Quantum Programming Foundations: History and Overview

Jens Palsberg

Jan 4, 2022

Outline

Hook: Quantum computers are here! We have algorithms for them but how do we actually run a quantum algorithm on a quantum computer? To answer this question, we need a sense of what the algorithms are like, how to program them in a programming language, and how to run a program.

Purpose: Persuade you that you can learn quantum programming.

Preview:

1. We will take the time to go through the details.
2. The math is linear algebra, probabilities, and complex numbers.
3. The quantum programming languages look familiar.

Transition to Body: First let me tell you about the pace of the course.

Main Point 1: We will take the time to go through the details.
[January is about foundations and algorithms, ending with a midterm exam]
[February is about languages and error correction]
[March is about implementation and quantum advantage, ending with a final exam]

Transition to MP2: The math that we need is stuff that many other courses use as well.

Main Point 2: The math is linear algebra, probabilities, and complex numbers.
[Double-slit and probabilities as complex numbers]
[Computing with vectors of complex numbers]
[Two levels of probabilities]

Transition to MP3: How weird are the quantum languages?

Main Point 3: The quantum programming languages look familiar.
[Some of the quantum languages are Python libraries]
[Some of them are entirely new designs, yet have familiar aspects]
[The languages tend to abstract away the connections between physical qubits]

Transition to Close: So there you have it.

Review: We will go slow, we will see some math that many people learn as undergraduates, and we will learn the quantum languages.

Strong finish: We have a dream team to do the teaching effort. In addition to me, we have Auguste Hirth as the teaching assistant for the fourth time in a row; he knows this course inside out. We also have one of the veterans from the 2021 version as our reader, who will grade the homework and give you all the feedback you need to succeed.

Call to action: Get going on the first homeworks today! We are ramping up fast, both on the math and on the programming.

Detailed presentation

Hook: Quantum computers are here! Many companies are working on quantum computing, more and more researchers work on quantum computing, and in 2019 the U.S. government started the National Quantum Initiative that allocated more than $1 billion to research in quantum computing. We have algorithms for quantum computers but how do we actually run a quantum algorithm on a quantum computer? To answer this question, we need a sense of what the algorithms are like, how to program them in a programming language, and how to run a program. This course will teach all that.

Purpose: Persuade you that you can learn quantum programming.

Preview:

1. We will take the time to go through the details.
2. The math is linear algebra, probabilities, and complex numbers.
3. The quantum programming languages look familiar.

Transition to Body: First let me tell you about the pace of the course.

Main Point 1: We will take the time to go through the details.

[January is about foundations and algorithms, ending with a midterm exam]
The goal for January is to get a detailed understanding of a handful of algorithms that we will run in February. Those algorithms are short, if we count lines of code, yet also complex, if we count the brain cycles needed for a newcomer to understand them. The way we will get there is to begin with two weeks on foundations and then continue with two weeks on algorithms. In the two weeks about foundations, I will cover the math needed to understand what quantum computing is and how the algorithms work. Let me get into this a little bit already now.

Quantum computing may fundamentally change what is efficiently computable. How? The idea is to scale computation exponentially with the size of the computer. This may allow us to solve otherwise intractable problems in optimization, chemistry, and machine learning. Those three areas are indeed the most promising application areas for quantum computing. Among those three areas, the front runner is optimization. We have all heard of Shor's algorithm for factoring numbers, but actually using it to factor the numbers we use in crypto is way into the future. Much more realistic for near-term quantum computers is a quantum algorithm for finding approximate solutions to optimization problems. Those problems can be NP-complete problems like graph coloring, traveling salesman, and integer linear programming. A paper from MIT in 2014 by Farhi and his colleagues presented such an algorithm and it generated a ton of interest. DARPA has a program devoted entirely to this algorithm and it descendants. If we can use quantum computers to quickly get good approximations to large NP-complete problems, it will be a game changer. We will cover that paper later in the course and you will get to program the algorithm.

Where are we with building quantum computers? IBM released a 127-qubit quantum computer in November 2021, USTC in China has 76-qubit quantum computer, and Google has a 72-qubit quantum computer. So, computers with 100+ qubits are here and 1,000 qubits seem within reach.

How many qubits can we simulate on a classical computer? Recently, NASA simulated 70 qubits on its supercomputer, and academics have simulated 49 qubits. The key here is that the challenge doubles for every additional qubit and nobody believes we can go beyond simulation of

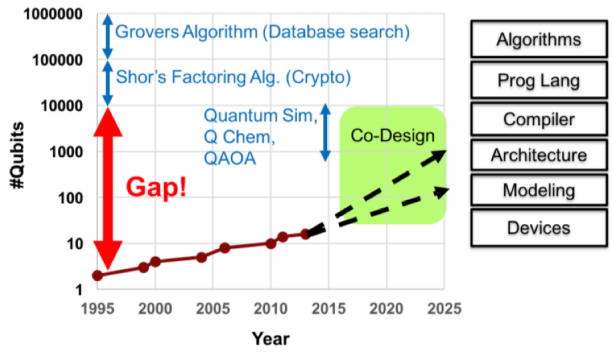
100 qubits. Hartmut Neven at Google has said that they prefer to go no higher than 37 qubits in their simulations. Any higher than that takes way too many resources and he says that they see no new phenomena above 37 qubits. I take all this as a sign that we are moving towards a quantum advantage. You will get both to use a quantum simulator and to program a quantum simulator.

Current quantum computers have high error rates, or, as John Preskill at Caltech has said: they are noisy. He called them NISQ computers, short for Noisy Intermediate-Scale Quantum computers. The errors will limit the potential until better error correction kicks in.

Let's compare two computers: one classical and one quantum.

The classical computer in our comparison is Intel Xeon Phi Processor, also known as "Knight's Corner". It is a commonly used processor in supercomputing clusters. This processor has a clock frequency of 1 GHz. In 2017, researchers measured the "soft error rate", which are radiation-induced errors that can flip states. The got the soft error rate to be 1 error per 1 thousand years.

In contrast, the IBM Q16 Rueschlikon is a quantum computer. It clocks at 5.26 GHz. The statistics are obtained on the IBM calibration data page in October 2018. Its single-qubit gate error is 2 errors for every 1,000 operations, its multi-qubit gate error 44 errors for every 1,000 operations, and its read out error is 6 errors for every 100 measurements. All of those error rates are orders of magnitude more frequent than the error rate of the Knight's Corner.



Two famous quantum algorithms are Shor's algorithm and Grover's algorithm. Shor's algorithm factors integers, which is useful for breaking crypto, while Grover's algorithm searches an unstructured database super quickly. As it happens, those algorithms require more than 10,000 qubits to do anything useful. Optimizing compilers may have a role to play here. Once we look closer at the optimization problem, it looks a lot like a design automation problem of the kind known from FPGA design and the like.

How good is a quantum computer? The key concept is the *quantum volume*, or the space-time product:

$$\text{quantum volume} \;=\; \#\text{qubits} \times \#\text{operations}$$

However, this must be tempered with the error rate of each operation. As we saw before, an operation on a couple of qubits has a much higher error rate than an operation on a single qubit. How far can we go in the near future? People are hoping for:

$$\text{near-future quantum volume} \;=\; 100 \text{ qubits} \times 1{,}000 \text{ operations}$$

The quantum computer stops working when decoherence kicks in, that is, when the quantumness goes away. Today, this happens within a second. Quantum computers today are what classical computers were in the 1950s. As an analogy that I learned from Alan Ho at Google, One of Google's current quantum computers has 72 qubits, and it just so happens that the Harvard Mark 1 computer from 1944 could store 72 numbers. By the way, one of the first programs on the Mark 1 was run by John von Neumann who was working on the Manhattan Project and needed some computation done. So, 72 qubits in 2021 and 72 numbers in 1944. Today's quantum computers are too small and too unreliable to be practical.

Each qubit doubles the size of the computer. Additionally, the reliability of each piece of the quantum computer is getting better all the time. The result is a double-exponential growth.

Let me compare with Moore's law. Moore's law says that the number of transistors on a square-inch die doubles every two years. We can translate this to say that the number of bits available for computing doubles every two years. Intuitively, if we have 2^{100} bits, we have a state space with 2^{100} dimensions. Let us contrast this with quantum computing. In quantum computing, every additional qubit doubles the number of dimensions of the state space. So, if we have 100 qubits, we have a state space with 2^{100} dimensions. Now I am waiting for the quantum equivalent of Moore's law that will talk about that the number of qubits increases by one every so often. How often? At the moment, my impression is that the number of qubits increases by one every month. This is exciting!

Alan Ho at Google talks about Neven's law, after Hartmut Neven. At Google, the reliability of gates is improving linearly over time. Neven observed that as you improve the reliability of the 2-qubit gates, you can linearly increase both the entanglement and the depth of the circuit. Because the equivalent classical computational power is exponential in both number of qubits and gate depth, this leads to a double exponential in growth. At the moment, the most reliable qubit in the world is from UCLA, from Eric Hudson's group in Physics, at 99.97 percent reliability.

What kind of algorithm is a good candidate for running on a quantum computer?

1. Small input, lots of computation, small output. Example: Shor's algorithm (input = an integer; output = two integers).
2. A result that we can verify easily. Example: Grover's algorithm (check that we found what we are looking for).
3. A subroutine for a classical computation. Example: Simon's algorithm (the quantum computer outputs equations, while the classical computer solves the equations).

We are going to spend January on the foundations and on some of the algorithms of quantum computing. On Bruinlearn you will find three useful links to material for January.

The first link is to Jack Hidary's book, which is good for three reasons. First, it gives a brief and accessible overview of quantum computing, both hardware and software. Second, it has a wonderful second half of the book on the math tool kit that we all need for learning quantum computing. Please take a look at whether you know all this and fill in any gaps that you discover. Third, the book uses Cirq for its program examples. Cirq is Google's quantum language and we will use it in this course.

The second link is to John Watrous lecture notes. John Watrous writes super clearly and is great at explaining proofs of quantum algorithms. Likely I have learned more quantum computing from reading John Watrous than from reading any other source.

The third link is to the quantum circuit simulator Quirk that Craig Gidney at Google wrote. This simulator makes it easy to get going on executing quantum programs and it plays a big role in homeworks in January.

Bruinlearn has a ton of homework for January. Some of the homework is on the foundations, which means math that boils down to mostly linear algebra and Boolean functions. I give you this homework for two reasons. One is to get you to page in some linear algebra that you already know and to perhaps learn some more linear algebra. The other reason is to think through some points that are directly relevant to the algorithms that we will cover later in January.

Some other homework in January is programming homework. Some of it is about running quantum circuits, to try out some ideas in the simplest possible setting. Some other programming homework is about getting into the details of the problems that the quantum algorithms can solve. You will do that by programming solutions on classical computers. This means that once we get to the quantum algorithms and the problems they solve, you have already solved each of them on a classical computer. You can use any language you like for those homeworks.

Overall, January has homework due twice a week, for the purpose of getting you ready for programming quantum computers in February. You will do the homework in January individually, and we will have a midterm exam in early February.

We will also have ten online quizzes throughout the course. I give quizzes for three reasons.

The first reason is to communicate some points in a lecture that I think are essential. The second reason is to have a different way to learn. When I took an MBA in UCLA Anderson some years ago, some of the professors gave us online quizzes, which were enjoyable. The third reason is that I see quizzes as part of your participation grade. You can all get 100 percent on the quizzes by taking them again and again so in a way I am not grading you on right or wrong. But I am grading your participation.

[February is about languages and error correction]

Every company that is building quantum computers also has its own quantum programming language. Google has Cirq, IBM has Qiskit, Microsoft has Q#, and Rigetti has PyQuil. In addition to languages from companies, we also have languages from universities, another handful of them. Until recently, each of the companies had built approximately one quantum computer, so we could say that in the quantum world, we had one language per computer. No standardization on that front! The goal should be the same as for classical computing: a language should run on many different computers. Write once, run everywhere. We are slowly moving in this direction. For example, Amazon Braket is working on the "connect everything" problem, and Cambridge Quantum Computing is working on "compile to any target" problem.

Which language will emerge as the winner? This is way to early to say. My guess is: none of them; people will design better languages. But we should try to grasp what language designers care about, how language designers envision language support for quantum algorithms, and how the ideas compare. We will get into all that in February by programming and running the algorithms that we learned in January. We will run on IBM's quantum computer and I am trying to get us access to IonQ's quantum computer.

The homework in February will cover three quantum languages. We will do the programming in Cirq, which is Google's quantum language. The Cirq compiler compiles Cirq programs to QASM, which is a widely used exchange format for quantum programs. Now we can import the QASM program into Qiskit, which is IBM's quantum language. Now we are ready to run on IBM's quantum computer.

In February and March you will work in groups. The maximum group size is three. We have 60+ people in the course so if every group has three people, we will have 20+ groups. Each group will submit quantum programs in February and March. You will form the groups, the deadline for group formation is end of January. Submit the group member names and the group name.

[March is about implementation and quantum advantage, ending with a final exam]
In March, I will talk about how to implement quantum languages, both via interpreters and via compilers. I will also give a lecture on where we are with getting a quantum advantage. I will talk about research questions to give you a sense of my perspective on where the field is going, which obstacles have to be overcome, and how long it will take to get there. Some of you may be interested in getting involved in research on quantum computing yourself; if so, please come and talk with me.

The course will have a final exam.

My plan for grading is straightforward. This is a graduate course and I would much prefer to give only A, B, C grades. If I sense that you really tried, even if you have lots of stuff wrong, I will give you either an A or a B, I hope that this will be true for everybody. My plan is to limit the A's to 45 percent, so my ideal case is that 45 percent of you gets an A and that 55 percent gets a B. However, I reserve the right to give a C or even below if I see little evidence of effort.

Let me make a shoutout to the undergraduate students in the course. I am delighted that you are here and encourage you to speak up. I know that you are outnumbered by the graduate students, yet please contribute to the discussions as much as you can.

Surely you have noticed that I have no slides. This is how I am going to roll the entire quarter. I am going to use the blackboard. The reason is that I want to force myself to slow down. Hopefully the slow pace will give you time to think and to ask questions. I hope you will ask a ton of questions.

We will use piazza in the course and I am going to give ten percent of your grade based on participation on piazza. Some examples of participation: ask a question, answer a question, and send a link to a newspaper article and comment on it. Some other examples could be to list a typo in my lecture notes and suggest a fix, show a work-through of a new example, and link to a research paper and comment on it. Plan on posting at least ten times on piazza under your own name this quarter. The goal is to build a community around quantum computing at UCLA.

Transition to MP2: The math that we need is stuff that many other courses use as well.

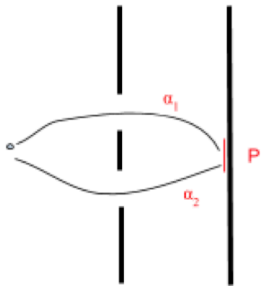
Main Point 2: The math is linear algebra, probabilities, and complex numbers.

[Double-slit and probabilities as complex numbers]
What is a qubit? Answer: it is a vector of two complex numbers:

$$\begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

If we have 72 qubits, then we are working with 2^{72} complex numbers. This exponential is the key to the power of quantum computing. If one day we have 300 qubits, we will be computing with more qubits than the number of atoms in the known universe. That day is likely to come within a few years.

Why are all the numbers complex? This goes to the heart of quantum mechanics. Feynman said that the essence of quantum mechanics is the Double Slit Experiment. This experiment goes back to Thomas Young in 1801.



The experiment works with of a source of light, a wall with two slits, and a back wall. In the experiment, we shoot photons toward the wall with the two slits. Where each photon lands on a second wall is probabilistic. If we plot where photons appear on the back wall, some places are highly likely, some unlikely.

So far this is straightforward: we might justify this behavior by a theory where each photon has a degree of freedom that determines which way it goes. What is weird is the following. For some interval on the second wall, let us define three probabilities:

- let P be the probability that the photon lands in the interval with both slits open;
- let P_1 be the probability that the photon lands in the interval if only slit 1 is open; and
- let P_2 be the probability that the photon lands in the interval if only slit 2 is open.

We would think that $P = P_1 + P_2$. But experiments show that this is false! Even places that are never hit when both slits are open, can sometimes be hit if only one slit is open.

Slogan: God plays dice, but they aren't normal dice.

The way to make this work is to change probabilities from real numbers to complex numbers. The people in quantum mechanics use the word *amplitudes* instead of probabilities.

The central idea of quantum mechanics is that to fully describe a system, you need a probability (an amplitude) for each possible configuration.

The Born rule: if the amplitude of a configuration is α, then the probability P of seeing that configuration as an outcome is

$$P = |\alpha|^2$$

Now we can redo the Double Slit Experiment with amplitudes instead of probabilities.

- let α be the amplitude that the photon lands in the interval with both slits open;
- let α_1 be the amplitude that the photon lands in the interval if only slit 1 is open; and
- let α_2 be the amplitude that the photon lands in the interval if only slit 2 is open.

We do get $\alpha = \alpha_1 + \alpha_2$. Example: suppose $\alpha_1 = \frac{1}{\sqrt{2}}$ and $\alpha_2 = -\frac{1}{\sqrt{2}}$. By the Born rule, the probability that the photon lands in the interval with one bit slit open is $\alpha_1^2 = \alpha_2^2 = \frac{1}{2}$. But if both slits are open

$$P = |\alpha_1 + \alpha_2|^2 = |\frac{1}{\sqrt{2}} + (-\frac{1}{\sqrt{2}})|^2 = 0$$

So, in quantum mechanics, probabilities (amplitudes!) can cancel each other out.

Now let us ask "how does a particle that went through the first slit know that the other slit is open?" In quantum mechanics, this question is ill-formed. Particles don't have trajectories, but rather take all paths simultaneously, in superposition. This is where we get the power of quantum computing.

[Computing with vectors of complex numbers]
Here is a way to sum up a central idea of quantum mechanics. Question: what if probabilities could be negative? Answer: we get quantum computing. Those complex numbers from quantum mechanics are the building blocks. Classical computing is about computing with bits, while quantum computing is about computing with complex numbers. Just like the state of a classical computer is given by a vector of bits, the state of a quantum computer is given by a vector of complex numbers. What can we do with those complex numbers? That is the topic of this course.

[Two levels of probabilities]
Quantum computing has two levels of probabilities. Probabilities on top of probabilities. At the bottom level we have probabilities from quantum mechanics. At the top level we have probabilities of the kind that we find in statements like: this algorithm gives the correct answer with high probability. The essence of quantum programming is to connect those levels. The top layer turns out to be what quantum algorithms are good at: finding approximate answers really fast. In contrast, quantum computers cannot get exact answers to hard problems much faster than a classical computer. For example, a quantum computer cannot find exact solutions to NP-complete problems in polynomial time. However, a quantum computer may be much better than a classical computer at finding approximate solutions to NP-complete problems.

Transition to MP3: How weird are the quantum languages?

Main Point 3: The quantum programming languages look familiar.

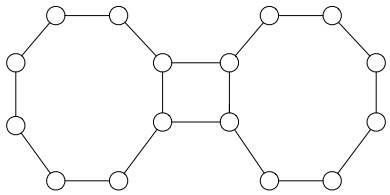
[Some of the quantum languages are Python libraries]
Cirq from Google and PyQuil from Rigetti are both Python libraries. This means that we can write Python code that runs on a classical computer and makes calls to a quantum computer. This is convenient because a lot of the code we need to write is not quantum at all. Rather, much of the code is for set up, post-processing, and input-output, which all can be written in Python. So, Python is the host language and all the quantum stuff is done by calling a quantum library.

