Chapter 2: Quantum Computation: General Features and Some Simple Examples

Warren Alphonso

March 29, 2019

1 The General Computational Process

Suppose we have a quantum computer that we want to act on a number x to produce another number $f(x)$ for some specified function f. If we specify x as an n-bit integer and $f(x)$ as an m-bit integer, then we need at least $n+m$ Qbits; n for the *input register* and m for the *output register*. This is critical because if f maps multiple inputs to the same output, it would not be reversible unless we kept track of the input.
Remember that any classically meaningful reversible transformation on Cbits will have a linear extension to a unitary transformation on Qbits that acts as the classical transformation when restricted to states of the classical basis. This fact is crucial for defining unitary transformations on Qbits.
We can write the unitary transformation for f as

$$U_f(|x\rangle_n |y\rangle_m) = |x\rangle_n |y \oplus f(x)\rangle_m$$

where $\oplus$ again means the modulo-2 bitwise addition or XOR gate. (For example, $1101 \oplus 0111 = 1010$.
Starting with an initial value of $y = 0$ in the output register, we have

$$U_f(|x\rangle_n |0\rangle_m = |x\rangle_n |f(x)\rangle_m$$

so we do indeed end up with $f(x)$ in the output register.
This transformation is invertible, since U_f is its own inverse

$$U_f U_f(|x\rangle |y\rangle) = |x\rangle |y \oplus f(x) \oplus f(x)\rangle = |x\rangle |y\rangle$$

since $w \oplus w = 0$ for any w.
Now we can apply an important trick of quantum computation: applying to each Qbit in the two-Qbit state $|0\rangle|0\rangle$ the 1-Qbit Hadamard transformation:

$$(H \otimes H)(|0\rangle |0\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{2}(|0\rangle_2 + |1\rangle_2 + |2\rangle_2 + |3\rangle_2)$$

This generalizes to:

$$H^{\otimes n} |0\rangle_n = \frac{1}{2^{n/2}} \sum_{0\leq x<2^n} |x\rangle_n \tag{1}$$

The above tells us that if the initial state of the input register is $|0\rangle_n$ and we apply an n-fold Hadamard transformation on that register, its state becomes an **equally** weighted superposition of all possible n-Qbit inputs.
This allows us to take advantage of the power of quantum computing. For example, if we apply a Hadamard transformation to every Qbit in a 100 Qbit input register initially in the state $|0\rangle_{100}$, and then apply U_f, the final state can contain the results of 2^{100} evaluations of the function f: a billion billion trillion evaluations! This miracle is called *quantum parallelism*.
Before drawing extravagant conclusions from this, recall that when we have a collection of Qbits in a definite but unknown state, there is in general no way to find out what the state is. We are forced to collapse it to get any information. So although we can learn something from the output, it is nothing more than what we would have learned if we ran the computation on a classical computer with a random input. Still, a hint of a miracle remains: we notice that the random selection of x when we collapse *only happens after* the computation has been carried out. This is what is called "quantum weirdness."
If there were a way to make copies of the output state prior to making the measurement, without running the whole computation agian, then we could learn the values of f (with high probability) for several random values of x. However, such copying is prohibited by the **no-cloning theorem**, which states there is no unitary transformation that can take a state $|\psi\rangle_n |0\rangle_n$ into the state $|\psi\rangle |\psi\rangle$ for arbitrary $|\psi\rangle_n$.
The theorem is a consequence of linearity. If

$$U(|\psi\rangle |0\rangle) = |\psi\rangle |\psi\rangle$$

then, we can apply U after distributing the following:

$$U(a|\psi\rangle + b|\phi\rangle)|0\rangle = aU(|\psi\rangle |0\rangle) + bU(|\phi\rangle |0\rangle) = a|\psi\rangle |\psi\rangle + b|\phi\rangle |\phi\rangle$$

However, because U is linear, we can also apply it in a different order:

$$U(a|\psi\rangle + b|\phi\rangle)|0\rangle = (a|\psi\rangle + b|\phi\rangle)(a|\psi\rangle + b|\phi\rangle)$$

$$= a^2 |\psi\rangle |\psi\rangle + b^2 |\phi\rangle |\phi\rangle + ab|\psi\rangle |\phi\rangle + ab|\phi\rangle |\psi\rangle$$

which differs from our first equation unless a or b is zero. Thus, we cannot clone inputs.
The uncertainty principle tells us that there will be a tradeoff in the information we attain and can attain. So it is wrong to say that the quantum computer has evaluated the function $f(x)$ for all x in the entire range. There are still some tricks we can employ to permit quantum computers to compute things classical computers can never accomplish.

