Surviving as a Quantum Computer in a Classical World

Daniel Gottesman

March 5, 2024

Contents

Part I

Quantum Error Correcting Codes

Chapter 1

Know Your Enemy: Quantum Errors

In this book, you will learn how to seek out and destroy errors on quantum states. Quantum errors are nasty, unforgiving things. If you don't know what you are doing, a single misstep can result in the destruction of irreplaceable quantum information. Of course, nobody's perfect, and in part II, you will learn how to handle your own fallibility. (Short answer: very carefully.) For now, we will assume you don't make any mistakes, but that doesn't mean things will be easy. The quantum errors are still out there, hungering to consume quantum information. You need to get the errors before they get you.

The key to most error hunts is preparation. There are two forms of preparation. One is dressing your quantum information properly: that is, encoding it in an appropriate quantum error-correcting code. You will learn about that in the other chapters in part I. This chapter will focus on the other aspect of preparation: studying the habits of the errors you are hunting.

1.1 The Quantum Channel

The most common way of characterizing a source of errors in a quantum system is as a quantum channel. "Quantum channel" is a general term for an opportunity for the environment to introduce errors into your quantum system. Usually, we consider a situation as depicted in figure 1.1. Alice wants to send quantum information to Bob. Alice and Bob both have perfect quantum computers, but the connection between them is possibly imperfect: the quantum channel. (We will drop the assumption that Alice and Bob have perfect quantum computers in part II.) While the quantum information is in transit, the external world (the "environment") has a chance to interact with the system, thereby changing it in some way, introducing errors (or "noise") relative to the state that Alice sent.

A common example of a quantum channel is an optical fiber. Single photons can pass through the optical fiber, but they may be lost or altered en route. Other possibilities are sending a photon through the air, physically moving an atom with information encoded in the state of the electron or nuclear spin, successively swapping the states of neighboring qubits arranged on a line, or even using quantum teleportation to send

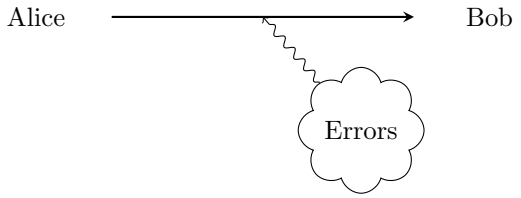


Figure 1.1: Alice, who has a perfect quantum computer, wants to send qubits to Bob, who also has a perfect quantum computer. However, the communications channel between them is *not* perfect.

a quantum state from Alice to Bob with classical communication and some sort of entangled state. This is not an exhaustive list. Indeed, any sort of communication, even a classical telephone call, can be considered as a quantum channel. If you try to send a qubit through a regular telephone connection, no amount of quantum error correction will allow you to recover the full quantum state afterwards, but that doesn't affect the telephone line's status as a quantum channel — it is simply a *very noisy* quantum channel.

Another common situation is when Bob is replaced by Alice's future self. In this case, we really want a "quantum memory": Alice wants to prepare some quantum information, go off and do other things, and then return and manipulate her stored quantum information again. If we assume that Alice's manipulations at the beginning and the end of the process are perfect, we can consider the "memory" portion, when the qubit is stored but subject to noise, as a quantum channel.

It is worth noting that the notion of a quantum channel only applies when we can look at a single communications link in isolation. That is, to have a quantum channel, the quantum state that exits the channel should only depend on the quantum state that goes into the channel. That may seem like a tautology, but it is not. Imagine that Alice has a quantum memory, and prepares a qubit to store in the memory at time 0. At time 1, she returns and fiddles some more with the stored system, then goes away again and comes back at time 2. The storage from time 0 to time 1 is a quantum channel (assuming Alice's initial preparation of the qubit is perfect), but the storage from time 1 to time 2 might not be. The problem is that the environment might remember what happened between time 0 and 1 (and more importantly, might remember something about the *state* that was stored between time 0 and 1) and use that to influence what it does to the state during the second time interval. The error now no longer depends only on what state is stored at time 1; it also depends on what state was stored at time 0. Of course, some environments have a very short memory, and in that case, it is a very good approximation to consider the time interval 1 to 2 to be independent of the time interval 0 to 1, and with that approximation, the second time interval *is* a quantum channel. When the environment has no memory, and every time interval is independent of any other non-overlapping time interval, the environment (or the error source) is called *Markovian*. When the environment does remember over time scales long enough to matter, it is a *non-Markovian* environment.

In part I, we only consider the case depicted in figure 1.1. There the environment has only one opportunity to attack the quantum information. While that opportunity may last for an extended period of time, because Alice and Bob do not do anything with the quantum information during that time, we can lump together everything the environment does to the state into a single transformation, and consider the whole communications line as a single quantum channel. The question of whether the environment is Markovian or non-Markovian then becomes moot. If we were to generalize this picture and allow noise during Alice and Bob's processing of the qubit, then the question arises again, since the environment then gets more than one shot at the quantum information, but don't worry about that situation until part II.

Now it is time to formally define a quantum channel:

Definition 1.1. A *quantum channel* is a completely positive trace-preserving (CPTP) map.

Wasn't that easy? At least, it is if you know what a CPTP map is. If not, you should refer to appendix A, where you will learn that a CPTP map is the most general physically possible transformation for an operation where the output depends only on the input, so this is the right definition. It is frequently convenient to consider the Kraus form of a CPTP map:

$$\mathcal{E}(\rho) = \sum_k A_k \rho A_k^\dagger. \tag{1.1}$$

We can think of this channel as a collection of possible errors A_k, where error A_k occurs with probability $\mathrm{tr}(A_k \rho A_k^\dagger)$. However, note that the probability of A_k occurring is not a single value but actually depends on the state ρ. Furthermore, remember that the decomposition into A_k operators is not unique and that $\sum_k A_k^\dagger A_k = I$.

Frequently, instead of dealing explicitly with a quantum channel, I will instead refer to the *set of possible errors*. One way to write this set is $\mathcal{E} = \{A_k\}$, but it is frequently convenient to rescale the errors, so more generally $\mathcal{E} = \{E_k\}$, where each $A_k = p_k E_k$ for some scalar p_k.

1.2 Single-Qubit Example Channels

1.2.1 Unitary Channels, Pauli Errors

Let us start by discussing some examples of channels acting on a single-qubit input. The simplest case is when there is only a single value of k in the Kraus decomposition:

$$\mathcal{E}(\rho) = A\rho A^\dagger. \tag{1.2}$$

Since $\sum_k A_k^\dagger A_k = I$, it follows that A is unitary. Typographically, I will usually represent a unitary channel the same way as a unitary matrix, e.g. $A(\rho)$ versus $A(|\psi\rangle)$, even though they formally act on different kinds of objects (density matrices versus state vectors). The one exception is that I will usually write the identity channel as $\mathcal{I}$, as opposed to the identity unitary I.

There are of course infinitely many unitary maps, but some are more interesting than others. One set that you will be particularly sick of by the end of this book are the Pauli matrices:

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{1.3}$$

I is, of course, the identity matrix — no error. X is probably the first thing you think of when you think of an error, a classical bit flip:

$$X|0\rangle = |1\rangle \tag{1.4}$$

$$X|1\rangle = |0\rangle. \tag{1.5}$$

However, since it is a quantum operator, it also acts sensibly on superpositions:

$$X(\alpha|0\rangle + \beta|1\rangle) = \alpha|1\rangle + \beta|0\rangle. \tag{1.6}$$

Z is the most basic kind of truly quantum error, a phase flip:

$$Z(\alpha|0\rangle + \beta|1\rangle) = \alpha|0\rangle - \beta|1\rangle. \tag{1.7}$$

Y is then just a combined bit flip and phase flip error:

$$Y = iXZ \tag{1.8}$$

$$Y(\alpha|0\rangle + \beta|1\rangle) = i\alpha|1\rangle - i\beta|0\rangle. \tag{1.9}$$

Recall that an *overall phase* — one that affects all states uniformly — has no physical significance, so Y and XZ are really the same channel. In the form presented above, all the Pauli matrices are Hermitian as well as unitary, which is sometimes useful.

In other contexts, the Pauli matrices are often written as σ_x, σ_y, and σ_z or σ_1, σ_2, and σ_3, but those are too much writing and less easy to read. I'll be using the Pauli matrices a *lot* in this book, so I'll use the more straightforward notation X, Y, and Z. In some of the earlier quantum error-correction literature, $Y = XZ$ instead of iXZ. There is not a huge difference, but I think this convention is somewhat nicer overall.

There are many more single qubit unitary errors, and some are even interesting. For instance, we can have phase rotation by an arbitrary angle:

$$R_\theta = \begin{pmatrix} e^{-i\theta} & 0 \\ 0 & e^{i\theta} \end{pmatrix} = e^{-i\theta} \begin{pmatrix} 1 & 0 \\ 0 & e^{2i\theta} \end{pmatrix}. \tag{1.10}$$

Again, we can ignore an overall phase, so R_θ is the same channel as $\mathrm{diag}(1, e^{2i\theta})$. The full set of physically distinct one-qubit unitary channels is the group $\mathsf{SU}(2)$.

In the Bloch sphere picture of the state space for a qubit, a unitary map is just a rotation of the sphere. X, Y, and Z are π rotations around the X, Y, and Z axes, as one might expect. R_θ is a rotation by angle 2θ around the Z axis. (I have defined R_θ this way in order to agree with the prevailing terminology for phase rotations, which results from talking about spin-1/2 particles.) Note that a reflection of the Bloch sphere is not a completely positive map. For instance, the transpose map is a reflection in the XZ plane, and when applied to part of an entangled state, the transpose gives something non-positive.

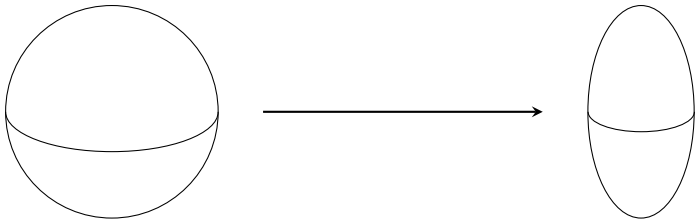


Figure 1.2: Bloch sphere transformation induced by a dephasing channel

1.2.2 Dephasing Channel

One very commonly-encountered channel is the dephasing channel.