[Some of them are entirely new designs, yet have familiar aspects]
Q# is a domain-specific language that was designed entirely for the purpose of expressing quantum algorithms. Program execution proceeds in a manner similar to the Python libraries: a classical computer controls the computation and makes calls to a quantum computer. However, in Q# the boundary is more fluid and up to the compiler.

[The languages tend to abstract away the connections between physical qubits]
For classical computers, while many companies make them, really they are all the same in one important respect: they are all based on CMOS. For quantum computers, this is not at all the case. Different companies build qubits in radically different ways. For example, IBM, Google, and Intel build superconducting qubits, while IonQ builds ion-trapped qubits. This affects quantum computing in particularly one important way: what if we want to do an operation on two qubits;

are they connected? On superconducting quantum computers, any operation on two qubits requires that they are connected. On ion-trapped quantum computers, an operation on two qubits can work on any two qubits.

Rigetti has a superconducting quantum computer with 16 qubits that are arranged as two octagons with two connections that cut across.



Quantum languages tend to abstract away the connections between physical qubits. The idea is "write once, run everywhere". of writing efficient programs. The compiler plays a key role in making that work. The problem of mapping the qubits in a program to the qubits on a specific quantum computer is akin to problems in design automation. Specifically, the mapping problem for quantum computing reminds of the mapping problems in FPGA design and ASIC design.

Transition to Close: So there you have it.

Review: We will go slow, we will see some math that many people learn as undergraduates, and we will learn the quantum languages.

Strong finish: We have a dream team to do the teaching effort. In addition to me, we have Auguste Hirth as the teaching assistant for the fourth time in a row; he knows this course inside out. We also have one of the veterans from the 2021 version as our reader, who will grade the homework and give you all the feedback you need to succeed.

Let me introduce the Quantum Computing Drinking Game. In the company of friends and fellow students, you fire up a youtube video on quantum computing or you read lecture notes. Now you listen and watch for the following quotes.

> Albert Einstein: "God does not play dice." (1926)
> Richard Feynman: "Nature isn't classical, dammit." (1981)
> Everybody: "$\frac{1}{\sqrt{2}}$" (every year)

Every time you hear or see one the quotes, you take a sip.

This course is the first in a series of two courses on quantum computing. The second course will be in Spring and my hidden agenda is to make as many of you as possible interested in taking the second course.

Call to action: Get going on the first homeworks today! We are ramping up fast, both on the math and on the programming.

Quantum Programming
Foundations: Interface to Quantum Mechanics.

Jens Palsberg

Jan 6, 2021

Outline

Foundations: interface to quantum mechanics; 100 minutes; Jan 6, 2021

Hook: Quantum computing has its roots in quantum mechanics. Does that mean that we have to learn quantum mechanics? No! Fortunately, we can understand quantum computing in simpler terms that abstract away from the physics.

Purpose: Persuade you that we can understand quantum programming without knowing physics.

Preview:

1. Quantum programming is based on four postulates about quantum mechanics.
2. We have an upgrade path from classical to probabilistic to quantum computing.
3. A grammar for quantum programs can be simple.

Transition to Body: First let me explain how to keep quantum mechanics at an arm's length.

Main Point 1: Quantum programming is based on four postulates about quantum mechanics.
[We can avoid calculus]
[The four postulates about quantum mechanics]
[First compute, then measure]

Transition to MP2: Is quantum computing an extension of classical computing?

Main Point 2: We have an upgrade path from classical to probabilistic to quantum computing.
[Think of classical computing in terms of applying matrices]
[Probabilistic computing uses more advanced matrices]
[Quantum computing uses even more advanced matrices]

Transition to MP3: So, quantum programs have a straightforward structure.

Main Point 3: A grammar for quantum programs can be simple.
[We need some qubits]
[We apply matrices to those qubits]
[We measure at the end]

Transition to Close: So there you have it.

Review: Quantum computing is like classical computing in that we have a simple interface to what the underlying hardware is doing.

Strong finish: This enables us as programmers to focus on what we do best: abstract, combine, optimize. In short, we solve problems, and we can do that on a classical computer or on a quantum computer without knowing how the hardware works.

Call to action: Take on board that quantum computing is applied linear algebra, learn the basics of the math, and get ready to program.

Detailed presentation

Hook: Quantum computing has its roots in quantum mechanics. Does that mean that we have to learn quantum mechanics? No! Fortunately, we can understand quantum computing in simpler terms that abstract away from the physics.

Purpose: Persuade you that we can understand quantum programming without knowing physics.

Preview:

1. Quantum programming is based on four postulates about quantum mechanics.
2. We have an upgrade path from classical to probabilistic to quantum computing.
3. A grammar for quantum programs can be simple.

Transition to Body: First let me explain how to keep quantum mechanics at an arm's length.

Main Point 1: Quantum programming is based on four postulates about quantum mechanics.

[We can avoid calculus]
Let us begin with a comparison of classical computing and quantum computing.

	Classical computing	Quantum computing
Software	***Boolean algebra***	***Linear algebra***
Hardware	***Classical mechanics***	***Quantum mechanics***
example	***Semiconductors***	***Superconductors***

What is the math of quantum mechanics? Linear algebra and calculus. On the calculus side, famous stuff like Schroedinger equations, Feynman diagrams, and Noether's theorem. The goal of today is to avoid calculus!

[The four postulates about quantum mechanics]
Authors use slightly different names for the four postulates; I use names that I remember easily: the state space rule, the composition rule, the step rule, and the measurement rule.

The state space rule: this rule tells us what are the allowable (possible) states of a given quantum system. The superposition principle says that if a quantum system can be in one of two states then it can also be placed in a linear combination of these states with complex coefficients. For example, if $|0\rangle$ and $|1\rangle$ are the two possible basic states, then the state can also be:

$$\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

We can think of the first $\frac{1}{\sqrt{2}}$ as a measure of the inclination towards the state $|0\rangle$.

The composition rule: the rule says that qubits can be composed in a way that we describe by a tensor product. The tensor product is the key to understanding why n qubits correspond to a

vector of length 2^n. For bits, we describe the state space of multiple bits by a Cartesian product; for qubits, we use a tensor product. When we use Cartesian product, adding a bit means adding a dimension. When we use tensor product, adding a qubit means doubling the number of dimensions. We will use Dirac notation like $\alpha|01\rangle$ to mean that we have two qubits and that with amplitude α we have that the first qubit is $|0\rangle$ and the second qubit is $|1\rangle$.

The step rule: this rule governs how the state of the quantum system changes over time. The postulate is that change is unitary: we take a step from one state to the next by applying a unitary operation. Intuitively, a unitary transformation transforms the state vector yet preserves its length; it is is a rotation or a reflection.

The measurement rule: this rule governs how much information about the state we can access. Measurement provides the only way of probing the quantum state vector and thereby bring information from the quantum world to the classical world. A measurement of a n qubits yields one of 2^n possible outcomes, that is, an bitstring with n bits. If the amplitude of $|j\rangle$ is α_j, then measuring in the standard basis yields $|j\rangle$ with probability $|\alpha_j|^2$.

Measurement alters the state of the quantum system: after measurement, the new state is exactly the outcome of the measurement. That is, if the outcome of the measurement is j, then after the measurement, the qubit is in state $|j\rangle$. Analogy: if you bake a souffle and open the oven to see how it is doing, the souffle collapses and you have to start over.

[First compute, then measure]
The paradigm of computing that emerges here is: set up a start state, then apply one matrix after the other, and finally measure to get the output.

Transition to MP2: Is quantum computing an extension of classical computing?

Main Point 2: We have an upgrade path from classical to probabilistic to quantum computing.

[Think of classical computing in terms of applying matrices]
Let us first rethink how we represent the state of a classical computation and how we compute with such states. The idea is that the state is a bit vector and we take a step to the next state by applying a matrix.

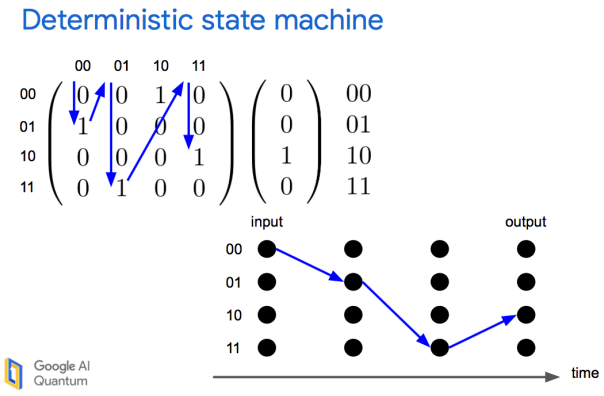
The state space rule: the state is a bit vector. Normally we think of the state of a classical computation as a bit vector of size n; now instead we will use a bit vector of size 2^n. The idea is that this much longer bit vector of size 2^n indicates which bit vector of size n is represented. This is done by having a single entry that is 1; the other entries are 0. Notice that the entries sum to 1.

The composition rule: we compose bits by tensor product.

The step rule: For deterministic, classical computing, we take a step by applying a matrix in which every column has a single 1 and otherwise 0.

The measurement rule: We can read the bits at any time.

Let us consider an example with two bits and a matrix that represents a program. The state space has $2 \times 2 = 4$ states and we apply the matrix to get from an input state to an output state.



[Probabilistic computing uses more advanced matrices]
Let us go over the four rules, this time for probabilistic computing.

The state space rule: the state is a probability vector, such as

$$\begin{pmatrix} p_0 \\ p_1 \end{pmatrix}$$

where p_0, p_1 are real numbers between 0 and 1, and $p_0 + p_1 = 1$. Here p_0 is the probability that the system is in configuration 0, while p_1 is the probability that the system is in configuration 1.

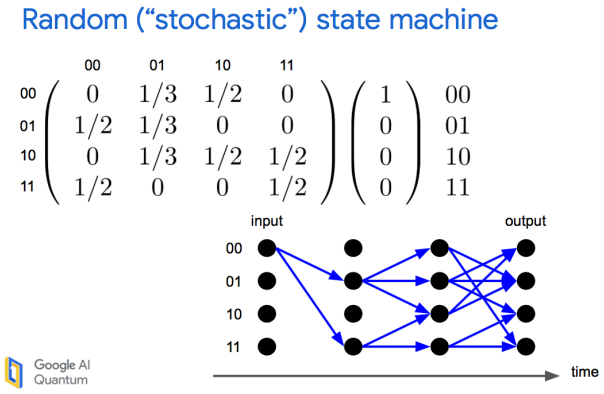
A pure state $|\Psi\rangle$ is a linear combination of base states, $|s\rangle$. That is, $|\Psi\rangle = \Sigma_s\ p_s|s\rangle$, where p_s is the probability of being in base state $|s\rangle$, and $\Sigma_s\ p_s = 1$, and $0 \leq p_s \leq 1$.

The composition rule: Combine two state spaces by using tensor product. For example:

$$\begin{pmatrix} p \\ 1-p \end{pmatrix} \otimes \begin{pmatrix} q \\ 1-q \end{pmatrix} = \begin{pmatrix} pq \\ p(1-q) \\ (1-p)q \\ (1-p)(1-q) \end{pmatrix}$$

The step rule: apply a stochastic matrix to a state to get the next state. In a stochastic matrix, all entries have to be non-negative and in each column, the entries must sum to 1. Thus, in a stochastic matrix, every column is a probability vector.

The measurement rule: If the i^{th} entry in the probability vector is p_i and we measure and find the system is in configuration i, then this happens with probability p_i and afterwards state "collapses" to be in configuration i.



Consider the probabilities of what a coin lands as after a flip:

$$\begin{pmatrix} p \\ q \end{pmatrix}$$

where p is the probability of tails (0) and q is the probability of heads (1). What happens after the coin lands and you look at the coin? You will not see a probability vector, of course. Rather, you will see tails or heads. Now you effectively change the description of your knowledge of the coin.

If you see tails, your knowledge is: $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$, while if you see heads, your knowledge is: $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$

Now let use matrices to model different tasks. The format of the transformation matrix is:

$$T = \begin{pmatrix} P(\text{tails}|\text{tails}) & P(\text{tails}|\text{heads}) \\ P(\text{heads}|\text{tails}) & P(\text{heads}|\text{heads}) \end{pmatrix}$$

We can interpret T_{ij} as the probability of entering state i after applying T to state j.

Now let us model, after the coin has landed, turning the coin over:

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} q \\ p \end{pmatrix}$$

Now let us model a fair flip of the coin:

$$\begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix} \begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} \frac{1}{2}p + \frac{1}{2}q \\ \frac{1}{2}p + \frac{1}{2}q \end{pmatrix} = \begin{pmatrix} \frac{1}{2}(p+q) \\ \frac{1}{2}(p+q) \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \end{pmatrix}$$

The last step is correct because $p + q = 1$, which is true because we work with probability vectors. Thus, regardless of the previous position, heads and tails are equally likely.

Now let us model that we flip the coin, and if we get heads we flip again, but if we get tails we turn it to heads.

$$\begin{matrix} & T & H \\ T & 0 & \frac{1}{2} \\ H & 1 & \frac{1}{2} \end{matrix} \begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} \frac{q}{2} \\ p + \frac{q}{2} \end{pmatrix}$$

Does that make sense? If we say that p, q are $P(\text{tails})$ and $P(\text{heads})$ after the first flip, we have two cases. First, the probability the coin will land on tails in the end is: 0 if (it lands on tails on the first flip) and $\frac{1}{2}$ if (it lands on heads and we flip again). Second, the probability that the coin will land on heads in the end is: 1 if (it lands on tails on the first flip) and $\frac{1}{2}$ if (it lands on heads and we flip again).

Now let us flip two coins (really, flip two bits). Now we can use a tensor product to describe the situation:

$$\begin{pmatrix} a \\ b \end{pmatrix} \otimes \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} P_{00} \\ P_{01} \\ P_{10} \\ P_{11} \end{pmatrix} = \begin{pmatrix} ac \\ ad \\ bc \\ bd \end{pmatrix}$$

Can all possible 4-element vectors arise by tensoring two 2-element vectors? No! For example:

$$\begin{pmatrix} ac \\ ad \\ bc \\ bd \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ 0 \\ 0 \\ \frac{1}{2} \end{pmatrix}$$

Here we see that this can work only if

$$\frac{1}{4} = \frac{1}{2} \times \frac{1}{2} = (ac) \times (bd) = (ad) \times (bc) = 0 \times 0 = 0$$

which is a contradiction. The two bits are always equal: they are both 0 half of the time, and they are both 1 half of the time. Thus, learning about one bit tells us something about the other bit.

Let us say that if the first bit is 1 then we want to flip the second bit; we can do that with a Controlled Not or $CNOT$ matrix:

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{2} \\ 0 \\ \frac{1}{2} \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ 0 \\ 0 \\ \frac{1}{2} \end{pmatrix}$$

As another example, let us say that if the first bit is 1, we want to replace the second bit with a random bit. The matrix for that is:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

Suppose we measure the first bit of the following state:

$$\frac{5}{8}|000\rangle + \frac{1}{8}|100\rangle + \frac{1}{8}|011\rangle + \frac{1}{8}|111\rangle$$

The probability that we observe $|0\rangle$ is $\frac{5}{8} + \frac{1}{8} = \frac{6}{8}$. Now, both according to intuition and according to the law of conditional probability, the state of the system "collapses" to

$$\frac{5/8}{6/8}|000\rangle + \frac{1/8}{6/8}|011\rangle \quad = \quad \frac{5}{6}|000\rangle + \frac{1}{6}|011\rangle$$

So, the bitstrings that survive are the ones that are consistent with the outcome, after which the probabilities for those bitstrings are normalized.

[Quantum computing uses even more advanced matrices]

Quantum computing is like probabilistic computing, but now the probabilities can be negative.

Probabilistic	Quantum
real numbers	complex numbers
vector of probabilities	vector of amplitudes
$\Sigma_{i=1}^{n} p_i = 1$	$\Sigma_{i=1}^{n} \lvert\alpha_i\rvert^2 = 1$
stochastic matrices	unitary matrices
preserve $\Sigma_{i=1}^{n} p_i = 1$	preserve $\Sigma_{i=1}^{n} \lvert\alpha_i\rvert^2 = 1$

The idea of the matrices in quantum computing is similar to that of the matrix in probabilistic computing. In probabilistic programming, a state is a probability vector; in quantum programming, a state is a superposition, also known as a qubit. The difference is that now we use amplitudes instead of probabilities, and instead of preserving that columns sum up to 1, we preserve that, for a vector with elements a_i, we have $\Sigma_i |a_i|^2 = 1$.

Similar to the probabilistic case, if you measure a qubit

$$\begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

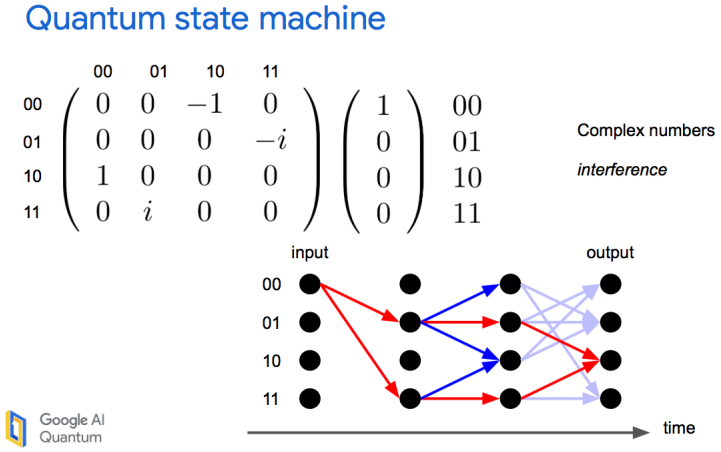
you will not see a superposition. Rather, you will see 0 or 1, just like before. Specifically, you will see 0 with probability $|\alpha|^2$, and you will see 1 with probability $|\beta|^2$. This is why we have the condition $|\alpha_i|^2 + |\beta_i|^2 = 1$; the probabilities have to sum to 1.

Similarly to the probabilistic case, after we look at a superposition, it becomes

$$\text{either } \begin{pmatrix} 1 \\ 0 \end{pmatrix} \text{ or } \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

So far, the quantum model seems a lot like the probabilistic model. Here is an important difference between probabilistic computing and quantum computing. In probabilistic computing, the probabilities represent uncertainty about the bits. Each bit is either 0 and 1, which just don't know which one. In contrast, in quantum computing, the superposition is the true state of the circuit. Literally, nature keeps track of 2^n amplitudes. If we work with 300 qubits, nature keeps track of 2^{300} complex numbers, which is more than the number of atoms in the known universe. Amazing that nature does exponential bookkeeping; let us use this power to do computing for us.

The operations in the quantum model are different from the operations in the probabilistic model. Rather than stochastic matrices, the quantum model works with unitary matrices.



A matrix is unitary if and only if it preserves the Euclidian norm. This is equivalent to saying that U is unitary if and only if

$$U^\dagger U \;=\; UU^\dagger \;=\; I$$

where $U^\dagger$ is the conjugate transpose of U. Notice that the above says that U is invertible. Thus, a computation step by a unitary matrix is reversible. Indeed, every quantum computation is a reversible computation.

Reversible computing means that the mapping from input to output is a bijection.

Consider again the matrix that models a flip of a fair coin:

$$\begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix} \;=\; \frac{1}{2}\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

Is this matrix unitary? No! Indeed, the matrix is of rank 1 so it isn't invertible, and we have

$$(\frac{1}{2})^2 + (\frac{1}{2})^2 \;=\; \frac{1}{2} \;\neq\; 1$$

So, for quantum programming we need a different matrix to model a flip of a fair coin. This is where the Hadamard matrix comes in:

$$H \;=\; \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad \text{where we notice } (\frac{1}{\sqrt{2}})^2 + (\frac{1}{\sqrt{2}})^2 = 1$$

If we apply the Hadamard gate to base states, we get the intuitive "fair coin" result. That is, regardless of which base state we are in, we end up with 50% probability of being in base state $|0\rangle$ and 50% probability of being in base state $|1\rangle$. Let us calculate this in detail.

