Quantum Computation **(CMU 18-859BB, Fall 2015)**

Lecture 1: Introduction to the Quantum Circuit Model

September 9, 2015

Lecturer: Ryan O'Donnell *Scribe: Ryan O'Donnell*

1 Overview of what is to come

1.1 An incredibly brief history of quantum computation

The idea of quantum computation was pioneered in the 1980s mainly by Feynman [Fey82, Fey86] and Deutsch [Deu85, Deu89], with Albert [Alb83] independently introducing quantum automata and with Benioff [Ben80] analyzing the link between quantum mechanics and reversible classical computation. The initial idea of Feynman was the following: Although it is perfectly possible to use a (normal) computer to simulate the behavior of n-particle systems evolving according to the laws of quantum, it seems be extremely inefficient. In particular, it seems to take an amount of time/space that is exponential in n. This is peculiar because the actual particles can be viewed as simulating *themselves* efficiently. So why not call the particles themselves a "computer"? After all, although we have sophisticated theoretical models of (normal) computation, in the end computers are ultimately physical objects operating according to the laws of physics. If we simply regard the particles following their natural quantum-mechanical behavior as a computer, then this "quantum computer" appears to be performing a certain computation (namely, simulating a quantum system) exponentially more efficiently than we know how to perform it with a normal, "classical" computer. Perhaps we can carefully engineer multi-particle systems in such a way that their natural quantum behavior will do *other* interesting computations exponentially more efficiently than classical computers can.

This is the basic idea behind quantum computers. As it turns out, you *can* get (seemingly) exponential speedups for a (seemingly) small number of natural computational problems by carefully designing a multi-particle quantum system and letting it evolve according to the (100-year old, extremely well-confirmed) laws of quantum mechanics. By far the most spectacular example is Shor's factoring algorithm [Sho97], an algorithm implementable on a quantum computer that can factor any n-digit integer (with high probability) in roughly n^2 time. This is contrast to the fact that the fastest known "classical" algorithm for factoring n-digit integers seems to require roughly $2^{n^{1/3}}$ time, and in fact the presumed computational difficulty of factoring is relied upon in an enormous number of real-world cryptographic applications (e.g., the computations done whenever you type `https://` into your browser).

1.2 Plans for this course and this lecture

Very briefly, in this course we will:

- *Mathematically* formulate the tiny bit of quantum mechanics that is relevant for the field of quantum computation. (We should mention that this course will be heavily slanted towards theoretical computer science and mathematics, and will contain almost no physics.)

- See some quantum algorithms that solve certain computational problems much faster than they are known to be solvable classically.

- Investigate the *limits* of quantum computation. (It is *not* the case that quantum computation automatically provides speedup over classical computation for all problems, or even for a wide class of problems.)

- Study some quantum information theory.

The goal for this first lecture is to give a lightning-fast, as-barebones-as-possible definition of the quantum circuit model of computation. After this lecture, you will theoretically know all you need to know in order to implement and analyze, e.g., Shor's algorithm. (Of course, we will subsequently make a more thorough and leisurely development of quantum computation before actually getting around to sophisticated algorithms.)

90% of the understanding of the quantum circuit model is achieved by reviewing three purely "classical" topics: classical Boolean circuits; reversible classical circuits; and randomized computation. The first and third of these topics should be very familiar to anyone who has studied the basics of theoretical computer science. And the second topic is very cute and elementary. Once we have these three concepts in hand, quantum circuits become practically just a tiny "twist" on randomized computation — what you might get if you tried to invent a model of randomized computation in which "probabilities" can be *negative*...

2 Classical Boolean circuits

Several models of computation/algorithms are studied in the classical theory of computation: Turing Machines, high-level programming languages, and Boolean circuits. It turns out that for the study of quantum computation, the Boolean circuit model is by far the easiest model to generalize (being as it the closest model of the physical reality of computers).

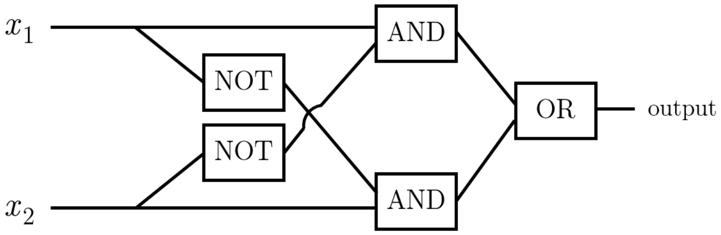
We begin with the following well known fact, stating that any computational task (modeled by a Boolean function) we might want to do is doable with an AND/OR/NOT Boolean circuit.

Proposition 2.1. *Any Boolean function $f : \{0,1\}^n \to \{0,1\}^m$ is computable by a Boolean circuit C using just* AND, OR, *and* NOT *gates. I.e.,* AND, OR, *and* NOT *gates are* universal.

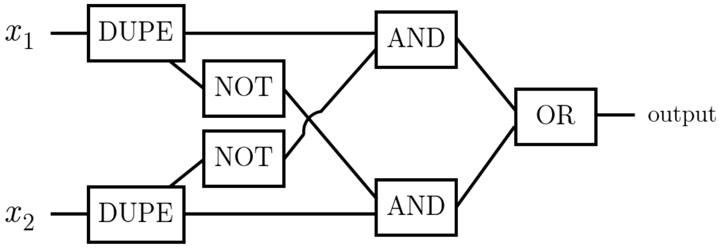
Remark 2.2. The AND and OR gates mentioned in this proposition take 2 input bits and produce 1 output bit. The NOT gate takes 1 input bit and produces 1 output bit.

