". The nature of quantum processors are to be devices which coordinate with classical computers, not compete against them. The notion of claiming quantum supremacy is no less ridiculous than claiming "GPU supremacy" because a GPU was shown to be faster at one specific task. The term quantum supremacy was blown out of proportion by the media and broadly misunderstood by the general public.

Chapter 5: Quantum Networks

it cool

5.1 No-Cloning Theorem

Heisenberg uncertainty principal

Cloning via recomputing

5.2 Teleportation

not like the movies

moving a qubit state over a long distance

distributing quantum pairs ahead of time

destroys sending qubit (measures)

decoherence

5.3 Superdense Coding

teleportation in reverse

2 bits for the price of 1 (kinda)

can't use Huffman encoding in quantum, so we gotta think of new encoding schemes

5.4 Pseudo Clones

Work arounds and cloning with a chance

Chapter 6: Quantum Cryptography

BB84 and other quantum key distributions

Quantum cryptography vs post-quantum cryptography

Chapter 7: Quantum Error Correction

Up to this point we have been assuming we have 'perfect' qubits

decoherence and fidelity

probability correction

phase correction

physical qubits vs logical qubits

Chapter 8: Quantum Computer Architecture

superconducting qubits (Google, IBM, Rigetti, Intel)

  dilution refrigeration (~10mK), low vibration, magnetic shielding

  current flow

  Josephson junction

ion trap (IonQ)

  lasers - slow but stable

  qubit storage (quantum RAM), quantum hard drives don't really exist yet

Quantum annealers (D-WAVE)

  different qubits than the gate model. Low degrees of freedom comparatively

Quantum walks on graphs

  wacky

Chapter 9: Quantum Compilers

phase normal form

#of cnots in circuit > (n-1)!

parallelization & race conditions

Chapter 10: Simulating Quantum Computers

we trying

10.1 Schrödinger State Simulation

matrix form

algorithm form: bitwise CNOT & bitwise X&Y&Z

  cubeOffset instead of cubeDepth

  phase != 0 instead of == 1

storing 2x2 gates as ZYZ rotations instead of 4 complex numbers, more memory efficient

10.2 Schrödinger State Optimizations

GPU optimization

10.3 Feynman Path Integrals

simulation in linear memory - each final amplitude separate. Depends on 2^d where d is depth

10.4 Hybrid Algorithms

tensor contraction

10.5 Quantum State Estimation

The pervious algorithms described how to perfectly simulate a quantum computer on a classical one. The problems in simulating a quantum computer arise from the exponential computing cost in terms of the circuit size. There is another approach to the problem: what if we don't require the simulation to be perfect, but close enough? The algorithm described in this section creates an estimation of the final quantum state which runs much faster than a perfect simulation.

First choose hyperparameters c and k based on how powerful the simulating computer is. c is a memory constraint, approximately 20-25 for a modern desktop. k is a time constraint, approximately 30-35.

Begin simulating the circuit using Schrodinger state simulation, one state per qubit. Only tensor states together when necessary, as a 2 qubit gate requires it. Halt Schrodinger state simulation when a gate would create a tensor larger than 2^c. Switch to Feynman path simulation. Choose the largest 2^c amplitudes out of the entire state, which can be found from all the separate tensors. Simulate those amplitudes through k superposition gates using Feynman path simulation.

After the first two steps, we have created a collection of the largest 2^c amplitudes. These amplitudes have a plurality of the probability mass compared to any other group of 2^c amplitudes, which theoretically means they hold the most information. We assume (incorrectly, but efficiently) that the rest of the amplitudes follow a uniform distribution, each of them close to zero. We then continue to simulate these 2^c amplitudes, dropping an amplitude if it becomes too small, and promoting an amplitude if it becomes large enough.

After every gate in the circuit has been processed, we ideally have the 2^c largest amplitudes of the actual state. If the circuit obeys our assumption that the remainder of the amplitudes follow a uniform distribution for each gate, then we legitimately do have the actual state's largest amplitudes. However, that assumption is almost always wrong since there will be slight variations between the amplitudes, which is why the result is an estimation. The estimation's accuracy is dependent on how close remaining amplitudes are to uniformity.

There is a question we need to ask ourselves: what is the purpose of simulating a quantum circuit? In most use cases, the entire resulting quantum state is not useful, we only care about the largest amplitudes since those are the results of whatever we were trying to compute. For lots of quantum algorithms, the answer we are seeking is in the largest 10 amplitudes. When we run this quantum state estimation algorithm, we are hoping the real state's largest 10 amplitudes are captured in the largest 2^c estimated amplitudes.

This simulation algorithm relies on the idea that the problem the circuit is solving is in the NP complexity class. That is, a solution to the problem can be verified in polynomial time. Once the largest 2^c estimated amplitudes have been found, each of them are considered a potential solution to the problem, so they all must be verified. Starting with the largest amplitude and working down, if an amplitude has been verified as correct, the estimation is considered to be a success. This algorithm is not guaranteed to produce a success for every circuit. Increasing the values of c and k make the likelihood of a success higher, but cannot guarantee a success until c equals the number of qubits or k equals the number of superposition gates in the circuit.