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

$$H(\,|0\rangle\,) \;=\; \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} \;=\; \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix} \;=\; \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

$$H(\,|1\rangle\,) \;=\; \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} \;=\; \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -1 \end{pmatrix} \;=\; \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$$

So, if we can live with negative probabilities and be ready to measure in the quantum way of measurement, we have a flip of a fair coin! Here, the minus sign in front of $|1\rangle$ is the key; it is where we stray beyond the usual probabilities between 0 and 1.

Suppose your friend has a qubit that he knows is in one of the two states:

$$v_0 = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} \quad \text{or} \quad v_1 = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix}$$

but doesn't know which. How can you help him determine which one it is? Measuring right away is fruitless; you would see a random bit in either case. Better is to perform the Hadamard transform and then measure. Performing the Hadamard transform changes the superposition as follows:

$$Hv_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \text{or} \quad Hv_1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Now you can measure and you will see 0 (with probability 1) if the original state was v_0, and you will see 1 (with probability 1) if the original state was v_1.

What about the classical *NAND* gate? The matrix that represents this transformation is:

$T(x,b) = (x \text{ NAND } b, b)$

$$T = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

00 01 10 11

00 → 1

01 → 1

10 → 1

11 → 0

Is this matrix unitary? No! Indeed, the matrix is of rank 3 so it isn't invertible.

How do we get anything done in quantum programming now that many of our beloved logical operations don't fit the idea of unitary matrices? This brings us to the first programming technique of quantum programming. If we have an operation

$$f : \{0,1\} \to \{0,1\}$$

then we can represent it as an invertible operation U_f:

$$\begin{aligned} U_f &: \{0,1\}^2 \to \{0,1\}^2 \\ U_f(x,b) &= (x, b \oplus f(x)) \end{aligned}$$

We use $\oplus$ to denote addition (mod 2). Here x is the input to f, while b is setting up for receiving the output, an idea known from compilers. A compiler can compile a function by leaving a word on the stack in which to receive the output of the function. Notice that U_f is invertible:

$$\begin{aligned} (U_f \circ U_f)(x,b) &= U_f(U_f(x,b)) \\ &= U_f(x, b \oplus f(x)) \\ &= (x, (b \oplus f(x)) \oplus f(x)) \\ &= (x, b \oplus (f(x) \oplus f(x))) \\ &= (x,b) \end{aligned}$$

So, we conclude that $U_f \circ U_f = I$. Thus, we have transformed computing with f into reversible computing with f. Now all we need is that U_f can be described by a unitary matrix, which it can.

Quantum computing has the same computational paths as probabilistic computing; quantum computing gets its power from interference among these paths. Specifically, in quantum computing some of those paths can cancel out because of negative probabilities. Intuitively, we want paths for correct answers to reinforce each other by adding constructively, and wrong answers to cancel each other out destructively.

Transition to MP3: So, quantum programs have a straightforward structure.

Main Point 3: A grammar for quantum programs can be simple.

[We need some qubits]
We will need both some input qubits and some helper qubits.

[We apply matrices to those qubits]
We will apply matrices, one after the other. Which matrices? Turns out, the set of H and $CCNOT$ is a solid choice. Here, H works on a single qubit, while $CCNOT$ works on three qubits.

[We measure at the end]
We postpone all measurement to the end. So, a program can have the form:

The input qubits are $x_1, x_2, \ldots x_n$
Initialize each of the helper qubits $x_{n+1}, \ldots, x_{n+c}$ to $|1\rangle$.
$\ldots$
$H(x_i)$
$\ldots$
$CCNOT(x_i, x_j, x_k)$
$\ldots$
Measure registers $x_1, \ldots, x_m$ and output the result.

Transition to Close: So there you have it.

Review: Quantum computing is like classical computing in that we have a simple interface to what the underlying hardware is doing.

Strong finish: This enables us as programmers to focus on what we do best: abstract, combine, optimize. In short, we solve problems, and we can do that on a classical computer or on a quantum computer without knowing how the hardware works.

Call to action: Take on board that quantum computing is applied linear algebra, learn the basics of the math, and get ready to program.

Quantum Programming Foundations: Linear Algebra

Jens Palsberg

Jan 11, 2022

Outline

Hook: The math that we need for quantum computing is linear algebra. The notation for vectors is from quantum mechanics and differs a bit from what we are used to in math, yet we can learn it. Most of the linear algebra that we need for quantum computing fits into a single lecture.

Purpose: Persuade you that the linear algebra is mostly stuff that you know already.

Preview:

1. The basic concept is Hilbert spaces.
2. We can explain qubits, superposition, entanglement, and measurement.
3. The possibly unfamiliar stuff is Dirac notation and tensor products.

Transition to Body: First let me introduce the basic mathematical structure that we will use.

Main Point 1: The basic concept is Hilbert spaces.
[A Hilbert space is a set with some operations]
[The key operations that we need for quantum computing are unitary, linear functions]
[The concept of a basis]

Transition to MP2: The concepts of quantum computing have mathematical counterparts.

Main Point 2: We can explain qubits, superposition, entanglement, and measurement.
[Qubits and superposition]
[Measurement]
[Entanglement and quantum chess]

Transition to MP3: Now have covered the basic stuff and can move on to advanced topics.

Main Point 3: The possibly unfamiliar stuff is Dirac notation and tensor products.
[Dirac notation]
[Tensor products]
[Laws]

Transition to Close: So there you have it.

Review: Based on Hilbert spaces, we can explain all the concepts of quantum computing, and when we use tensor products and Dirac notation, the notation is compact.

Strong finish: Now we are ready for quantum computing and we will get into it next lecture.

Call to action: Brush up on basic linear algebra by reviewing the second half of Hidary's book.

Detailed presentation

Hook: The math that we need for quantum computing is linear algebra. Some of the notation for vectors is from quantum mechanics, which differs a bit from what we are used to in math, yet we can learn it. All the linear algebra that we need for quantum computing fits into a single lecture.

Purpose: Persuade you that the linear algebra is mostly stuff that you know already.

Preview:

1. The basic concept is Hilbert spaces.
2. We can explain qubits, superposition, entanglement, and measurement.
3. The possibly unfamiliar stuff is Dirac notation and tensor products.

Transition to Body: First let me introduce the basic mathematical structure that we will use.

Main Point 1: The basic concept is Hilbert spaces.

[A Hilbert space is a set with some operations]

If $z = a+bi$, then the complex *conjugate* of z is $z^* = a-bi$. The length $|z|$ of z is a nonnegative real number given by $|z|^2 = zz^* = (a+bi)(a-bi) = a^2+b^2$. Euler's formula says that $e^{i\theta} = \cos\theta + i\sin\theta$, so we can represent any complex number as $re^{i\theta}$.

In quantum computing, the state of the system is a unit vector in a Hilbert space. A unit vector is a vector of length 1. A Hilbert space is a vector space with an inner product that satisfies some conditions. In our case, the Hilbert space consists of vectors of complex numbers and scalars that are also complex numbers. Vector addition and scalar multiplication work as usual, and for two vectors,

$$\begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_{k-1} \end{pmatrix} \quad \begin{pmatrix} \beta_0 \\ \vdots \\ \beta_{k-1} \end{pmatrix}$$

the inner product is $\Sigma_i \alpha_i^* \beta_i$. Notation: we write the inner product of $|v\rangle$ and $|w\rangle$ as $\langle v|w\rangle$, which is shorthand for $\langle v||w\rangle$, which itself is the matrix product of a row vector and column vector. Two vectors are orthogonal if their inner product is 0. Examples:

$$\begin{aligned}
|v\rangle &= a_0|0\rangle + a_1|1\rangle \\
|w\rangle &= b_0|0\rangle + b_1|1\rangle \\
\langle v|w\rangle &= (a_0^* \; a_1^*)\begin{pmatrix} b_0 \\ b_1 \end{pmatrix} = a_0^* b_0 + a_1^* b_1 \\
|+\rangle &= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\
|-\rangle &= \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \\
\langle 0|1\rangle &= (1 \; 0)\begin{pmatrix} 0 \\ 1 \end{pmatrix} = 1\times 0 + 0 \times 1 = 0 \\
\langle +|-\rangle &= (\frac{1}{\sqrt{2}} \; \frac{1}{\sqrt{2}})\begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix} = \frac{1}{2} - \frac{1}{2} = 0
\end{aligned}$$

The outer product of two vectors is defined as follows:

$$\begin{pmatrix}\alpha_1\\ \beta_1\end{pmatrix}(\alpha_2^*\ \beta_2^*) = \begin{pmatrix}\alpha_1\alpha_2^* & \alpha_1\beta_2^*\\ \beta_1\alpha_2^* & \beta_1\beta_2^*\end{pmatrix}$$

The outer product is the matrix product of a column vector and a row vector; the result is a matrix.

[The key operations that we need for quantum computing are unitary, linear functions]
A linear operation U is unitary iff $UU^\dagger = U^\dagger U = I$. Here $U^\dagger$ is the conjugate transpose of U. We have $U^{-1} = U^\dagger$. Examples:

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} \qquad R_y(\theta) = \begin{pmatrix}\cos(\theta/2) & -\sin(\theta/2)\\ \sin(\theta/2) & \cos(\theta/2)\end{pmatrix} \qquad S = \begin{pmatrix}1 & 0\\ 0 & i\end{pmatrix}$$
$$NOT = X = \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix} \qquad Y = \begin{pmatrix}0 & -i\\ i & 0\end{pmatrix} \qquad Z = \begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}$$

The matrices X, Y, Z, are called the Pauli gates. Note that $H^\dagger = H$ and $H^2 = I$. Note that $H|0\rangle = |+\rangle$ and $H|1\rangle = |-\rangle$. Here are detailed calculations:

$$H|0\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}1\\ 0\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\ 1\end{pmatrix} = |+\rangle$$
$$H|1\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}0\\ 1\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\ -1\end{pmatrix} = |-\rangle$$

Note that $Z|+\rangle = |-\rangle$ and $Z|-\rangle = |+\rangle$.

Here is how to decompose H into a sum of outer products:

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\ 1\end{pmatrix}(1\ 0) + \frac{1}{\sqrt{2}}\begin{pmatrix}1\\ -1\end{pmatrix}(0\ 1) = (|+\rangle\langle 0|) + (|-\rangle\langle 1|)$$

Let us check that NOT works like a classical Boolean negation operator.

$$NOT|0\rangle = \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}\begin{pmatrix}1\\ 0\end{pmatrix} = \begin{pmatrix}0\\ 1\end{pmatrix} = |1\rangle$$
$$NOT|1\rangle = |0\rangle$$
$$NOT(\alpha|0\rangle + \beta|1\rangle) = \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}\begin{pmatrix}\alpha\\ \beta\end{pmatrix} = \begin{pmatrix}\beta\\ \alpha\end{pmatrix} = \beta|0\rangle + \alpha|1\rangle$$

[The concept of a basis]
A generating set for a vector space is a finite set of vectors such that every vector in this space can be written as a linear combination of these vectors, that is, if $\{|e_i\rangle\}$ is a generating set for a vector space, then every element of the vector space can be expressed as $\Sigma_i\ v_i|e_i\rangle$. A set of vectors $\{|e_i\rangle\}$ is linearly independent if $\Sigma_i\ v_i|e_i\rangle = 0$ if and only if $v_i = 0$ for all i. A basis is a generating set of vectors that are all linearly independent. An orthonormal basis is a basis for which the basis vectors are unit vectors and are orthogonal to each other.

Note that $\{|0\rangle, |1\rangle\}$ forms an orthonormal basis. Note that $\{|+\rangle, |-\rangle\}$ forms an orthonormal basis. The Hadamard gate goes back and forth between the $\{|0\rangle, |1\rangle\}$ basis and the $\{|+\rangle, |-\rangle\}$-basis.

Generating Set, Linearly Independent, Basis, Orthonormal basis

Transition to MP2: The concepts of quantum computing have mathematical counterparts.

Main Point 2: We can explain qubits, superposition, entanglement, and measurement.

[Qubits and superposition]

A qubit is a vector $\begin{pmatrix}\alpha\\ \beta\end{pmatrix}$, where α, β are complex numbers that satisfy $|\alpha|^2 + |\beta|^2 = 1$. Here, α, β are amplitudes (probabilities as complex numbers). Quantum mechanics people prefer to write $\alpha|0\rangle + \beta|1\rangle$. Here we see a linear combination, also known as a superposition: the qubit is 0 and 1 at the same time, with some amplitudes.

[Measurement]

We cannot measure the amplitudes in a qubit; this is enshrined in the measurement postulate of quantum mechanics. What we can do is to choose an orthonormal basis $\{|v\rangle, |w\rangle\}$ and measure the qubit in that basis. For example, when we measure the qubit $\alpha|0\rangle + \beta|1\rangle$ in the basis $\{|0\rangle, |1\rangle\}$, we use the Born rule and get:

$$\begin{cases} |0\rangle & \text{with probability } |\alpha|^2 \\ |1\rangle & \text{with probability } |\beta|^2 \end{cases}$$

More generally, we can choose the orthonormal basis $\{|v\rangle, |w\rangle\}$, rewrite the qubit in that basis: $\alpha|0\rangle + \beta|1\rangle = \alpha'|v\rangle + \beta'|w\rangle$, and now the outcome of the measurement is v with probability $|\alpha'|^2$, and w with probability $|\beta'|^2$. The outcome of the measurement is also the new state of the qubit. We can wonder why measurement alters the state; researchers have proposed three interpretations.

The first is the Copenhagen interpretation (Niels Bohr), which says that nature operates in a quantum world, but we live in a classical world. The idea that we get information from the quantum world via measurement is an axiom. Intuitively, asking in quantum mechanics "what is a measurement?" is equivalent to asking the axioms of euclidean geometry "what is a point?".

The second is the many-world interpretation (Hugh Everett), which says that every time one measures a quantum object, the universe branches into two equally real universes.

The third is the non-local hidden variables interpretation (David Bohm), which says that both of the previous answers are unacceptable and that quantum mechanics is an incomplete theory. Non-local hidden variables is one among several proposals to fill the gap.

In general, for an orthogonal basis $\{|v\rangle, |w\rangle\}$, measurement of $|\psi\rangle$ gives $|v\rangle$ with probability $|\langle v|\psi\rangle|^2$, and gives $|w\rangle$ with probability $|\langle w|\psi\rangle|^2$.

Suppose we have two qubits:

$$|\psi\rangle \quad = \quad \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle \qquad \text{where } \Sigma_{ij}\ |\alpha_{ij}|^2 = 1$$

The probability that we measure $|ij\rangle$, where $i, j \in \{0,1\}^2$ is $|\alpha_{ij}|^2$. Additionally, after the measurement, the state of the two qubits is $|ij\rangle$.

Suppose Alice measures the first qubit in the standard basis and observes $|0\rangle$, which she will do with probability $|\alpha_{00}|^2 + |\alpha_{01}|^2$. After the measurement, the state of the two qubits is:

$$\frac{\alpha_{00}|00\rangle + \alpha_{01}|01\rangle}{\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}} \quad = \quad |0\rangle \otimes \frac{\alpha_{00}|0\rangle + \alpha_{01}|1\rangle}{\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}}$$

The probability rule for measuring the second qubit is now given by the rules of conditional probability. Now a measurement of the second qubit in the standard basis will give:

$$\begin{cases} |0\rangle & \text{with probability } \frac{|\alpha_{00}|^2}{|\alpha_{00}|^2+|\alpha_{01}|^2} \\ |1\rangle & \text{with probability } \frac{|\alpha_{01}|^2}{|\alpha_{00}|^2+|\alpha_{01}|^2} \end{cases}$$

Notice that once Alice made her first measurement, the probability of observing each of $|10\rangle$ and $|11\rangle$ becomes zero. Notice also that measuring the first qubit and then measuring the second qubit gives the same result as measuring both qubits at once.

Example of how measuring along way can change the computation. We have $HH|0\rangle = |0\rangle$. In contrast, if we do $H|0\rangle$, then measure, then apply H again, then half of the time we get $|+\rangle$ and half of the time we get $|-\rangle$.

[Entanglement and quantum chess]
Suppose we have two qubits:

$$|\psi\rangle \quad = \quad \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle \qquad \text{where } \Sigma_{ij}\ |\alpha_{ij}|^2 = 1$$

One of the possibilities here is a so-called Bell pair, which is an entangled state:

Entanglement

$$\boxed{\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle}$$

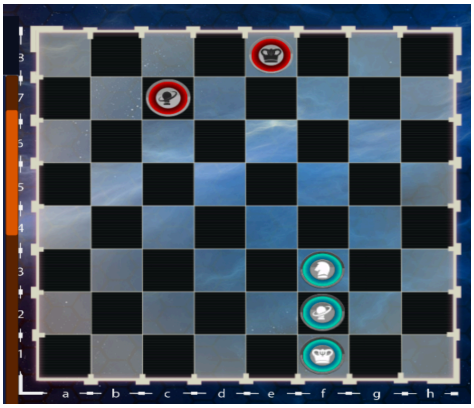
Notice that if we measure one of the qubits as zero, the other qubit has to measure zero as well. In general, we say that either a state is a tensor product or, otherwise, it is entangled.

Here is an example. Assume there are three characters, Alice, Bob and Charlie. Charlie has two pairs of gloves. Charlie decides based on a random coin flip whether to pick the left-hand gloves or the right-hand gloves. Then Charlie gives each of Alice and Bob a box with one of the picked gloves. Charlie keeps all this secret from Alice and Bob. Afterwards, Alice goes to Mars, while Bob stays on Earth. When Alice opens her box, she finds out she has a left glove, which means that she knows immediately that Bob has a left glove. Since Alice and Bob don't communicate and thus information does not travel between them, this does not violate relativity theory.

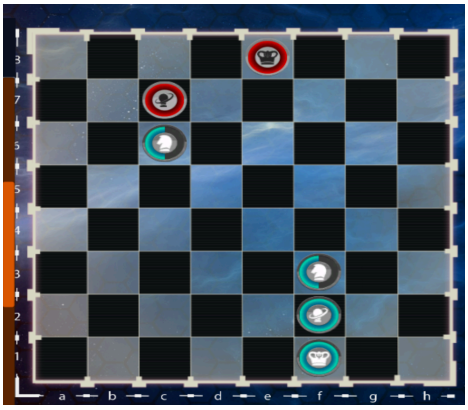
This example doesn't show the full power of quantum because Charlie picks the gloves before Alice goes to Mars. Thus, Alice is traveling with a classical glove rather than a quantum glove, but she won't know which one until later. Thus, this example is more of a probabilistic computing example than a quantum computing example.

As an example that may help understand superposition and entanglement, I will discuss quantum chess. Chris Cantwell at USC developed quantum chess and he had the idea to add a layer of quantumness on top of chess. What did he do?

In quantum chess, a piece can exist in superposition. A quantum move consists of up to two standard moves that don't capture anything. The chance that the piece moved is 50 percent, and the chance that it didn't move is 50 percent. A measurement may find the piece on either square.

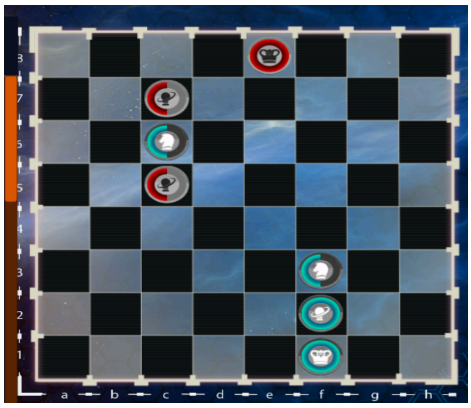


Above, we have a white king, a white pawn, and a white knight. We also have a black king and a black pawn. Now let us do a quantum move with the white knight.



