Lecture 16, Tues March 21: Quantum Computing, Universal Gate Sets

Guest Lecture by Tom Wong

Having seen lots of quantum protocols, we're finally ready to tackle the holy grail of the field: a *programmable quantum computer*, a single machine that could do *any* short series of quantum-mechanical operations.

Quantum computation has two intellectual origins.

One comes from David Deutsch, who was thinking about experimentally testing the Many-Worlds Interpretation (of which he was and remains a firm believer), during his time as a postdoc here at UT Austin. Much like with Wigner's friend, Deutsch imagined creating an equal superposition of a human brain having measured a qubit as $|0\rangle$, and the same brain having measured the qubit as $|1\rangle$. In some sense, if you ever had to ascribe such a superposition state to *yourself*, then we'd have to discard the Copenhagen Interpretation.

But how could we ever test this? Given the practical impossibility of isolating all the degrees of freedom in a human brain, *Step 1* would presumably have to be: take a complete description of a human brain, and upload it to a computer. Then *Step 2* would be to put the computer into a superposition of states corresponding to different thoughts.

But then, this leads to more general questions: *could* anything as complicated as a computer be maintained in a superposition of "semantically different" states? How would you arrange that, in practice? And would such a computer be able to use its superposition power to do anything that it couldn't do otherwise?

The other, more "respectable," path to the same questions came from Richard Feynman, who gave a famous lecture in 1982 concerned with the question, "how do you simulate quantum mechanics on a classical computer?"

Chemists and physicists had known for decades that this is hard, basically because the number of amplitudes that you need to keep track of increases exponentially with the number of particles. This is the case because, as we know, an n-qubit state can be highly entangled.

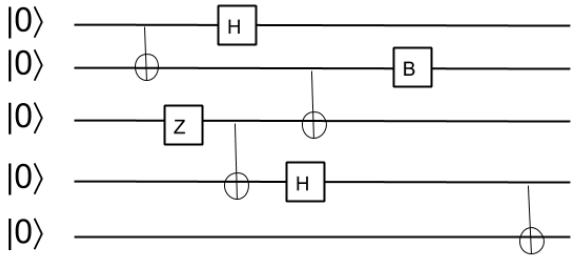
$$\text{The state } |\Psi\rangle = \sum_{x\epsilon\{0,1\}^n} \alpha x_n |x\rangle \text{ must be described by the vector } \begin{bmatrix} \alpha_{00\ldots0} \\ \alpha_{00\ldots1} \\ \vdots \end{bmatrix} \text{ of length } 2^n$$

Chemists and physicists know many approximation methods for such problems—going by names like Density Functional Theory and Quantum Monte Carlo—that often work in practice. In the worst case, though, even to solve for the energy of the system, or for the state of some particular qubit, there's no known shortcut to dealing with the whole vector of 2^n amplitudes. So Feynman raised the question, "Why don't we build computers *out of qubits*, which themselves could take advantage of superposition and interference and entanglement?" Of course he then faced the question: supposing we built such a

computer, what would it be useful for? At that time, he was really only able to suggest one answer to that question: namely, it would be good for simulating quantum mechanics itself! If and when quantum computers really become practical, that's probably still the most important application we know for them. In any case, no one knew at the time whether quantum computers would be useful for "classical" tasks as well: that's a different, harder question, which will occupy us for much of the rest of the course.

We already have all the tools we need to discuss quantum computers.

The basic picture of a quantum computer is that it's just a quantum circuit, but we're jumping from working with 1, 2, or 3 qubits at a time to n qubits—where n could be, say, a million or more. You apply a sequence of gates on these qubits, each gate acting on just a few qubits (say, 1 or 2 of them), then measure some or all of the qubits.



Let's address a few conceptual questions before proceeding further:

1. Will we able to solve any problem on a quantum computer that literally *can't* be solved on a classical computer, regardless of resources?

It's easy to see that the answer is no. Using a classical computer, one can always simulate a quantum computer with *n* qubits, by just explicitly storing the vector of 2^n amplitudes (to some suitable precision, enough to approximate the final probabilities), and updating the vector whenever a unitary transformation gets applied. For this reason, we see that a quantum computer could "only, at most" achieve an exponential speedup over a classical computer: it could "only" falsify the Extended Church-Turing Thesis, rather than the original Church-Turing Thesis. Quantum computers might change what's *computable in polynomial time*—how drastically, we don't yet know—but they're not going to change what's computable at all, by letting us solve the halting problem or anything of that kind.