Definition 1.2. A channel of the form

$$\mathcal{R}_p(\rho) = (1-p)\rho + pZ\rho Z^\dagger. \tag{1.11}$$

is a *dephasing channel*. When $p = 1/2$, we have the *completely dephasing channel*. We usually restrict attention to $p \leq 1/2$ since channels with $p > 1/2$ are related to channels with $p < 1/2$ by a Z operation.

The Kraus operators of a dephasing channel in this form are $\sqrt{1-p}I$ and $\sqrt{p}Z$. Since both are proportional to unitary maps, the probabilities of these two errors occurring do not depend on the input state. Thus, this channel corresponds to no error with probability $1-p$ and a phase flip with probability p. We can calculate how it acts on the density matrix in component form:

$$\mathcal{R}_p : \begin{pmatrix} a & b \\ c & d \end{pmatrix} \mapsto \begin{pmatrix} a & (1-2p)b \\ (1-2p)c & d \end{pmatrix}. \tag{1.12}$$

In other words, a dephasing channel shrinks the off-diagonal components of the density matrix. In the completely dephasing channel, the off-diagonal terms disappear completely.

An alternate Kraus decomposition is also edifying.

$$\mathcal{R}_p(\rho) = (1-2p)\rho + \frac{2p}{\pi}\int_0^\pi R_\theta \rho R_\theta^\dagger d\theta. \tag{1.13}$$

The dephasing channel is a channel for which, with probability $1-2p$, nothing happens to the state, and with probability $2p$, the phase between $|0\rangle$ and $|1\rangle$ is completely randomized. This seems to conflict with the decomposition into I and Z, where the state is left unchanged with probability $1-p$, not with probability $1-2p$. However, when the dephasing angle θ is chosen uniformly at random, there is a reasonable chance that θ is small and the state does not change very much. Of course, the probability that the angle θ is *exactly* zero is just $1-2p$, but using the magic of quantum mechanics, the small-θ cases cancel out just right so that if we break the channel up into I and Z, we find the probability of error is only p.

This example illustrates a rather disturbing principle: in quantum mechanics, the notion of "probability of error" is inherently somewhat subjective. When error correction is concerned, the decomposition into I and Z is the better choice for the dephasing channel, for reasons that will become clearer in chapter 2. However, there are many channels for which none of the decompositions is particularly favored, even for the specific application of quantum error correction.

In the Bloch sphere picture, a dephasing channel shrinks the sphere into an ellipsoid, leaving the Z axis unchanged, as in figure 1.2. A completely dephasing channel shrinks the Bloch sphere down to just the segment on the Z axis.

The dephasing channel is a physically very interesting channel. A dephasing channel occurs in any system where the environment learns about the qubit in the standard basis but does not otherwise interfere with the state. Even in a more realistic system where there are more complicated interactions between the

system and the environment, there is frequently a large dephasing component to the noise. The prevalence of approximate dephasing channels is one of the reasons that the macroscopic world appears classical to us — a completely dephasing channel converts a qubit into a probabilistic classical bit.

We can make a simple model of a dephasing channel by having one environment qubit interact with the system qubit via a Hamiltonian $H = \omega Z \otimes Z$. The environment qubit starts in the pure state $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$, and the system qubit starts in state $|\psi_0\rangle = \alpha|0\rangle + \beta|1\rangle$. Then, after a time t, the state of the two qubits is

$$e^{-i\omega t Z\otimes Z}|\psi_0\rangle \otimes |+\rangle = \frac{1}{\sqrt{2}}(\alpha e^{-i\omega t}|00\rangle + \alpha e^{+i\omega t}|01\rangle + \beta e^{+i\omega t}|10\rangle + \beta e^{-i\omega t}|11\rangle. \tag{1.14}$$

(Taking $\hbar = 1$.) Tracing over the second (environment) qubit, we find that the system qubit at time t is in an equal mixture of $R_{-\omega t}|\psi_0\rangle$ and $R_{+\omega t}|\psi_0\rangle$, giving it density matrix

$$\begin{pmatrix} |\alpha|^2 & \alpha\beta^* \cos(2\omega t) \\ \alpha^*\beta\cos(2\omega t) & |\beta|^2 \end{pmatrix}. \tag{1.15}$$

In other words, at time t, we have the dephasing channel $\mathcal{R}_{(1-\cos(2\omega t))/2}$. In this simple model, the system dephases at short times, but after a longer time $t = \pi/\omega$, it returns to its starting state.

In a more realistic system, there are many different environment qubits interacting with the system, and/or there are additional qubits interacting with the environment qubits. If we take a Markovian form of our simple model, and assume that the environment's internal interaction effectively resets the environment qubit after a very short time, phase coherence instead decays steadily:

$$\begin{pmatrix} |\alpha|^2 & \alpha\beta^* e^{-t/T_2} \\ \alpha^*\beta e^{-t/T_2} & |\beta|^2 \end{pmatrix}, \tag{1.16}$$

corresponding to a dephasing channel $\mathcal{R}_{(1-e^{-t/t_2})/2}$. The time constant T_2 of the exponential decay is known as the t_2 *time*. (Otherwise I probably would have used a different symbol.) One way to think about this behavior is that there is a constant probability $1/T_2$ per unit time that the phase is completely randomized as an instantaneous "quantum jump".

Another common source of dephasing is a varying energy difference between the $|0\rangle$ and $|1\rangle$ states. If $|a\rangle$ has energy E_a, after time t, $|a\rangle$ has evolved into $e^{-iE_a t}|a\rangle$. We can ignore the global phase, but there is still a relative phase difference $e^{-i(E_1-E_0)t}$ between $|0\rangle$ and $|1\rangle$. However, when E_0 and E_1 are known constants, we can generally ignore the relative phase too: if we keep track of the time t, the relative phase is known, and we can take it into account in any operation we want to perform. In some systems, there is a phase reference (such as the phase of a laser) which automatically compensates for the phase difference. However, when the energy difference varies unpredictably, we can no longer keep precise track of the relative phase, resulting in an uncompensated relative phase shift accumulating randomly over time. This results in dephasing. In some cases, the relative phase shift is not random, but is simply unknown, and more sophisticated techniques may be able to compensate.

Experimentally, the signature of dephasing is often a decay of coherent interference effects. In a Rabi oscillation experiment, the system cycles $\cos(\Omega_R t)|0\rangle + \sin(\Omega_R t)|1\rangle$. After time t, the system is measured to test if it is $|0\rangle$ or $|1\rangle$. If the system were perfect, if we were to plot the probability of 1 against the time, we would get a perfect oscillation $\sin^2(\Omega_R t)$ for all times. Instead, we get something more like figure 1.3, with oscillations decreasing in amplitude.

Let us imagine a Markovian environment, and assume that the T_2 time is much longer than the Rabi frequency Ω_R, so that we can assume the dephasing and oscillation are independent. If there is a quantum jump causing full dephasing at time t_0, the pure state $\cos(\Omega_R t_0)|0\rangle + \sin(\Omega_R t_0)|1\rangle$ becomes the mixed state with probability $\cos^2(\Omega_R t_0)$ of $|0\rangle$ and probability $\sin^2(\Omega_R t_0)$ of $|1\rangle$. Then $|0\rangle$ and $|1\rangle$ continue with their

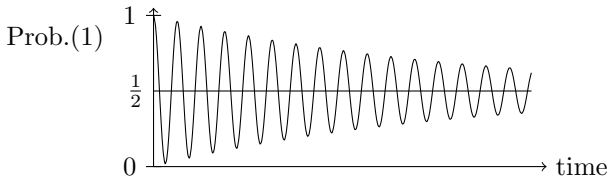


Figure 1.3: Decay of Rabi oscillations due to dephasing

own oscillations, but out of phase with each other. This mixture has density matrix

$$\begin{pmatrix} \cos^2(\Omega_R t_0)\cos^2(\Omega_R t') + \sin^2(\Omega_R t_0)\sin^2(\Omega_R t') & (\cos^2(\Omega_R t_0) - \sin^2(\Omega_R t_0))\sin(\Omega_R t')\cos(\Omega_R t') \\ (\cos^2(\Omega_R t_0) - \sin^2(\Omega_R t_0))\sin(\Omega_R t')\cos(\Omega_R t') & \cos^2(\Omega_R t_0)\sin^2(\Omega_R t') + \sin^2(\Omega_R t_0)\cos^2(\Omega_R t') \end{pmatrix} \tag{1.17}$$

$$= \begin{pmatrix} \sin^2(\Omega_R t_0) + \cos(2\Omega_R t_0)\cos^2(\Omega_R t') & \cos(2\Omega_R t_0)\sin(\Omega_R t')\cos(\Omega_R t') \\ \cos(2\Omega_R t_0)\sin(\Omega_R t')\cos(\Omega_R t') & \sin^2(\Omega_R t_0) + \cos(2\Omega_R t_0)\sin^2(\Omega_R t') \end{pmatrix} \tag{1.18}$$

$$= \sin^2(\Omega_R t_0) I + \cos(2\Omega_R t_0)\rho(t'), \tag{1.19}$$

where $t' = t - t_0$ and $\rho(t')$ is the density matrix of a Rabi oscillation running for time t'. We will still see Rabi oscillation, but with a reduced amplitude.

1.2.3 Depolarizing Channel and Pauli Channel

While the dephasing channel is perhaps the favorite (or least favorite, depending on your point of view) of experimentalists, it lacks a certain quantumness. It only involves one kind of non-trivial error, and indeed, in an appropriate choice of basis, it can be viewed as a purely classical noisy channel. Theorists are instead enamored of the depolarizing channel, which does experience the full range of quantum errors and is highly symmetric. It is not so common in the real world, which is less symmetric than theorists would prefer, but it is still extremely useful for exploring quantum error correction.

Definition 1.3. The *depolarizing channel* is the quantum channel

$$\mathcal{D}_p(\rho) = (1-p)\rho + \frac{p}{3}X\rho X^\dagger + \frac{p}{3}Y\rho Y^\dagger + \frac{p}{3}Z\rho Z^\dagger. \tag{1.20}$$

$\mathcal{D}_{3/4}$ is the *completely depolarizing channel*.