2 Deutsch's Problem

Deutsch's problem is the simplest example of a tradeoff that sacrifices particular information for relational information. Here's how it works: Let both input and output registers contain only a single Qbit. We explore functions f that take a single bit into a single bit. All of these functions are outlined in the table.

	$\mathbf{x = 0}$	$\mathbf{x = 1}$
f_0	0	0
f_1	0	1
f_2	1	0
f_3	1	1

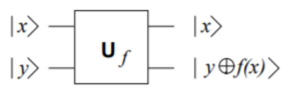
The input to these functions are at the top of the table (either $x = 0$ or $x = 1$). The rest of the table denotes the one bit output attained by the output register of:

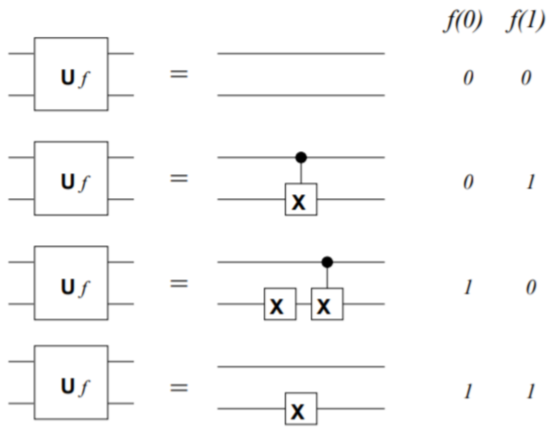
$$U_f(|x\rangle |y\rangle) = |x\rangle |y \oplus f(x)\rangle$$

Verify that the f in the table above are:

$$U_{f_0} = 1, U_{f_1} = C_{io}, U_{f_2} = C_{io}X_o, U_{f_3} = X_o$$

To better visualize these operators, remember that the output register is initially in $|0\rangle$ state and use the following figure:





Suppose we are given a black box that executes U_f for one of these four functions, but we don't know which it is. We can clearly find out by letting the black box

act twice - first on $|0\rangle|0\rangle$ and then on $|1\rangle|0\rangle$. But what if we are only allowed to let the box act once?
In a classical computer, we can either learn the value of $f(0)$ (by letting U_f act on either $|0\rangle|0\rangle$ or $|0\rangle|1\rangle$). If we get $f(0) = 0$, we know it is either f_0 or f_1. If we instead learn the value of $f(1)$, and we find $f(1) = 0$, then we can restrict U_f to be either f_0 or f_2.
Another approach we could use is to learn whether f is constant $\Big(f(0) = f(1)$, satisfied by f_0 and $f_3\Big)$. To do this in a classical computer, we must evaluate both $f(0)$ and $f(1)$ and compare them. We have to run U_f twice.
Remarkably, with a quantum computer, we can do this in a *single run*. When we do this, however, we learn nothing about the individual values of $f(0)$ and $f(1)$, but we are able to answer whether they are the same!
To do this we use our standard trick, preparing the input register in the superposition $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. The final state of the input and output registers would then be

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

which derives from Figure 1.
However, all this does is tell us $f(x)$ for a certain x and is thus not an improvement over the classical computer.
It was later noticed that there are unitary transformations one can apply to the state before carrying out the measurement, that, depending on the outcome, enable you half the time state with assurance whether or not $f(0) = f(1)$. Some time after that, it was discovered we can *always* answer the question, provided we apply the appropriate unitary transformations before and after running the computation. Here is how we execute this trick:
In our above quantum attempt, we made the input to U_f to be the state

$$(H \otimes 1)(|0\rangle|0\rangle)$$

Instead of this, we again start with both input and output registers in the state $|0\rangle$ but then we apply the NOT operation X to both registers, followed by an application of the Hadamard transformation to both. Since $X|0\rangle = |1\rangle$ and $H|1\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$, the input to U_f is now described by the state

$$\begin{aligned}(H \otimes H)(X \otimes X)(|0\rangle|0\rangle) &= (H \otimes H)(|1\rangle|1\rangle) \\ &= (\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle)(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle) \\ &= \frac{1}{2}(|0\rangle|0\rangle - |1\rangle|0\rangle - |0\rangle|1\rangle + |1\rangle|1\rangle\end{aligned} \tag{2}$$