2. Why does each gate act on only a few qubits? Where is this assumption coming from?

It's similar to how classical computers don't have gates act on arbitrarily large numbers of bits, and instead use small gates like AND, OR, and NOT to build up complex circuits. The laws of physics provide us with *local* interactions—one particle colliding with another one, two currents flowing into a transistor, etc.—and it's up to us to string together those local interactions into something more complicated.

In the quantum case, you could imagine a giant unitary matrix U, which takes n qubits, interprets them as encoding an instance of the 3SAT problem (or some other NP-complete problem), and then cNOTs the answer (1 for yes, 0 for no) into an $(n+1)^{\text{st}}$ qubit. That U formally exists, is formally allowed by the rules of quantum mechanics. OK, but how would you go about actually implementing it? Well, you'd need to build it up out of smaller components—say, components that act on only a few qubits at a time.

It turns out that it's possible to implement any unitary U you want, using exponentially many simple components (we'll say more about that later in the lecture). The question that will interest us, in quantum computing theory, is which U's can be implemented using only polynomially many simple components. Those are the U's that we'll consider "feasible" or "efficient."

3. What is the role of interference in quantum computing?

Quantum amplitudes can cancel each other out; that's the most important way in which they differ from classical probabilities. The goal, in quantum computing, is always to choreograph a pattern of interference such that, for each wrong answer, some of the contributions to its amplitude are positive and others are negative (or, they point every which way in the complex plane), so on the whole they *interfere destructively* and cancel each other out. Meanwhile, the contributions to the right answer's amplitude should *interfere constructively* (say, by being all positive or negative). If we can arrange that, then when we measure, the right answer will be observed with high probability.

Note that, if it weren't for interference, then we might as well have just used a classical computer with a random-number generator, and saved the effort of building a quantum computer. In that sense, all quantum-computational advantage relies on interference.

4. What is the role of entanglement in quantum computing?

In some sense, we can develop the entire theory of quantum computing without ever talking about entanglement. However, pretty much any time we're doing an interesting quantum computation, entanglement *is* something that will be there, "along for the ride." The reason for this is simply that an *un*entangled pure state of *n* qubits can always be written as

$$(\alpha_1|0\rangle + \beta_1|1\rangle) \otimes (\alpha_2|0\rangle + \beta_2|1\rangle) \otimes \cdots \otimes (\alpha_n|0\rangle + \beta_n|1\rangle).$$

Specifying such a state requires only $2n$ amplitudes, so we can store it efficiently. Thus, if for some reason the (pure) state of our quantum computer never had any entanglement in it, then the computer would be easy to simulate classically, and would be unable to achieve any speedup.

By contrast, a general entangled state of n qubits, $\sum_{x \in \{0,1\}^n} \alpha_x |x\rangle$, requires 2^n amplitudes to specify.

It very quickly becomes hopelessly intractable to store and manipulate all these amplitudes on a classical computer.

With 300 qubits, we already have more amplitudes to deal with than there are atoms in the observable universe.

Just to recap: it's a theorem, which we won't prove in this class, that *any* unitary transformation on *any* number of qubits n can be decomposed as a product of 1- and 2-qubit gates. However, if you just run the decomposition blindly, it will produce a quantum circuit with something like 4^n gates—just like, if you use the method of truth tables to build a circuit to compute some arbitrary Boolean function $f: \{0,1\}^n \rightarrow \{0,1\}$, you'll get something with about 2^n AND, OR, and NOT gates. Just like in the classical world, our real interest is in which Boolean functions can be *efficiently* computed—say, using a polynomial number of AND, OR, and NOT gates—so too in the quantum world, our real interest is in which n-qubit unitary transformations can be realized using only polynomially many 1- and 2-qubit gates.

But that being so, it behooves us to ask: is it possible that *all* n-qubit unitaries U can be realized by polynomial-size circuits?

The answer turns out to be no: in fact, just like "almost all" Boolean functions require exponentially large circuits to compute them, so too "almost all" unitaries require exponentially large quantum circuits. As in the classical case, the way to prove this is using a...