Remark 2.3. Once we know that every Boolean function is computable by some circuit, we usually become interested in computing it *efficiently*; i.e., with a circuit C of small *size*. The size of the circuit, $\mathrm{size}(C)$, is defined to be the number of gates it uses. Circuit size fairly closely corresponds to *running time* in the Turing Machine (sequential algorithm) model. For example, it is known that a circuit of size s can be evaluated in time $O(s \log s)$ by a Turing Machine, and conversely, a Turing Machine operating in time t on length-n inputs can be converted to an n-input circuit of size $O(t \log t)$.

Here is a simple example of a circuit computing the XOR function, $f(x_1, x_2) = x_1 \oplus x_2$:

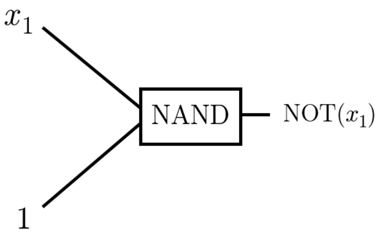


The lines in this diagram are called "wires", and the things inside the rectangles are called "gates". In the diagram we have followed a traditional circuit-theory convention by allowing wires to "branch"; i.e., split into two copies. In reality, some physical mechanism must exist at these branches, and in the future it will be convenient to make this explicit. So we will introduce a new kind of gate called a DUPE (*duplicate*) gate which takes 1 input bit and outputs 2 duplicate copies of that bit. We will then redraw the above diagram as follows:



With this convention, it would be more accurate to say that AND, OR, NOT, and DUPE gates are universal for Boolean circuit computation.

It is also a well known fact that one can get smaller universal gate sets; in fact, one can replace AND/OR/NOT gates with just NAND gates. (Recall that $\mathrm{NAND}(x_1, x_2) = \mathrm{NOT}(\mathrm{AND}(x_1, x_2))$.) To see this, first note that we can eliminate OR gates using De Morgan's rule: $\mathrm{OR}(x_1, x_2) = \mathrm{NOT}(\mathrm{AND}(\mathrm{NOT}(x_1), \mathrm{NOT}(x_2)))$. Then we can eliminate AND gates in favor of NAND gates via $\mathrm{AND}(x_1, x_2) = \mathrm{NOT}(\mathrm{NAND}(x_1, x_2))$. Finally, we need to show that NOT gates can be eliminated using NAND gates. One way to implement $\mathrm{NOT}(x_1)$ with a NAND gate is as follows:



On the lower left in this diagram, we have what is called an *ancilla* bit: an input that is "hardwired" to the constant bit 1, for the purposes of assisting the computation. It's actually possible to implement NOT(x_1) using NAND and DUPE without the use of ancillas (specifically, via NAND(DUPE(x_1))). However the above method gives us a good opportunity to introduce the notion of ancillas.

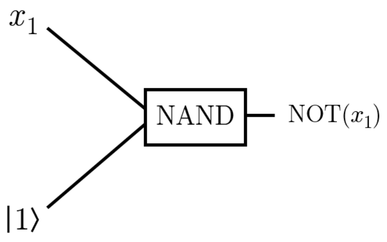
What we have just shown is the following:

Proposition 2.4. *Boolean* NAND *and* DUPE *gates (along with the use of ancillas) are* universal *for computation.*

Remark 2.5. In fact, we have shown something stronger: Not only can every AND/OR/NOT/DUPE circuit C be converted to an equivalent AND/OR/NOT circuit C', this conversion can be done very *efficiently*; there is an efficient algorithm carrying out the conversion, and $\text{size}(C') = O(\text{size}(C))$.

2.1 Bra-ket notation

We take this opportunity to introduce a bit of unusual notation that will play an essential role in the remainder of the course. This is the "bra-ket" notation invented by Paul Dirac. Actually, we will postpone the mathematical definitions to the next lecture; for now we will just introduce it as pure symbolism. We will henceforth enclose bits and bit-strings in asymmetrical brackets called *kets*, writing $|0\rangle$ and $|1\rangle$ instead of 0 and 1. We will also usually eliminate internal brackets when writing strings; e.g., writing $|011\rangle$ instead of $|0\rangle\,|1\rangle\,|1\rangle$. As a small example of this notation, we will redraw the previous diagram as follows:



3 Reversible computation

In actual physical reality, a theoretical bit ($|0\rangle$ or $|1\rangle$) is implemented by a particle or bunch of particles (e.g., high or low voltage on a physical wire). Similarly, a gate is implemented by a physical object (a "switch" or some other gadget) that manipulates the bit-representations. We then would ideally like to think of the circuit as a "closed physical system". Unfortunately, for a typical AND/OR/NOT/DUPE circuit, this is not possible. The reason is that the laws of physics governing microscopic systems (both classical and quantum) are *reversible* with respect to time, but this is not true of most gates we would like to physically implement.