If we use this state as input to U_f, then by linearity the resulting state is

$$\frac{1}{2}(|0\rangle|f(0)\rangle - |1\rangle|f(1)\rangle - |0\rangle|\bar{f}(0)\rangle + |1\rangle|\bar{f}(1)\rangle)$$

where $\bar{x} = 1 \oplus x$

Thus, if $f(0) = f(1)$, the resulting state is

$$\frac{1}{2}\Big(|0\rangle - |1\rangle\Big)\Big(|f(0)\rangle - |\bar{f}(0)\rangle\Big)$$

assuming $f(0) = f(1)$
However if $f(0) \neq f(1)$, then $f(1) = \bar{f}(0)$, so the resulting state is

$$\frac{1}{2}\Big(|0\rangle + |1\rangle\Big)\Big(|f(0)\rangle - |\bar{f}(0)\rangle\Big)$$

assuming $f(0) \neq f(1)$.
Finally, if we apply a Hadamard transformation to only the input register, these become

$$|1\rangle\,\frac{1}{\sqrt{2}}\Big(|f(0)\rangle - |\bar{f}(0)\rangle\Big), f(0) = f(1)$$

$$|0\rangle\,\frac{1}{\sqrt{2}}\Big(|f(0)\rangle - |\bar{f}(0)\rangle\Big), f(0) \neq f(1)$$

Putting together all the operations so far, we have

$$(H \otimes 1)U_f(H \otimes H)(X \otimes X)(|0\rangle\,|0\rangle) = \begin{cases} |1\rangle\,\frac{1}{\sqrt{2}}\Big(|f(0)\rangle - |\bar{f}(0)\rangle\Big), f(0) = f(1) \\ |0\rangle\,\frac{1}{\sqrt{2}}\Big(|f(0)\rangle - |\bar{f}(0)\rangle\Big), f(0) \neq f(1) \end{cases}$$

Thus, the state of the input register ends up as $|1\rangle$ or $|0\rangle$ depending on whether or not $f(0) = f(1)$, so by measuring the *input* register, we can determine if $f(0) = f(1)$. Notice that because the output registers are the same, one learns absolutely nothing by measuring it. This information still only narrows our search for what U_f is to two functions, but it does this in 1 calculation whereas a classical computer requires 2 in order to compare $f(0)$ and $f(1)$.
Another way to think of this is as follows: Suppose $f(x)$ = millionth bit of $\sqrt{2+x}$. Then, it is quite startling that we can determine whether the millionth bits of $\sqrt{2}$ and $\sqrt{3}$ are the same with the exact same effort as it takes to calculate the millionth bit of either $\sqrt{2}$ or $\sqrt{3}$!

3 Why Additional Subroutine Qbits Needn't Mess Things Up

In our secondary way of thinking of Deutsch's problem, we may need to use many more Qbits to represent the other bits of $\sqrt{2}$ and $\sqrt{3}$. Then the action of the computer must only induce a transformation on the input and output registers n and m, while leaving the additional r Qbits alone. Otherwise, the input and output registers will become entangled with the additional r Qbits.
Let the additional r Qbits start off in some initial state $|\psi\rangle_r$ so that we can define the initial state of the input register, output register, and additional Qbits as

$$|\Psi\rangle_{n+m+r} = |x\rangle_n\,|y\rangle_m\,|\psi\rangle_r$$

Though the r additional Qbits might become entangled with the input and output registers during the calculation (in fact, they *must* in order to serve any useful purpose) we require that the final state is of the form

$$W\left|\Psi\right\rangle_{n+m+r} = \left|x\right\rangle_n \left|y \oplus f(x)\right\rangle_m \left|\phi\right\rangle_r$$

where the additional r Qbits have a state $\left|\phi\right\rangle_r$ which is independent of the initial state of the input and output registers.
Because the initial and final states $\left|\psi\right\rangle_r$ and $\left|\phi\right\rangle_r$ of the additional Qbits are independent of the initial state of the input and output, we know the final states of the input and output registers will be related by a transformation U_f that is linear.
Therefore, we can ignore the additional r Qbits needed to compute the function f, provided both the initial and final states of the additional Qbits are *entirely independent* of the initial state of the input and output registers. This independence is achieved by taking advantage of the fact that unitary transformations are reversible. Actually, it is this very need to disentangle additional registers from input and output that is the **fundamental reason why quantum computation must be reversible**.
Concretely, we begin the computation by applying a unitary transformation V that acts only on the input register and the r additional Qbits, leaving the output register alone. This gives us $f(x)$ after acting on x. Now, we change the output register to $y \oplus f(x)$ without altering any Qbits outside the output register, by using m cNOT gates. The m control bits are among the $n + r$ that represent the result of the computation of $f(x)$. Since the state of the $n + r$ Qbits is not altered by cNOT gates, we can apply the inverse transformation $V^\dagger$ to restore them to their original state.