The depolarizing channel has an equal chance of an X, Y, or Z error, each with probability $p/3$. We can analyze what it does to the density matrix:

$$\mathcal{D}_p : \begin{pmatrix} a & b \\ c & d \end{pmatrix} \mapsto \begin{pmatrix} (1-2p/3)a + (2p/3)d & (1-4p/3)b \\ (1-4p/3)c & (1-2p/3)d + (2p/3)a \end{pmatrix}. \tag{1.21}$$

This is not particularly enlightening until you remember that $\operatorname{tr}\rho = 1$, in which case you can see that

$$\mathcal{D}_p(\rho) = (1-4p/3)\rho + (4p/3)(I/2). \tag{1.22}$$

In other words, with probability $1 - 4p/3$, the qubit is left alone, but with probability $4p/3$, it is replaced with the completely mixed state. It also should now be clear why I defined $p = 3/4$ as the completely depolarizing channel: in that case, the input state is always replaced with the maximally mixed state. As with the dephasing channel, there is some ambiguity as to what the "true" error rate is for the depolarizing channel, but the two representations can be reconciled by recognizing that the completely mixed state does contain a component of the original input state.

The other thing to notice about the second representation is that it is much more symmetric than the first one. The decomposition into Paulis has a certain amount of symmetry, treating X, Y, and Z on the

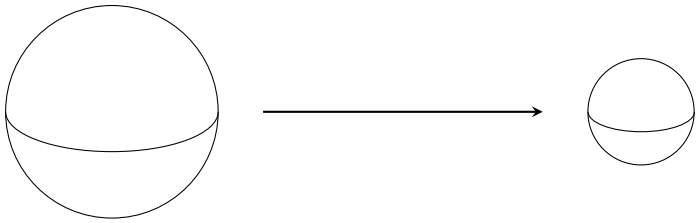


Figure 1.4: Bloch sphere transformation induced by a depolarizing channel

same footing, but the second decomposition has even more: there are no preferred bases or unitary operators at all appearing in it. This is the true beauty of the depolarizing channel — it can at once be considered as a simple mixture of the very basic Pauli errors, but also is invariant under any kind of unitary rotation. This symmetry property suggests a decomposition for the depolarizing channel akin to equation (1.13), and indeed there is one:

$$\mathcal{D}_p(\rho) = (1 - 4p/3)\rho + \int_{\mathsf{SU}(2)} U\rho U^\dagger dU, \tag{1.23}$$

where the integral uses the Haar measure, the unitarily-invariant measure over $\mathsf{SU}(2)$.

We can generalize the depolarizing channel by giving up its symmetry but keeping its decomposition into Paulis:

Definition 1.4. A channel of the form

$$\mathcal{E}(\rho) = p_I\rho + p_X X\rho X^\dagger + p_Y Y\rho Y^\dagger + p_Z Z\rho Z^\dagger \tag{1.24}$$

is a *Pauli channel*.

A Pauli channel has potentially different probabilities for the four Pauli matrices I, X, Y, and Z. They don't *have* to be different — dephasing channels and depolarizing channels are both examples of Pauli channels — but once you've generalized to a Pauli channel, you might as well take advantage of the opportunity to have some variety among the Paulis. Naturally, $p_I + p_X + p_Y + p_Z = 1$ so that the total probability adds up to 1.

The depolarizing channel uniformly shrinks the Bloch sphere into a smaller sphere still centered on the origin, or to a single point (the maximally mixed state) if we have the completely depolarizing channel. A more general Pauli map shrinks the Bloch sphere into an ellipsoid centered on the origin.

1.2.4 Amplitude Damping Channel

Another physically motivated channel is the amplitude damping channel. It occurs, for instance, if $|0\rangle$ and $|1\rangle$ are the ground and excited state of a two-level atom, and the excited state can decay to the ground state by spontaneously emitting a photon. The amplitude damping channel also occurs for a photonic qubit where $|0\rangle$ is no photon and $|1\rangle$ is 1 photon — damping occurs when the photon escapes or is absorbed somewhere along the way. There should be 2 Kraus operators for the amplitude damping channel, one representing the cases when there is no emission (or loss), and one representing the "no-jump" case. It is worthwhile trying to figure out yourself what the form of the Kraus operators should be before looking at the definition below.

Definition 1.5. An *amplitude damping channel* has the following form:

$$\mathcal{A}_p(\rho) = A_0\rho A_0^\dagger + A_1\rho A_1^\dagger, \tag{1.25}$$

where

$$A_0 = \begin{pmatrix} 1 & 0 \\ 0 & \sqrt{1-p} \end{pmatrix}, \quad A_1 = \begin{pmatrix} 0 & \sqrt{p} \\ 0 & 0 \end{pmatrix}. \tag{1.26}$$

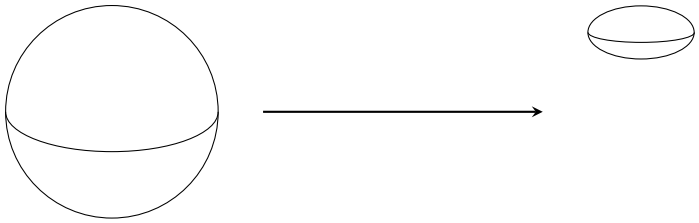


Figure 1.5: Bloch sphere transformation induced by an amplitude damping channel

Probably you were able to get the right form for A_1 (perhaps without the square root, which is a matter of convention depending on how we choose to parametrize amplitude damping channels). It represents the possibility that $|1\rangle$ can become $|0\rangle$. However, $|0\rangle$ never spontaneously gains energy in this idealized channel; even in the real world, it is fairly rare, since it requires that a stray photon of about the right energy be wandering by. Therefore the lower left corner of A_1 is 0.

A_0 might surprise you. (If not, then good work.) The most obvious guess is that since there is no decay, nothing should happen, and A_0 should be proportional to the identity. However, you won't be able to satisfy the constraint that $A_0^\dagger A_0 + A_1^\dagger A_1 = I$ if you choose A_1 as above and $A_0 \propto I$. Conceptually, the reason for this is that A_1 can only occur if the initial state was $|1\rangle$. Therefore, if A_1 *doesn't* happen, it means that the initial state was *more likely* to be $|0\rangle$, and A_0 reflects that, reducing the amplitude of $|1\rangle$ in the initial state. The same phenomenon can be found in classical probability theory, for instance in the "Monty Haul problem."

In the Bloch sphere picture, amplitude damping results in a uniform shrinkage to a smaller sphere. It differs from the depolarizing channel in that the center of the sphere is no longer fixed. Instead, the south pole (the $|0\rangle$ state) is fixed. In the limit $p = 1$, the whole sphere shrinks down to the south pole.

The characteristic decay time for a system undergoing continuous amplitude damping is the "T_1 time." (After all, there had to be a T_1 to go with T_2 for the dephasing time scale.)

1.2.5 Photon Loss Channels

Amplitude damping can occur due to photon loss from a photon channel when the presence or absence of a photon represents $|0\rangle$ or $|1\rangle$. However, this is not a particularly common encoding used for photonic qubits. More often, the qubit is stored in some other state of a single photon (such as the polarization), or by being in one of two modes (a "dual-rail encoding"), or some more complicated structure possibly involving multiple photons. When we use one of these encodings, loss of a photon is no longer represented by an amplitude damping channel.

Consider, for instance, a polarization encoding, with horizontal polarization being $|0\rangle$ and vertical polarization being $|1\rangle$. Now, photon loss can affect either $|0\rangle$ or $|1\rangle$, and what results if the photon does escape is neither of those states, instead being a third state $|\text{vacuum}\rangle$. Therefore, a photon loss channel for polarization encoding takes a qubit input but its output is a 3-dimensional *qutrit*. Assuming both polarizations have equal probability p to lose a photon, the channel is then very simple. There are three Kraus operators, corresponding to no photon loss, loss of a horizontally polarized photon, and loss of a vertically polarized photon:

$$A_0 = 1 - p \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}, A_1 = p \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{pmatrix}, \text{ and } A_2 = p \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}. \tag{1.27}$$

The same channel works for photon loss from a dual-rail encoding.

When the encoding involves multiple photons, there is still a sensible description as a quantum channel, but the analysis requires a bit more quantum optics, and is beyond the scope of this book.

1.2.6 Erasure Errors

The example of the photon loss qubit channel suggests another interesting kind of error. When a photon is not lost, the state is not changed at all. When a photon *is* lost, by monitoring the photon number (but not the polarization), we can, in principle, tell that a photon has been lost. In that case, we do not know what the original state of the system was, but at least we know that something has happened to it. Measuring the photon number without destroying the polarization state is technologically difficult, so for this particular example, this is more an issue of principle than of practice. There are, however, other systems where monitoring for loss of the qubit is easier.

Definition 1.6. A *qubit erasure error* is an error A acting on a single qubit which maps all qubit states to a third state $|\perp\rangle$ orthogonal to the qubit Hilbert space.

Erasure channels may seem to be difficult channels to correct, since the information is completely lost, but that's not the case. By measuring if the state is $|\perp\rangle$ is present (without collapsing superpositions of $|0\rangle$ and $|1\rangle$) — for instance, by measuring the number of photons — we can determine if an erasure error has occurred. Because erasure channels provide some classical information about which qubits underwent errors, erasure channels are actually easier to correct than more general channels.

1.3 Multiple-Qubit Channels

It's hard to do too much with only one qubit. Even if we have only one qubit of data, we'll need more than one qubit to create a real quantum error-correcting code. Therefore, we should also think some about channels acting on multiple qubits.

1.3.1 Pauli Channels

One route to defining multiple-qubit channels is to largely forgot about the tensor product structure and just pick some completely positive map acting on the Hilbert space as a whole. Such channels can be extremely complicated, since they act on a Hilbert space of dimension 2^n when there are n qubits. Sometimes this is necessary, as it reflects the actual physics of the system, but quantum computers are usually built out of systems that naturally break up into qubits, and many-qubit interactions are rare.