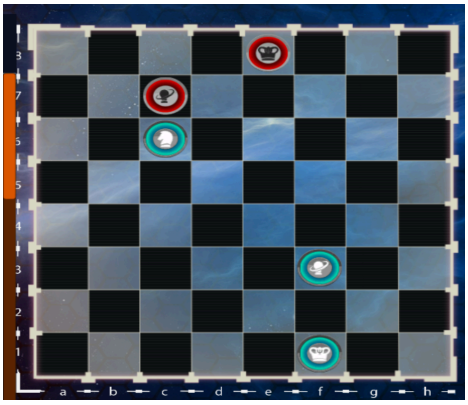
Above, the quantum knight has done a double move, which means that now it is in the new square with 50 percent probability and in the old square with 50 percent probability. We can see this indicated with the two half circles.

Now let us move the black pawn two squares forward.

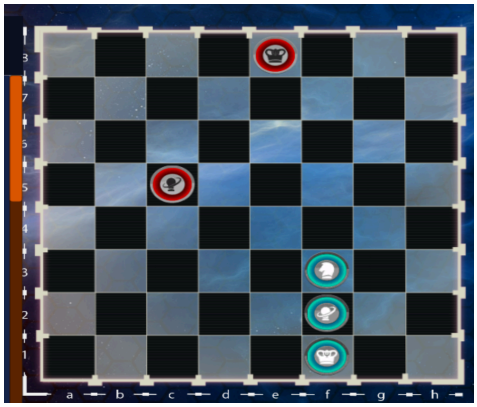


This moves depends on whether white knight is in the way so the move of the black pawn entangles the white knight and the black pawn. This means that the black pawn moved with 50 percent probability and stayed put with 50 percent probability.

Now let us try to move the white pawn one square forward. This triggers a measurement to see whether the white knight is there.



Above, the measurement gave that the white knight had moved so the white pawn could advance. Because the white knight did move, the black pawn was unable to move. So now we also know that the black pawn is on its original square, and all the probabilities are back to 100 percent.



In contrast, above, the measurement gave that the white knight didn't move so the white pawn had to stay put. This means that the white knight was not blocking the black pawn, hence the black pawn advanced two squares. Notice that all the probabilities are back to 100 percent.

We saw that superposition is possible and we saw how pieces can get entangled.

Transition to MP3: Now have covered the basic stuff and can move on to advanced topics.

Main Point 3: The possibly unfamiliar stuff is Dirac notation and tensor products.

[Dirac notation]
We will use Dirac's ket notation, which is a compact way to write a column vector:

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

Those vectors are pairwise orthogonal unit vectors so they form an orthonormal basis for the vector space; we call it the *standard basis*. Notice that:

$$\langle i|j\rangle = \begin{cases} 1 & \text{if } i=j \\ 0 & \text{otherwise} \end{cases}$$

The advantage of the ket notation is that the it labels the basis vectors explicitly.

Dirac's word "ket" is part of the word *bracket*; he also has a "bra" notation for row vectors.

"Bra" Notation

$$\langle 0| = (1\ 0) \quad \text{and} \quad \langle 1| = (0\ 1)$$

Technically, $\langle i|$ is the conjugate transpose of $|i\rangle$, so $\langle\psi| = |\psi\rangle^\dagger$.

Examples:

$$\begin{array}{rclcrcl} |\psi\rangle &=& \begin{pmatrix}\frac{1+i}{2}\\ \frac{1}{\sqrt{2}}\end{pmatrix} & & \langle\psi| &=& (\frac{1-i}{2} \quad \frac{1}{\sqrt{2}}) \\ |\varphi\rangle &=& \frac{i}{\sqrt{2}}|0\rangle + \frac{1+i}{2}|1\rangle & & \langle\varphi| &=& \frac{-i}{\sqrt{2}}\langle 0| + \frac{1-i}{2}\langle 1| \end{array}$$

In ket notation, the outer product looks like $|\Psi_1\rangle\ \langle\Psi_2|$. The matrix $|\Psi\rangle\ \langle\Psi|$ is called the density matrix of the quantum state Ψ.

[Tensor products]
The tensor product is also known as the Kronecker product; it is defined for two vectors as follows:

$$\begin{pmatrix}\alpha_1\\ \beta_1\end{pmatrix} \otimes \begin{pmatrix}\alpha_2\\ \beta_2\end{pmatrix} = \begin{pmatrix}\alpha_1\begin{pmatrix}\alpha_2\\ \beta_2\end{pmatrix}\\ \beta_1\begin{pmatrix}\alpha_2\\ \beta_2\end{pmatrix}\end{pmatrix} = \begin{pmatrix}\alpha_1\alpha_2\\ \alpha_1\beta_2\\ \beta_1\alpha_2\\ \beta_1\beta_2\end{pmatrix}$$

We can show that

$$|\alpha_1|^2 + |\beta_1|^2 = 1 \text{ and } |\alpha_2|^2 + |\beta_2|^2 = 1 \iff |\alpha_1\alpha_2|^2 + |\alpha_1\beta_2|^2 + |\beta_1\alpha_2|^2 + |\beta_1\beta_2|^2 = 1$$

The above definition generalizes to a tensor product of two matrices.

$$A \otimes B = \begin{pmatrix} A_{11}B & A_{12}B & \dots & A_{1m}B \\ A_{21}B & A_{22}B & \dots & A_{2m}B \\ \vdots & \vdots & \vdots & \vdots \\ A_{n1}B & A_{n2}B & \dots & A_{nm}B \end{pmatrix}$$

We will write $|\psi\rangle|\varphi\rangle$ as an abbreviation for $|\psi\rangle \otimes |\varphi\rangle$. Additionally, we will often eliminate internal brackets. Thus, $|0\rangle \otimes |1\rangle \otimes |1\rangle$ and $|0\rangle|1\rangle|1\rangle$ and $|011\rangle$ all denote the same vector.

If $s \in \{0,1\}^n$, then $|s\rangle$ denotes a column vector with 2^n entries. If we think of s as a number in binary notation, then $|s\rangle$ is a column vector that is 0 everywhere except that it is 1 in the position indexed by the number denoted by s.

Examples:

$$(\alpha_1|0\rangle + \beta_1|1\rangle) \otimes (\alpha_2|0\rangle + \beta_2|1\rangle) = \alpha_1\alpha_2|00\rangle + \alpha_1\beta_2|01\rangle + \beta_1\alpha_2|10\rangle + \beta_1\beta_2|11\rangle$$

$$NOT \otimes H = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \\ 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \end{pmatrix}$$

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

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

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

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

$$\frac{1}{\sqrt{2}}|000000\rangle + \frac{1}{\sqrt{2}}|111111\rangle = \ldots = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} \quad \text{here we have 62 zeroes!}$$

Example:

$$H \otimes I = \tfrac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix} \qquad \varphi = \tfrac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} \qquad (H \otimes I)\varphi = \tfrac{1}{2} \begin{pmatrix} 1 \\ 1 \\ 1 \\ -1 \end{pmatrix}$$

Or, in Dirac notation:

$$(H \otimes I)(\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle) = \frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle - \frac{1}{2}|11\rangle$$

Now let us use properties of the tensor product to do the calculation.

$$\varphi = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle = \frac{1}{\sqrt{2}}|0\rangle|0\rangle + \frac{1}{\sqrt{2}}|1\rangle|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle \otimes |0\rangle + |1\rangle \otimes |1\rangle)$$

$$
\begin{aligned}
(H \otimes I)(\varphi) &= (H \otimes I)(\frac{1}{\sqrt{2}}(|0\rangle \otimes |0\rangle + |1\rangle \otimes |1\rangle)) \\
&= \frac{1}{\sqrt{2}}((H|0\rangle \otimes I|0\rangle) + (H|1\rangle \otimes I|1\rangle)) \\
&= \frac{1}{\sqrt{2}}(((\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle) \otimes |0\rangle) + ((\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle) \otimes |1\rangle)) \\
&= ((\frac{1}{2}|0\rangle + \frac{1}{2}|1\rangle) \otimes |0\rangle) + ((\frac{1}{2}|0\rangle - \frac{1}{2}|1\rangle) \otimes |1\rangle) \\
&= \frac{1}{2}|00\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|01\rangle - \frac{1}{2}|11\rangle \\
(I \otimes X)(\varphi) &= (I \otimes X)(\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)) = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle)
\end{aligned}
$$

[Laws]

The tensor product of two unitary matrices is unitary. Tensor product is associative, distributes over matrix addition, can be applied to a tensor product, and satisfies a "floating scalars" law:

$$
\begin{aligned}
(A \otimes B) \otimes C &= A \otimes (B \otimes C) \\
A \otimes (B + C) &= (A \otimes B) + (A \otimes C) \\
(A + B) \otimes C &= (A \otimes C) + (B \otimes C) \\
(A \otimes B)(C \otimes D) &= (AC) \otimes (BD) \\
(\alpha A) \otimes B &= A \otimes (\alpha B) = \alpha(A \otimes B)
\end{aligned}
$$

In particular, if $|a\rangle = \Sigma_j\ \alpha_j|a_j\rangle$ and $|b\rangle = \Sigma_k\ \beta_k|b_k\rangle$, then

$$
|a\rangle \otimes |b\rangle = \Sigma_j\ \Sigma_k\ \alpha_j\beta_k(|a_j\rangle \otimes |b_k\rangle) = \Sigma_j\ \Sigma_k\ \alpha_j\beta_k|a_jb_k\rangle
$$

In general, the tensor product fails to be commutative: $|0\rangle \otimes |1\rangle = |01\rangle \neq |10\rangle = |1\rangle \otimes |0\rangle$.

Laws about outer product:

$$
\begin{aligned}
|\psi\rangle\ \langle\varphi|\ |\gamma\rangle &= |\psi\rangle\ \langle\varphi|\gamma\rangle = \langle\varphi|\gamma\rangle\ |\psi\rangle \\
(\alpha|v\rangle\langle w|M)^\dagger &= M^\dagger(\langle w|)^\dagger(|v\rangle)^\dagger(\alpha)^\dagger = M^\dagger|w\rangle\langle v|\alpha^* = \alpha^*M^\dagger|w\rangle\langle v|
\end{aligned}
$$

We can express any matrix M as a sum of outer product terms:

$$
\begin{aligned}
M &= \Sigma_{i,j}\ M_{ij}|i\rangle\langle j| \\
M^\dagger &= \Sigma_{i,j}\ M^*_{ji}|i\rangle\langle j| \\
I &= \Sigma_i\ |i\rangle\langle i| \\
|v\rangle &= \Sigma_k\ v_k|k\rangle \\
M|v\rangle &= \Sigma_{i,j}\ M_{ij}|i\rangle\langle j|\ \Sigma_k\ v_k|k\rangle = \Sigma_{i,j,k}\ M_{ij}v_k|i\rangle\langle j|\ |k\rangle = \Sigma_{i,j,k}\ M_{ij}v_k|i\rangle\langle j|k\rangle \\
&= \Sigma_{i,j}\ M_{ij}v_j|i\rangle
\end{aligned}
$$

Transition to Close: So there you have it.

Review: Based on Hilbert spaces, we can explain all the concepts of quantum computing, and when we use tensor products and Dirac notation, the notation is compact.

Strong finish: Now we are ready for quantum computing and we will get into it next lecture.

Call to action: Brush up on basic linear algebra by reviewing the second half of Hidary's book.

Quantum Programming Foundations: Quantum Circuits

Jens Palsberg

Jan 13, 2022

Outline

Hook: The machine model of quantum computing is a quantum circuit. A quantum circuit can be drawn as a diagram, which shows which qubits we work with, which matrices we apply, and when. All this is just as simple as it sounds, but the things we can do with quantum circuits are amazing.

Purpose: Persuade you that quantum circuits are both simple and expressive.

Preview:

1. Diagram notation for quantum circuits.
2. Superdense coding, teleportation, and no-cloning.
3. Universal sets of operations.

Transition to Body: Let us first look at how to draw a quantum circuit.

Main Point 1: Diagram notation for quantum circuits.
[Input qubits]
[Unitary matrices]
[Measurement]

Transition to MP2: Now let us look at what we can and cannot do with quantum circuits.

Main Point 2: Superdense coding, teleportation, and no-cloning.
[Superdense coding]
[Teleportation]
[No-cloning]

Transition to MP3: What is the machine language of quantum circuits?

Main Point 3: Universal sets of operations.
[Universal sets exists]
[Universal sets can be quite different]
[Different quantum computers support different sets of matrices]

Transition to Close: So there you have it.

Review: We can draw circuit diagrams and do amazing little tasks, and we can talk about universal sets of operations.

Strong finish: Now we are ready to study quantum algorithms, which we will express as quantum circuits.

Call to action: Play around with the quantum circuit simulator that we linked online.

Detailed presentation

Hook: The machine model of quantum computing is a quantum circuit. A quantum circuit can be drawn as a diagram, which shows which qubits we work with, which matrices we apply, and when. All this is just as simple as it sounds, but the things we can do with quantum circuits are amazing.

Purpose: Persuade you that quantum circuits are both simple and expressive.

Preview:

1. Diagram notation for quantum circuits.
2. Superdense coding, teleportation, and no-cloning.
3. Universal sets of operations.

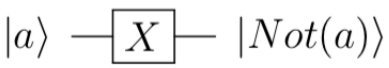
Transition to Body: Let us first look at how to draw a quantum circuit.

Main Point 1: Diagram notation for quantum circuits.

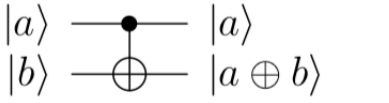
[Input qubits]
The input qubits appear on the left. Each qubit has its own line that shows what happens to it as time moves from left to right.

[Unitary matrices]
Since quantum gates needs to be reversible, quantum gates must have the same number of inputs and outputs, unlike classical gates. Examples:

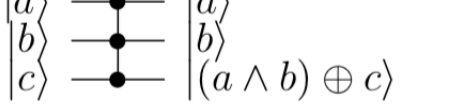
- NOT(X) gate



- CNOT(X) gate

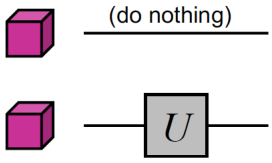


- Toffoli (CCNOT) gate



Note that the CNOT gate has the control bit on top (the filled circle) and the target bit on the bottom (the empty circle).

Let us apply a one-qubit matrix to two qubits.



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

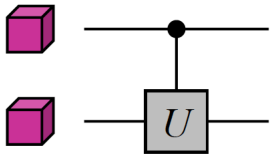
Maps basis states as:

$|0\rangle|0\rangle \rightarrow |0\rangle U|0\rangle$
$|0\rangle|1\rangle \rightarrow |0\rangle U|1\rangle$
$|1\rangle|0\rangle \rightarrow |1\rangle U|0\rangle$
$|1\rangle|1\rangle \rightarrow |1\rangle U|1\rangle$

The resulting 4x4 matrix is

$$I \otimes U = \begin{bmatrix} u_{00} & u_{01} & 0 & 0 \\ u_{10} & u_{11} & 0 & 0 \\ 0 & 0 & u_{00} & u_{01} \\ 0 & 0 & u_{10} & u_{11} \end{bmatrix}$$

Now let us use one qubit to control the application of a matrix U to another qubit.



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

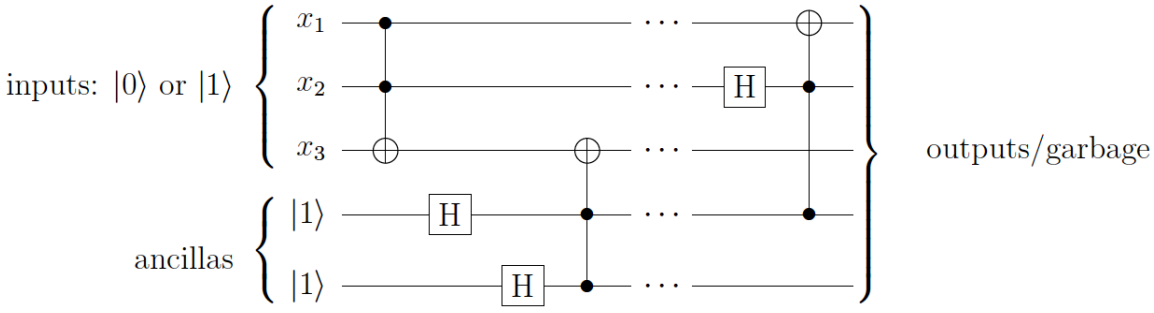
Resulting 4x4 matrix is
controlled-U =

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

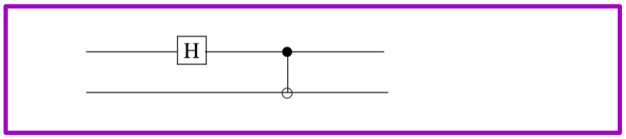
Maps basis states as:

$|0\rangle|0\rangle \rightarrow |0\rangle|0\rangle$
$|0\rangle|1\rangle \rightarrow |0\rangle|1\rangle$
$|1\rangle|0\rangle \rightarrow |1\rangle U|0\rangle$
$|1\rangle|1\rangle \rightarrow |1\rangle U|1\rangle$

General idea of a circuit:



We can map $|0\rangle \otimes |0\rangle$ to the Bell state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ with the following circuit that uses an Hadamard and a *CNOT*.



Let us check the math. The first application of H to the first qubit of $|0\rangle \otimes |0\rangle$ gives us

$$|+\rangle \otimes |0\rangle \quad = \quad \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|10\rangle$$

and now we can apply $CNOT$ and get

$$\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$$

The four Bell states are:

$$\begin{aligned} |\Phi^+\rangle &= \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \\ |\Phi^-\rangle &= \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle) \\ |\Psi^+\rangle &= \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle) \\ |\Psi^-\rangle &= \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle) \end{aligned}$$

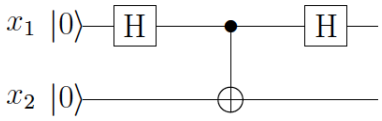
Exercise: show how to generate the other three Bell states by circuits.

Exercise: show that the four Bell states form an orthonormal basis.

For example, we could devise a quantum circuit that carries out Shor's algorithm: it takes as input an n-bit integer, uses roughly n^2 CCNOT and H gates, and has the property that when you measure the final output qubits, they give (with probability at least 99%) the binary encoding of the prime factorization of the input integer (plus some garbage bits).

[Measurement]

Example:



The initial state is $|00\rangle$.

After the first Hadamard gate, the state is

$$\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|10\rangle$$

After the CNOT, the state is

$$\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$$

After the final Hadamard gate, the state is:

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

So we get each of the four outcomes with probability $\frac{1}{4}$.

Transition to MP2: Now let us look at what we can and cannot do with quantum circuits.

Main Point 2: Superdense coding, teleportation, and no-cloning.

[Superdense coding]

Suppose we have n qubits. How many classical bits is that? Holevo's theorem (1973) gives the answer: n bits.

What are the consequences of Holevo's theorem? Example: suppose Alice is trying to encode two bits as a single qubit. Holevo's theorem says that this is impossible: if you have a single qubit, it boils down to a single bit.

However, Alice still wants to send two bits ab to Bob, by sending qubits. Does Alice have to send two qubits? Here is a different idea, called superdense coding, in which Alice sends a single qubit to Bob. The idea is that first Bob sends a qubit to Alice; perhaps way ahead of time.