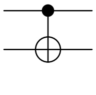
Take for example an AND gate. Suppose its output is $|0\rangle$. Can we infer what its inputs were? The answer is no — they could have been $|00\rangle$, $|01\rangle$, or $|10\rangle$. The AND process is not reversible: information sometimes needs to be deleted; "entropy" is lost. According to the 2nd Law of Thermodynamics, a physical system consisting of a single AND gate cannot be "closed"; its operation must dissipate some energy — typically as escaping heat. On the other hand, a NOT gate *is* theoretically "reversible": its output can be determined from its input; no information is created or destroyed in switching $|0\rangle$ to a $|1\rangle$ or vice versa. Thus, in principle, it is possible to construct a completely closed physical system implementing a NOT gate, without the need for energy dissipation.

These issues were studied in the 1960s and 1970s by Landauer [Lan61] and Bennett [Ben73], among others. They raised the question of whether there are Boolean gates that are both reversible and universal. If so, then by using them it would be possible — at least according to the theoretical laws of physics — to have circuits doing general computation without dissipating any energy. On one hand, as we will see shortly, it *is* possible to find universal reversible gates. On the other hand, it turned out that from a practical point of view, the energy dissipation of standard electronic circuits did not prove to be a major problem (although laptops sometimes *do* get rather hot in your lap). On the other other hand, it turns out to be important for the quantum circuit model that universal reversible computation is possible. So we will now explain how to do it. We begin with a definition:

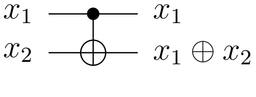
Definition 3.1. A Boolean gate G is said to be *reversible* if it has the same number of inputs as outputs, and its mapping from input strings to output strings is a bijection.

Thus a NOT gate is reversible, whereas most other "standard" gates (e.g., AND, OR, NAND, and DUPE) cannot be reversible since they do not have an equal number of inputs and outputs.

Let's introduce a new, simple, reversible gate, the CNOT (*controlled*-NOT) gate. It has 2 input bits and 2 output bits, and is drawn like this:



Its behavior is as follows:



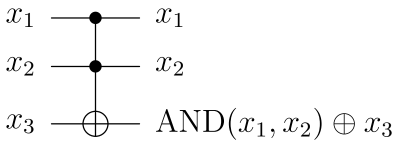
That is, the first input bit x_1 is always passed through directly; the second bit gets NOT applied to it if and only if the "control" bit x_1 is $|1\rangle$. To be even more explicit, CNOT has the following truth table:

CNOT:

input	output
$\|00\rangle$	$\|00\rangle$
$\|01\rangle$	$\|01\rangle$
$\|10\rangle$	$\|11\rangle$
$\|11\rangle$	$\|10\rangle$

You can see that this mapping is indeed a bijection, confirming that CNOT is a reversible gate.

We now describe a small but important generalization, called the CCNOT (*controlled-controlled*-NOT) or *Toffoli gate*. The below diagram indicates how this 3-input, 3-output gate is drawn, as well as its behavior:



In other words, the first two inputs to a CCNOT gate are passed through directly, and the third input is negated if and only if the first two "control" input bits are both $|1\rangle$.[1] Explicitly, we have the following truth table, showing that CCNOT is reversible:

CCNOT:

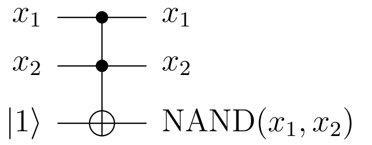
input	output
$\|000\rangle$	$\|000\rangle$
$\|001\rangle$	$\|001\rangle$
$\|010\rangle$	$\|010\rangle$
$\|011\rangle$	$\|011\rangle$
$\|100\rangle$	$\|100\rangle$
$\|101\rangle$	$\|101\rangle$
$\|110\rangle$	$\|111\rangle$
$\|111\rangle$	$\|110\rangle$

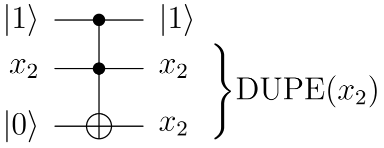
Remark 3.2. The three examples of reversible gates we have seen so far — NOT, CNOT, CCNOT — also have the extra property that *they are their own inverse*; i.e., applying them twice in succession restores the original bits. This is a bit of a coincidence, insofar as it is *not* a property we insist on for reversible gates. We merely insist that reversible gates have *some* inverse gate; the inverse doesn't have to be the gate itself.

The CCNOT gate is extremely handy: as the following two pictures show, we can use it to simulate both NAND gates *and* DUPE gates (assuming, as always, that ancillas are

[1] In general, we use the convention that attaching a dot to a k-input/k-output gate G with a vertical line means creating a "controlled-G" gate. This is the $(k+1)$-input/$(k+1)$-output gate that passes through its first, "control", bit, and which either applies G or doesn't depending on whether the control bit is $|1\rangle$ or $|0\rangle$. Assuming that a NOT gate is drawn as $\oplus$, this explains the picture used for CNOT and CCNOT gates.