Why begin with Schrodinger and Feynman style simulation for the first two steps before switching to an estimation algorithm? For one reason, it is more efficient to begin with the other styles until the exponential nature becomes too difficult to compute. The quantum state estimation algorithm has time complexity $\Theta(d)$ where d is the number of gates and space complexity $\Theta(1)$. It seems odd to say an exponential style of simulation is more efficient, but it is true for low gate depth. The time and space complexities of the estimation algorithm are deceptive because it ignores the very high constants defined by the c hyperparameter.

The other reason why the other two styles of simulation are used before switching to estimation is for accuracy. The estimation assumes all other amplitudes not currently tracked are in a uniform distribution, which is a bad assumption for low gate counts. It is difficult to get a good estimation of lightly entangled states using this method, heavily entangled states produce better estimations.

Since this algorithm is an estimation, it can be considered a form of noise. Future research should be conducted to see if error correcting circuits can be designed specifically for this type of noise to increase the accuracy and perhaps the efficiency of the estimation.

qiskit universal gate form

deferred form partitioning

Some sets of universal gates are faster than others

Chapter 11: Quantum Algorithms

General format of a quantum algorithm: classical preprocessing, quantum circuit, classical postprocessing

General goal of a quantum algorithm: Minimize probability of 'wrong' answers & maximize probability of 'right' answers

QFT, Shors, Grovers, Deutsch-Jozsa Problem, Simons Problem, etc.

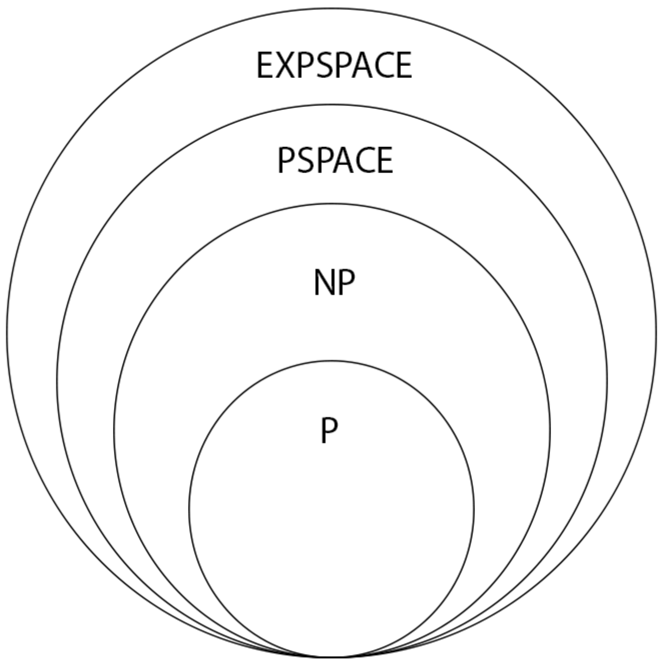


Figure 1.2.1 Common Complexity Classes

The study of *computational complexity classes* is from the theory side of computer science which describes how long a computer program takes to solve a certain problem based off the size of the input.

BQP complexity class

https://www.cs.virginia.edu/~robins/The_Limits_of_Quantum_Computers.pdf

Chapter 12: Quantum Artificial Intelligence

I know very little right now

Chapter 13: Chemical Simulation

I know **nothing** right now, but it seems supa cool

Bibliography

Resources ordered by the chapter they are most heavily related to. Referencing format: CSE Manual.

Chapter 1

a

Chapter 2

Nielsen M, Chuang I. 2010. Quantum Computation and Quantum Information. Cambridge (England): Cambridge University Press.

Sutor R. 2019. Dancing with Qubits. Birmingham (England): Packt.

Chapter 3

Kitaev A. 1997. Quantum computations: algorithms and error correction. Russian Mathematical Surveys. 52(6):1191-1249.

Ekert A, Hayden P, Inamori H. 2008. Basic Concepts in Quantum Computation. arXiv.

Zetie K, Adams S, Tocknell R. 2000. How does a Mach-Zender iterferometer work?. Physics Education. 35(1):46-48.

Glendinning I. 2010. Rotations on the Bloch Sphere Lecture Notes. University of Vienna.

Lala P. 2019. Quantum Computing: A Beginner's Introduction. New York (NY): McGraw-Hill Education.

Aaronson S. 2008. The Limits of Quantum. Scientific American. 298(3):62-69.

Garcia-Escartin J, Chamorro-Posada P. 2011. Equivalent Quantum Circuits. arXiv.

Gidney G. 2015. Constructing Large Controlled Nots. Algorithmic Assertions.

Chapter 4

Crooks G. 2019. Gates, States, and Circuits. ThreePlusOne.

Arute F, Arya K, Babbush R, *et al.* 2019. Quantum supremacy using a programmable superconducting processor. Nature. 574(7779):505–510.