1. Bob creates the state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ and sends the first qubit A of that state to Alice but keeps the second qubit B.
2. Alice does the following (she creates one of the four states in the Bell basis):
   (a) If $a = 1$, then she applies Z to the qubit A.
   (b) If $b = 1$, then she applies X to the possibly transformed qubit A.
   (c) She sends the possibly transformed qubit A to Bob.
3. Bob does the following (he measures the qubits in the Bell basis):
   (a) He applies CNOT(A, B).
   (b) He applies H to A.
   (c) He measures both A and B.

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

ab	After 2a	After 2b	After 3a	After 3b	Bob sees
00	$\frac{1}{\sqrt{2}}(\|00\rangle + \|11\rangle)$	$\frac{1}{\sqrt{2}}(\|00\rangle + \|11\rangle)$	$\frac{1}{\sqrt{2}}(\|00\rangle + \|10\rangle) = \frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)\|0\rangle$	$\|00\rangle$	00
01	$\frac{1}{\sqrt{2}}(\|00\rangle + \|11\rangle)$	$\frac{1}{\sqrt{2}}(\|10\rangle + \|01\rangle)$	$\frac{1}{\sqrt{2}}(\|11\rangle + \|01\rangle) = \frac{1}{\sqrt{2}}(\|1\rangle + \|0\rangle)\|1\rangle$	$\|01\rangle$	01
10	$\frac{1}{\sqrt{2}}(\|00\rangle - \|11\rangle)$	$\frac{1}{\sqrt{2}}(\|00\rangle - \|11\rangle)$	$\frac{1}{\sqrt{2}}(\|00\rangle - \|10\rangle) = \frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)\|0\rangle$	$\|10\rangle$	10
11	$\frac{1}{\sqrt{2}}(\|00\rangle - \|11\rangle)$	$\frac{1}{\sqrt{2}}(\|10\rangle - \|01\rangle)$	$\frac{1}{\sqrt{2}}(\|11\rangle - \|01\rangle) = \frac{1}{\sqrt{2}}(\|1\rangle - \|0\rangle)\|1\rangle$	$-\|11\rangle$	11

[Teleportation]

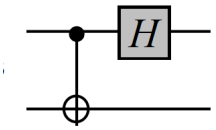
The task: Alice has a qubit $\alpha|0\rangle + \beta|1\rangle$ and wants to tell Bob what it is; she wants to do that by sending classical bits to Bob.

Like in superdense coding, Bob creates the state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ and sends the first qubit of that state to Alice.

Let us look at the state of all three qubits, of which the first two belong to Alice and the third belong to Bob:

$$(\alpha|0\rangle + \beta|1\rangle) \otimes \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

Now Alice measures her two qubits in the Bell basis and sends the result to Bob. Specifically, Alice applies the following to her two qubits:



after which Alice measures her two qubits and gets one of 00, 01, 10, 11, with probability $\frac{1}{4}$, and then sends those two bits to Bob.

Let us go through the steps of how this works. Let us rewrite the state of all three qubits:

$$(\alpha|0\rangle + \beta|1\rangle) \otimes \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \quad = \quad \frac{1}{\sqrt{2}}(\alpha|000\rangle + \alpha|011\rangle + \beta|100\rangle + \beta|111\rangle)$$

Now, after Alice has applied CNOT, the state of all three qubits is:

$$\frac{1}{\sqrt{2}}(\alpha|000\rangle + \alpha|011\rangle + \beta|110\rangle + \beta|101\rangle)$$

Then, after Alice has applied H to the first qubit, the state of all three qubits is:

$$\begin{aligned} & \frac{1}{2}(\alpha|000\rangle + \alpha|100\rangle + \alpha|011\rangle + \alpha|111\rangle + \beta|010\rangle - \beta|110\rangle + \beta|001\rangle - \beta|101\rangle) \\ = \quad & \frac{1}{2}(|00\rangle(\alpha|0\rangle + \beta|1\rangle) + |01\rangle(\alpha|1\rangle + \beta|0\rangle) + |10\rangle(\alpha|0\rangle - \beta|1\rangle) + |11\rangle(\alpha|1\rangle - \beta|0\rangle) \end{aligned}$$

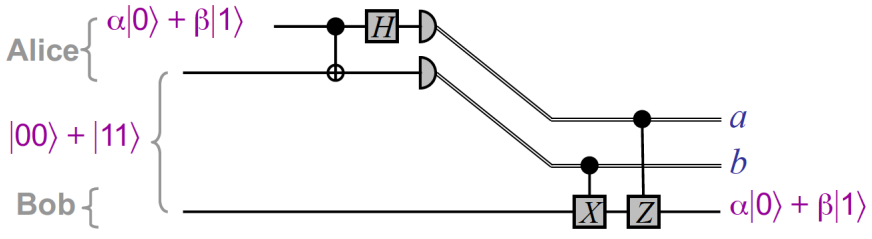
Now Alice measures her two qubits. What is the chance that she sees 00? We can see above that it is the square of $\frac{1}{2}$, which is $\frac{1}{4}$. Similar reasoning applies to the other three cases.

Bob receives the two classical bits ab from Alice, and then Bob does the following:

- If $b = 1$, then he applies X to his qubit.
- If $a = 1$, then he applies Z to his qubit.

The result is:

$$\begin{cases} 00 & : & \alpha|0\rangle + \beta|1\rangle \\ 01 & : & X(\alpha|1\rangle + \beta|0\rangle) \;=\; \alpha|0\rangle + \beta|1\rangle \\ 10 & : & Z(\alpha|0\rangle - \beta|1\rangle) \;=\; \alpha|0\rangle + \beta|1\rangle \\ 11 & : & ZX(\alpha|1\rangle - \beta|0\rangle) \;=\; \alpha|0\rangle + \beta|1\rangle \end{cases}$$



Note that in the process of teleporting a qubit from Alice to Bob, we destroyed Alice's qubit. This is the way it is supposed to be: we cannot clone an unknown qubit, as we will discuss next.

[No-cloning]

We can copy a classical bit; what about a qubit?

Theorem: No quantum operation maps $|\psi\rangle|0\rangle$ to $|\psi\rangle|\psi\rangle$.

Proof: Suppose we have such an operation U, so, for every $|\psi\rangle$:

$$U|\psi\rangle|0\rangle \quad = \quad |\psi\rangle|\psi\rangle \tag{1}$$

Let us pick $|\psi_1\rangle$ and $|\psi_2\rangle$, such that they are not orthogonal and not proportional, that is:

$$\langle\psi_1|\psi_2\rangle \quad \neq \quad 0 \qquad \wedge \qquad \langle\psi_1|\psi_2\rangle \quad \neq \quad 1 \tag{2}$$

Now we will do a calculation that uses the following property:

$$\langle(v_1 \otimes v_2)|(w_1 \otimes w_2)\rangle \quad = \quad \langle v_1|w_1\rangle \ \langle v_2|w_2\rangle \tag{3}$$

We calculate as follows.

$$\begin{array}{rcll}
\langle\psi_1|\psi_2\rangle & = & \langle\psi_1|\psi_2\rangle \ \langle 0|0\rangle & (\langle 0|0\rangle = 1) \\
& = & \langle(\psi_1 \otimes |0\rangle)|(\psi_2 \otimes |0\rangle)\rangle & (\text{use property (3)}) \\
& = & \langle U(\psi_1 \otimes |0\rangle)|U(\psi_2 \otimes |0\rangle)\rangle & (U \text{ preserves inner products}) \\
& = & \langle(\psi_1 \otimes |\psi_1\rangle)|(\psi_2 \otimes |\psi_2\rangle)\rangle & (\text{use our assumption (1)}) \\
& = & \langle\psi_1|\psi_2\rangle^2 & (\text{use property (3)})
\end{array}$$

From this we get that either $\langle\psi_1|\psi_2\rangle = 0$ or $\langle\psi_1|\psi_2\rangle = 1$. This contradicts Equation (2). QED.

Transition to MP3: What is the machine language of quantum circuits?

Main Point 3: Universal sets of operations.

[Universal sets exists]
S is a universal set for a model if any gate from that model can be realized using only combinations of gates from S.

For classical computing, {NAND} is a universal set. Let us review why. We will get there in three steps. Our starting point is that we know that $\{\text{AND}, \text{OR}, \text{NOT}\}$ is universal. First, we implement OR in terms of AND and NOT gates using De Morgan's rule:

$$\text{OR}(x_1, x_2) \quad = \quad \text{NOT}(\text{AND}(\text{NOT}(x_1), \text{NOT}(x_2)))$$

We can also use standard notation:

$$x_1 \vee x_2 \quad = \quad \neg((\neg x_1) \wedge (\neg x_2))$$

Second, we implement AND in terms of NAND gates:

$$\text{AND}(x_1, x_2) \quad = \quad \text{NOT}(\text{NAND}(x_1, x_2)$$

Third, we implement NOT in terms of NAND gates:

$$\text{NOT}(x_1) \quad = \quad \text{NAND}(x_1, 1)$$

Notice that we used a helper bit, initialized to 1. So, the statement that, for classical computing, $\{NAND\}$ is a universal set means that NAND gates plus helper bits is a universal set. We know that helper bits are important in reversible computing and we will use them again and again.

For reversible computing, we observe that NAND is not reversible, so, we have to think harder. Fortunately, NOT is reversible: it is a bijection on $\{0, 1\}$, as the following table shows:

input	output
0	1
1	0

Indeed, NOT(NOT x) $= x$.

Now let us look at CNOT, also known as controlled-NOT:

$$\text{CNOT}(x_1, x_2) \quad = \quad (x_1, x_1 \oplus x_2)$$

Or, in table form:

input	output
00	00
01	01
10	11
11	10

This mapping is a bijection so CNOT is good for reversible computing. Indeed,

$$\text{CNOT}(\text{CNOT } xy \text{ }) \quad = \quad xy$$

Now let us take the CNOT idea a step further and look at CCNOT, also known as controlled-controlled-NOT, and in quantum computing known as the Toffoli gate:

$$\text{CCNOT}(x_1, x_2, x_3) \quad = \quad (x_1, x_2, \text{AND}(x_1, x_2) \oplus x_3)$$

Or, in table form:

input	output
000	000
001	001
010	010
011	011
100	100
101	101
110	111
111	110

This mapping is a bijection so CCNOT is good for reversible computing. Indeed,

$$\text{CCNOT}(\text{CCNOT } xyz\ ) \quad = \quad xyz$$

Now for the crux about reversible computing: we can implement NAND using CCNOT:

$$\text{NAND}(x_1, x_2) \quad = \quad \text{CCNOT}(x_1, x_2, 1)$$

So, for reversible computing, CCNOT gates plus helper bits is a universal set.

Additionally, let us note that we can get $|1\rangle$ from $|0\rangle$ by using CCNOT:

$$\text{CCNOT}(1, 1, 0) \quad = \quad (1, 1, 1)$$

So, helper bits can always be initialized to 0. Or, if we prefer, helper bits can always be initialized to 1.

For probabilistic computing, $\{\text{NAND}, \text{CoinFlip}_p\}$ is a universal set. Here, CoinFlip_p is a coin flip operation on a single bit operating with bias p. In practice, $\{\text{NAND}, \text{CoinFlip}_{\frac{1}{2}}\}$ is a universal set; the fair coin can simulate any biased coin with arbitrary precision.

For quantum computing, $\{\text{CCNOT}, H\}$ is a universal set. This set can express all real unitary matrices. Once we add S, we can also express all complex unitary matrices.

$$S \quad = \quad \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$$

The foundational people love the gate set $\{\text{CCNOT}, H, S\}$ because they can prove easily that it is universal.

[Universal sets can be quite different]

For quantum programming, the gate set $\{\text{CCNOT}, H, S\}$ is difficult to work with. Rather than using CCNOT for everything, CNOT would be easier. So, for quantum programming, people prefer to use the universal gate set $\{\text{CNOT}, H, T\}$, where

$$T \quad = \quad \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$$

How do we know that $\{\text{CNOT}, H, T\}$ is universal? All we need to do is to implement everything in the universal gate set $\{\text{CCNOT}, H, S\}$, which boils down to implementing CCNOT and S. Turns out, CCNOT has a well-known implementation that uses six CNOT gates and some H gates and T gates. Additionally, the foundational people have shown that no combination of five or fewer CNOT gates, plus 1-qubit gates, implements CCNOT. Also, S has an easy implementation in terms of T; indeed, $S = (T\ T)$:

[Different quantum computers support different sets of matrices]
Aside from a gate set that we can easily prove universal and a gate set that we can use for programming, we also need a gate set that we can implement without too much trouble. The concept of a finite universal gate set is welcome here; it gives the implementers a finite list of tasks to do.

Which finite gate set is easy to implement? Turns out, this depends on the qubit technology. Here are two examples.

Quantum computer	Gate set
IBMQX5	$u_1, u_2, u_3, \texttt{CNOT}$
Ion Trap (Duke U.)	$R_x, R_y, R_z, \texttt{XX}$

The gates are defined as follows.

$$
\begin{aligned}
u_1(\lambda) &= \begin{pmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{pmatrix} \\
u_2(\varphi, \lambda) &= \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -e^{i\lambda} \\ e^{i\phi} & e^{i(\lambda+\phi)} \end{pmatrix} \\
u_3(\theta, \phi, \lambda) &= \begin{pmatrix} \cos(\theta/2) & -e^{i\lambda}\sin(\theta/2) \\ e^{i\phi}\sin(\theta/2) & e^{i(\lambda+\phi)}\cos(\theta/2) \end{pmatrix} \\
R_x(\theta) &= e^{-i\theta X/2} = \begin{pmatrix} \cos(\theta/2) & -i\sin(\theta/2) \\ -i\sin(\theta/2) & \cos(\theta/2) \end{pmatrix} \\
R_y(\theta) &= e^{-i\theta Y/2} = \begin{pmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{pmatrix} \\
R_z(\theta) &= e^{-i\theta Z/2} = \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix} \\
\texttt{XX}(\phi) &= \frac{1}{\sqrt{2}} \begin{pmatrix} \cos\phi & 0 & 0 & -i\sin\phi \\ 0 & \cos\phi & -i\sin\phi & 0 \\ 0 & -i\sin\phi & \cos\phi & 0 \\ -i\sin\phi & 0 & 0 & \cos\phi \end{pmatrix}
\end{aligned}
$$

For each gate set, we can prove that it is universal like we did above: show that we can implement everything in a universal gate set.

Notice that having one gate set for programming and another gate set at the level of the quantum computer creates a compilation task. Specifically, we must translate each program level gate to some quantum-computer-level gates.

Transition to Close: So there you have it.

Review: We can draw circuit diagrams and do amazing little tasks, and we can talk about universal sets of operations.

Strong finish: Now we are ready to study quantum algorithms, which we will express as quantum circuits.

Call to action: Play around with the quantum circuit simulator that we linked online.

Quantum Programming Algorithms: Deutsch-Jozsa

Jens Palsberg

Jan 18, 2022

Outline

Hook: Our first quantum algorithm is the Deutsch-Jozsa algorithm. It solves an artificial problem, but does so faster on a quantum computer than we can ever do on a classical computer.

Purpose: Persuade you that you can understand this algorithm.

Preview:

1. We will make sure that everybody understands the problem.
2. We will begin with a simple version called Deutsch's algorithm.
3. We will cover full details of how and why the Deutsch-Jozsa algorithm works.

Transition to Body: Now let us talk about the problem that this algorithm solves.

Main Point 1: We will make sure that everybody understands the problem.
[Most of you have done the homework on solving the problem on a classical computer]
[We can represent the black-box function in multiple ways]
[We can program the solution in any classical language]

Transition to MP2: Now let us look at a solution on a quantum computer.

Main Point 2: We will begin with a simple version called Deutsch's algorithm.
[We make the black-box computation reversible]
[We construct a circuit that puts each of two qubits in superposition]
[The reasoning about correctness uncovers the phase-kickback trick]

Transition to MP3: Now we are ready for the full solution.

Main Point 3: We will cover full details of how and why the Deutsch-Jozsa algorithm works.
[We can make a black box with n inputs reversible]
[The structure is the same as in Deutsch's algorithm]
[The reasoning about the phase-kickback trick is more complicated with n bits]

Transition to Close: So there you have it.

Review: The Deutsch-Jozsa algorithm makes a single call to the black-box function, which works because it tries all combinations of $|0\rangle$ and $|1\rangle$ at the same time.

Strong finish: The Deutsch-Jozsa algorithm is the first indication that a quantum computer can be faster than a classical computer. We will see more examples of that in this course.

Call to action: Learn how to reason about correctness by getting good at calculating with Dirac notation.

Detailed presentation

Hook: Our first quantum algorithm is the Deutsch-Jozsa algorithm. It solves an artificial problem, but does so faster on a quantum computer than we can ever do on a classical computer.

Purpose: Persuade you that you can understand this algorithm.

Preview:

1. We will make sure that everybody understands the problem.
2. We will begin with a simple version called Deutsch's algorithm.
3. We will cover full details of how and why the Deutsch-Jozsa algorithm works.

Transition to Body: Now let us talk about the problem that this algorithm solves.

Main Point 1: We will make sure that everybody understands the problem.

[Most of you have done the homework on solving the problem on a classical computer]
Here is the homework problem.

> The Deutsch-Jozsa problem:
> Input: a function $f : \{0,1\}^n \to \{0,1\}$.
> Assumption: f is either constant or balanced.
> Output: 0 if f is constant; 1 if f is balanced.
> Notation: $\{0,1\}$ is the set of bits, and $\{0,1\}^n$ is the set of bit strings of length n.
> Terminology:
> Constant: f is constant if either f always outputs 0 or f always outputs 1.
> Balanced: f is balanced if f outputs 0 on exactly half of the inputs.
>
> The assignment:
> On a classical computer, in a classical language of your choice (such as C, Java, Python, etc), program solutions to the Deutsch-Jozsa problem. Treat the input function f as black box that you can call but cannot inspect in any way at all. Each solution will be code that includes one or more calls of f.

[We can represent the black-box function in multiple ways]
For the case of $n = 1$, we have that f can be one of four functions, here called f_0, f_1, f_2, f_3:

Input	f_0	f_1	f_2	f_3
0	0	0	1	1
1	0	1	0	1

Notice that f_0 and f_3 are constant, while f_1 and f_2 are balanced.

How do we keep the black-box function at an arms length such that we cannot get tempted to inspect it? In C we can use a function pointer. In Java we can use a lambda expression. In Python we can use an anonymous function. Other languages have similar constructs that will enable us to call the function, while giving us no way to inspect it.

[We can program the solution in any classical language]
You wrote the solutions to the homework in several languages.

If we want an exact solution, we need $2^{n-1}+1$ queries in the worst case. If we can tolerate that we get the right answer with high probability, we can randomly choose k inputs; evaluate f on each of the k inputs, and if the outputs are all the same, then output *constant*, and otherwise output *balanced*. If the function really is constant, this method is correct every time, and if the function is balanced, this method will be wrong (and answer *constant*) with probability $2^{-(k-1)}$.

Transition to MP2: Now let us look at a solution on a quantum computer.

Main Point 2: We will begin with a simple version called Deutsch's algorithm.

[We make the black-box computation reversible]

Here we bring in the technique from the previous lecture. Specifically, if we have an operation

$$f \;:\; \{0,1\} \to \{0,1\}$$

then we can represent it as an invertible operation U_f:

$$\begin{aligned} U_f &: \mathit{Qubit}^{\otimes 2} \to \mathit{Qubit}^{\otimes 2} \\ U_f|x\rangle|b\rangle &= |x\rangle|b \oplus f(x)\rangle \end{aligned}$$

We use $\mathit{Qubit}^{\otimes n}$ to denote the tensor product of n vectors spaces of single qubits. We use $\oplus$ to denote addition (mod 2).