4 Some More Substantial Speed-ups with a Quantum Computer

4.1 The Bernstein-Vazirani Problem

While this is not as applicable as Shor's algorithm, the significance of this problem is that is can be solved unambiguously faster on a quantum computer.
Let a be an unknown non-negative integer less than 2^n. Let $f(x)$ take any integer x into the mod 2 sum of the products of corresponding bits of a and x, which we denote by $a \cdot x$:

$$a \cdot x = a_0x_0 \oplus a_1x_1 \oplus a_2x_2 \cdots$$

If we want to determine the value of a, how many times do we have to call $f(x)$? Notice that the m-th bit of a is $a \cdot 2^m$ (with both terms in binary), since the binary expansion of 2^m has 1 in position m and 0 in all the other positions. Thus, with a classical computer, we can learn the n bits of a by applying f to the n values $x = 2^m; 0 \leq m < n$, which requires n different invocations of

the subroutine. However, we will see that with a quantum computer a *single* invocation is enough to determine a completely, regardless of how big n is!
If we prepare the 1-Qbit *output register* in the state

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

Then, since U_f applied to basis $\left|x\right\rangle_n \left|y\right\rangle_1$ flips the value y *only if* $f(x) = 1$, we have

$$U_f\left|x\right\rangle_n \frac{1}{\sqrt{2}}(\left|0\right\rangle - \left|1\right\rangle) = \left|x\right\rangle_n \frac{1}{\sqrt{2}}(\left|0\right\rangle - (-1)^{f(x)}\left|1\right\rangle)$$

This allows us to convert a bit flip to a change of sign. Thus, the action of H on $\left|x\right\rangle_1$ can be written as:

$$H\left|x\right\rangle_1 = \frac{1}{\sqrt{2}}(\left|0\right\rangle + (-1)^x\left|1\right\rangle) = \frac{1}{\sqrt{2}}\sum_{y=0}^{1}(-1)^{xy}\left|y\right\rangle$$

Generalizing $H^{\otimes n}$ to an n-Qbit $\left|x\right\rangle_n$ can be expressed as:

$$H^{\otimes n}\left|x\right\rangle_n = \frac{1}{2^{n/2}}\sum_{y=0}^{2^n-1}(-1)^{xy}\left|y\right\rangle_n$$

Now if we start with the n-Qbit input register in the standard initial state $H^{\otimes n}\left|0\right\rangle$, put the 1-Qbit output register into the state $H\left|1\right\rangle$, apply U_f, and then apply $H^{\otimes n}$ to only the input register, we get

$$(H^{\otimes n} \otimes 1)U_f(H^{\otimes n} \otimes H)\left|0\right\rangle_n \left|1\right\rangle_1 =$$

$$(H^{\otimes n} \otimes 1)U_f\Big(\frac{1}{2^{n/2}}\sum_{x=0}^{2^n-1}\left|x\right\rangle\Big)\frac{1}{\sqrt{2}}(\left|0\right\rangle - \left|1\right\rangle) =$$

$$\frac{1}{2^{n/2}}\Big(H^{\otimes n}\sum_{x=0}^{2^n-1}(-1)^{f(x)}\left|x\right\rangle\Big)\frac{1}{\sqrt{2}}(\left|0\right\rangle - \left|1\right\rangle) =$$

$$\frac{1}{2^n}\sum_{x=0}^{2^n-1}\sum_{y=0}^{2^n-1}(-1)^{f(x)+xy}\left|y\right\rangle\frac{1}{\sqrt{2}}(\left|0\right\rangle - \left|1\right\rangle)$$

We'll do the sum over x first. If the function $f(x)$ is $a \cdot x$, then

$$\sum_{x=0}^{2^n-1}(-1)^{ax}(-1)^{xy} = \prod_{j=1}^{n}\sum_{x_j=1}^{1}(-1)^{(a_j+y_j)x_j}$$

At least one term in the product vanishes unless every bit y_j of $y = a_j$ of a (ie $y = a$). Therefore, the entire computation reduces to

$$H^{\otimes(n+1)}U_f H^{\otimes(n+1)}\left|0\right\rangle_n \left|1\right\rangle_1 = \left|a\right\rangle_n \left|1\right\rangle_1$$

This means that by preprocessing the input register appropriately, we only need to call the subroutine once, followed by application of $H^{\otimes n}$ to the input register. This results in the input register becoming $|a\rangle$, so all n bits of a can be determined by measuring the input register.
Alternate way of looking at this with circuit...