Mathematically, it is usually too difficult to deal with an arbitrary n-qubit channel, but the n-qubit generalization of the Pauli channel is sufficiently well-behaved to be sometimes useful.

Definition 1.7. A channel of the form

$$\mathcal{E}(\rho) = \sum_P p_P P \rho P^\dagger, \tag{1.28}$$

where the sum is taken over P which are tensor products of I, X, Y, and Z, is a *Pauli channel*.

In a Pauli channel, the operator P occurs with probability p_P. Pauli channels are a reasonable quantum analogue of classical channels. There is a definite probability of error, and the errors that occur are large and discrete. However, since we can have a mix of bit flip and phase errors, a Pauli channel does have enough quantum features to be interesting.

1.3.2 Independent/Memoryless Channels

The simplest way to create a channel on n qubits is to just treat each qubit separately.

Definition 1.8. An *independent* channel on n qudits (each of dimension q) has the form $\otimes_{i=1}^n \mathcal{E}_i$, where each $\mathcal{E}_i$ is a single-qudit channel.

A dimension-q qudit is a single system whose state space is a Hilbert space of dimension q, so for instance, $q = 2$ gives us a qubit, and for the moment, we will restrict attention to qubits. Often, we set all $\mathcal{E}_i$'s to be equal to $\mathcal{E}$, so that all qubits are treated equally.

As a simple example, we can consider $\mathcal{E} = \mathcal{R}_p$, the dephasing channel. Let us stick to $n = 3$ in order to be more explicit. There are a total of 8 possible Kraus operators for this channel, with the following probabilities:

Probability	Errors
$(1-p)^3$	$I \otimes I \otimes I$
$p(1-p)^2$	$Z \otimes I \otimes I$, $I \otimes Z \otimes I$, $I \otimes I \otimes Z$
$p^2(1-p)$	$Z \otimes Z \otimes I$, $Z \otimes I \otimes Z$, $I \otimes Z \otimes Z$
p^3	$Z \otimes Z \otimes Z$

The total probability of having a Z error on exactly one qubit is then $3p(1-p)^2$, and the probability of having two Z errors is $3p^2(1-p)$.

The chance of having at least one qubit with an error on it is therefore larger than the chance of having a single qubit by itself go wrong under the same dephasing channel $\mathcal{R}_p$. This makes sense, since there are more places for errors to occur. However, when p is small, most of the time, there will only be 0 or 1 Z error, and the other qubits will have I acting on them. This is the case we want to address through quantum error correction — errors are rare, but not negligibly so. In the case of this 3-qubit dephasing channel, we can make a good approximation by considering only the no-error or one-error possibilities, and ignoring the two- and three-qubit error cases.

1.3.3 Definition of t-Qubit Errors

More generally, we can define a t-qubit error to be one that acts on t qubits. However, there are two technical points in exactly how we want to define it:

Definition 1.9. We say that a linear operator A acting on n qubits has *weight* t if it is the tensor product of the identity on $n-t$ qubits with some matrix on the remaining t qubits. The weight of A is denoted wt A. We say that a weight t operator A has *support* on the t qubits on which it acts non-trivially. Generally (but not always), we cite the weight and support for the minimal set of qubits for which A acts non-trivially. A linear operator B acting on n qubits is a *t-qubit error* if it has the form

$$B = \sum_i B_i, \tag{1.29}$$

where each B_i has weight at most t, not necessarily with support on the same set of qubits for different i. An n-qubit quantum channel is a *t-qubit error channel* if it has a Kraus decomposition for which all Kraus operators are t-qubit errors. We can similarly define a *t-qubit error map* acting linearly on n-qubit density matrices but which may not be trace preserving.

Technical point number one is that even though a weight-t operator A acts non-trivially on only t qubits, it can act *arbitrarily* on those t qubits. In particular, it does not need to be a tensor product of errors on the separate qubits — it can entangle them however it likes. For instance, CNOT $\otimes I$ is a weight 2 operator acting on 3 qubits.

Technical point number two is that a t-qubit error B can be the sum of terms acting on different sets of t qubits. This naturally means that more than t qubits will be altered under the action of this error, but it turns out that they are altered in a way that is no more harmful than if we had a channel that had a possibility of altering each set of t qubits as separate Kraus operators. This point will be discussed in greater length in section 2.4.3, once we have a real quantum error-correcting code to examine.

We can also define t-qubit erasure errors. For instance, imagine we have many qubits each stored as the polarization of a photon, and each one undergoes a slight amount of photon loss.

Definition 1.10. A *t-qubit erasure error* is a weight t operator which is the tensor product of erasure errors on the qubits in its support. A *t-qubit erasure channel* is a quantum channel with a Kraus representation

where all Kraus operators are s-qubit erasure errors, with $s \leq t$. s can be different for different Kraus operators.

Note that here I am requiring that each Kraus operator has a specific set of qubits which can be erased, not a superposition of different sets of qubits. This reflects the idea that an erasure is in some sense a classical event, because the set of qubits that were erased can be measured.

1.3.4 Connection Between Independent Channel and t-Qubit Errors

An independent channel seems much more physically plausible (albeit still an approximation) than a t-qubit channel. However, it turns out to be more mathematically straightforward to design quantum error-correcting codes for t-qubit channels (or to correct a set of t-qubit errors) than for independent channels. Luckily, as suggested by the example of the 3-qubit dephasing channel, an n-qubit independent channel can be well approximated by a t-qubit channel (for some $t < n$) when the single-qubit channels making up the n-qubit channel are all close to the identity.

Theorem 1.1. *Let $\mathcal{I}$ be the 1-qubit identity channel and $\mathcal{E} = \otimes_{i=1}^{n} \mathcal{E}_i$ be an n-qubit independent channel, with $\|\mathcal{E}_i - \mathcal{I}\|_\diamond < \epsilon \leq \frac{t+1}{n-t-1}$, and $\epsilon \leq 1/3$ as well. Then*

$$\|\mathcal{E} - \tilde{\mathcal{E}}\|_\diamond < 5\binom{n}{t+1}[(4e+2)\epsilon]^{t+1} \tag{1.30}$$

for some t-qubit error map $\tilde{\mathcal{E}}$.

The significance of the theorem is that if we have a quantum error-correcting code which is designed to correct t-qubit error channels, it will automatically also correct independent channels where the single-qubit tensor factors are sufficiently close to the identity. Actually, that's not completely true: While the theorem does show that, this is not really the significance of the theorem, since there is much easier proof (which we will see in chapter 2) that a quantum error-correcting code that corrects t-qubit errors also corrects small independent error channels. Rather, this theorem provides a motivation for thinking about t-qubit error channels, which might otherwise seem rather bizarre.

When $\|\mathcal{E}_i - \mathcal{I}\|_\diamond < \epsilon$, we are getting about an ϵ chance of error per qubit, so with n qubits, we expect around ϵn errors. Therefore, we would not expect to be able to approximate $\mathcal{E}$ well by a t-qubit error map unless $t \gtrsim \epsilon n$, which is reflected in the bound on ϵ. The other detailed constants in the theorem shouldn't be taken too seriously, as the proof could likely be tightened considerably to get better constants. But if you do insist on taking those constants seriously, you might find, for example, that when $n = 5$ and $t = 1$, you would get $\|\mathcal{E} - \tilde{\mathcal{E}}\|_\diamond \lesssim 8286\epsilon^2$, which only gives a non-trivial bound when ϵ is less than about 0.01.

Two elements of equation (1.30) that *are* significant are the combinatorial factor $\binom{n}{t+1}$, which reflects the number of $(t+1)$-qubit subsets of the n qubits, and the exponent $t+1$ for ϵ, which tells us that the closeness of the approximation improves exponentially as we allow more qubits to have errors. In the limit of large n and any constant ratio t/n, the combination of the combinatorial factor and the exponent means that there will be a threshold value of ϵ below which we get a good approximation for all large n (and indeed, a better one as n gets larger).

Proof. We begin with a lemma on sums of error probabilities or amplitudes that will also be helpful later.

Lemma 1.2. *If $0 < t < n$, then*

a) For any $0 \leq \epsilon \leq 1$, $\sum_{j=t+1}^{n} \binom{n}{j} \epsilon^j (1-\epsilon)^{n-j} \leq \binom{n}{t+1} \epsilon^{t+1}$

b) When $0 \leq \epsilon \leq \frac{t+1}{n-t-1}$, then $\sum_{j=t+1}^{n} \binom{n}{j} \epsilon^j \leq \binom{n}{t+1} (e\epsilon)^{t+1}$

Proof of lemma. There are a number of ways to prove part a. One straightforward method is to interpret ϵ as a probability of some event (which is the main application we will have for this lemma). Then the sum is the probability that the event occurs at least $t+1$ times in n independent trials. We can upper bound

this probability by considering each subset of $t+1$ trials. The total probability of having the event occur in all trials in the subset, without regard to what happens on the other $n-t-1$ trials, is ϵ^{t+1}. There are $\binom{n}{t+1}$ subsets of size $t+1$, so by the union bound, the total probability of having some set of $t+1$ trials with the event is at most $\binom{n}{t+1}\epsilon^{t+1}$. Whenever the event occurs $j > t+1$ times, we have over-counted, since we included that probability as part of all $\binom{j}{t+1}$ sets of size $t+1$ which had the event.

For part b, note that

$$(1-\epsilon)^{n-t-1} \geq \left(1 - \frac{t+1}{n-t-1}\right)^{n-t-1} \geq e^{-(t+1)}. \tag{1.31}$$

Then

$$\sum_{j=t+1}^{n} \binom{n}{j}\epsilon^j = \sum_{j=t+1}^{n} \binom{n}{j}\epsilon^j(1-\epsilon)^{n-j}\frac{1}{(1-\epsilon)^{n-j}} \tag{1.32}$$

$$\leq \sum_{j=t+1}^{n} \binom{n}{j}\epsilon^j(1-\epsilon)^{n-j}\frac{1}{(1-\epsilon)^{n-t-1}} \tag{1.33}$$

$$\leq \binom{n}{t+1}\epsilon^{t+1}e^{t+1} \tag{1.34}$$

by part a and equation (1.31). □

As a warm-up to prove the theorem, let us consider the case when for all i, $\mathcal{E}_i$ has a Kraus operator $(1-\epsilon)I$. In this case, we can say that $\mathcal{E}_i$ has probability ϵ of having an error (one of the other Kraus operators), and a probability of $1-\epsilon$ of having no error. Part a of lemma 1.2 applies, so the probability of having at least $t+1$ errors is at most $\binom{n}{t+1}\epsilon^{t+1}$. In this case, the t-qubit error map $\mathcal{F}$ has all combinations of up to t "error" Kraus operators with the "good" Kraus operator $(1-\epsilon)I$ on the other qubits. The map $\mathcal{F}$ is completely positive, but is not trace preserving, since we have discarded the Kraus operators with more than t errors.