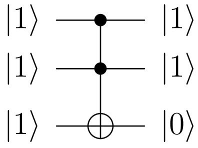
allowed):





Note that, in addition to producing the desired NAND and DUPE outputs, these conversions also produce extra, unneeded bits (namely, x_1 and x_2 in the NAND case, and the top $|1\rangle$ in the DUPE case). This is somewhat inevitable, given that reversible gates are required to have equally many input and output bits. We call such unwanted outputs *garbage*.

As one more very minor note, the above conversions use both $|0\rangle$ ancillas and $|1\rangle$ ancillas. We can also use CCNOT gates to generate $|0\rangle$'s from $|1\rangle$ ancillas as follows:



We have therefore established the following key theorem, which shows that we can do universal computation reversibly.

Theorem 3.3. *The* CCNOT *gate is universal, assuming ancilla inputs (all set to* $|1\rangle$*) and garbage outputs are allowed; any standard* AND/OR/NOT *circuit for a function* $f : \{0,1\}^n \to \{0,1\}^m$ *may be efficiently transformed into a reversible one that looks like Figure 1.*

Remark 3.4. In reversible circuits we will always have $n + \#\text{ancillas} = m + \#\text{garbage}$.

Remark 3.5. "In practice", when doing reversible computing we usually also allow ourselves NOT and CNOT gates. (Given NOT gates, we may assume that all ancillas are fixed to $|0\rangle$ rather than $|1\rangle$, and this is actually a more traditional assumption.)

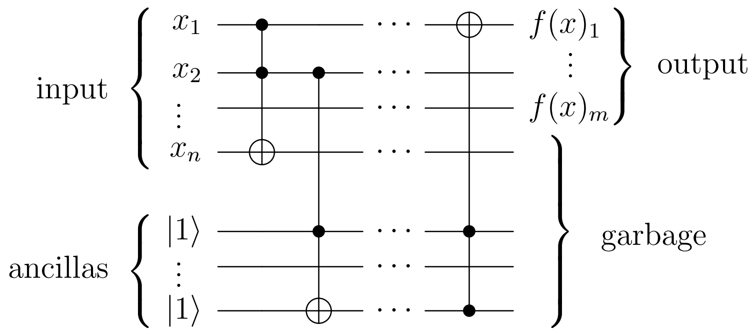


Figure 1: A typical reversible circuit using CCNOT gates. We remark that the output bits need not be the "topmost" m bits on the right; we could designate any of the m bits on the right as the outputs.

With "standard" circuits, the number of wires carrying a bit at any one "time" (vertical slice) may vary. However with reversible circuits, this will always equal the number of inputs+ancillas, as you can see above. Indeed, one sort of stops thinking about wires and instead thinks of each input/ancilla bit being carried in its own *register*, which maintains its "identity" throughout the computation. It's very helpful to think of circuits not just as diagrams but also as "lists of instructions performed on registers", as in the following description, which is completely equivalent to the diagram in Figure 1:

```
Input is in registers $x_1, x_2, \dots, x_n$.
"Attach" $c$ ancillas in $x_{n+1}, \dots, x_{n+c}$, initialized to $|1\rangle$.

    • CCNOT($x_1, x_2, x_n$)
    • CCNOT($x_2, x_{n+1}, x_{n+c}$)
    • $\cdots$
    • CCNOT($x_{n+c}, x_{n+1}, x_1$)

Output is in registers $x_1, \dots, x_m$.
```

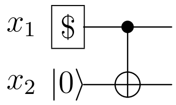
4 Randomized computation

Although we are well used to it now, randomized computation is a bit like the "quantum computation of the '60s and '70s" — a creative new twist on classical computation, seemingly realizable in practice and potentially allowing for big speedups over deterministic computation, but one requiring its investigators to make a big investment in a new area of math (i.e., probability). Indeed, there are some computational tasks which we know how to provably solve efficiently using randomized computation, but which we don't know how to provably solve efficiently using only deterministic computation. (An example: on input "n", generate an n-digit prime number.) Unlike with quantum computation, however, we believe this is mainly due to our lack of skill in proving things, rather than an inherent major advantage of randomized computation. (E.g., we know a deterministic algorithm that we *believe* efficiently generates n-digit prime numbers; we just can't prove its efficiency.)

It is very easy to upgrade the circuit model of computation to a randomized model: we just introduce a single new gate called the COIN gate, drawn like this: $\boxed{\text{COIN}}$— or $\boxed{\$}$—. It has 0 inputs and 1 output; the output is a "fair coin flip", viz., $|0\rangle$ with probability $\frac{1}{2}$ and $|1\rangle$ with probability $\frac{1}{2}$.

Remark 4.1. You might also imagine allowing other kinds of randomized gates; for example, a $\text{COIN}_{\frac{1}{3}}$ gate that outputs $|1\rangle$ with probability $\frac{1}{3}$ and $|0\rangle$ with probability $\frac{2}{3}$. It turns out that allowing such gates does not fundamentally change the model of randomized computing; although a fair COIN gate cannot simulate a $\text{COIN}_{\frac{1}{3}}$ gate *exactly*, it can simulate it close-to-exactly enough that it doesn't really matter. For this reason we say that the plain COIN gate is (effectively) *universal* for randomized computation. See Homework 1 for more details.

