Lecture 28, Tues May 2: Stabilizer Formalism

Today we'll see a beautiful formalism that was originally invented to describe quantum-error correcting codes, but now plays many different roles in quantum computation. First, some definitions:

Stabilizer Gates

are the gates CNOT, Hadamard, and $P = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$ (also called the "phase gate")

Stabilizer Circuits

are quantum circuits made entirely of stabilizer gates

Stabilizer States

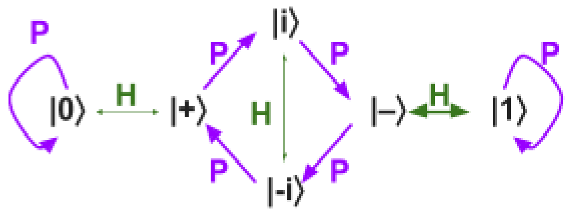
are states that a stabilizer circuit can generate, starting from $|00\ldots0\rangle$

We briefly met stabilizer gates earlier in the course, when we discussed universal quantum gate sets, and needed to include a warning that the set S={CNOT,Hadamard,P} is *not* universal. At first, this failure of universality might be surprising. After all, the set S seems to have everything: the Hadamard gate can create superpositions, CNOT acts on two qubits and (in concert with Hadamard) can create complicated entangled states, and P can even add complex phases.

What's more, many of the weird quantum effects and protocols that we saw in this course can be demonstrated entirely using stabilizer gates. Examples include superdense coding, quantum teleportation, BB84 quantum key distribution, Wiesner's quantum money, the Deutsch-Jozsa and Bernstein-Vazirani algorithms, and the Shor 9-qubit code.

So then what prevents S from being universal? Well, if you try playing around with the CNOT, Hadamard, and Phase gates, you'll notice that you tend to reach certain discrete states, but never anything between them. You'll also notice that, whenever you can create an n-qubit superposition that assigns nonzero amplitudes to the strings in some set $A \in \{0,1\}^n$, it's always an *equal* superposition over A (possibly with $+1,-1,+i,-i$ phases), and furthermore A is always an affine subspace of F_2^n (so in particular, $|A|$ is always a power of 2).

With only 1 qubit, the H and P gates can only get us to 6 states in total (ignoring global phases), via the reachability diagram shown on the right. These 6 states--$|0\rangle,|1\rangle,|+\rangle,|-\rangle,|i\rangle,|-i\rangle$--are the 1-qubit stabilizer states.



What about with two qubits?

Now you can reach some more interesting states, like $\frac{|00\rangle + i\,|11\rangle}{\sqrt{2}}$ or $\frac{|01\rangle - i\,|10\rangle}{\sqrt{2}}$. But these always follow certain patterns, as mentioned above. For example, they're always equal superpositions over power-of-2 numbers of strings, and a measurement of a given qubit in the $\{|0\rangle,|1\rangle\}$ basis always produces either

(1) always $|0\rangle$,

(2) always $|1\rangle$, or
(3) $|0\rangle$ and $|1\rangle$ with equal probabilities.

So what gives?
To answer that question, it will help to define a few concepts.

We say that a unitary U stabilizes a pure state $|\Psi\rangle$ if $U|\Psi\rangle = |\Psi\rangle$.

In other words, if $|\Psi\rangle$ is an eigenstate of U with eigenvalue +1. Crucially, global phase matters here! If $U|\Psi\rangle = -|\Psi\rangle$, then U does not stabilize $|\Psi\rangle$.

Notice that if U and V both stabilize $|\Psi\rangle$, then any product of them, like UV or VU, *also* stabilizes $|\Psi\rangle$, as do their inverses U^{-1} and V^{-1}. Also, the identity matrix, I, stabilizes everything.

This means that the set the unitaries that stabilize $|\Psi\rangle$ form a *group* under multiplication.

We already know that unitaries have inverses and are associative.

The next ingredient we need is the **Pauli Matrices**.

These four matrices come up a lot in quantum physics. They are:

$$\mathbf{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \mathbf{X} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \mathbf{Y} = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad \mathbf{Z} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Notice that these matrices match up with the errors we need to worry about in quantum error-correction:

No error $\mathbf{I}|1\rangle = |1\rangle$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$	Bit flip $\mathbf{X}|1\rangle = |0\rangle$	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$
Phase flip $\mathbf{Z}|1\rangle = -|1\rangle$	$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ -1 \end{pmatrix}$	and Both $\mathbf{Y}|1\rangle = -i|0\rangle$	$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} -i \\ 0 \end{pmatrix}$

That's not a coincidence!