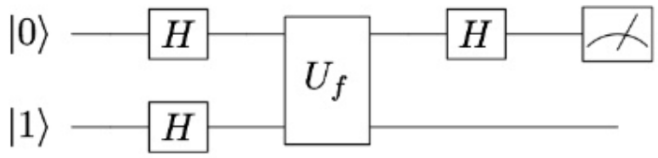
Let us check that for every one of the four possibilities of f, we have that U_f is a unitary function.

$$U_{f_0} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = I \qquad U_{f_1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} = \mathit{CNOT}$$

$$U_{f_2} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad U_{f_3} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Notice that each of the four matrices above is a permutation matrix: every entry is 0 or 1, every column has a single 1, and every row has a single 1. Every permutation matrix is unitary.

[We construct a circuit that puts each of two qubits in superposition]



If we measure 0, we conclude that the function is *constant*, and otherwise it is *balanced*.

[The reasoning about correctness uncovers the phase-kickback trick]

Lemma 1. $\forall a \in \{0,1\} : |0 \oplus a\rangle - |1 \oplus a\rangle \;=\; (-1)^a\;(|0\rangle - |1\rangle)$.

Proof. Let us try both possibilities for a in turn.
For $a = 0$, we have:

$$\begin{array}{rrcccc} \text{left-hand side:} & |0 \oplus a\rangle - |1 \oplus a\rangle & = & |0 \oplus 0\rangle - |1 \oplus 0\rangle & = & |0\rangle - |1\rangle \\ \text{right-hand side:} & (-1)^a\;(|0\rangle - |1\rangle) & = & (-1)^0\;(|0\rangle - |1\rangle) & = & |0\rangle - |1\rangle \end{array}$$

For $a = 1$, we have:

$$\begin{array}{rrcccc} \text{left-hand side:} & |0 \oplus a\rangle - |1 \oplus a\rangle & = & |0 \oplus 1\rangle - |1 \oplus 1\rangle & = & |1\rangle - |0\rangle \\ \text{right-hand side:} & (-1)^a\;(|0\rangle - |1\rangle) & = & (-1)^1\;(|0\rangle - |1\rangle) & = & |1\rangle - |0\rangle \end{array}$$

□

Lemma 2 (Phase kickback)**.** $\forall x \in \{0,1\}^n :\; U_f|x\rangle|-\rangle \;=\; (-1)^{f(x)}|x\rangle|-\rangle$.

Proof.

$$\begin{array}{rcl} & & U_f|x\rangle|-\rangle \\ & = & \dfrac{1}{\sqrt{2}}\;(U_f|x\rangle|0\rangle - U_f|x\rangle|1\rangle) \\ & = & \dfrac{1}{\sqrt{2}}\;(|x\rangle|f(x)\rangle - |x\rangle|1 \oplus f(x)\rangle) \\ & = & |x\rangle \otimes \dfrac{1}{\sqrt{2}}\;(|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle) \\ & = & |x\rangle \otimes \dfrac{1}{\sqrt{2}}\;(-1)^{f(x)}\;(|0\rangle - |1\rangle) \\ & = & (-1)^{f(x)}\;|x\rangle \otimes \dfrac{1}{\sqrt{2}}\;(|0\rangle - |1\rangle) \\ & = & (-1)^{f(x)}\;|x\rangle|-\rangle \end{array}$$

In second step, we used $|-\rangle = \frac{1}{\sqrt{2}}\;(|0\rangle - |1\rangle)$. In the fourth step, we used Lemma 1. □

Lemma 3. $\forall a \in \{0,1\} : H\left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}(-1)^a|1\rangle\right) \;=\; |a\rangle$.

Proof. Let us try both possibilities for a in turn.
For $a = 0$, we have:

$$\begin{array}{rcccccccc} H\left(\dfrac{1}{\sqrt{2}}|0\rangle + \dfrac{1}{\sqrt{2}}(-1)^a|1\rangle\right) & = & H\left(\dfrac{1}{\sqrt{2}}|0\rangle + \dfrac{1}{\sqrt{2}}(-1)^0|1\rangle\right) & = & H\left(\dfrac{1}{\sqrt{2}}|0\rangle + \dfrac{1}{\sqrt{2}}|1\rangle\right) & = & H|+\rangle \\ & = & |0\rangle & = & |a\rangle & & \end{array}$$

For $a = 1$, we have:

$$\begin{array}{rcccccccc} H\left(\dfrac{1}{\sqrt{2}}|0\rangle + \dfrac{1}{\sqrt{2}}(-1)^a|1\rangle\right) & = & H\left(\dfrac{1}{\sqrt{2}}|0\rangle + \dfrac{1}{\sqrt{2}}(-1)^1|1\rangle\right) & = & H\left(\dfrac{1}{\sqrt{2}}|0\rangle - \dfrac{1}{\sqrt{2}}|1\rangle\right) & = & H|-\rangle \\ & = & |1\rangle & = & |a\rangle & & \end{array}$$

□

Theorem 4. *Deutsch's algorithm is correct.*

Proof. Initially, the state of the two qubits is $|01\rangle$.

After the two uses of H, followed by the use of U_f, followed by the single use of H, the state of the two qubits is:

$$\begin{aligned}
& (H \otimes I) \circ U_f \circ (H \otimes H)(|01\rangle) \\
= \; & (H \otimes I) \circ U_f \; (|+\rangle \otimes |-\rangle) \\
= \; & (H \otimes I) \circ U_f \; (\frac{1}{\sqrt{2}} \; (|0\rangle + |1\rangle) \otimes |-\rangle) \\
= \; & (H \otimes I) \circ U_f \; (\frac{1}{\sqrt{2}} \; (\Sigma_{x \in \{0,1\}} |x\rangle) \otimes |-\rangle) \\
= \; & (H \otimes I) \frac{1}{\sqrt{2}} \; \Sigma_{x \in \{0,1\}} \; (-1)^{f(x)} |x\rangle \otimes |-\rangle \\
= \; & (H \otimes I)(\frac{1}{\sqrt{2}} \; ((-1)^{f(0)} |0\rangle \; + \; (-1)^{f(1)} |1\rangle)) \; \otimes \; |-\rangle \\
= \; & (H \otimes I)((-1)^{f(0)} \; \frac{1}{\sqrt{2}} \; (|0\rangle \; + \; (-1)^{f(0) \oplus f(1)} |1\rangle)) \; \otimes \; |-\rangle \\
= \; & (H \otimes I)((-1)^{f(0)} \; (\frac{1}{\sqrt{2}} \; |0\rangle \; + \; \frac{1}{\sqrt{2}} \; (-1)^{f(0) \oplus f(1)} |1\rangle)) \; \otimes \; |-\rangle \\
= \; & ((-1)^{f(0)} \; |f(0) \oplus f(1)\rangle) \; \otimes \; |-\rangle
\end{aligned}$$

In the fourth step, we use Lemma 2. In the last step, we use Lemma 3. So, if we measure the first qubit, we will get $f(0) \oplus f(1)$ with probability

$$|((-1)^{f(0)}|^2 \;\; = \;\; 1^2 \;\; = \;\; 1$$

Notice that

$$f(0) \oplus f(1) \;\; = \;\; \begin{cases} 0 & f \text{ is constant} \\ 1 & f \text{ is balanced} \end{cases}$$

The grand conclusion is that if we measure 0, we output "constant", and if we measure 1, we output "balanced". □

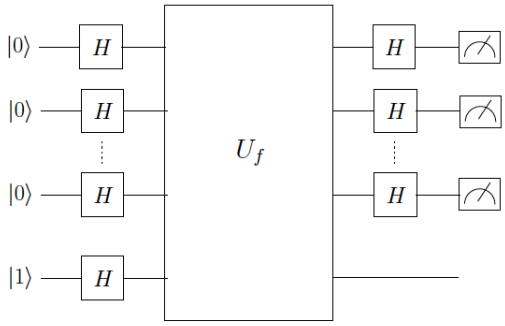
Transition to MP3: Now we are ready for the full solution.

Main Point 3: We will cover full details of how and why the Deutsch-Jozsa algorithm works.

[We can make a black box with n inputs reversible]
The above construction of permutation matrices to support reversible computing generalizes to functions f from n bits to m bits.

[The structure is the same as in Deutsch's algorithm]

Like in the case of Deutsch's algorithm, we use a single call to U_f.



The idea is that we get n bits from the measurements. If all those bits are 0, we conclude that the function is *constant*, and otherwise it is *balanced*. Notice that we call U_f just once.

[The reasoning about the phase-kickback trick is more complicated with n bits]

Lemma 5. $\forall x \in \{0,1\}: \quad H|x\rangle \;=\; \frac{1}{\sqrt{2}} \Sigma_{y\in\{0,1\}} \; (-1)^{x\cdot y} \; |y\rangle.$

Proof. The left-hand side:

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

Here, we use Lemma 3.

The right-hand side:

$$\frac{1}{\sqrt{2}} \Sigma_{y\in\{0,1\}} \; (-1)^{x\cdot y} \; |y\rangle \quad = \quad \frac{1}{\sqrt{2}} \left((-1)^{x\cdot 0} \; |0\rangle + (-1)^{x\cdot 1} \; |1\rangle\right) \quad = \quad \frac{1}{\sqrt{2}} \left(|0\rangle + (-1)^x|1\rangle\right)$$

We see that the two sides are equal. □

We use $H^{\otimes n}$ to denote $H \otimes H \otimes \ldots \otimes H$ (n occurrences of H). If x and y are bit-strings of the same length, then $x \cdot y$ denotes the dot-product (mod 2) of x and y.

Lemma 6. $H^{\otimes n}|x\rangle \;=\; \frac{1}{\sqrt{2^n}} \Sigma_{y\in\{0,1\}^n} \; (-1)^{x\cdot y} \; |y\rangle$

Proof.

$$\begin{aligned}
& H^{\otimes n}|x\rangle \\
= \;& H|x_1\rangle \otimes \ldots \otimes H|x_n\rangle \\
= \;& \left(\frac{1}{\sqrt{2}} \Sigma_{y_1\in\{0,1\}} \; (-1)^{x_1\cdot y_1} \; |y_1\rangle\right) \otimes \ldots \otimes \left(\frac{1}{\sqrt{2}} \Sigma_{y_n\in\{0,1\}} \; (-1)^{x_n\cdot y_n} \; |y_n\rangle\right) \\
= \;& \frac{1}{\sqrt{2^n}} \Sigma_{y_1\in\{0,1\}} \; \ldots \Sigma_{y_n\in\{0,1\}} \; (-1)^{x_1\cdot y_1} \; \ldots \; (-1)^{x_n\cdot y_n} \; |y_1\rangle \ldots \; |y_n\rangle \\
= \;& \frac{1}{\sqrt{2^n}} \Sigma_{y\in\{0,1\}^n} \; (-1)^{x\cdot y} \; |y\rangle
\end{aligned}$$

In the second step, we use Lemma 5 a total of n times. □

Theorem 7. *The Deutsch-Jozsa algorithm is correct.*

Proof. Initially, the state of the $n+1$ qubits is $|00\ldots01\rangle$.

After the $n+1$ uses of H followed by the use of U_f followed by the n uses of H, the state of the $n+1$ qubits is:

$$\begin{aligned}
& (H^{\otimes n} \otimes I) \circ U_f \circ H^{\otimes n+1}(|00\ldots01\rangle) \\
= \; & (H^{\otimes n} \otimes I) \circ U_f \; \frac{1}{\sqrt{2^n}} \; \Sigma_{x\in\{0,1\}^n} \; |x\rangle \otimes |-\rangle \\
= \; & (H^{\otimes n} \otimes I) \; \frac{1}{\sqrt{2^n}} \; \Sigma_{x\in\{0,1\}^n} \; (-1)^{f(x)} \; |x\rangle \otimes |-\rangle \\
= \; & \frac{1}{\sqrt{2^n}} \; \Sigma_{x\in\{0,1\}^n} \; (-1)^{f(x)} \; (\frac{1}{\sqrt{2^n}} \; \Sigma_{y\in\{0,1\}^n} \; (-1)^{x\cdot y} \; |y\rangle) \otimes |-\rangle \\
= \; & \frac{1}{2^n} \; \Sigma_{x\in\{0,1\}^n} \; \Sigma_{y\in\{0,1\}^n} \; (-1)^{x\cdot y \oplus f(x)} \; |y\rangle \otimes |-\rangle
\end{aligned}$$

In the first step, we used Lemma 6; in the second step, we used Lemma 2; and in the third step, we used Lemma 6.

Now we measure the first n qubits. Turns out, the state $|00\ldots0-\rangle$ is important:

$$\text{the amplitude of } |00\ldots0-\rangle \;\; = \;\; \frac{1}{2^n} \; \Sigma_{x\in\{0,1\}^n} \; (-1)^{f(x)}$$

Let us consider the two cases of f in turn.

First, suppose f is constant.

$$\text{the amplitude of } |00\ldots0-\rangle \;\; = \;\; \begin{cases} 1 & \text{if } f(x) = 0 \text{ for all } x \\ -1 & \text{if } f(x) = 1 \text{ for all } x \end{cases}$$

We conclude that if f is constant, a measurement of the first n qubits will give us $00\ldots0$ with probability 1.

Second, suppose f is balanced.

$$\text{the amplitude of } |00\ldots0-\rangle \;\; = \;\; 0$$

The reason is that $(-1)^{f(x)}$ will be 1 half of the time and -1 half of time; they will cancel either other out. We conclude that if f is balanced, a measurement of the first n qubits will give us $00\ldots0$ with probability 0

The grand conclusion is that if we measure $00\ldots0$, we output "constant", and if we measure anything else, we output "balanced". □

Transition to Close: So there you have it.

Review: The Deutsch-Jozsa algorithm makes a single call to the black-box function, which works because it tries all combinations of $|0\rangle$ and $|1\rangle$ at the same time.

Strong finish: The Deutsch-Jozsa algorithm is the first indication that a quantum computer can be faster than a classical computer. We will see more examples of that in this course.

Call to action: Learn how to reason about correctness by getting good at calculating with Dirac notation.

Quantum Programming Algorithms: Bernstein-Vazirani

Jens Palsberg

Jan 20, 2022

Outline

Hook: Our second quantum algorithm is the Bernstein-Vazirani algorithm. It solves a natural problem and does so faster on a quantum computer than we can ever do on a classical computer.

Purpose: Persuade you that you can understand this algorithm.

Preview:

1. We will make sure that everybody understands the problem.
2. We will cover full details of how and why the algorithm works.
3. We will show an additional algorithm that uses similar ideas.

Transition to Body: Now let us talk about the problem that this algorithm solves.

Main Point 1: We will make sure that everybody understands the problem.
[Most of you have done the homework on solving the problem on a classical computer]
[We can represent the black-box function in multiple ways]
[We can program the solution in any classical language]

Transition to MP2: Now let us look at how to solve the problem.

Main Point 2: We will cover full details of how and why the algorithm works.
[We will mix classical and quantum computing]
[The structure of the quantum part is the same as in Deutsch-Josza]
[We can use some of the lemmas about Deutsch-Josza]

Transition to MP3: Now let us apply our knowledge to a different case.

Main Point 3: We will show an additional algorithm that uses similar ideas.
[The problem is a special case of Grover's problem]
[The solution uses a different circuit]
[The reasoning uses some of the same lemmas that we proved earlier]

Transition to Close: So there you have it.

Review: The Bernstein-Vazirani algorithm combines classical and quantum computing.

Strong finish: The Bernstein-Vazirani algorithm is the second indication that a quantum computer can be faster than a classical computer. We will see more examples of that in this course.

Call to action: Look for other natural problems that may be solvable on quantum computers.

Detailed presentation

Hook: Our second quantum algorithm is the Bernstein-Vazirani algorithm. It solves a natural problem and does so faster on a quantum computer than we can ever do on a classical computer.

Purpose: Persuade you that you can understand this algorithm.

Preview:

1. We will make sure that everybody understands the problem.
2. We will cover full details of how and why the algorithm works.
3. We will show an additional algorithm that uses similar ideas.

Transition to Body: Now let us talk about the problem that this algorithm solves.

Main Point 1: We will make sure that everybody understands the problem.

[Most of you have done the homework on solving the problem on a classical computer]
Here is the homework problem.

> The Bernstein-Vazirani problem:
> Input: a function $f : \{0,1\}^n \to \{0,1\}$.
> Assumption: $f(x) = (a \cdot x) \oplus b$.
> Output: a, b.
> Notation: $\{0,1\}^n$ is the set of bit strings of length n, a is an unknown bit string of length n, $\cdot$ is dot product mod 2, $\oplus$ is addition mod 2, and b is an unknown single bit.
>
> The assignment:
> On a classical computer, in a classical language of your choice (such as C, Java, Python, etc), program solutions to the Bernstein-Vazirani problem. Treat the input function f as black box that you can call but cannot inspect in any way at all. Each solution will be code that includes one or more calls of f.

Notice that if $a = 00\ldots0$, then f is *constant*. Additionally, if $a \neq 00\ldots0$, then f is *balanced*. So, solving Bernstein-Vazirani is a lot like solving Deutsch-Josza.

[We can represent the black-box function in multiple ways]
How do we keep the black-box function at an arms length such that we cannot get tempted to inspect it? In C we can use a function pointer. In Java we can use a lambda expression. In Python we can use an anonymous function. Other languages have similar constructs that will enable us to call the function, while giving us no way to inspect it.

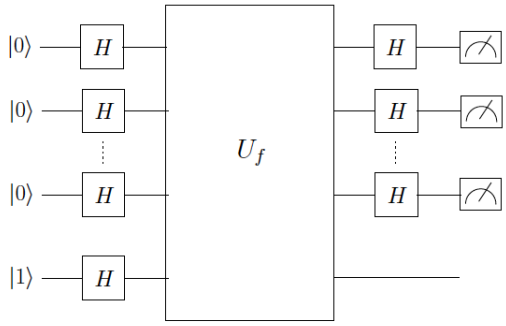
[We can program the solution in any classical language]
You wrote the solutions to the homework in several languages. We need to make several calls to f: a total of $n + 1$ calls to f are needed. The reason is that we need to call $f(00\ldots0) = b$, and then we need to do n calls to determine a, one call for each bit of a.

Transition to MP2: Now let us look at how to solve the problem.

Main Point 2: We will cover full details of how and why the algorithm works.

[We will mix classical and quantum computing]
First do a classical call $f(00\ldots0) = b$. Then use a quantum circuit to determine a.

[The structure of the quantum part is the same as in Deutsch-Josza]



[We can use some of the lemmas about Deutsch-Josza]
After the second round of uses of H, we know that the state of the first n qubits is:

$$\begin{aligned} & \frac{1}{2^n} \Sigma_{x\in\{0,1\}^n} \Sigma_{y\in\{0,1\}^n} (-1)^{(x\cdot y)\oplus f(x)} |y\rangle \\ = & \frac{1}{2^n} \Sigma_{x\in\{0,1\}^n} \Sigma_{y\in\{0,1\}^n} (-1)^{(x\cdot y)\oplus((a\cdot x)\oplus b)} |y\rangle \\ = & \frac{1}{2^n} (-1)^b \Sigma_{x\in\{0,1\}^n} \Sigma_{y\in\{0,1\}^n} (-1)^{(a\oplus y)\cdot x} |y\rangle \\ = & (-1)^b |a\rangle \end{aligned}$$

In the third step, we rely on this calculation:

$$\begin{aligned} & \text{The amplitude of } |a\rangle \\ = & \frac{1}{2^n} (-1)^b \Sigma_{x\in\{0,1\}^n} (-1)^{(a\oplus a)\cdot x} \\ = & \frac{1}{2^n} (-1)^b \Sigma_{x\in\{0,1\}^n} 1 \\ = & (-1)^b \end{aligned}$$

So, when we measure the first n qubits, we will observe $|a\rangle$ with probability $|(-1)^b|^2 = 1^2 = 1$.