We will now describe how to "analyze" randomized circuits. Although our style of analysis will be very simple and pedantic, it will be great practice for analyzing the closely related quantum circuits. Here is an example randomized circuit:



Alternatively, we could think of this circuit as the following "program":

1. x_1 initialized to $\boxed{\$}$
2. x_2 initialized to $|0\rangle$
3. CNOT(x_1, x_2)

As you can easily see, the output of this circuit is $|00\rangle$ with probability $\frac{1}{2}$ (if the coin flip is $|0\rangle$) and is $|11\rangle$ with probability $\frac{1}{2}$ (if the coin flip is $|1\rangle$). Nevertheless, let's patiently "analyze" it.

The state of x_1 after the coin flip — equivalently, after Line 1 of the program — is

$$\frac{1}{2} \text{ probability of } |0\rangle, \quad \frac{1}{2} \text{ probability of } |1\rangle. \tag{1}$$

Let us introduce some funny notation for this:

Notation 4.2. *We will write* (1) *as*

$$\frac{1}{2} \cdot |0\rangle + \frac{1}{2} \cdot |1\rangle.$$

In the next lecture we will "make mathematical sense" of this notation using linear algebra, but for this lecture you should be perfectly happy just treating it as some "formal notation".

The state of x_2 after Line 2 of the program is

$$1 \text{ probability of } |0\rangle, \quad 0 \text{ probability of } |1\rangle,$$

which we will write in our new notation as

$$1 \cdot |0\rangle + 0 \cdot |1\rangle = |0\rangle.$$

Here we have used the "usual" laws and notation of arithmetic in the equality. (Again, continue to think of this as shorthand notation.)

Finally, what happens at the end of the circuit, after Line 3? One could say that we have:

$$\begin{aligned}\text{State of } x_1: &\quad \frac{1}{2}\left|0\right\rangle + \frac{1}{2}\left|1\right\rangle \\ \text{State of } x_2: &\quad \frac{1}{2}\left|0\right\rangle + \frac{1}{2}\left|1\right\rangle\end{aligned}$$

While in some sense this is true (each "register" *is* equally likely to be $\left|0\right\rangle$ or $\left|1\right\rangle$), it's grossly misleading. It makes it looks as if the two bits are independent, when in fact they are **correlated**. So the above analysis is true but incomplete; to truly capture the correlations in the system we should say:

$$\text{Joint state of } x_1, x_2: \quad \frac{1}{2}\left|00\right\rangle + \frac{1}{2}\left|11\right\rangle.$$

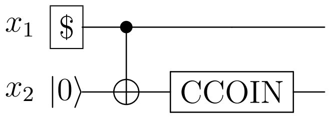
In our analyses of randomized circuits, we will keep track of the joint state of all registers all along. For example, in the circuit we have been analyzing the joint state just *prior* to Line 3 would be

$$\frac{1}{2}\left|00\right\rangle + \frac{1}{2}\left|10\right\rangle.$$

Let's practice some more analysis of "r-bit circuits" (where "r" standards for "randomized"). For the purposes of practice we'll invent a new randomized gate, $-\boxed{\text{CCOIN}}-$ ("controlled-coin"), which has 1 input and 1 output. Its behavior is the following:

	input	output
CCOIN:	$\lvert 0\rangle$	$\lvert 0\rangle$
	$\lvert 1\rangle$	$\begin{cases}\lvert 0\rangle & \text{with prob. } \frac{1}{2} \\ \lvert 1\rangle & \text{with prob. } \frac{1}{2}\end{cases}$

Now let's extend the 2 r-bit circuit we had previously been analyzing, as follows:



Equivalently, we are adding the instruction "4. CCOIN(x_1, x_2)" to our program. Now prior to the CCOIN gate, the joint state of the system is

$$\frac{1}{2}\left|00\right\rangle + \frac{1}{2}\left|11\right\rangle. \tag{2}$$

What is the state after the new CCOIN gate? Here is how you would say it in words:

> Prior to the CCOIN gate, (2) tells us that there is a $\frac{1}{2}$ probability that x_1 is $\left|0\right\rangle$ and x_2 is $\left|0\right\rangle$. In this case, the CCOIN does not touch x_1, so it stays $\left|0\right\rangle$, and the CCOIN gate leaves x_2 as $\left|0\right\rangle$ as per its definition. Thus the final state in this case

> is still $|00\rangle$. On the other hand, (2) tells us that there is a $\frac{1}{2}$ probability that x_1 is $|1\rangle$ and x_2 is $|1\rangle$. In this case, the CCOIN does not touch x_1, so it stays $|1\rangle$, and the CCOIN gate changes x_2 to $|0\rangle$ with probability $\frac{1}{2}$ and to $|1\rangle$ with probability $\frac{1}{2}$, as per its definition. Thus overall the final state is $|00\rangle$ with probability $\frac{1}{2}$, is $|10\rangle$ with probability $\frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$, and is $|11\rangle$ with probability $\frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$.

Here is the math symbolism you would write to exactly model those words:

$$\text{the final state is} \quad \frac{1}{2}|00\rangle + \frac{1}{2}\left(\frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle\right) = \frac{1}{2}|00\rangle + \frac{1}{4}|10\rangle + \frac{1}{4}|11\rangle.$$

As you can see, the natural "arithmetic" you would write with this formalism matches up with the actual probabilistic calculations.