The Pauli matrices satisfy several beautiful identities:

$$X^2 = Y^2 = Z^2 = I \qquad \begin{array}{ll} XY = iZ & YX = -iZ \\ YZ = iX & ZY = -iX \\ ZX = iY & XZ = -iY \end{array}$$

If you've seen the quaternions, you might recall that they're defined using the same kinds of relations.
This is also not a coincidence! Nothing is a coincidence in math!

Also, all four Pauli matrices are both unitary and Hermitian.

So what does each Pauli matrix stabilize?

I stabilizes everything

–I stabilizes nothing Remember: global phase matters, so $-I|\Psi\rangle \neq |\Psi\rangle$.

X stabilizes $|+\rangle$
–X stabilizes $|-\rangle$
Z stabilizes $|0\rangle$
–Z stabilizes $|1\rangle$
Y stabilizes $|i\rangle$
–Y stabilizes $|-i\rangle$

So each of the six 1-qubit stabilizer states corresponds to a Pauli matrix that stabilizes it.

Next, given an n-qubit pure state $|\Psi\rangle$, we define $|\Psi\rangle$'s stabilizer group as:
The group of all tensor products of Pauli matrices that stabilize $|\Psi\rangle$.

We know this is a group since being a Pauli matrix is closed under multiplication, and so is stabilizing $|\Psi\rangle$. As you can check, stabilizer groups have the additional interesting property of being abelian.

For example, the stabilizer group of $|0\rangle$ is { I, Z } closed because $Z^2 = I$
while the stabilizer group of $|+\rangle$ is { I, X }

The stabilizer group of $|0\rangle\otimes|+\rangle$ will be the Cartesian product of those two groups:
{ I⊗I, I⊗X, Z⊗I, Z⊗X }
as a convention, from now on we omit the ⊗'s, so that for example the above is just { II, IX, ZI, ZX}

For a slightly more interesting example, what's the stabilizer group of a Bell pair?
We know XX is in it because $\frac{X|0\rangle\otimes X|0\rangle + X|1\rangle\otimes X|1\rangle}{\sqrt{2}} = \frac{|11\rangle + |00\rangle}{\sqrt{2}} = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$.
A similar argument can be made for –YY.
We can get another element by doing component-wise multiplication: XX · –YY = $-(iZ)(iZ)$ = ZZ
So the stabilizer group of $\frac{|00\rangle + |11\rangle}{\sqrt{2}}$ contains { II, XX, –YY, ZZ }. And you can check that it doesn't contain anything else.
You can similarly compute the stabilizer group of $\frac{|00\rangle - |11\rangle}{\sqrt{2}}$ to be { II, –XX, YY, ZZ }.

Now, here's an amazing fact, which we won't give a proof of:

The n-qubit stabilizer states are exactly the n-qubit states that have a stabilizer group of size 2^n.

So the 1-qubit stabilizer states are those states with a 2-element stabilizer group, the 2-qubit stabilizer states are those states with a 4-element stabilizer group, and so on.

This is a completely different characterization of stabilizer states, a structural one. It makes no mention of stabilizer circuits, but tells us something about the invariant that stabilizer circuits are preserving.

OK, so suppose we have an n-qubit stabilizer state, which (by the above) has a 2^n-element stabilizer group G. Then here's the next thing we might want to know:

How can we succinctly specify *G*? Does *G* always have a small generating set--that is, a few elements from which we can get all the others by multiplication?

While we again won't prove it, the answer turns out to be yes. Given any n-qubit stabilizer state, its stabilizer group is always generated by only *n* elements (i.e., ± tensor products of Pauli matrices). So, to specify a stabilizer group (and hence, a stabilizer state), you only need to specify *n* such generators.

Let's see an example. To specify the Bell pair, which has stabilizer group { II, XX, –YY, ZZ }, it's enough to give the following generating set:

(X X)
(Z Z)

Or we could also give a different generating set, like

(X X)
-(Y Y)

Now we come to a crucial point:
How many bits does it take to store such a generating set in your computer?
Well, there are *n* generators, and each one takes $2n+1$ bits to specify: 2 bits for each of the *n* Pauli matrices, plus 1 additional bit for the ± sign. So the total number of bits is

$$n(2n+1) = 2n^2 + n = O(n^2).$$

Naïvely writing out the entire amplitude vector, or the entire stabilizer group, would have taken $\sim 2^n$ bits, so we've gotten an exponential savings. We're already starting to see the power of the stabilizer formalism.