4.2 Simon's Problem

Simon's problem also has an n-bit non-zero number a built into the action of subroutine U_f, and the aim is to learn the value of a with as few invocations of the subroutine as possible. In the Bernstein-Vazirani problem, a classical computer had to invoke the subroutine n times while a quantum computer invoked the subroutine 1 time. In Simon's problem, the difference is even more dramatic: a classical computer invokes n^2 times, while a quantum computer only invokes n times.
The subroutine U_f in Simon's problem evaluates a function f on n bits that is two to one ie a function from n to $n-1$ bits. It is specifically constructed so that $f(x) = f(y)$ iff the n-bit integers x and y are related by $x = y \oplus a$, which is equivalently $x \oplus y = a$. f is periodic under modulo-2 addition: $f(x \oplus a) = f(x)$ for all x and the problem is to find a.
To find a in a classical computer, we must pass in x_1, x_2, x_3, ... to $f(x)$ until we stumble upon an x_j that yields some previously computed value $f(x_j)$, which means $a = x_j \oplus x_i$. If at some point we have picked m different values and we know $a \neq x_i \oplus x_j$ for all pairs, then we have eliminated $\frac{1}{2}m(m-1)$ values of a. Since there are 2^n possible values, we are unlikely to find a until $m = 2^{n/2}$.
In contrast, a quantum computer can determine a with high probability after calling the subroutine about n times. Here's how:
Like before, we apply U_f only after applying $H^{\otimes n}$ to the input register:

$$\frac{1}{2^{n/2}} \sum_{x=0}^{2^n-1} |x\rangle |f(x)\rangle$$

If we measure the input register, the generalized Born rule tells us that it is in the state

$$\frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus a\rangle)$$

for some random x_0 for which $f(x_0)$ agrees with the value given by output register measurement. This might seem like good progress but isn't really since we can't learn what the input state is. If we measure it, we get either x_0 or $x_0 + a$ which are both random values. Instead, we must do something similar to what we did in Deutsch's problem and try to learn the relationship between the two, instead of something about each of them individually.
Let's apply a Hadamard transformation to the input:

$$H^{\otimes n} \frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus a\rangle) = \frac{1}{2^{(n+1)/2}} \sum_{y=0}^{2^n-1} \Big((-1)^{x_0 \cdot y} + (-1)^{(x_0 \oplus a) \cdot y} |y\rangle$$

Since $(-1)^{(x_0 \oplus a)\cdot y} = (-1)^{x_0 \cdot y}(-1)^{a\cdot y}$, the coefficient for $|y\rangle$ is 0 if $a \cdot y = 1$ and $2(-1)^{x_0 \cdot y}$ if $a \cdot y = 0$. Thus, we can simplify to

$$\frac{1}{2^{(n-1)/2}} \sum_{a\cdot y=0} (-1)^{x_0 \cdot y} |y\rangle$$

So measuring the input register now gives us any of the values of y for which $a \cdot y = 0$ ie for which

$$\sum_{i=0}^{n-1} y_i a_i = 0 \pmod 2$$

where a_i and y_i correspond to ith bit in binary expansions of a and y.
With this algorithm, we learn nothing only if we're unlucky enough to get $y = 0$ (which happens with probability $\frac{1}{2^{n-1}}$. So if we learn a nonzero value of y, if we get $n-1$ bits of a, we can figure out the last bit using y. Thus, we have cut the number of possible choices by a factor of 2. Thus, if we repeat the procedure a few more than n times, we have a very high chance of learning a.

5 The Importance of cNOT gates

While 1-Qbit unitary gates are relatively straightforward to construct, 2-Qbit gates are much tougher wince they require one to manipulate the interaction between 2 physical Qbits. The cNOT gate is the workhorse of 2-Qbit gates because any arithmetic operation can be carried out with 1-Qbit gates acting with 2-Qbit cNOT gates. This is because of:

1. Any arithmetic operation can be built on a reversible classical computer out of *three*-Cbit controlled-controlled-not gates (ccNOT gates, called *Toffoli gates*)
2. The quantum ccNOT gate can be built out of cNOT gates and controlled-U gates, defined as cU_{ij} does nothing to j if i is $|0\rangle$ but applies unitary transformation U to j if i is $|1\rangle$.
3. Controlled-U gates can be built out of cNOT gates and 1-Qbit unitary gates.