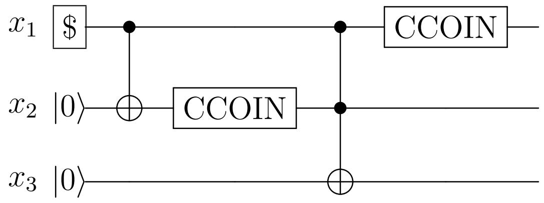
Let's add a few more twists. Suppose that x_3 is a new register that was in the system all along (we forgot to tell you about it), initialized to $|0\rangle$ and never touched. Then we would say that the final state of the system is

$$\frac{1}{2}|000\rangle + \frac{1}{4}|100\rangle + \frac{1}{4}|110\rangle.$$

Suppose we now added a CCNOT(x_1, x_2, x_3) gate to the end of the circuit. What would the new state be? Clearly we just go through the above state and change each of the bit-strings according to CCNOT's operation, leaving the probabilities unchanged:

$$\frac{1}{2}|000\rangle + \frac{1}{4}|100\rangle + \frac{1}{4}|111\rangle. \tag{3}$$

Finally, suppose we now added a CCOIN(x_1) instruction, so that the final circuit looked like this:



Now we can calculate the final state as follows. We start with state (3) and proceed through the "terms" (probabilistic cases) in it. For each one, the last two r-bits in the string will be unchanged, since the final CCOIN gate only operates on x_1. If the first r-bit is $|0\rangle$ then it will stay $|0\rangle$, as per CCOIN's definition. On the other hand, if the first r-bit is $|1\rangle$ then it will become $|0\rangle$ with probability $\frac{1}{2}$ and become $|1\rangle$ with probability $\frac{1}{2}$ (generating two "terms"). Then we simplify. The calculation is:

$$\begin{aligned} &\frac{1}{2}|000\rangle + \frac{1}{4}\left(\frac{1}{2}|000\rangle + \frac{1}{2}|100\rangle\right) + \frac{1}{4}\left(\frac{1}{2}|011\rangle + \frac{1}{2}|111\rangle\right) && (4) \\ = &\frac{5}{8}|000\rangle + \frac{1}{8}|100\rangle + \frac{1}{8}|011\rangle + \frac{1}{8}|111\rangle. && (5) \end{aligned}$$

And indeed, had you been asked to compute the final joint state of the 3 r-bits in the above circuit, however you analyzed it would ultimately be pretty close to our pedantic style, and you would have indeed computed that there's a $\frac{5}{8}$ chance of ending with $|000\rangle$, a 0 chance of ending with $|001\rangle$, a $\frac{1}{8}$ chance of ending with $|100\rangle$, etc.

Remark 4.3. An obvious yet important takeaway from this kind of analysis is the following: Suppose we have a circuit with n r-bit registers. At any time, the state of the circuit can be written as

$$\sum_{x \in \{0,1\}^n} p_x |x\rangle ,$$

where the "coefficient" probabilities p_x are *nonnegative* and *summing to* 1.

4.1 On measurement

As a small note, we typically imagine that we provide the inputs to a randomized circuit, and then we observe (or *measure*) the outputs. The probabilistic "state" of the registers at some intermediate time in the circuit's execution reflects only the uncertainty that we, the observers, have about the registers' values. Of course, in reality the registers always have some definite value; it's merely that these variables are "hidden" to us. Analytically, once we observe one or more of the r-bits, the probabilistic state "collapses" to reflect the information we learned.

For example, in the randomized circuit we analyzed in the previous section, the final state (5) is

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

Suppose for example we measure just the first register, x_1. The probability we observe a $|0\rangle$ is

$$\frac{5}{8} + \frac{1}{8} = \frac{6}{8} = \frac{3}{4}.$$

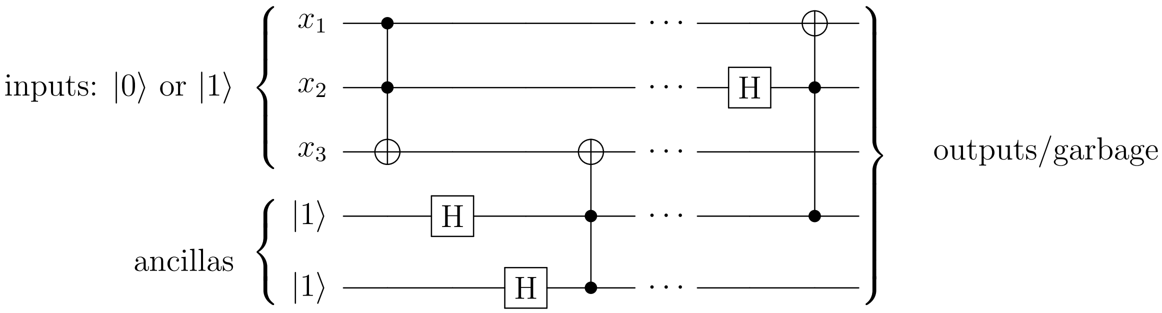
Supposing we do observe a $|0\rangle$, if we wanted to continue the analysis we would use the law of conditional probability to deduce that the state of the system "collapses" to

$$\frac{5/8}{3/4}|000\rangle + \frac{1/8}{3/4}|011\rangle = \frac{5}{6}|000\rangle + \frac{1}{6}|011\rangle .$$

Here, since we observed that the first bit was $|0\rangle$, only the strings consistent with that outcome survive, and the remaining probabilities are renormalized.