For the general case, first we need a better characterization of single-qubit channels $\mathcal{G}$ which are close to the identity.

Lemma 1.3. *If $\mathcal{E}$ is a quantum channel from $\mathcal{H}_D$ to $\mathcal{H}_D$ satisfying $\|\mathcal{E}-\mathcal{I}_1\|_\diamond < \epsilon \leq 1/3$, then $\mathcal{E}$ has a Kraus representation $\mathcal{E}(\rho) = \sum_k A_k\rho A_k^\dagger$ such that $\|A_0 - I\|_\infty < \sqrt{2D}\,(\epsilon+\epsilon^2) + (\epsilon/2+\epsilon^2)$ and $\sum_{k\neq 0}\|A_k\|_\infty^2 < D\epsilon(1/2+\epsilon)$.*

Proof of lemma. Using the Choi-Jamiolkowski isomorphism, the channel $\mathcal{E}$ corresponds to the entangled state $\Phi_\mathcal{E} = (I\otimes\mathcal{E})(|\Phi^+\rangle\langle\Phi^+|)$, with $|\Phi^+\rangle = \frac{1}{\sqrt{D}}\sum_a |aa\rangle$. (Note that I have normalized the state to make it easier to apply common identities, which is not always the convention used with the Choi-Jamiolkowski isomorphism.) Since $\|\mathcal{E}-\mathcal{I}_1\|_\diamond < \epsilon$,

$$\|\Phi_\mathcal{E} - |\Phi^+\rangle\langle\Phi^+|\|_1 < \epsilon \tag{1.35}$$

as well. Since the trace distance between these two states is small, the fidelity between them is high:

$$\langle\Phi^+|\Phi_\mathcal{E}|\Phi^+\rangle > 1-\epsilon/2. \tag{1.36}$$

Now, $|\Phi^+\rangle\langle\Phi^+|$ has one eigenvalue $+1$ and the remaining eigenvalues 0. Let us also write $\Phi_\mathcal{E}$ in terms of an eigenbasis,

$$\Phi_\mathcal{E} = \sum_{i=0}^{D^2-1}\lambda_i|\phi_i\rangle\langle\phi_i|. \tag{1.37}$$

$\Phi_\mathcal{E}$ is positive and has trace 1, so $\lambda_i \geq 0$ and $\sum_i \lambda_i = 1$. Now consider

$$|\langle\phi_i|\Phi^+\rangle\langle\Phi^+|\phi_j\rangle| = |\langle\phi_i|\left(\Phi_\mathcal{E} - |\Phi^+\rangle\langle\Phi^+|\right)|\phi_j\rangle| < \epsilon \tag{1.38}$$

for $i \neq j$. In addition,

$$\langle \Phi^+ | \Phi_{\mathcal{E}} | \Phi^+ \rangle = \sum_{i=0}^{D^2-1} \lambda_i |\langle \Phi^+ | \phi_i \rangle|^2 > 1 - \epsilon/2. \tag{1.39}$$

We can choose the phase of the eigenstates $|\phi_i\rangle$ to ensure that $\langle \Phi^+ | \phi_i \rangle$ is real and non-negative. Letting $a_i = \langle \Phi^+ | \phi_i \rangle$, we have $a_i \geq 0$, $\sum \lambda_i a_i^2 > 1 - \epsilon/2$, and $a_i a_j < \epsilon$ for $i \neq j$. Assume without loss of generality that a_0 is the largest of the a_is. Then $\sum \lambda_i a_i^2 \leq \sum \lambda_i a_0^2 = a_0^2$, so

$$a_0 > \sqrt{1 - \epsilon/2} \geq 1 - \epsilon/2. \tag{1.40}$$

Thus

$$a_i < \epsilon / a_0 < \frac{\epsilon}{1 - \epsilon/2} \tag{1.41}$$

for $i \neq 0$. Then

$$\sum_{i=0}^{D^2-1} \lambda_i a_i^2 \leq \lambda_0 + \sum_{i=1}^{D^2-1} \lambda_i \frac{\epsilon}{1 - \epsilon/2} \tag{1.42}$$

$$= \lambda_0 + (1 - \lambda_0) \frac{\epsilon}{1 - \epsilon/2} \tag{1.43}$$

$$= \frac{\epsilon + (1 - 3\epsilon/2)\lambda_0}{1 - \epsilon/2}, \tag{1.44}$$

since $\sum \lambda_i = 1$. Therefore,

$$\lambda_0 \geq \frac{(1 - \epsilon/2)^2 - \epsilon}{1 - 3\epsilon/2} \geq 1 - \epsilon/2 - \epsilon^2. \tag{1.45}$$

(assuming $\epsilon \leq 1/3$), which means

$$\sum_{i \neq 0} \lambda_i = 1 - \lambda_0 \leq \epsilon/2 + \epsilon^2 \tag{1.46}$$

for $i \neq 0$.

We have now bounded all the terms we need, but we'd like a tighter bound on a_0. We can do that by going back and plugging in the bound on $\sum \lambda_i$.

$$\| |\phi_0\rangle \langle \phi_0| - |\Phi^+\rangle \langle \Phi^+| \|_1 \leq (1 - \lambda_0) + \| \lambda_0 |\phi_0\rangle \langle \phi_0| - |\Phi^+\rangle \langle \Phi^+| \|_1 \tag{1.47}$$

$$\leq (1 - \lambda_0) + \| \sum_{i \neq 0} \lambda_i |\phi_i\rangle \langle \phi_i| \|_1 + \| \Phi_{\mathcal{E}} - |\Phi^+\rangle \langle \Phi^+| \|_1 \tag{1.48}$$

$$\leq 2\epsilon + 2\epsilon^2. \tag{1.49}$$

But the 1-norm distance between two pure states is just given by

$$\| |\phi_0\rangle \langle \phi_0| - |\Phi^+\rangle \langle \Phi^+| \|_1 = 2\sqrt{1 - |\langle \phi_0 | \Phi^+ \rangle|^2}, \tag{1.50}$$

meaning

$$\sqrt{1 - |\langle \phi_0 | \Phi^+ \rangle|} \leq \frac{\epsilon + \epsilon^2}{\sqrt{1 + |\langle \phi_0 | \Phi^+ \rangle|}} \leq \epsilon + \epsilon^2. \tag{1.51}$$

Thus, in the Choi-Jamiolkowski isomorphism form of the channel, the state has one large eigenvalue, for which the eigenstate is close to the maximally-entangled state, and the other eigenvalues are all small, with the eigenstates far from the maximally entangled state $|\Phi^+\rangle$. (Of course, they could be close to *other* maximally entangled states.) We can recover a set of Kraus operators for the channel by letting

$$A_k |a\rangle = \sqrt{D \lambda_k} \left(\langle a| \otimes I \right) |\phi_k\rangle \tag{1.52}$$

for basis states $|a\rangle$, extended linearly to the full Hilbert space $\mathcal{H}_D$. (Recall that $|\phi_i\rangle$ is a state in $\mathcal{H}_D \otimes \mathcal{H}_D$, so the right-hand side of equation (1.52) is in $\mathcal{H}_D$.) Then $\|A_k\|_1^2 \leq D\lambda_k$, so $\sum_{k\neq 0} \|A_k\|_\infty^2 \leq D\epsilon(1/2+\epsilon)$, as desired.

To get the bound on A_0, note that $A_0|\psi\rangle = \sqrt{D\lambda_0}\langle\psi^*|\phi_0\rangle$, where $|\psi^*\rangle$ is the complex conjugate of $|\psi\rangle$ in the basis $\{|a\rangle\}$. Furthermore, $|\psi\rangle = \sqrt{D}\langle\psi^*|\Phi^+\rangle$. Then

$$\|A_0 - \lambda_0 I\|_\infty = \max_{|\psi\rangle} |A_0|\psi\rangle - \lambda_0|\psi\rangle| \tag{1.53}$$

$$= \max_{|\psi\rangle} \sqrt{D\lambda_0}|\langle\psi^*|\phi_0\rangle - \langle\psi^*|\Phi^+\rangle| \tag{1.54}$$

$$= \sqrt{D\lambda_0}||\phi_0\rangle - |\Phi^+\rangle| \tag{1.55}$$

$$= \sqrt{D\lambda_0}\sqrt{2 - 2\,\mathrm{Re}\langle\Phi_+|\phi_0\rangle} \tag{1.56}$$

$$\leq \sqrt{2D}\,(\epsilon + \epsilon^2), \tag{1.57}$$

applying equation (1.51) in the last line, and recalling we have chosen $\langle\Phi_+|\phi_0\rangle$ to be real. Thus,

$$\|A_0 - I\|_\infty \leq \sqrt{2D}\,(\epsilon + \epsilon^2) + (\epsilon/2 + \epsilon^2) \tag{1.58}$$

□

For the case of qubits and applying $\epsilon \leq 1/3$, the lemma gives $\|A_0 - I\|_\infty \leq 7\epsilon/2$ and $\sum_{k\neq 0} \|A_k\|_1^2 \leq 5\epsilon/3$ for $k \neq 0$. We will round to $\|A_0 - I\|_1 < 4\epsilon$ and $\sum_{k\neq 0} \|A_k\|_1^2 \leq 2\epsilon$ for $k \neq 0$.