But that power turns out to go much further. Around 1998, Daniel Gottesman and Manny Knill proved the...

Gottesman-Knill Theorem

which says that there's a polynomial-time classical algorithm to simulate any stabilizer circuit that acts on a stabilizer initial state like $|00\ldots0\rangle$.

Here, "simulate" means pretty much anything you could ask for: you can compute the probability of any possible sequence of measurement outcomes, or you can *simulate* the measurement outcomes if given access to a random bit source.

A more negative interpretation is:
stabilizer states and gates, by themselves, are useless for producing superpolynomial quantum speedups.

So, how does the classical simulation work? Just by keeping track, at each point in time, of a list of generators for the current state's stabilizer group! And updating the list whenever a CNOT, Hadamard, Phase, or measurement gate is applied.

Almost the only time that Professor Aaronson (being a theorist) ever wrote code that other people actually used, was when he did a project in grad school for a Computer Architecture course.

He wrote a fast simulator for stabilizer circuits called CHP, which could handle thousands of qubits on a normal laptop (limited only by the available RAM). He was only trying to pass the class, but the challenge of actually implementing the Gottesman-Knill algorithm in an optimized way led to the discovery of an *even faster* classical algorithm for simulating stabilizer circuits, so Aaronson ended up publishing a paper with Gottesman about this.

Truth be told, this project had very little to do with Computer Architecture. He's still not sure why the professor accepted it.

So how does the Gottesman-Knill algorithm work?

For simplicity, let's assume the initial state is |00…0⟩. Then the first step is to find a stabilizer representation (that is, a list of generators) for |00…0⟩.

We know the stabilizer group contains II…I, but we won't put that into the generating set: it's implied. Since |0⟩ is a +1 eigenstate of Z, you can check that the following generating set works:

ZIII…I
IZII…I
IIZI…I
:
IIII…Z

For purposes of the algorithm, it's useful to write these lists of generators in a slightly different way:

Tableau Representation

Here we'll keep track of two $n\times n$ matrices of 1's and 0's (as well as n signs). The two matrices can be combined entrywise to produce an {I,X,Y,Z} matrix like the one above. We call them:

The X Matrix and The Z Matrix

$$\begin{array}{llclll}
+(& 0\,0\,0\,0 & \mid & 1\,0\,0\,0 &) & \\
+(& 0\,0\,0\,0 & \mid & 0\,1\,0\,0 &) & \leftarrow \text{Each row represents one generator of the stabilizer group} \\
+(& 0\,0\,0\,0 & \mid & 0\,0\,1\,0 &) & \\
+(& 0\,0\,0\,0 & \mid & 0\,0\,0\,1 &) & \\
 & \uparrow & & \uparrow & & \\
 & \text{1 if X or Y} & & \text{1 if Z or Y} & & \\
 & \text{0 otherwise} & & \text{0 otherwise} & &
\end{array}$$

Thus, the first row of the above tableau represents the generator +ZIII, the second +IZII, the third +IIZI, and the fourth +IIIZ. So this is just another way to represent the generating set {ZIII, IZII, IIZI, IIIZ} for the state |0000⟩.

We're now going to provide rules for updating this tableau representation whenever a CNOT, Hadamard, or phase gate is applied. We won't prove that the rules are correct, but you should examine them one by one and see if you can convince yourself.

We're also going to cheat a little. Keeping track of the +'s and –'s is tricky and not particularly illuminating, so we'll just ignore them. What do we lose by ignoring them? Well, whenever measuring a qubit has a definite outcome (either |0⟩ or |1⟩), we need the +'s and -'s to figure out *which* of the two it is. On the other hand, if we only want to know whether measuring a qubit will give a definite outcome or a random outcome (and not *which* definite outcome, in the former case), then we can ignore the signs.

So what are the rules?

The gates available to us are CNOT, H, and P, so we need to figure out how to update the tableau for each.

- To apply H to the *i*th qubit:
    - Swap the *i*th column of the X matrix with the *i*th column of the Z matrix.