Counting Argument...

something that goes back to Claude Shannon in 1949. Shannon observed that almost every n-bit Boolean function requires a circuit of at least $\sim \frac{2^n}{n}$ AND, OR, and NOT gates to compute it. The reason for this, in one sentence, is that there are too many Boolean functions, and not enough small circuits! In more detail, there are 2^{2^n} different Boolean functions $f: \{0,1\}^n \rightarrow \{0,1\}$, but only $(n+T)^{O(T)}$ different circuits that take n inputs and that have at most T gates. Since each circuit can only compute one function, we can simply set the two expressions equal and solve to find that almost every function must require a circuit of nearly 2^n size.

Strikingly, and famously, this argument doesn't give us a single *example* of a hard-to-compute Boolean function. It merely tells us that such functions must exist, and be ubiquitous!

We can use a similar sort of argument in the quantum case—although, since $2^n \times 2^n$ unitary matrices form a continuous manifold, it's easier to talk in terms of dimensionality rather than cardinality. For simplicity, let's even restrict attention to those $2^n \times 2^n$ unitary matrices that are *diagonal*. Even then, specifying such a matrix clearly requires us to specify 2^n independent complex numbers of norm 1 (or $2^n - 1$, if we ignore global phase). By contrast, a quantum circuit with T gates, each acting on at most 2 qubits, can be specified using only $O(T)$ continuous parameters (plus some discrete choices about where to apply the gates, which won't affect the argument).

So we have a 2^n-dimensional manifold in the one case, and a union of $O(T)$-dimensional manifolds in the other. Clearly, for one to cover the other, we need T to grow at least like $\sim 2^n$. Hence there exist ≈-qubit unitary transformations U that require exponentially many gates to implement. In fact, "almost all" U's (now in the technical, measure-theory sense of 100% of them) have this property.

You might complain that we've only showed that exponentially many gates are needed to implement most n-qubit unitary transformations *exactly*. What about approximately implementing them—that is, implementing them up to some small error ε (say, in each entry)? In fact, though, a little algebraic geometry (which we won't go into here) is enough to show that exponentially many gates are needed to *approximate* most n-qubit unitaries as well, or even most n-qubit diagonal unitary transformations.

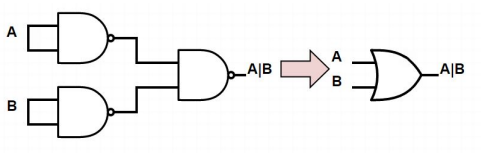
Once again, these arguments don't give us a single *example* of a hard-to-implement unitary transformation. They just tell us that that's the norm, that the easy-to-implement unitaries are the rare exceptions! Yet, rare though they might be, the subset of unitaries that are easy (i.e., that can be implemented by polynomial-size quantum circuits) are the main ones that will interest us in quantum computing.

Up till now, we've assumed that *any* 1- and 2-qubit gates are available, but is that assumption realistic? Is it necessary? This brings us to our next topic...

Universal Gate Sets

In classical computing, you're probably familiar with logic gates like AND, OR, NOT, NAND, etc. A (classical) gate set is called *universal* if, by stringing together enough gates from the set, you can express any Boolean function on any number of bits.

For example, the NAND gate by itself is universal. The diagram on the right shows how you'd construct an OR gate out of NANDs. You can also work out how to construct an AND gate and a NOT gate, and from there you can get anything else.



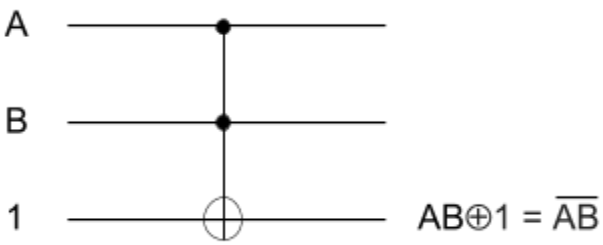
By contrast, the set {AND,OR} is *not* universal, because it can only express monotone Boolean functions: that is, changing an input bit from 0 to 1 can never change an output bit from 1 to 0. Likewise, the set {NOT,XOR} is not universal, because it can only express Boolean functions that are linear or affine mod 2. So, while "most" gate sets are universal, being universal isn't completely automatic.

Next let's discuss *classical reversible gates* (that is, reversible mappings from k-bit strings to k-bit strings), where we'll find that something similar happens. We call a set of reversible gates universal if, by composing enough of them, you can express any reversible transformation on any number of bits n.