Given lemma 1.3, we can use a similar approach for the general case as we did for the warm-up, which was essentially classical. Channel $\mathcal{E}_i$ has Kraus operators A_k^i, with A_0^i close to the identity and A_k^i small for $k \neq 0$. The n-qubit independent channel $\mathcal{E}$ has Kraus operators which are all possible tensor products $\bigotimes_i A_{k_i}^i$. Let $\mathcal{F}$ be the map whose Kraus operators are all tensor products $\bigotimes_i A_{k_i}^i$ with at most t values of i for which $k_i \neq 0$. Then

$$\|\mathcal{F} - \mathcal{E}\|_\diamond \leq \sum_{r=t+1}^{n} \sum_{|S|=r} \prod_{i\in S} \sum_{k_i\neq 0} \|A_{k_i}^i\|_\infty^2. \tag{1.59}$$

The sum over r represents the number of values of i for which $k_i \neq 0$, and S is the set of indices for which $k_i \neq 0$. Since $\sum_{k_i\neq 0} \|A_{k_i}\|_\infty^2 \leq 2\epsilon$, we get

$$\|\mathcal{F} - \mathcal{E}\|_\diamond \leq \sum_{r=t+1}^{n} \binom{n}{r}(2\epsilon)^r \leq \binom{n}{t+1}(2e\epsilon)^{t+1} \tag{1.60}$$

by lemma 1.2.

Now $\mathcal{F}$ is not yet a t-qubit error map because the A_0 terms include some errors. However, the A_0 terms are all near I, so we can expand them as $A_0^i = I + \delta A^i$, with $\|\delta A^i\|_\infty < 4\epsilon$. Given a Kraus operator for $\mathcal{F}$ $A_{\{k_i\}} = \bigotimes_i A_{k_i}^i$ with r indices $k_i \neq 0$, expand all A_0^is in this way and form $A'_{\{k_i\}}$ by discarding all terms in the expansion with more than $t - r$ δA^i factors. The resulting Kraus operators are composed of sums of terms with weight at most t, of which r non-identity factors come from $k_i \neq 0$ terms, and the remaining come from δA^i components of $k_i = 0$ factors.

$$\|A'_{\{k_i\}} - A_{\{k_i\}}\|_\infty \leq \left(\prod_{j|k_j\neq 0} \|A_{k_j}\|_\infty\right) \sum_{s=t+1-r}^{n-r} \sum_{|S|=s} \prod_{i\in S} \|\delta A^i\|_\infty \tag{1.61}$$

$$\leq \left(\prod_{j|k_j\neq 0} \|A_{k_j}\|_\infty\right) \sum_{s=t+1-r}^{n-r} \binom{n-r}{s}(4\epsilon)^s \tag{1.62}$$

$$\leq \binom{n-r}{t+1-r}(4e\epsilon)^{t+1-r} \left(\prod_{j|k_j\neq 0} \|A_{k_j}\|_\infty\right). \tag{1.63}$$

Let ρ be an arbitrary pure state, possibly entangled between $\mathcal{H}_D$ and a reference system. Then

$$\|A'_{\{k_i\}}\rho(A'_{\{k_i\}})^\dagger - A_{\{k_i\}}\rho A^\dagger_{\{k_i\}}\|_1 \leq \|(A'_{\{k_i\}} - A_{\{k_i\}})\rho(A'_{\{k_i\}})^\dagger\|_1 + \|A_{\{k_i\}}\rho((A'_{\{k_i\}})^\dagger - A^\dagger_{\{k_i\}})\|_1 \tag{1.64}$$

$$\leq 2\left(\prod_{j|k_j\neq 0}\|A_{k_j}\|_\infty\right)\|A'_{\{k_i\}} - A_{\{k_i\}}\|_\infty \tag{1.65}$$

$$\leq 2\binom{n-r}{t+1-r}(4e\epsilon)^{t+1-r}\left(\prod_{j|k_j\neq 0}\|A_{k_j}\|^2_\infty\right). \tag{1.66}$$

I have omitted the $\otimes I$ terms affecting the reference system in the first line; the equation is complicated enough as is. In the second line, we have used the property that $\|A|\psi\rangle\| \leq \|A\|_\infty$ and bounded $\|A_{\{k_i\}}\|_\infty$ and $\|A'_{\{k_i\}}\|_\infty$ by the norm of just the terms with $k_j \neq 0$. If we sum over all $\{k_i\}$ with the same locations for which $k_j \neq 0$, we get

$$\sum \|A'_{\{k_i\}}\rho(A'_{\{k_i\}})^\dagger - A_{\{k_i\}}\rho A^\dagger_{\{k_i\}}\|_\infty \leq 2\binom{n-r}{t+1-r}(4e\epsilon)^{t+1-r}(2\epsilon)^r. \tag{1.67}$$

Finally, let $\mathcal{G}$ be the linear map with Kraus operators $A'_{\{k_i\}}$. In $\mathcal{G}$, we have eliminated all Kraus operators with more than t errors. Then we get a bound on $\|\mathcal{G} - \mathcal{F}\|_\diamond$ by applying them to ρ, and considering the 1-norm of the resulting state, which involves summing equation (1.66) over all possible values of $\{k_i\}$ with at most t indices $k_j \neq 0$ (those with more have already been excluded from $\mathcal{F}$). We get

$$\|\mathcal{G} - \mathcal{F}\|_\diamond \leq 2\sum_{r=0}^{t}\binom{n}{r}\binom{n-r}{t+1-r}(4e\epsilon)^{t+1-r}(2\epsilon)^r \tag{1.68}$$

$$= 2\sum_{r=0}^{t}\binom{n}{t+1}\binom{t+1}{r}(4e\epsilon)^{t+1-r}(2\epsilon)^r \tag{1.69}$$

$$\leq 2\binom{n}{t+1}[(4e+2)\epsilon]^{t+1}. \tag{1.70}$$

Combining this with equation (1.60), we get

$$\|\mathcal{G} - \mathcal{E}\|_\diamond \leq 3\binom{n}{t+1}[(4e+2)\epsilon]^{t+1}. \tag{1.71}$$

There is one final step needed. It is possible that $\mathcal{G}$ is no longer completely positive, since it could be that $\|A'_{\{k_i\}}\rho(A'_{\{k_i\}})^\dagger\|_1 > \|A_{\{k_i\}}\rho(A_{\{k_i\}})^\dagger\|_1$, which could result in $\operatorname{tr}\mathcal{G}(\rho) > 1$. We should therefore scale $\mathcal{G}$ down to $C\mathcal{G}$, for appropriately chosen constant C, to get a map that is guaranteed to be completely positive. $\mathcal{F}$ is trace non-increasing, though, so

$$\operatorname{tr}\mathcal{G}(\rho) \leq 1 + \|\mathcal{G} - \mathcal{F}\|_\diamond \leq 1 + 2\binom{n}{t+1}[(4e+2)\epsilon]^{t+1} = 1/C. \tag{1.72}$$

Then

$$\|C\mathcal{G} - \mathcal{E}\|_\diamond \leq 3\binom{n}{t+1}[(4e+2)\epsilon]^{t+1} + 1 - C \tag{1.73}$$

$$\leq 5\binom{n}{t+1}[(4e+2)\epsilon]^{t+1}. \tag{1.74}$$

□

1.4 A Peek Ahead: Errors During Computation

In part II, we'll consider more general types of errors. In particular, quantum gates will be able to go wrong in various ways, and errors will occur multiple times during a computation. As noted above, the quantum channel picture is no longer completely general then, since the noise might be non-Markovian. Mostly we'll stick to Markovian noise, but even then matters are much more complicated. Since our control is no longer reliable, we'll have to deal with errors occurring even while we're trying to fix them. However, there are various subtler difficulties to contend with as well.

For one thing, we don't know when an error occurs, so we can't assume we do error correction immediately after each error. In particular, an error might occur right before a gate that we had intended for some other purpose. Then even if the gate itself works perfectly, it can cause the error to propagate, infecting an additional qubit with the same error. In addition, the effect of the gate can change the type of error. For instance, a Z error that occurs before a Hadamard gate will change into a X error after the gate. This phenomenon makes it much more difficult to take advantage of information we know about the errors. For instance, suppose the noise source is largely dephasing noise. We might want to use a code that is particularly good at correcting dephasing noise, but if we use Hadamard gates in our circuit, some of the Z errors that occur will have become X errors by the time we get around to correcting them, and our code won't work anywhere near as well as expected. A particularly insidious form of this phenomenon occurs when errors happen *during* the implementation of a gate, which, after all, should take a non-zero time. Even if the completed gate does not change the type of error, depending on how the gate is being implemented, the partial gate may in fact alter the structure of the noise.

We'll return to all these issues in part II, and discuss how to design fault-tolerant quantum circuits that allow reliable error correction and computation on encoded states despite the complications.

Chapter 2

Redundancy Without Repetition: Basics Of Quantum Error Correction

Now we are ready to start designing quantum error-correcting codes. A natural place to start for inspiration is to look at the theory of classical error-correcting codes, and indeed it can give us some guidance. However, we'll rapidly see that there are some major differences between classical and quantum error correction.

The simplest classical error-correcting code is the repetition code:

$$0 \mapsto 000 \tag{2.1}$$

$$1 \mapsto 111. \tag{2.2}$$

If we send this 3-bit encoding through a 1-bit error classical channel, it is clear that we will be able to correct for any error that occurs on a single bit. If all three bits are the same, we know there hasn't been an error, while if one of the three is different, for instance 010, we know that the one that's different is the one that's wrong. By enforcing a boring conformity among the bits, we can recover the original state. Recall that if we use an independent channel instead of a 1-bit error channel, then there is some chance of 2 or 3 errors, which would fool us. If there are two errors, the one bit that we think is wrong is actually the only bit that's correct, and our well-meaning attempt to fix it will actually complete the error, making all three bits wrong. Luckily, the chance of two errors occurring is only $O(p^2)$ when the probability of an error on a single bit is p (see section 1.3.2). When p is small, the chance of the encoded state ending up wrong is less than the chance that an unencoded bit can make it unchanged through the channel.

Throughout this chapter, we'll assume we have a t-qubit error channel. This is justified by theorem 1.1, which tells us that if we actually have an independent channel, it is very close to a t-qubit channel, at least when the error rate per qubit is low. We will actually reprove a version of theorem 1.1 with an easier proof and better constants specifically applicable to quantum error-correctiong codes.