5.1 Proving Part 1

In classical computation, the Toffoli gate flips the target bit if and only if *both* of the first two control bits are 1:

$$x, y, z \to x, y, z \oplus xy$$

We define its quantum extension as T. Since T is its own inverse, it is reversible. In classical computing, the Toffoli gate allows us to calculate the AND of two bits by setting the target bit to $|0\rangle$. Since all Boolean operations can be built out of AND and NOT, and since all arithmetic can be built out of Boolean operators, Toffoli gates can build up all of classical computing through reversible operations.

5.2 Proving Part 2

There is not way to construct a Toffoli gate out of 1-Cbit and 2-Cbit classical gates, but we can use a trick with quantum gates.
We begin by noting that we can create a quantum gate whose square is the NOT operator X.

$$\sqrt{X} = \frac{1}{1+i}\begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix}$$

$$(\sqrt{X})^2 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = X$$

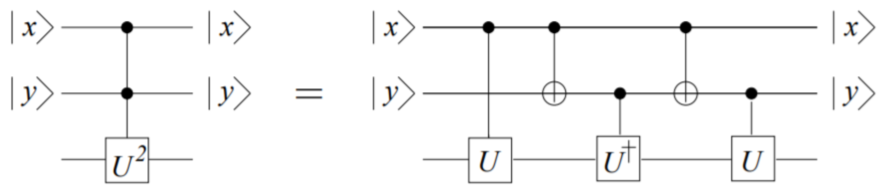
We can also verify that it is unitary:

$$\sqrt{X}^\dagger \sqrt{X} = 1$$

To build a 3-Qbit Toffoli gate, we need a *controlled* square root of NOT gate. Given a controlled-U gate cU_{ij} with Qbit i the control, j the target, and U a 1-Qbit unitary transformation, a doubly controlled U^2 gate, controlled by Qbits 2 and 1 and targeting Qbit 0, can be constructed like so:

$$cU_{10}C_{21}cU_{10}^\dagger C_{21}cU_{20}$$

A circuit diagram of an implementation of a Quantum Toffoli:



Verify that if both Qbits 2 and 1 are in state $|1\rangle$ the gate acts on Qbit 0 as U^2. An easier way to see this is to note that the middle three Qbits leaves Qbits 2 and 1 unchanged while acting on Qbit 0 as 1 if those states are both $|0\rangle$ or both $|1\rangle$, and acting on Qbit 0 as $U^\dagger$ if only one state is $|1\rangle$.

5.3 Proving Part 3

Finally, we show that a controlled-U gate can be built out of cNOT gates and 1-Qbit unitary gates. Let V and W be arbitrary 1-Qbit unitary transformations and consider their product:

$$V_0C_{10}V_0^\dagger W_0C_{10}W_0^\dagger$$

If the control bit Qbit 1 is $|0\rangle$, then this reduces to $(VV^\dagger)(WW^\dagger) = 1$. But if Qbit 1 is $|1\rangle$, then it instead becomes $(VXV^\dagger)(WXW^\dagger)$. Instead, let's think of X as its equivalent form as the *Pauli operator* $X = \sigma_x = x \cdot \sigma$. It is possible (though I don't really understand the proof for why) to pick U so that $U(x \cdot \sigma)U^\dagger = c{\cdot}\sigma$ for any unit vector c by taking U to be the transformation associated with a rotation of x to c. Thus, we choose V and W so that

$$\Big(V(x\cdot\sigma)V^\dagger\Big)\Big(W(x\cdot\sigma)W^\dagger\Big) = (a\cdot\sigma)(b\cdot\sigma) = a\cdot b + \text{ ia}\otimes b\cdot\sigma$$

for any desired unit vectors, a and b.
Remember our goal is to produce a certain unitary transformation. An arbitrary 1-Qbit unitary transformation has the form

$$U(n,\theta) = e^{\frac{i}{2}\theta(n\cdot\sigma)} = cos\frac{1}{2}\theta + i(n\cdot\sigma)sin\frac{1}{2}\theta$$

So by taking a and b to be in the plane perpendicular to n and taking the angle between them to be $\frac{1}{2}\theta$, we can reproduce a similar unitary transformation. Look in Chapter 1 Appendix A2 for more information.
Ultimately, **eight** cNOT gates are required to construct a Toffoli gate in this way: the construction of a doubly-controlled U^2 gate that we outlined above needs two cNOT gates and three controlled-unitary gates, and each controlled-unitary gate as described above requires two cNOT gates