Here we allow the use of "ancilla bits" (that is, bits other than the n being acted on), as long as the ancilla bits are returned to their initial states by the end of the computation. The use of ancilla bits is provably necessary for universality in this setting.

The most famous example of a universal reversible gate—the reversible analogue of the NAND gate, if you like—is called the **Toffoli gate**. The Toffoli gate, also known as controlled-controlled-NOT, is a 3-bit gate that flips the third bit *if and only if* the first two bits are both set to 1.

To show that Toffoli is at least capable of universal computation, we construct a NAND gate out of one (in the diagram on the right). Because Toffoli can simulate NAND, it can also simulate any ordinary (non-reversible) Boolean circuit.



The third bit ends up as 0 if only if both A and B are both 1, meaning that the third bit gets flipped. Thus, we've got a NAND gate.

Note that this argument does *not* yet establish that Toffoli is a universal reversible gate, in the sense we defined above—because for that we need to implement all possible reversible transformations, not just compute all Boolean functions. However, a more careful argument, which we'll give later in the course, shows that Toffoli really is universal in that stronger sense as well.

A good example of a gate that separates the two kinds of universality is the so-called **Fredkin** gate, which is also called **Controlled-SWAP** or **CSWAP**. This is a 3-bit gate that swaps the second and third bits, if and only if the first bit is set to 1. Just like the Toffoli gate, the Fredkin gate can simulate a NAND gate (exercise to see how), and is therefore capable of universal computation. However, Fredkin is *not* universal in the stronger sense, because it can never change the total number of 1's in the input string (the so-called *Hamming weight*). For example, there's no way, by composing any number of Fredkin gates, to map the string 000 to 111. A gate with this property, of always preserving the Hamming weight, is called *conservative*.

There are also reversible gates that are not even capable, on their own, of universal computation. An important example is the Controlled-NOT or CNOT gate, which we saw earlier. By composing CNOT gates, we can only express Boolean functions that are affine mod 2: for example, we can never express AND or OR. More generally, and in contrast to the irreversible case, it turns out that *no* 2-bit classical reversible gate is capable of universal computation. For universality, we need reversible gates (such as Toffoli or Fredkin) that act on at least 3 bits.

It's worth noting that any classical reversible gate can *also* be used as a quantum gate (i.e., it's unitary). So in particular, Toffoli can be used as a quantum gate. So we see that, if nothing else, a quantum circuit can compute any function that a classical circuit of similar size can compute: we simply need to transform the classical circuit into one made of Toffoli gates.

We're now ready to talk about universal *quantum* gate sets.

We'll call a set S of quantum gates *universal* if, by composing gates from S, you can approximate any unitary transformation on any number of qubits to any desired precision. Note that, if S is finite, then approximation is all we can hope for, because there are uncountably many unitary transformations, but only a countable infinity of quantum circuits built out of S-gates.

Just like with classical reversible gates, there are weaker kinds of universality for quantum gate sets that are often enough in practice. That is, even if gates *can't* be used to approximate an arbitrary unitary to any desired precision, they'll often anyway suffice for universal quantum computation. We'll see examples of this soon.

First, though, let's list some of the ways a quantum gate set could be limited, in such a way that it *fails* to be universal. We'll discuss such four ways: three of them obvious and one of them not.

1. Your gate set doesn't create interference/superposition

Example: The set {cNOT} can only map computational basis states, like $|10\rangle$, to other computational basis states, like $|11\rangle$. It can maintain existing superpositions, but it can't *create* a superposition of basis states where there wasn't one already.

2. Your gate set can create superpositions, but not entanglement

Example: The set {Hadamard} can map $|0\rangle$ to $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$, thereby creating a superposition. But it should be obvious that the Hadamard gate can't map a product state to an entangled state, since it acts on only one qubit. In fact, any quantum circuit made of Hadamard gates—or any 1-qubit gates, for that matter—will just act independently on each of the n qubits, so it will be trivial to simulate classically.