This should be pretty intuitive: the whole point of the Hadamard gate is to "swap the X and Z bases."

- To apply P to the *i*th qubit:
    - Bitwise XOR the *i*th column of the X matrix into the *i*th column of the Z matrix.

Note that P has no effect on the tableau representation of |00…0⟩.
Coincidence? I think not.

- To apply CNOT from the *i*th qubit to the *j*th qubit:
    - Bitwise XOR the *i*th column of the X matrix into the *j*th column of the X matrix.

That seems reasonable enough, *but* ... remember how a CNOT from *i* to *j* is equivalent, when viewed in the Hadamard basis, to a CNOT from *j* to *i*?
That means we also have to…

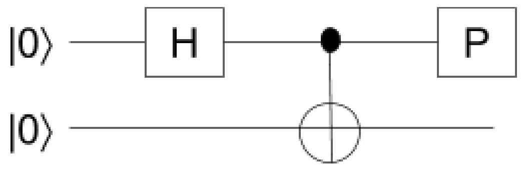
-
    - Bitwise XOR the *j*th column of the Z matrix into the *i*th column of the Z matrix.

Finally, whenever the *i*th qubit is measured in the {|0⟩,|1⟩} basis: the measurement will have a determinate outcome *if and only if* the *i*th column of the X matrix is all 0's. (Can you figure out why?)

There are also rules for updating the tableau in the case that the measurement outcome is not determinate, but we won't cover them here.

Here's another cool fact: in our state, the number of basis states that have nonzero amplitudes is just 2^k, where *k* is the rank of the X matrix.

In the above example, rank(X) = 0, corresponding to the fact that our "superposition" only contains a single basis state, namely |0000⟩.



Let's test this out, by keeping track of the tableau for the circuit on the right. We start with

(0 0 | 1 0)
(0 0 | 0 1)

After the Hadamard we get (swap 1st columns of X and Z)

(1 0 | 0 0)
(0 0 | 0 1)

You could convert this back into Pauli notation by saying that the current state is the one generated by +XI and +IZ.
That makes sense, since those do indeed generate the stabilizer group for $|0\rangle^{\otimes}|+\rangle$

After the CNOT: (In X: XOR 1st column into 2nd. In Z: XOR 2nd column into 1st)

$(1\ 1 \mid 0\ 0)$ This is (X X), the stabilizer generator for a Bell pair.

$(0\ 0 \mid 1\ 1)$ (Y Y)

After the phase gate: (XOR 1st column of X into 1st column of Z)

$(1\ 1 \mid 1\ 0)$

$(0\ 0 \mid 1\ 1)$

This corresponds to the state $\frac{|00\rangle + i|11\rangle}{\sqrt{2}}$.

A <u>stabilizer code</u> is a quantum error-correcting code in which the encoding and decoding (at least if there are no errors!) can be done entirely by stabilizer circuits. In particular, this means that all the code states are stabilizer states.

In quantum computing research, most of the error-correcting codes that have been seriously considered are stabilizer codes. The reason for this is similar to why linear codes play such a central role in classical error correction: namely,

(1) it makes everything *much* easier to calculate and reason about, and
(2) by insisting on it, we don't seem to give up any of the error-correcting properties we want.

As a result, the stabilizer formalism is the lingua franca of quantum error-correction; it's completely indispensable there.

To take an example: with Shor's 9-qubit code, we were dealing with states of the form

$$\left(\frac{|000\rangle \pm |111\rangle}{\sqrt{2}}\right)^{\otimes 3}$$

We claim that a generating set for the above state's stabilizer group is as follows:

$$\begin{array}{cccccccccccc}
\{ & Z & Z & I & I & I & I & I & I & I, & \\
 & I & Z & Z & I & I & I & I & I & I, & \\
 & I & I & I & Z & Z & I & I & I & I, & \\
 & I & I & I & I & Z & Z & I & I & I, & \\
 & I & I & I & I & I & I & Z & Z & I, & \\
 & I & I & I & I & I & I & I & Z & Z, & \\
 & X & X & X & X & X & X & I & I & I, & \\
 & I & I & I & X & X & X & X & X & X, & \\
\pm & X & X & X & X & X & X & X & X & X & \}
\end{array}$$

The last line can have either a + or –, encoding $|\bar{0}\rangle$ or $|\bar{1}\rangle$ respectively.