To make a quantum error-correcting code, we might want to imitate the classical repetition code, but that instantly runs into a few problems. We'll need to find a somewhat different way to protect our quantum information, one that adds redundancy without repeating the state.

2.1 Quantum Error Correction? Ridiculous!

Why am I so dead-set against a quantum repetition code? Perhaps we could encode

$$|\psi\rangle \mapsto |\psi\rangle|\psi\rangle|\psi\rangle. \tag{2.3}$$

If you've had much quantum information experience, you'll immediately see the problem with this encoding: it is forbidden by the No-Cloning Theorem.

1. No-cloning theorem prohibits repeating a quantum state.
2. Measuring the data while determining the error will destroy superpositions.
3. We must correct Y and Z errors in addition to X errors.
4. We must correct an infinite set of unitaries and also channels which decohere the state.

Table 2.1: Barriers to making a quantum error-correcting code.

Theorem 2.1 (No-Cloning Theorem)**.** *There is no quantum operation which maps an arbitrary state $|\psi\rangle$ to $|\psi\rangle|\psi\rangle$.*

Proof. The problem is linearity. Suppose

$$|0\rangle \mapsto |00\rangle \tag{2.4}$$
$$|1\rangle \mapsto |11\rangle. \tag{2.5}$$

Then, because quantum operations must be linear,

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \mapsto \alpha|00\rangle + \beta|11\rangle. \tag{2.6}$$

However, this is not equal to the cloned $|\psi\rangle$ state:

$$|\psi\rangle|\psi\rangle = \alpha^2|00\rangle + \beta^2|11\rangle + \alpha\beta(|01\rangle + |10\rangle). \tag{2.7}$$

□

Another problem has to do with how we correct errors for the repetition code. Given a 3-bit state coming out of the channel, we'd like to look at it and determine which of the three bits is different from the other two. However, when dealing with quantum states, looks really can kill. Or if not kill, at least decohere, destroying any superposition we have. And without the ability to be in a superposition, a qubit is no better than a classical bit. Somehow, to make a quantum error-correcting code, we'll need some way to correct errors without looking too closely at what we're doing.

Even if we can handle these problems, it's clear that quantum error correction will be more complicated than classical error correction. For a classical channel, basically all that can happen is a bit flip error, but quantum codes are going to need to handle phase flip errors as well, not to mention Y errors. While the 3-qubit repetition code is good at correcting bit flip errors, it doesn't do anything to help correct phase flips. Actually, it makes phase flip errors more common since there are now three qubits which could have phase flip errors instead of only one.

In addition to X, Y, and Z, we will also have to handle an infinite set of unitaries, such as the R_θ errors. Then there are more general channels, such as the dephasing channel or depolarizing channel, which turn pure states into mixed states. How can we come up with a correction operation that will turn the state from a mixed state back into a pure state?

At this point, the prospects for a quantum error-correcting code look bleak. The difficulties we've identified so far are listed in table 2.1. But don't worry — there are solutions to all of these problems. If there weren't, this book would be a lot thinner.

2.2 The 3-Qubit Code(s)

We won't try to handle all of these problems at once. Let's focus first on points 1 and 2. We'll try to make a quantum version of the three-bit repetition code. The quantum code will encode a qubit, but only protect against one kind of error, just like the classical three-bit code.

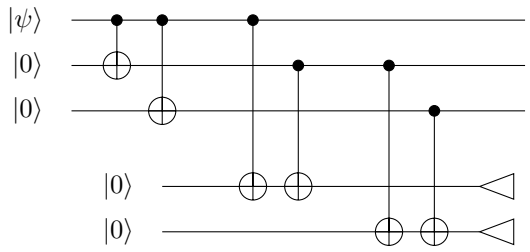


Figure 2.1: Encoding and syndrome measurement circuit for the 3-qubit bit flip correction code.

2.2.1 Correcting Bit Flips for Superposition States

Suppose we apply the classical repetition encoding to the basis states and extend to arbitrary superpositions by linearity:

$$\alpha|0\rangle + \beta|1\rangle \mapsto \alpha|000\rangle + \beta|111\rangle. \tag{2.8}$$

The first thing to notice is that this is an allowed encoding, and doesn't violate the no-cloning theorem. There is no rule against copying basis states — they act just like classical bits. The no-cloning theorem only kicks in if you try to copy superpositions as well. Rather than encoding a superposition $|\psi\rangle$ as three copies of $|\psi\rangle$ (which is impossible), we encode superpositions as *entangled* states. That takes care of difficulty 1.

Let's look at what happens to this state when there's been a bit flip error on the second qubit:

$$X_2(\alpha|000\rangle + \beta|111\rangle) = \alpha|010\rangle + \beta|101\rangle. \tag{2.9}$$

(In this book, I shall use the notation X_2 to indicate that the error X is acting on qubit 2. The same notation applies for gates. However, sometimes I shall omit the subscripts and instead write out a tensor product explicitly; in this case, that would be $I \otimes X \otimes I$.)

As with the classical repetition code, the error can be identified by noticing that the middle qubit is different from the first and third qubits. The critical point is that *this is true for both branches of the superposition.* It is therefore possible to measure the fact that the second qubit is different without measuring whether we have an encoded zero or an encoded one. If we don't measure the data that's encoded in the code, we don't destroy an encoded superposition. This is how we resolve the second barrier: we can measure the error without measuring the information we're trying to preserve.

2.2.2 Encoding Circuit and Error Syndrome Measurement

To understand this point in more detail, look at figure 2.1. The first part of the figure gives the encoding circuit for the 3-qubit code. We start with one qubit in an arbitrary unencoded state and add two additional qubits which become entangled with the first qubit. After the encoding, it is no longer particularly meaningful to ask which was the original data qubit and which are the ones we added. They are now all treated on an equal basis. The encoding of eqnrefeqn:threequbitbitflip is symmetric between all three qubits.

The second part of the circuit contains the error correction process. We wish to know whether one of the qubits is different from the other two, and if so, which one is different. We can break that down into two pieces: We ask whether the first two qubits are the same or different, and then we ask whether the second and third qubits are the same or different. In other words, we wish to know the *parity* of the first two qubits and the parity of the last two qubits. We will store the answers to these two questions on two more extra qubits. Extra qubits like these that are used for this or any other helpful purpose during a computation are known as *ancilla* qubits.

To tell whether two qubits are the same or different, the circuit in figure 2.1 uses two CNOT gates. If they are the same, either neither CNOT flips the corresponding ancilla, or both do; if they are different, exactly one of the CNOT gates flips the ancilla qubit. Thus, when we measure the ancilla qubit for one

of the parity measurements, the output we get is equal to the parity: 0 for same (even parity), and 1 for opposite (odd parity). Notice that the result does not depend at all on the encoded state, simply whether there is a bit flip error on the qubits we are measuring. That means the measurement can be done without disturbing a superposition of the encoded data.

Together, the measurement results for the two ancillas form a bit string known as the *error syndrome*. The error syndrome encapsulates all the information we have about the error. For instance, when the bit flip error is X_2, the error syndrome is 11 because both the first pair and the second pair of qubits are different. Similarly, error syndromes 10 and 01 correspond to the errors X_1 and X_3, respectively, and error syndrome 00 tells us that there is no error. Thus, every possible error for a one-qubit bit flip channel is accounted for. Once we know what the error is, we can simply correct it by performing another bit flip on the appropriate qubit.

Notice that the choice of error correction circuit and corresponding error syndrome is not unique. For instance, we could have measured the parity of the first and third qubits instead of measuring the parity of the second and third qubits. This would have given us the same information, but the correspondence between error syndromes and errors would have been shuffled. We could have measured the parities of all three pairs of qubits, but in that case the error syndrome (which would be 3 bits long) would be redundant: from any two error syndrome bits, we could deduce the third. It is not a coincidence that the number of syndrome bits we need is equal to the number of extra qubits we added to the data in order to perform the original encoding. This is a general property of quantum error-correcting codes, as discussed in chapter 3, although there are certain cases where we don't need all of the information encoded in the syndrome. (See section 17.2 for a description of that case.)

2.2.3 Phase Error Correction

The three-qubit code described above corrects a single bit flip error, but cannot correct any phase flip errors. It is also straightforward to make a code that works the other way: it corrects a single phase flip error, but cannot correct any bit flip errors.

The key to making this code is to notice that the Hadamard transform H switches the role of X and Z errors:

$$|0\rangle \leftrightarrow |+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{2.10}$$

$$|1\rangle \leftrightarrow |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{2.11}$$

X acting on the $|0\rangle$, $|1\rangle$ basis is a bit flip, but in the $|+\rangle$, $|-\rangle$ basis, it maps

$$X|+\rangle = |+\rangle \tag{2.12}$$

$$X|-\rangle = -|-\rangle, \tag{2.13}$$

acting as a phase flip. Similarly, Z switches $|+\rangle$ and $|-\rangle$, so it acts as a bit flip in the Hadamard-rotated basis.

Therefore, we can make a three-qubit *phase* correcting code by just applying H to every qubit in the three-qubit bit flip correcting code:

$$|0\rangle \mapsto |\overline{0}\rangle = |+\rangle|+\rangle|+\rangle \tag{2.14}$$

$$|1\rangle \mapsto |\overline{1}\rangle = |-\rangle|-\rangle|-\rangle, \tag{2.15}$$

extended by linearity so that $\alpha|0\rangle + \beta|1\rangle \mapsto \alpha|\overline{0}\rangle + \beta|\overline{1}\rangle$. In this book, I shall use lines over a state or operator to indicate the encoded version of the object.

If there is a single phase flip error, for instance, Z_2, one of the three qubits will be different than the other two in the Hadamard basis. E.g., $Z_2|\overline{0}\rangle = |+\rangle|-\rangle|+\rangle$. We can locate the error in just the same way as before, except that now we should rotate the control qubits for the CNOTs in the error correction circuit by a Hadamard transform to work in the correct basis. The modified encoding and error correction circuit is pictured in figure 2.2.