5 Quantum computation

Finally we can introduce the (barest essentials) of the quantum circuit model of computation. As mentioned, it is kind of like what you would get if you took randomized computation but found a way to allow the "probabilities" to be negative. It can also arguably be described as classical reversible computation augmented with the *Hadamard gate* $-\boxed{\mathrm{H}}-$. In other words, a typical quantum circuit with 5 **qubit** (*quantum bit*) registers might look like this:



As usual, it is sufficient to just use CCNOT gates in addition to Hadamard gates, but for convenience we also allow NOT and CNOT gates too.

Just as in randomized computation, to analyze such a circuit we need to keep track of the joint state of all 5 qubits as time passes; i.e., as they proceed through the gates. Even though each gate only affects a small subset of all qubits, nevertheless just as in randomized computation we must track the state of all qubits at all times in order to keep track of the "correlations", which are called **entanglement** in the context of qubits.

Further, just as in randomized computation, at each time step the state of the 5 qubits is given by an expression that looks like this:

$$\alpha\,|00000\rangle + \beta\,|00001\rangle + \gamma\,|00010\rangle + \cdots + \omega\,|11111\rangle\,, \tag{6}$$

where the 32 (in general, 2^n) coefficients are **possibly negative** real numbers.[2] Since these numbers may be negative, we don't call them probabilities any more; we call them **amplitudes** (and a state like (6) is called a **superposition** of $|00000\rangle, \ldots, |11111\rangle$). Finally, just as in Remark 4.3, there is a restriction on what amplitudes are possible; this restriction is that the sum of their squares[3] is always 1:

$$\alpha^2 + \beta^2 + \cdots + \omega^2 = 1.$$

Note that this restriction is indeed always satisfied at the input. For example, if the actual input to the above quantum circuit is $|101\rangle$, then the initial state (when ancillas are included) is $|10111\rangle$. Here the amplitude on $|10111\rangle$ is 1, the amplitude on the other 31 strings is 0, and indeed $0^2 + \cdots + 0^2 + 1^2 + 0^2 + \cdots + 0^2 = 1$.

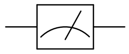
Here are all the rules of how to analyze quantum circuits:

[2] Actually, they can even be *complex numbers*. However if we stick to quantum circuits containing only Hadamard gates and reversible classical gates, then they will always just be real numbers. Further, it is known that these two gates are *universal* for quantum computation, and hence real numbers are universal for quantum computation. I.e., strictly speaking you don't need to worry about complex numbers if you don't want to; though ultimately, it will be more convenient (and physically accurate) to allow them.

[3] Squared magnitudes, if they're complex numbers

- CCNOT gates (and other classical reversible gates like NOT and CNOT) are analyzed exactly as in randomized circuits.
- Hadamard gates $-\boxed{\mathrm{H}}-$ have the following input/output behavior: $|0\rangle \mapsto \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$ and $|1\rangle \mapsto \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$.
- Follow all the same arithmetic rules as for analyzing "r-bit (randomized) circuits", except say the word "amplitude" whenever you were going to say "probability".
- At the end, when you *measure* the output bits, the rule is probabilistic: if the final state is as in (6), then you "see"...

  $$\begin{aligned}
  |00000\rangle &\text{ with probability } \alpha^2,\\
  |00001\rangle &\text{ with probability } \beta^2,\\
  &\vdots\\
  |11111\rangle &\text{ with probability } \omega^2.
  \end{aligned}$$

  For now we will assume that you always measure all the qubits at the end of a circuit's computation, and nowhere else. The picture for a measurement is 

That's it! We could now, in theory, describe an enormous 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). Of course, that would be like if someone described a circuit for computing the maximum flow in a graph immediately after explaining what a NAND gate is. But it's possible.

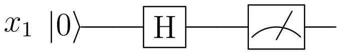
Remark 5.1. As mentioned in a footnote, it is a theorem that CCNOT and Hadamard gates together are universal for quantum computation. But they are not the only gates that are allowed by the physical laws of quantum computation. In the next lecture we will see just what gates nature *does* allow. For now, though, we will mention that any classical reversible gate is okay, and it's convenient to allow NOT and CNOT.

In fact, it is not complete obvious that the reversible gates and the Hadamard gate preserve the key property of quantum states — that the sum of the squares of the (magnitudes of the) amplitudes is 1. You will check this on the homework. In fact, the set of gates that quantum mechanics allows is *exactly* the set of linear transformations of amplitudes which preserve this property.

Remark 5.2. Quantum circuits are at least as powerful/efficient as classical circuits, since we can just not use Hadamard gates, and we're precisely left with reversible classical computation.

Remark 5.3. Quantum circuits are also at least as powerful/efficient as randomized circuits. This will be mostly clear from the following example, which shows how quantum circuits can generate COIN gates. (See the homework for a few clarifying details.)