3. Your gate set only has real gates

Example: The set {cNOT, Hadamard} is getting closer to universality, as it's capable of creating entangled superposition states. But it's still not there, as can be seen from the fact that the cNOT and Hadamard matrices have real entries only. So composing them could never give us a unitary transformation like

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

4. Your gate set is "contained in the stabilizer set"

This is the non-obvious case. Later in the course, we'll explain stabilizer gates in more detail, as well as their central importance for quantum error correction. For now, though, the *stabilizer gates* are the following three quantum gates: cNOT, Hadamard, and

Phase $= P = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$

These gates pass every test for universality that we saw before: they can generate superposition, *and* entanglement, *and* amplitudes that aren't real. Furthermore, they're enough to demonstrate many (though not all) of the quantum phenomena that we've seen in this course, such as teleportation and superdense coding.

Nevertheless, it turns out that the particular set

{cNOT, Hadamard, P}

is not universal. Indeed, a famous result called the **Gottesman-Knill Theorem** shows that these gates generate only a discrete subset of unitary transformations—and that furthermore, any circuit made of stabilizer gates, and applied to the initial state $|0 \ldots 0\rangle$, can be simulated in polynomial time on a classical computer. Thus, despite containing an interesting subset of quantum mechanics, these gates still aren't enough to realize exponential quantum speedups.

Are there any other ways for a set of quantum gates, acting on qubits, to fail to be universal?

That's currently an open question! It's one of Prof. Aaronson's personal favorites. Not many people in the field care about this particular question, since "we have lots of universal gate sets that work, so just roll with it," but it would be nice to know, and you should go solve it anyway.

So then which quantum gate sets *are* universal?

It turns out that, in the stabilizer set {cNOT,Hadamard,P}, if you swap out the Hadamard gate for nearly anything else, then the set becomes universal.

So for example, {cNOT, $R_{\frac{\pi}{8}} = \begin{bmatrix} \cos\frac{\pi}{8} & -\sin\frac{\pi}{8} \\ \sin\frac{\pi}{8} & \cos\frac{\pi}{8} \end{bmatrix}$, P } is universal.

Also, {Toffoli, Hadamard, P} is universal, by a 2002 result of Yaoyun Shi.

Also, if you just pick a 2-qubit gate uniformly at random, then it's known to have a 100% chance of being universal. With universality, the whole difficulty comes from the remaining 0% of gates!

Above, we listed bad cases—ways of failing to be universal—but there are also general criteria such that if your gate set meets them then it *is* universal. We won't cover this, but see (for example) the paper of Shi, or earlier literature on the subject, for examples of such criteria.

So far, in discussing universal gate sets, we've swept an important question under the rug. Namely, a universal gate set lets us approximate any unitary to any desired accuracy ε—but how does the number of gates needed with ε? If, for example, the number scaled like $2^{1/\varepsilon}$, then our gate set would be

"universal" in only the most theoretical sense. Fortunately, there's a central result in quantum computing theory, called the...

Solovay-Kitaev Theorem

which assures us that this never happens! In more detail, call a gate set S *closed under inverses* if, whenever a gate G is in S, the inverse gate G^{-1} is also in S. Then the Solovay-Kitaev Theorem says that, using any universal gate set G that's closed under inverses, we can approximate any unitary on n qubits to within precision ε (say, entrywise precision) using only $O(4^n \ \mathrm{polylog}\frac{1}{\epsilon})$ gates. In other words, if we treat n as fixed—say, if we're just trying to emulate a gate from one universal set using gates from a different universal set—then the complexity scales only like some power of $\log\left(\frac{1}{\varepsilon}\right)$. It shows that *all* universal gate sets, or at least the ones closed under inverses, "fill in the space of all unitary transformations reasonably quickly." Furthermore, the gate sequences that achieve the Solovay-Kitaev bound can actually be found by a reasonably fast algorithm.

Whether the closed-under-inverses condition can be removed remains an unsolved problem to this day.

With the original proofs of Solovay-Kitaev, from the late 1990s, the number of gates needed grew like $\log^{3.97}\left(\frac{1}{\varepsilon}\right)$. However, more recently it's been shown that, at least if we use special universal gate sets arising from algebra and number theory, we can get the number of gates to grow only like $\log\left(\frac{1}{\varepsilon}\right)$ —which is not only much more practical, but also the best one could possibly hope for on information-theoretic grounds.

The proof of Solovay-Kitaev is beyond the scope of this course, but it's contained in many quantum computing textbooks (including Nielsen & Chuang), and is something you should know is out there.