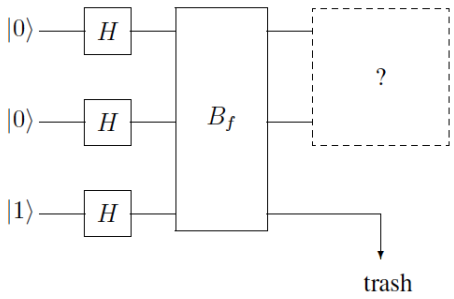
Transition to MP3: Now let us apply our knowledge to a different case.

Main Point 3: We will show an additional algorithm that uses similar ideas.

[The problem is a special case of Grover's problem]
Suppose we are given $f : \{0,1\}^2 \to \{0,1\}$ and we are promised that f returns 1 for a single input and that f returns 0 otherwise. Our goal is to determine which input makes f return 1. In classical computing, we have to make three calls of f to solve the problem.

Grover's problem generalizes this situation to a function f of n bits.

[The solution uses a different circuit]



Above, the diagram uses B_f instead of U_f. The box with the "?" denotes the following matrix:

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

[The reasoning uses some of the same lemmas that we proved earlier]
Let us give names to four particular superpositions:

$$\begin{aligned} \phi_{00} &= \frac{1}{2}(-|00\rangle + |01\rangle + |10\rangle + |11\rangle) \\ \phi_{01} &= \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle + |11\rangle) \\ \phi_{10} &= \frac{1}{2}(|00\rangle + |01\rangle - |10\rangle + |11\rangle) \\ \phi_{11} &= \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle - |11\rangle) \end{aligned}$$

Lemma 1. $\forall c, d \in \{0,1\} : \; U(\, \phi_{cd} \,) = |cd\rangle$.

Proof. The proof has four cases. We will show one of the cases; the others are similar.

$$U(\, \phi_{00} \,) = \frac{1}{2}\begin{pmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{pmatrix} \left(\frac{1}{2}\begin{pmatrix} -1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \right) = \frac{1}{4}\begin{pmatrix} 4 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = |00\rangle$$

□

Theorem 2. *The algorithm works.*

Proof. The initial state of the three qubits is $|001\rangle$.

After using the three H, followed by the use of U_f, followed by the use of U, the state of the three qubits is:

$$\begin{aligned}
& (U \otimes I) \circ U_f \circ (H \otimes H \otimes H)\ |001\rangle \\
= \ & (U \otimes I) \circ U_f\ |{+}{+}{-}\rangle \\
= \ & (U \otimes I) \circ U_f\ \frac{1}{2}(|00-\rangle + |01-\rangle + |10-\rangle + |11-\rangle) \\
= \ & (U \otimes I)\frac{1}{2}((-1)^{f(00)}|00-\rangle + (-1)^{f(01)}|01-\rangle + (-1)^{f(10)}|10-\rangle + (-1)^{f(11)}|11-\rangle) \\
= \ & (U \otimes I)\frac{1}{2}(((-1)^{f(00)}|00\rangle + (-1)^{f(01)}|01\rangle + (-1)^{f(10)}|10\rangle + (-1)^{f(11)}|11\rangle) \otimes |-\rangle) \\
= \ & |cd\rangle|-\rangle \quad \text{where } f(cd) = 1
\end{aligned}$$

In the third step, we used the phase-kickback lemma. In the fifth step, we used Lemma 1.

Now we measure the first two qubits; with probability $|1|^2 = 1$, we get cd where $f(cd) = 1$. □

Transition to Close: So there you have it.

Review: The Bernstein-Vazirani algorithm combines classical and quantum computing.

Strong finish: The Bernstein-Vazirani algorithm is the second indication that a quantum computer can be faster than a classical computer. We will see more examples of that in this course.

Call to action: Look for other natural problems that may be solvable on quantum computers.

Quantum Programming Algorithms: Simon

Jens Palsberg

Jan 25, 2022

Outline

Hook: Our next quantum algorithm is Simon's algorithm. It solves a natural problem and does so faster on a quantum computer than we can ever do on a classical computer.

Purpose: Persuade you that you can understand this algorithm.

Preview:

1. We will make sure that everybody understands the problem.
2. We will look at how Simon's algorithm mixes quantum and classical computing.
3. We will cover full details of how Simon's algorithm works.

Transition to Body: Now let us talk about the problem that this algorithm solves.

Main Point 1: We will make sure that everybody understands the problem.
[Most of you have done the homework on solving the problem on a classical computer]
[We can represent the black-box function in multiple ways]
[We can program the solution in any classical language]

Transition to MP2: Now let us look at how to solve the problem.

Main Point 2: We will look at how Simon's algorithm mixes quantum and classical computing.
[First quantum produces a set of equations, and then classical solves them]
[We can solve the equations with an off-the-shelf classical solver]
[Simon's algorithm is an approximation algorithm]

Transition to MP3: Now let us look at the quantum side of Simon's algorithm.

Main Point 3: We will cover full details of Simon's algorithm works.
[The quantum circuit is a variation of what we have seen before]
[The number of helper bits is equal to the number of input bits]
[The reasoning about correctness relies on some known lemmas]

Transition to Close: So there you have it.

Review: Simon's algorithm combines classical and quantum computing.

Strong finish: Simon's algorithm is a powerful indication that a quantum computer can be faster than a classical computer. We will see more examples of that in this course.

Call to action: Look for other natural problems that may be solvable on quantum computers.

Detailed presentation

Hook: Our next quantum algorithm is Simon's algorithm. It solves a natural problem and does so faster on a quantum computer than we can ever do on a classical computer.

Purpose: Persuade you that you can understand this algorithm.

Preview:

1. We will make sure that everybody understands the problem.
2. We will look at how Simon's algorithm mixes quantum and classical computing.
3. We will cover full details of how Simon's algorithm works.

Transition to Body: Now let us talk about the problem that this algorithm solves.

Main Point 1: We will make sure that everybody understands the problem.

[Most of you have done the homework on solving the problem on a classical computer] Here is the homework problem.

Simon's problem:
Input: a function $f : \{0,1\}^n \to \{0,1\}^n$.
Assumption: there exists $s \in \{0,1\}^n$ such that

$$\forall x, y : \ [f(x) = f(y)] \text{ iff } [(x + y) \in \{0^n, s\}].$$

Output: s.
Notation: $\{0,1\}^n$ is the set of bit strings of length n, s is an unknown bit string of length n, $=$ is comparison of bit strings of length n, $+$ is pointwise addition mod 2 of bit strings of length n, and 0^n is a bit string of length n with all 0.

The assignment:
On a classical computer, in a classical language of your choice (such as C, Java, Python, etc), program a solution to Simon's problem. Treat the input function f as black box that you can call but cannot inspect in any way at all. Each solution will be code that includes one or more calls of f.

Let us consider the assumption in more detail. Specifically, let us exhibit a particular f that satisfies the assumption. Suppose $n = 3$ and let f be the function given by the following table:

x	$f(x)$
000	101
001	010
010	000
011	110
100	000
101	110
110	101
111	010

For the above f, the assumption is correct and $s = 110$. In particular, notice that every output of f occurs twice. Indeed, f has four different outputs: 000, 010, 101, 110. We can check that $s = 110$ is what we need by considering each of the four outputs in turn.

For output 000, the two inputs are 010 and 100 and we see that $010 \oplus 100 = 110 = s$.
For output 010, the two inputs are 001 and 111 and we see that $001 \oplus 111 = 110 = s$.
For output 101, the two inputs are 000 and 110 and we see that $000 \oplus 110 = 110 = s$.
For output 110, the two inputs are 011 and 101 and we see that $011 \oplus 101 = 110 = s$.
We conclude that f satisfies the assumption.

Any f that satisfies the assumption deserves to be called a 2-to-1 function. As a special case, we may have an f that satisfies the assumption with $s = 0^n$, in which case f is a 1-to-1 function.

[We can represent the black-box function in multiple ways]
How do we keep the black-box function at an arms length such that we cannot get tempted to inspect it? In C we can use a function pointer. In Java we can use a lambda expression. In Python we can use an anonymous function. Other languages have similar constructs that will enable us to call the function, while giving us no way to inspect it.

[We can program the solution in any classical language]
You wrote the solutions to the homework in several languages. We need to make many calls to f: the reason is that we need to find two different inputs x, y such that $f(x) = f(y)$. We need to try $\Omega(\sqrt{2^n})$ inputs before we have a reasonable chance of finding such x, y.

Transition to MP2: Now let us look at how to solve the problem.

Main Point 2: We will look at how Simon's algorithm mixes quantum and classical computing.

[First quantum produces a set of equations, and then classical solves them]
The idea is to use quantum computing for what it is good at, and then do the rest by classical computing. We will do that by using quantum computing to map the input function f to equations that captures everything we need to know about f. Then we will use classical computing to map the equations to s. The equations is the interface between quantum and classical. The above process may fail so we will repeat it until the chance of success is high. Specifically, the classical part may fail to produce s. Thus, we can illustrate the entire process with the followed diagram.

$$\text{repeat } \left(f \xrightarrow{\text{quantum}} \text{equations} \xrightarrow{\text{classical}} s \right) \text{ until the chance of success is high}$$

The above interface consists of a set of $n - 1$ equations:

$$\begin{aligned} y_1 \cdot s &= 0 \\ y_2 \cdot s &= 0 \\ &\vdots \\ y_{n-1} \cdot s &= 0 \end{aligned}$$

Above, $y_1, \ldots, y_{n-1}$ are known bit strings, while s is the unknown bit string that is part of the assumption about f.

From one angle, we can view the above as facts about s; the equations are properties that s satisfy.

From another angle, we can view the above as equations that we can solve to get s. Recall that s is a bit string of length n, so we can view each bit in s as an unknown. Then, the above equation system has $n - 1$ equations and n unknowns.

[We can solve the equations with an off-the-shelf classical solver]
We have $n-1$ equations and n unknowns, and we have that 0^n is a solution. We can solve the set of equations in polynomial time using Gaussian elimination, as usual.

If $y_1, \ldots, y_{n-1}$ are linearly independent, the equations have two solutions, namely 0^n and a vector s that is orthogonal to the others. Otherwise, the set of equations has more than two solutions.

If $y_1, \ldots, y_{n-1}$ are chosen independently of each other, the probability that they are linearly independent is more than $\frac{1}{4}$.

[Simon's algorithm is an approximation algorithm]
If we run the entire process of quantum followed by classical a single time, we have only a little more than $\frac{1}{4}$ chance of finding s. So, Simon's algorithm runs the entire process a total of $4m$ times, where m is a parameter that we must determine.

$$\begin{array}{ll} & \text{Probability of failing to find } s \text{ after } 4m \text{ iterations} \\ < & \left(1 - \frac{1}{4}\right)^{4m} \\ < & e^{-m} \end{array}$$

In the first step, we use that the iterations are independent. In the second step, we use one of Bernoulli's formulas.

Notice that $\forall \epsilon > 0 : \exists m : e^{-m} < \epsilon$. So, for example, if we want Simon's algorithm to succeed with probability at least 99 percent, we choose $m = 5$ and run the process $4 \times 5 = 20$ times.

$e^{-m} < 1/100$

$-m < \ln(1/100)$

$-m < -\ln(100)$

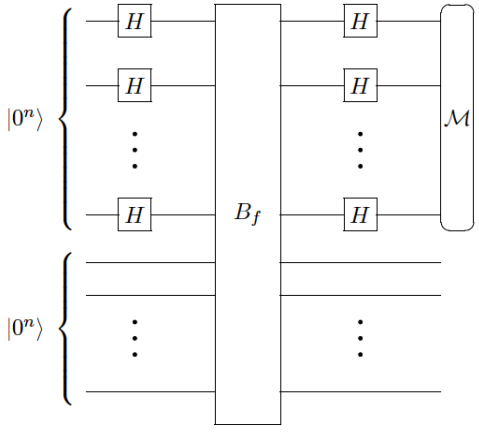
$m > \ln(100)$

$m > 4.605$

Transition to MP3: Now let us look at the quantum side of Simon's algorithm.

Main Point 3: We will cover full details of how Simon's algorithm works.

[The quantum circuit is a variation of what we have seen before]



The diagram above uses the notation B_f for what I will call U_f.

[The number of helper bits is equal to the number of input bits]

The main reason is that f produces a bit string of length n. We will represent f as an invertible operation U_f, as usual:

$$\begin{aligned} U_f &: Qubit^{\otimes 2n} \rightarrow Qubit^{\otimes 2n} \\ U_f|x\rangle|b\rangle &= |x\rangle|b \oplus f(x)\rangle \end{aligned}$$

Here, $x, b \in \{0,1\}^n$.

[The reasoning about correctness relies on some known lemmas]
Initially, the state of the $2n$ qubits is $|0^n\rangle|0^n\rangle$.

After the first n uses of H, followed by the use of U_f, followed by the n uses of H, the state of the $2n$ qubits is as follows.

$$\begin{aligned} & (H^{\otimes n} \otimes I^{\otimes n}) \circ U_f \circ (H^{\otimes n} \otimes I^{\otimes n}) \ |0^n\rangle|0^n\rangle \\ = \ & (H^{\otimes n} \otimes I^{\otimes n}) \circ U_f \ \frac{1}{\sqrt{2^n}} \ \Sigma_{x\in\{0,1\}^n}|x\rangle|0^n\rangle \\ = \ & (H^{\otimes n} \otimes I^{\otimes n}) \ \frac{1}{\sqrt{2^n}} \ \Sigma_{x\in\{0,1\}^n}|x\rangle|f(x)\rangle \\ = \ & \frac{1}{2^n} \ \Sigma_{x\in\{0,1\}^n}\Sigma_{y\in\{0,1\}^n} \ (-1)^{x\cdot y} \ |y\rangle|f(x)\rangle \\ = \ & \Sigma_{y\in\{0,1\}^n} \ |y\rangle \ \left(\frac{1}{2^n} \ \Sigma_{x\in\{0,1\}^n}(-1)^{x\cdot y} \ |f(x)\rangle \right) \end{aligned}$$

Given a particular $y \in \{0,1\}^n$, let us calculate the probability of measuring y. We divide the analysis into two cases.

First, consider $s = 0^n$. In this case, f is 1-to-1, so the sum over $x \in \{0,1\}^n$ is a sum of 2^n different, orthonormal vectors $|f(x)\rangle$. For each of those 2^n vectors, the term $(-1)^{x\cdot y}$ is either -1 or $+1$. So, the probability of measuring y is:

$$\begin{aligned} || \ \frac{1}{2^n} \ \Sigma_{x\in\{0,1\}^n} \ (-1)^{x\cdot y} \ |f(x)\rangle \ ||^2 &= \frac{1}{2^{2n}} \ || \ \Sigma_{x\in\{0,1\}^n} \ (-1)^{x\cdot y} \ |f(x)\rangle \ ||^2 \\ &= \frac{1}{2^{2n}} \ (\sqrt{2^n})^2 \\ &= \frac{2^n}{2^{2n}} \\ &= \frac{1}{2^n} \end{aligned}$$

In the second step, we use that the $|f(x)\rangle$ vectors are orthonormal so $\Sigma_{x\in\{0,1\}^n} \ (-1)^{x\cdot y} \ |f(x)\rangle$ is a vector of length 2^n in which every entry is -1 or 1; the norm of such a vector is $\sqrt{2^n}$. We conclude that each of the 2^n different y are equally likely outcomes of the measurement. Notice that y satisfies

$$y \cdot s = 0$$

because $s = 0^n$.

Second, consider $s \neq 0^n$. Let A denote the range of f. For each $z \in A$, we have 2 distinct $x_z, x'_z \in \{0,1\}^n$ such that

$$f(x_z) \;=\; f(x'_z) \;=\; z$$

From the above and the assumption about f, we also have:

$$\begin{aligned} x_z \oplus x'_z &= s \quad \text{which is equivalent to} \\ x'_z &= x_z \oplus s \end{aligned}$$

So, the probability of measuring y is:

$$\begin{aligned} & || \frac{1}{2^n} \Sigma_{x \in \{0,1\}^n} (-1)^{x \cdot y} |f(x)\rangle ||^2 \\ = & || \frac{1}{2^n} \Sigma_{z \in A} \left((-1)^{x_z \cdot y} + (-1)^{x'_z \cdot y} \right) |z\rangle ||^2 \\ = & || \frac{1}{2^n} \Sigma_{z \in A} \left((-1)^{x_z \cdot y} + (-1)^{(x_z \oplus s) \cdot y} \right) |z\rangle ||^2 \\ = & || \frac{1}{2^n} \Sigma_{z \in A} (-1)^{x_z \cdot y} \left(1 + (-1)^{s \cdot y}\right) |z\rangle ||^2 \\ = & \begin{cases} 2^{-(n-1)} & \text{if } s \cdot y = 0 \\ 0 & \text{if } s \cdot y = 1 \end{cases} \end{aligned}$$

In the third step, we used the property

$$(x_z \oplus s) \cdot y \;=\; (x_z \cdot y) \oplus (s \cdot y)$$

We conclude that each of 2^{n-1} different y are equally likely outcomes of the measurement. Additionally, the y that we get as result of the measurement satisfies the equation

$$y \cdot s \;=\; 0$$

In summary, the measurement produces a bit string y that satisfies $y \cdot s = 0$, and the distribution is uniform across all such bit strings.

Thus, we can satisfy the interface to the classical part by running the above quantum circuit $n-1$ times and collecting the equations.

For example, let us run the quantum circuit on the specific f for which we listed a table earlier and for which we have $s = 110$. Just before measurement, the state of the $2 \times 3 = 6$ qubits is:

$$\begin{aligned}
& \Sigma_{y\in\{0,1\}^3}\ |y\rangle\ \left(\frac{1}{2^3}\ \Sigma_{x\in\{0,1\}^3}(-1)^{x\cdot y}\ |f(x)\rangle\right) \\
= \ & \Sigma_{y\in\{0,1\}^3}\ |y\rangle\ (\frac{1}{8}\ (((-1)^{010\cdot y}+(-1)^{100\cdot y})\ |000\rangle + \\
& \qquad ((-1)^{001\cdot y}+(-1)^{111\cdot y})\ |010\rangle + \\
& \qquad ((-1)^{000\cdot y}+(-1)^{110\cdot y})\ |101\rangle + \\
& \qquad ((-1)^{011\cdot y}+(-1)^{101\cdot y})\ |110\rangle)) \\
= \ & \Sigma_{y\in\{0,1\}^3}\ |y\rangle\ (\frac{1}{8}\ (((-1)^{010\cdot y}+(-1)^{(010\oplus 110)\cdot y})\ |000\rangle + \\
& \qquad ((-1)^{001\cdot y}+(-1)^{(001\oplus 110)\cdot y})\ |010\rangle + \\
& \qquad ((-1)^{000\cdot y}+(-1)^{(000\oplus 110)\cdot y})\ |101\rangle + \\
& \qquad ((-1)^{011\cdot y}+(-1)^{(011\oplus 110)\cdot y})\ |110\rangle)) \\
= \ & \Sigma_{y\in\{0,1\}^3}\ |y\rangle\ (\frac{1}{8}\ ((-1)^{010\cdot y}(1+(-1)^{110\cdot y})\ |000\rangle + \\
& \qquad (-1)^{001\cdot y}(1+(-1)^{110\cdot y})\ |010\rangle + \\
& \qquad (-1)^{000\cdot y}(1+(-1)^{110\cdot y})\ |101\rangle + \\
& \qquad (-1)^{011\cdot y}(1+(-1)^{110\cdot y})\ |110\rangle))
\end{aligned}$$

So, when we measure and get y, we know that $110 \cdot y = 0$ and that all such cases of y are equally likely. We need to run at least $3 - 1 = 2$ times to produce two such y that will form the interface to the classical part.