Why are the above elements in the stabilizer group? Well, phase-flips applied to any pair of qubits in the same block cancel each other out. Bit-flips also take us back to where we started, except possibly with the addition of a global -1 phase.

You then just need to check that these 9 elements are linearly independent of each other, meaning that there aren't any more to be found.

Now that we know the stabilizer formalism, we're finally ready to see an "optimal" (5-qubit) code for detecting and correcting an error in any one qubit. The codeword states would be a mess if we wrote them out explicitly--superpositions over 32 different 5-bit strings! But everything is much more compact if we use the stabilizer formalism. Here's the code:

{ XZZXI,
IXZZX,
XIXZZ,
ZXIXZ,
± XXXXX }

Once again, the sign on the last generator is + if we want the logical $|\bar{0}\rangle$ state, or - if we want the logical $|\bar{1}\rangle$ state.

One can check (we won't prove it here) that this code can indeed correct either a bit-flip or a phase-flip error on any one of the five qubits.

To conclude this lecture, let's say a tiny bit about doing actual quantum *computations* on qubits that are encoded using stabilizer codes.

Thus, suppose we have *n* logical qubits, each encoded with a stabilizer code, and we want to apply a gate to one or two of the logical qubits. The "obvious" way to do this would be:

1. Decode the qubits.
2. Apply the desired gate to the "bare," unencoded qubits.
3. Re-encode the result.

But doing all that is expensive, and creates lots of new opportunities for error! (E.g., while the qubits are unencoded, there's "nothing to protect them" from decoherence.)

So it would be awesome if we had a code where applying gates to encoded qubits was hardly more complicated than applying them to unencoded qubits. This motivates the following definition:

The gate G is transversal for the code C, if in order to apply G to qubits encoded using C, all you need to do is:

- Apply G to the first qubits of the codewords
- Apply G to the second qubits of the codewords
- etc.

So for example, the Hadamard gate is transversal if you can Hadamard a logical qubit by just separately Hadamarding each physical qubit in the codeword.

You should check that the Hadamard gate is transversal for Shor's 9-qubit code.

It turns out that there are quantum error-correcting codes for which the CNOT, Hadamard, and Phase gates are *all* transversal. Thus, if you use one of these codes, then applying *any* stabilizer circuit to the encoded qubits is extremely cheap and easy.

Unfortunately, we already saw that the stabilizer gates are non-universal---and there's a theorem that says that non-stabilizer gates *can't* all be transversal.

This means that, if we want universal quantum computer, we're going to need non-stabilizer gates like Toffoli or $R_{\pi/8}$ that *can't* be implemented transversally, but only via sequences of gates that are much more expensive.

So the quantum computer engineers tend to adopt a worldview wherein stabilizer gates are "free"---they're so cheap to implement that you might as well not even count them---and the "complexity" of a quantum circuit equals the number of non-stabilizer gates. The non-stabilizer gates are so much more expensive that they completely dominate the running time.

In practice, a lot of quantum computer engineering has boiled down to designing improved methods for getting non-stabilizer gates into a circuit. There are various tricks, a famous example being Magic State Distillation.

The idea there is that, if you can just produce certain non-stabilizer states like $\cos(\pi/8)|0\rangle + \sin(\pi/8)|1\rangle$ -- those are the "magic states" -- then applying stabilizer operations to those states, together with measurements (and adapting based on the outcome of the measurements), is enough to *simulate* the effect of non-stabilizer gates. In other words, with help from magic states, stabilizer operations can break out of the Gottesman-Knill prison and get all the way up to universal quantum computation. On the other hand, actually realizing this idea seems to require building a quantum computer where the *overwhelming majority* of the work would happen in "magic state factories," with the actual quantum computation on the magic states almost an afterthought.

There's a different way to understand the importance of non-stabilizer states for quantum computation. The paper by Aaronson and Gottesman from 2004, mentioned earlier, also proved the following result:

> Suppose we have a quantum circuit on n qubits, which contains mostly stabilizer gates---say, $n^{O(1)}$ of them---but also a small number T of non-stabilizer gates. Then there's a classical algorithm to simulate the circuit in time that's polynomial in n and exponential in T.

This tells us that, if we want an exponential quantum speedup, then not only do we need non-stabilizer gates in our circuit, we need a polynomial *number* of such gates.