Let us give an extremely basic example of a quantum circuit:

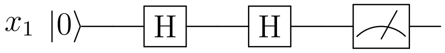


Here the single qubit x_1 is initialized to $|0\rangle$. By definition, after the Hadamard gate, the state of the register is

$$\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle . \tag{7}$$

Finally, when the register is measured at the end, the measurement rule tells us that we observe $|0\rangle$ with probability $\left(\frac{1}{\sqrt{2}}\right)^2 = \frac{1}{2}$ and we observe $|1\rangle$ with probability $\left(\frac{1}{\sqrt{2}}\right)^2 = \frac{1}{2}$. Thus the outcome is just like a fair coin flip. (This would also have been the case had x_1 been initialized to $|1\rangle$; in that case, we would still see the output $|1\rangle$ with probability $\left(-\frac{1}{\sqrt{2}}\right)^2 = \frac{1}{2}$.)

However an interesting thing happens if, instead of measuring the register after the Hadamard gate is applied, we first apply another Hadamard gate and *then* measure:



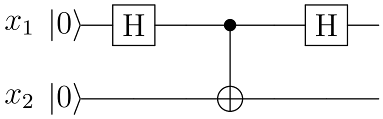
Let us do the analysis here. Just prior to the second Hadamard gate, the state of the register is as in (7). What is the new state after the second Hadamard gate? In words, we say almost exactly the same thing we would say in the case of a probabilistic analysis:

> With ~~probability~~ amplitude $\frac{1}{\sqrt{2}}$, the input is $|0\rangle$, in which case by definition the Hadamard gate outputs $|0\rangle$ with ~~probability~~ amplitude $\frac{1}{\sqrt{2}}$ and outputs $|1\rangle$ with amplitude $\frac{1}{\sqrt{2}}$. On the other hand, with amplitude $\frac{1}{\sqrt{2}}$, the input is $|1\rangle$, in which case by definition the Hadamard gate outputs $|0\rangle$ with amplitude $\frac{1}{\sqrt{2}}$ and outputs $|1\rangle$ with amplitude $-\frac{1}{\sqrt{2}}$. Thus the final state is...

$$\frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right) + \frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle\right) = \left(\frac{1}{2}+\frac{1}{2}\right)|0\rangle + \left(\frac{1}{2}-\frac{1}{2}\right)|1\rangle = |0\rangle \text{ !}$$

Now when we measure the register, we will *always* see $|0\rangle$. It's *kind of* like we "unflipped the coin". The positive/negative cancelation achieved here is precisely both the power and the mystery of quantum computation.

Let us do one more complicated example for practice. We will analyze this circuit:



The initial state is $|00\rangle$.

After the first Hadamard gate, the state is $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|10\rangle$ (the second qubit is always unchanged).

After the CNOT gate, the state is

$$\frac{1}{\sqrt{2}}\left|00\right\rangle + \frac{1}{\sqrt{2}}\left|11\right\rangle . \tag{8}$$

(As you can see, when applying classical gates, you really just need to change bit-strings according to the gate's definition; there's no need to do any arithmetic.) This state (8) is a famous entangled state; it is called an *EPR pair* after Einstein, Podolsky, and Rosen.[4] *If* we were to measure the two qubits at this point, we would see $\left|00\right\rangle$ with probability $\frac{1}{2}$ and $\left|11\right\rangle$ with probability $\frac{1}{2}$.

Finally, after the final Hadamard gate applied to state (8), we get

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

That's it. If we were to measure now, the rule tells us we would observe each of the four outcomes $\left|00\right\rangle, \left|01\right\rangle, \left|10\right\rangle, \left|11\right\rangle$ with probability $\frac{1}{4}$ each.

5.1 On measurement

It is important to stress the following distinction with randomized computation. As mentioned, in the middle of a randomized circuit's computation, the probabilistic state of the registers only represents the uncertainty we (the analyzers) have about the bits' values. However, it's not like the bits have this state in Nature. Rather, the bits are *actually* in some deterministic state; we just don't know what it is.

This is **not** the case in quantum computation. According to the laws of physics, in the middle of a quantum circuit's computation, the superposition state that the n qubits are in is literally the true state they're in in Nature. They are not secretly in one of the basic states; Nature is literally keeping track of the 2^n amplitudes. (This gives you a hint of the potential computational power of quantum mechanics!) In a later lecture we will describe the theory that *proves* that this is the case, as well as the experimental work (including outstanding work from earlier this month) that backs up the theory.

References

[Alb83] David Albert. On quantum-mechanical automata. *Physics Letters A*, 98(5):249–252, 1983.

[Ben73] Charles Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17(6):525–532, 1973.

[4] These three people wrote a paper together doubting that the laws of quantum mechanic could be true, based on the nature of this state. However, EP&R were proven wrong.

[Ben80] Paul Benioff. The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. *Journal of Statistical Physics*, 22(5):563–591, 1980.

[Deu85] David Deutsch. Quantum theory, the Church–Turing principle and the universal quantum computer. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 400, pages 97–117, 1985.

[Deu89] David Deutsch. Quantum computational networks. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 425, pages 73–90, 1989.

[Fey82] Richard Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6/7):467–488, 1982.

[Fey86] Richard Feynman. Quantum mechanical computers. *Foundations of Physics*, 16(6):507–531, 1986.

[Lan61] Rolf Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5(3):183–191, 1961.

[Sho97] Peter Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM journal on computing*, 26(5):1484–1509, 1997.