We can take the example further and suppose that the two runs produced 111 and 001. Notice that we have $111 \cdot 110 = 0$ and $001 \cdot 110 = 0$.

We see that 111 and 001 are linearly independent. Now we can give the two equations

$$\begin{aligned}
111 \cdot s &= 0 \\
001 \cdot s &= 0
\end{aligned}$$

to a constraint solver. The constraint solver will tell us that the equations have a single solution that is different from 000, namely 110. Thus, we have succeeded in taking f as input and producing 110 as output.

Notice that Simon's algorithm uses $(n-1) \times 4m$ iterations, which is much fewer than the $\sqrt{2^n}$ that are needed with classical computing. Thus, Simon's algorithm gave an exponential speed-up.

Transition to Close: So there you have it.

Review: Simon's algorithm combines classical and quantum computing.

Strong finish: Simon's algorithm is a powerful indication that a quantum computer can be faster than a classical computer. We will see more examples of that in this course.

Call to action: Look for other natural problems that may be solvable on quantum computers.

Quantum Programming Algorithms: Grover

Jens Palsberg

Jan 27, 2022

Outline

Hook: Our next quantum algorithm is Grover's algorithm. It solves a natural problem and does so faster on a quantum computer than we can ever do on a classical computer.

Purpose: Persuade you that you can understand this algorithm.

Preview:

1. We will make sure that everybody understands the problem.
2. We will show how Grover's algorithm iterates the application of a unitary matrix.
3. We will cover full details of why Grover's algorithm works.

Transition to Body: Now let us talk about the problem that this algorithm solves.

Main Point 1: We will make sure that everybody understands the problem.
[Most of you have done the homework on solving the problem on a classical computer]
[We can represent the black-box function in multiple ways]
[We can program the solution in any classical language]

Transition to MP2: Now let us look at how to solve the problem.

Main Point 2: We will show how Grover's algorithm iterates the application of a unitary matrix.
[First we need two particular unitary operations]
[Second we compose them in a particular way]
[Finally we iterate the use of the resulting matrix]

Transition to MP3: Now let us get into correctness.

Main Point 3: We will cover full details of why Grover's algorithm works.
[We can define a subspace with two dimensions]
[Each iteration performs a rotation on the subspace]
[After some iterations, a measurement gives what we are looking for, with high probability]

Transition to Close: So there you have it.

Review: Grover's algorithm iterates to set up for a measurement that will give a good outcome with high probability.

Strong finish: Grover's algorithm is a powerful indication that a quantum computer can be faster than a classical computer. We will see more examples of that in this course.

Call to action: Look for other natural problems that may be solvable on quantum computers.

Detailed presentation

Hook: Our next quantum algorithm is Grover's algorithm. It solves a natural problem and does so faster on a quantum computer than we can ever do on a classical computer.

Purpose: Persuade you that you can understand this algorithm.

Preview:

1. We will make sure that everybody understands the problem.
2. We will show how Grover's algorithm iterates the application of a unitary matrix.
3. We will cover full details of why Grover's algorithm works.

Transition to Body: Now let us talk about the problem that this algorithm solves.

Main Point 1: We will make sure that everybody understands the problem.

[Most of you have done the homework on solving the problem on a classical computer]
Here is the homework problem.

> Grover's problem:
> Input: a function $f : \{0,1\}^n \to \{0,1\}$.
> Output: 1 if there exists $x \in \{0,1\}^n$ such that $f(x) = 1$, and 0 otherwise.
> Notation: $\{0,1\}^n$ is the set of bit strings of length n.
>
> The assignment:
> On a classical computer, in a classical language of your choice (such as C, Java, Python, etc), program a solution to Grover's problem. Treat the input function f as black box that you can call but cannot inspect in any way at all. Each solution will be code that includes one or more calls of f.

[We can represent the black-box function in multiple ways]
How do we keep the black-box function at an arms length such that we cannot get tempted to inspect it? In C we can use a function pointer. In Java we can use a lambda expression. In Python we can use an anonymous function. Other languages have similar constructs that will enable us to call the function, while giving us no way to inspect it.

[We can program the solution in any classical language]
You wrote the solutions to the homework in several languages. We need to make many calls to f: in the worst case, we need to try f on every bit string of length n. This means 2^n calls to f.

Transition to MP2: Now let us look at how to solve the problem.

Main Point 2: We will show how Grover's algorithm iterates the application of a unitary matrix.

[First we need two particular unitary operations]
Each of those operations map n qubits (plus helper qubits) to n qubits (plus helper qubits). If we

ignore the helper qubits, those operations behave as follows. We use x to range over $\{0,1\}^n$.

$$\begin{aligned} Z_f\ |x\rangle &= (-1)^{f(x)}\ |x\rangle \\ Z_0\ |x\rangle &= \begin{cases} -|x\rangle & \text{if } x = 0^n \\ |x\rangle & \text{if } x \neq 0^n \end{cases} \end{aligned}$$

[Second we compose them in a particular way]
Define

$$G = -H^{\otimes n}\ Z_0\ H^{\otimes n}\ Z_f$$

[Finally we iterate the use of the resulting matrix]
Let X be a collection of n qubits that initially is $|0^n\rangle$. Grover's algorithm is:

1. Apply $H^{\otimes n}$ to X.
2. Repeat { apply G to X } $O(\sqrt{2^n})$ times.
3. Measure X and output the result.

Let us try Grover's algorithm on an example. Suppose $n = 2$. Let $f : \{0,1\}^2 \to \{0,1\}$ be defined by: $f(00) = 0$ and $f(01) = 0$ and $f(10) = 0$ and $f(11) = 1$.

Now we initialize two qubits to $|00\rangle$. Now let us execute the algorithm. How many times should we iterate in Step 2? Turns out that the correct number of iterations is 1, as we will see later.

Here is a property of $H^{\otimes 2}$ that will be useful later:

$$\begin{aligned} H^{\otimes 2}\ (|00\rangle + |01\rangle + |10\rangle - |11\rangle) &= \frac{1}{2}\ ((|00\rangle + |01\rangle + |10\rangle + |11\rangle) + (|00\rangle - |01\rangle + |10\rangle - |11\rangle) + \\ &\qquad (|00\rangle + |01\rangle - |10\rangle - |11\rangle) - (|00\rangle - |01\rangle - |10\rangle + |11\rangle)) \\ &= |00\rangle + |01\rangle + |10\rangle - |11\rangle \end{aligned}$$

Now we calculate Grover's algorithm for the example:

$$\begin{aligned} G\ H^{\otimes 2}\ |00\rangle &= G\ \frac{1}{2}\ (|00\rangle + |01\rangle + |10\rangle + |11\rangle) \\ &= -H^{\otimes 2}\ Z_0\ H^{\otimes 2}\ Z_f\ \frac{1}{2}\ (|00\rangle + |01\rangle + |10\rangle + |11\rangle) \\ &= -H^{\otimes 2}\ Z_0\ H^{\otimes 2}\ \frac{1}{2}\ (|00\rangle + |01\rangle + |10\rangle - |11\rangle) \\ &= -H^{\otimes 2}\ Z_0\ \frac{1}{2}\ (|00\rangle + |01\rangle + |10\rangle - |11\rangle) \\ &= -H^{\otimes 2}\ \frac{1}{2}\ (-|00\rangle + |01\rangle + |10\rangle - |11\rangle) \\ &= -H^{\otimes 2}\ \frac{1}{2}\ (-2|00\rangle + (|00\rangle + |01\rangle + |10\rangle - |11\rangle)) \\ &= -\frac{1}{2}\ (-2\ (\frac{1}{2}\ (|00\rangle + |01\rangle + |10\rangle + |11\rangle)) + (|00\rangle + |01\rangle + |10\rangle - |11\rangle)) \\ &= -\frac{1}{2}\ (-2\ |11\rangle) \\ &= |11\rangle \end{aligned}$$

In the fourth and seventh step, we used the above property of $H^{\otimes 2}$. In conclusion, when we measure we get 11 with probability 1. Thus, we found 11 after a single use of f.

Transition to MP3: Now let us get into correctness.

Main Point 3: We will cover full details of why Grover's algorithm works.

[We can define a subspace with two dimensions]
Define two sets of bit strings as follows:

$$\begin{aligned} A &= \{\ x \in \{0,1\}^n \mid f(x) = 1\ \} \\ B &= \{\ x \in \{0,1\}^n \mid f(x) = 0\ \} \end{aligned}$$

We can think of A as the *Awesome* bit strings and B as the *Bad* bit strings. For convenience, define

$$\begin{aligned} N &= 2^n \\ a &= |A| \\ b &= |B| \end{aligned}$$

Notice that $N = a + b$.

Let us focus on the main case of $(a \neq 0 \wedge b \neq 0)$. Define

$$\begin{aligned} |A\rangle &= \frac{1}{\sqrt{a}} \Sigma_{x \in A}\ |x\rangle \\ |B\rangle &= \frac{1}{\sqrt{b}} \Sigma_{x \in B}\ |x\rangle \end{aligned}$$

Lemma 1. *$|A\rangle$ and $|B\rangle$ are orthogonal unit vectors.*

[Each iteration performs a rotation on the subspace]

Lemma 2.

$$\begin{aligned} G|A\rangle &= \left(1 - \frac{2a}{N}\right)|A\rangle\ -\ \frac{2\sqrt{ab}}{N}\ |B\rangle \\ G|B\rangle &= \frac{2\sqrt{ab}}{N}\ |A\rangle\ -\ \left(1 - \frac{2b}{N}\right)|B\rangle \end{aligned}$$

Proof. The proof has three main parts.

In the first part of the proof, define $|h\rangle$ to be the state of X after Step 1 of the algorithm.

$$|h\rangle\ =\ H^{\otimes n}\ |0^n\rangle\ =\ \frac{1}{\sqrt{N}}\ \Sigma_{x \in \{0,1\}^n}\ |x\rangle\ =\ \sqrt{\frac{a}{N}}\ |A\rangle\ +\ \sqrt{\frac{b}{N}}\ |B\rangle$$

Notice that in the third step, the renormalization is correct:

$$\left(\sqrt{\frac{a}{N}}\right)^2 + \left(\sqrt{\frac{b}{N}}\right)^2\ =\ \frac{a}{N} + \frac{b}{N}\ =\ \frac{a+b}{N}\ =\ 1$$

In the second part of the proof, notice that we can write

$$Z_0 \;=\; I - 2|0^n\rangle\langle 0^n|$$

This is because Z_0 is the matrix that looks almost entirely like the identity matrix but has a -1 (instead $+1$) in the top-left corner. The above way of writing Z_0 enables us to calculate:

$$H^{\otimes n}\; Z_0\; H^{\otimes n} \;=\; H^{\otimes n}\; (I - 2|0^n\rangle\langle 0^n|)\; H^{\otimes n} \;=\; I - 2|h\rangle\langle h|$$

Here we used that $H^\dagger = H$ and that if $|\psi\rangle = U|\phi\rangle$, then $\langle\psi| = \langle\phi|\; U^\dagger$.

In the third part of the proof, we consider the two equations stated in the lemma. We begin with the equation for $G|A\rangle$.

$$\begin{aligned}
G|A\rangle &= -H^{\otimes n}\; Z_0\; H^{\otimes n}\; Z_f\; |A\rangle \\
&= (I - 2|h\rangle\langle h|)\; (-Z_f)\; |A\rangle \\
&= (I - 2|h\rangle\langle h|)\; |A\rangle \\
&= |A\rangle - 2\langle h|A\rangle\; |h\rangle \\
&= |A\rangle - 2\sqrt{\frac{a}{N}}\left(\sqrt{\frac{a}{N}}\; |A\rangle \;+\; \sqrt{\frac{b}{N}}\; |B\rangle\right) \\
&= \left(1 - \frac{2a}{N}\right)|A\rangle \;-\; \frac{2\sqrt{ab}}{N}\; |B\rangle
\end{aligned}$$

In the third step, we use that we apply Z_f to $|A\rangle$, so for each vector in the sum for A, we have $f(x) = 1$, hence $(-1)^{f(x)} = -1$.

We continue with the equation for $G|B\rangle$.

$$\begin{aligned}
G|B\rangle &= -H^{\otimes n}\; Z_0\; H^{\otimes n}\; Z_f\; |B\rangle \\
&= -(I - 2|h\rangle\langle h|)\; Z_f\; |B\rangle \\
&= -(I - 2|h\rangle\langle h|)\; |B\rangle \\
&= -|B\rangle + 2\langle h|B\rangle\; |h\rangle \\
&= -|B\rangle + 2\sqrt{\frac{b}{N}}\left(\sqrt{\frac{a}{N}}\; |A\rangle \;+\; \sqrt{\frac{b}{N}}\; |B\rangle\right) \\
&= \frac{2\sqrt{ab}}{N}\; |A\rangle \;-\; \left(1 - \frac{2b}{N}\right)|B\rangle
\end{aligned}$$

In the third step, we use that we apply Z_f to $|B\rangle$, so for each vector in the sum for B, we have $f(x) = 0$, hence $(-1)^{f(x)} = 1$. □

Lemma 2 shows that G maps the subspace spanned by A and B to itself. Indeed, if we put $|B\rangle$ in the first row and first column, and we put $|A\rangle$ in the second row and second column, we see that we can represent G by the matrix:

$$M \;=\; \begin{pmatrix} -(1 - \frac{2b}{N}) & -\frac{2\sqrt{ab}}{N} \\ \frac{2\sqrt{ab}}{N} & 1 - \frac{2a}{N} \end{pmatrix}$$

Define $\theta \in (0, \frac{\pi}{2})$ to be the angle that satisfies:

$$\sin\theta \;=\; \sqrt{\tfrac{a}{N}} \qquad \text{and} \qquad \cos\theta \;=\; \sqrt{\tfrac{b}{N}}$$

Lemma 3. *G causes a rotation by an angle 2θ in the space spanned by A and B.*

Proof. The following matrix causes a rotation of θ:

$$R_\theta \;=\; \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

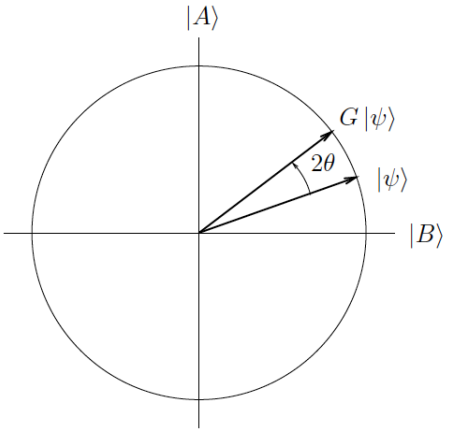
Notice that

$$R_\theta^2 \;=\; \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}^2 \;=\; \begin{pmatrix} \sqrt{\frac{b}{N}} & -\sqrt{\frac{a}{N}} \\ \sqrt{\frac{a}{N}} & \sqrt{\frac{b}{N}} \end{pmatrix}^2 \;=\; \begin{pmatrix} \frac{b-a}{N} & -\frac{2\sqrt{ab}}{N} \\ \frac{2\sqrt{ab}}{N} & \frac{b-a}{N} \end{pmatrix} \;=\; \begin{pmatrix} -(1-\frac{2b}{N}) & -\frac{2\sqrt{ab}}{N} \\ \frac{2\sqrt{ab}}{N} & 1-\frac{2a}{N} \end{pmatrix}$$
$$=\; M$$

In the second-to-last step we used $b-a = -(a-b) = -((a-b+2b)-2b) = -((a+b)-2b) = -(N-2b)$ and $b-a = (b-a+2a)-2a = (a+b)-2a = N-2a$.

We conclude that M does two applications of R_θ so M causes a rotation that is twice the angle of the rotation caused by R_θ. □

We can illustrate the effect of G as follows:



[After some iterations, a measurement gives what we are looking for, with high probability]

After Step 1 of the algorithm, the qubits are in the state

$$|h\rangle \;=\; \sqrt{\frac{b}{N}}\,|B\rangle \;+\; \sqrt{\frac{a}{N}}\,|A\rangle \;=\; \cos\theta\,|B\rangle \;+\; \sin\theta\,|A\rangle$$

We want to rotate until we are as close to $|A\rangle$ as we can get. If we do k iterations in Step 2, then the state of the qubits is:

$$\cos((2k+1)\theta)\,|B\rangle \;+\; \sin((2k+1)\theta)\,|A\rangle$$

We are going for

$$\begin{aligned} & \sin((2k+1)\theta) \;\approx\; 1 \\ \iff\; & (2k+1)\theta \;\approx\; \frac{\pi}{2} \\ \iff\; & 2k+1 \;\approx\; \frac{\pi}{2\theta} \\ \iff\; & k \;\approx\; \frac{\pi}{4\theta} - \frac{1}{2} \end{aligned}$$

Let us focus on the case of $a = 1$, that is, we are looking for a unique bit string. Then

$$\begin{aligned} \theta &= \sin^{-1}\sqrt{\frac{1}{N}} \approx \sqrt{\frac{1}{N}} = \frac{1}{\sqrt{N}} \\ k &\approx \frac{\pi}{4\theta} - \frac{1}{2} \approx \frac{\pi}{4\frac{1}{\sqrt{N}}} - \frac{1}{2} = \frac{\pi}{4}\sqrt{N} - \frac{1}{2} \end{aligned}$$

In the calculation of θ, in the second step (the one with $\approx$), we use that when y is close to 0, we have $\sin(y) \approx y$. Eventually, we will have to round k to the nearest integer.

In the example above, we have $N = 2^2 = 4$ so

$$k \approx \frac{\pi}{4}\sqrt{N} - \frac{1}{2} = \frac{\pi}{4}\sqrt{4} - \frac{1}{2} = \frac{\pi}{2} - \frac{1}{2} \approx 1$$

This justifies why we iterated the use of G a single time in the example.

We can check that we have good a good estimate of k by calculating the probability that we will measure and get a bit string x such that $f(x) = 1$:

$$\begin{aligned} |\ \sin((2k+1)\theta)\ |^2 &\approx |\ \sin(\ (2(\frac{\pi}{4}\sqrt{N} - \frac{1}{2}) + 1)\ \left(\frac{1}{\sqrt{N}}\right)\ )\ |^2 \\ &= |\ \sin(\ ((\frac{\pi}{2}\sqrt{N} - 1) + 1)\ \left(\frac{1}{\sqrt{N}}\right)\ )\ |^2 \\ &= |\ \sin(\ \frac{\pi}{2}\sqrt{N}\ \left(\frac{1}{\sqrt{N}}\right)\ )\ |^2 \\ &= |\ \sin(\frac{\pi}{2})\ |^2 \\ &= |1|^2 \\ &= 1 \end{aligned}$$

We can repeat the algorithm and evaluate f on the output each time; this will find the unique x such that $f(x) = 1$ with high probability.

If $a > 1$, we can use the same algorithm as for $a = 1$, and if it is unsuccessful, then we can try larger and larger values of k. This turns out to be an effective strategy.

If $a = 0$, then we can use the same algorithm as for $a > 1$, and if after a while, nothing has been found, we can be confident in concluding that $a = 0$.

Transition to Close: So there you have it.

Review: Grover's algorithm iterates to set up for a measurement that will give a good outcome with high probability.

Strong finish: Grover's algorithm is a powerful indication that a quantum computer can be faster than a classical computer. We will see more examples of that in this course.

Call to action: Look for other natural problems that may be solvable on quantum computers.