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.

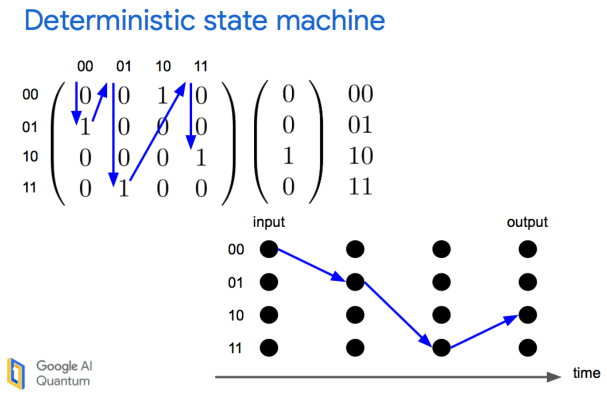
The state space rule: the state is a bit vector. Normally we think of the state of a classical computation as a bit vector of size n; now instead we will use a bit vector of size 2^n. The idea is that this much longer bit vector of size 2^n indicates which bit vector of size n is represented. This is 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.



$$\begin{array}{c|cccc} & 00 & 01 & 10 & 11 \\ \hline 00 & 0 & 0 & 1 & 0 \\ 01 & 1 & 0 & 0 & 0 \\ 10 & 0 & 0 & 0 & 1 \\ 11 & 0 & 1 & 0 & 0 \end{array} \quad \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \begin{array}{c} 00 \\ 01 \\ 10 \\ 11 \end{array}$$

[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.

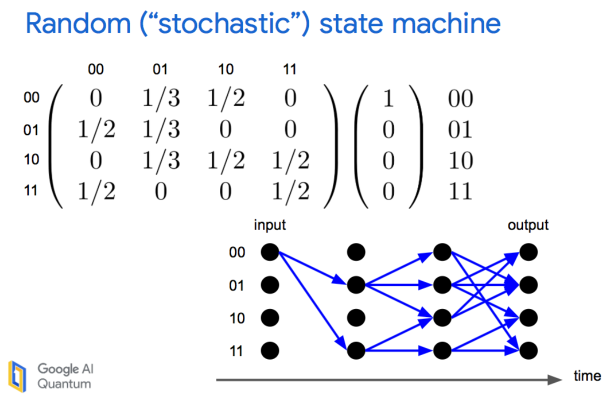
A pure state $|\Psi\rangle$ is a linear combination of base states, $|s\rangle$. That is, $|\Psi\rangle = \Sigma_s\, p_s|s\rangle$, where p_s is the probability of being in base state $|s\rangle$, and $\Sigma_s\, p_s = 1$, and $0 \le p_s \le 1$.

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

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

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.



Random ("stochastic") state machine

$$\begin{array}{c} \\ 00 \\ 01 \\ 10 \\ 11 \end{array}\begin{array}{c} \begin{array}{cccc} 00 & 01 & 10 & 11 \end{array} \\ \begin{pmatrix} 0 & 1/3 & 1/2 & 0 \\ 1/2 & 1/3 & 0 & 0 \\ 0 & 1/3 & 1/2 & 1/2 \\ 1/2 & 0 & 0 & 1/2 \end{pmatrix} \end{array} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \begin{array}{c} 00 \\ 01 \\ 10 \\ 11 \end{array}$$

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.

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

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:

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

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
$\Sigma_{i=1}^n p_i = 1$	$\Sigma_{i=1}^n \|\alpha_i\|^2 = 1$
stochastic matrices	unitary matrices
preserve $\Sigma_{i=1}^n p_i = 1$	preserve $\Sigma_{i=1}^n \|\alpha_i\|^2 = 1$

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

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

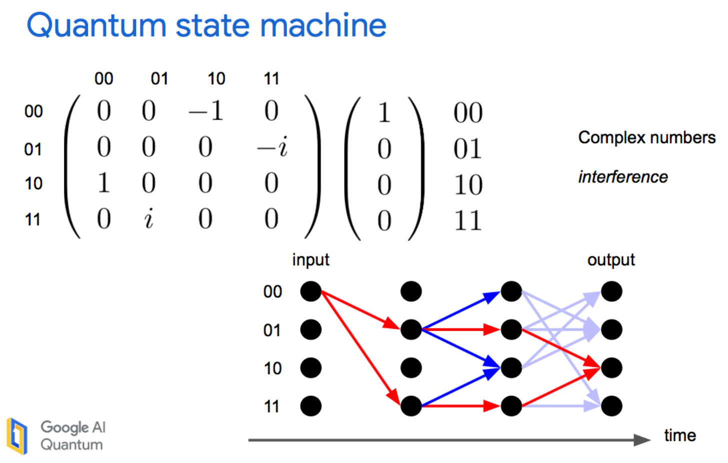
you will not see a superposition. Rather, you will see 0 or 1, just like before. Specifically, you will see 0 with probability $|\alpha|^2$, and you will see 1 with probability $|\beta|^2$. This is why we have the condition $|\alpha_i|^2 + |\beta_i|^2 = 1$; the probabilities have to sum to 1.

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.



Quantum state machine

$$\begin{array}{c|cccc} & 00 & 01 & 10 & 11 \\ \hline 00 & 0 & 0 & -1 & 0 \\ 01 & 0 & 0 & 0 & -i \\ 10 & 1 & 0 & 0 & 0 \\ 11 & 0 & i & 0 & 0 \end{array} \qquad \begin{array}{cc} 1 & 00 \\ 0 & 01 \\ 0 & 10 \\ 0 & 11 \end{array}$$

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:

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

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.

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

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

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

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:

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

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 $|1\rangle$.
    ...
    $H(x_i)$
    ...
    $CCNOT(x_i, x_j, x_k)$
    ...
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.