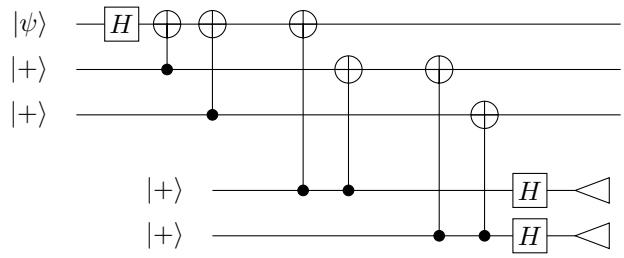


Figure 2.2: Encoding and syndrome measurement circuit for the 3-qubit phase correction code.

2.3 The 9-Qubit Code

The three-qubit codes resolve the first two barriers from table 2.1, but to address the third difficulty, we'll have to add more qubits. In order to make a code that corrects a bit flip error *or* a phase error, we'll encode one qubit into nine qubits, mixing together the encodings from both the bit and phase correcting three-qubit codes:

$$|0\rangle \mapsto |\overline{0}\rangle = \frac{1}{2\sqrt{2}}(|000\rangle + |111\rangle)(|000\rangle + |111\rangle)(|000\rangle + |111\rangle) \tag{2.16}$$

$$|1\rangle \mapsto |\overline{1}\rangle = \frac{1}{2\sqrt{2}}(|000\rangle - |111\rangle)(|000\rangle - |111\rangle)(|000\rangle - |111\rangle). \tag{2.17}$$

Again, we extend the encoding by linearity to work on superpositions. The 9-qubit code was discovered by Peter Shor and so is sometimes called the *Shor code*.

For the nine-qubit code, we will let the set of possible errors be $\{I, X_i, Y_i, Z_i | i = 1, \ldots, 9\}$, so we can have any single-qubit Pauli error. If we have an X error acting on a single qubit, that can be detected by looking at a single group of three qubits to see if one qubit is different from the other two in the standard basis. For instance,

$$X_5|\overline{0}\rangle = \frac{1}{2\sqrt{2}}(|000\rangle + |111\rangle)(|010\rangle + |101\rangle)(|000\rangle + |111\rangle), \tag{2.18}$$

and the error correction circuit of figure 2.1 applied to the middle set of three qubits will identify the error correctly. As before, once we identify the location of an X error, we can correct it by simply flipping the appropriate bit back to its original state. Also as before, the circuit tells us nothing about the encoded state.

If we have a Z error on a single qubit, matters are a little more complicated, but work basically the same way.

$$Z_5|\overline{0}\rangle = \frac{1}{2\sqrt{2}}(|000\rangle + |111\rangle)(|000\rangle - |111\rangle)(|000\rangle + |111\rangle). \tag{2.19}$$

Now we need to compare the three phases. Simply rotating into the Hadamard basis will not quite do it this time (although it gets us close), since we have to take into account the fact that the phase is distributed among a set of three qubits.

We will return to the nine-qubit code in chapter 3, and discuss exactly what to measure to find the error syndrome, but for now, let us just notice that it is possible in principle to make the desired measurement. The set of all correctly encoded states (the *code space*) forms a 2-dimensional subspace of the 2^9-dimensional Hilbert space of nine qubits. The two-dimensional subspace where the first phase is different from the other two (i.e., we have $-++$ or $+--$) is spanned by $Z_1|\overline{0}\rangle$ and $Z_1|\overline{1}\rangle$, and is orthogonal to the code space. Similarly, the subspaces where the second phase is different or the third phase is different are also orthogonal to the code space. Furthermore, these three erroneous code spaces are all orthogonal to each other. Thus, there is a measurement we can make that will distinguish them, and identify whether we have no phase error or one phase error, and if there is a phase error, on which set of three qubits it has occurred. Once we know the correct set of three, we can undo the phase error by applying Z to one of the qubits in that set of three.

A little consideration will show you that this argument applies to Y errors as well. The X and Z error correction procedures are essentially independent, and the argument that we can identify the block of three qubits with a Z error applies equally well if there is an X error on one of the nine qubits. Thus, the nine-qubit code can correct for both an X and a Z error. In particular, it can correct for a single-qubit Y error, which corresponds to having an X and a Z on the same qubit. In the set of possible errors, we only allowed single-qubit errors, but actually the code will still work if you extend the set of possible errors to allow one X error and one Z error on *any* pair of qubits.

2.4 Correcting General Errors

The nine-qubit code addresses problem 3, but it suggests that problem 4 might be troublesome. In order to correct a single bit-flip error, we needed 3 qubits, but to correct Y and Z errors as well, we went up to nine qubits. To solve problem 4 and correct general single-qubit errors, we'll need to handle an infinite set of errors. Does this mean we will have to use infinitely many qubits? Hopefully not.

2.4.1 Continuous Phase Rotation Example

Let's examine the specific case of the continuous phase rotation R_θ in more detail. We can rewrite

$$R_\theta = \begin{pmatrix} e^{-i\theta} & 0 \\ 0 & e^{i\theta} \end{pmatrix} = \cos\theta I - i\sin\theta Z. \tag{2.20}$$

Now suppose the usual nine-qubit code has experienced an error $(R_\theta)_i$ instead of the usual X, Y, or Z error, but imagine that we do not know that and use the usual error correction circuit for the nine-qubit code. We can figure out what happens to state by again applying the linearity of quantum mechanics.

$$(R_\theta)_i|\overline{\psi}\rangle = \cos\theta I|\overline{\psi}\rangle - i\sin\theta Z_i|\overline{\psi}\rangle. \tag{2.21}$$

We know what happens to $I|\overline{\psi}\rangle$ and $Z_i|\overline{\psi}\rangle$ under the error correction circuit: First, we interact with some ancilla qubits and determine the error syndrome corresponding to these errors, then we measure the ancilla qubits. If $|I\rangle_{\text{syn}}$ and $|Z_i\rangle_{\text{syn}}$ are the error syndromes corresponding to the errors I and Z_i, when we interact with the ancillas, we get the following:

$$(R_\theta)_i|\overline{\psi}\rangle \mapsto \cos\theta I|\overline{\psi}\rangle|I\rangle_{\text{syn}} - i\sin\theta Z_i|\overline{\psi}\rangle|Z_i\rangle_{\text{syn}}. \tag{2.22}$$

Right now, we have an entangled state between the code qubits and the ancilla qubits. When we measure the ancilla register, we'll collapse the superposition, getting one of two results:

$$\text{Prob.}(\cos^2\theta): \ I|\overline{\psi}\rangle|I\rangle_{\text{syn}} \tag{2.23}$$

$$\text{Prob.}(\sin^2\theta): \ Z_i|\overline{\psi}\rangle|Z_i\rangle_{\text{syn}}. \tag{2.24}$$

Something miraculous has happened: now we have either an I error (i.e., no error at all) or a Z error, and the outcome of the error syndrome tells us which and where the error is. We can then correct it in the usual way. Somehow, by pretending that we had an I, X, Y, or Z error, we made it actually true that the error was one of those. It's a triumph of wishful thinking.

For the nine-qubit code, the same procedure works with arbitrary single-qubit unitaries, and actually works for any Kraus operator A_k. Indeed, this is not a property of just the nine-qubit code, but of quantum error-correcting codes in general: correcting I, X, Y, and Z errors is sufficient to correct general errors. At this point, we could easily prove this for the nine-qubit code, but it is worthwhile to prove the generalization. To do that, we will want to first define precisely what a quantum error-correcting code is.

2.4.2 Definition of a Quantum Error Correcting Code

Generally speaking, a quantum error-correcting code is a subspace of a larger Hilbert space. Typically, that subspace has been chosen in some complicated entangled way in order to have special properties when errors occur on states from the code space. Therefore, it is most sensible to define a quantum error-correcting code together with the set of errors it corrects. Also, we'll frequently want to consider the subspace as a Hilbert space encoding some quantum data, so we'll define the subspace as a map from a smaller Hilbert space into the big Hilbert space.

Definition 2.1. A *quantum error-correcting code* $(U, \mathcal{E})$ (or *QECC* for short) is a partial isometry $U : \mathcal{H}_K \to \mathcal{H}_N$ with the set of *correctable errors* $\mathcal{E}$ (consisting of linear maps $E : \mathcal{H}_N \to \mathcal{H}_M$) with the following property: $\exists$ quantum operation $\mathcal{D} : \mathcal{H}_M \to \mathcal{H}_K$ such that $\forall E \in \mathcal{E}$, $\forall |\psi\rangle \in \mathcal{H}_K$,

$$\mathcal{D}(EU|\psi\rangle \langle\psi|U^\dagger E^\dagger) = c(E, |\psi\rangle)|\psi\rangle \langle\psi|. \tag{2.25}$$

U is known as the *encoding* operation for the code (or the *encoder*), and $\mathcal{D}$ is the *decoding* map (or *decoder*). $\mathcal{D}$ is also known as the *recovery* operation. We say that the QECC *corrects* E if $E \in \mathcal{E}$.

Sometimes, we don't consider a specific encoding map, and refer to Image(U) (more precisely known as the *code space*) as the QECC. A *codeword* is any state in the code space. Often, a QECC is mentioned without an explicit set of correctable errors, which should be determined by context. Sometimes the error set simply does not matter. When it does matter but is not specified, the set of correctable errors is often the set of all t-qubit errors for the largest possible t; except sometimes it is instead the set of the most likely errors in the system. $\mathcal{H}_K$ is sometimes referred to as the *logical* Hilbert space and $\mathcal{H}_N$ is the *physical* Hilbert space. If $\mathcal{E}$ is the set of all t-qubit errors, we say that U (or Image(U)) is a *t-error correcting code*, or that it *corrects t errors*.

In the above definition (and elsewhere in the book), $\mathcal{H}_D$ is a Hilbert space of dimension D. A partial isometry is like a unitary in that it preserves inner products, but might not be invertible because it may map a Hilbert space to a Hilbert space of strictly larger dimension. Of course $\mathcal{D}$ actually maps matrices on $\mathcal{H}_M$ to matrices on $\mathcal{H}_K$. Since E can be a general linear map, not necessarily unitary, $EU|\psi\rangle$ might not be normalized properly, which is why we need the $c(E, |\psi\rangle)$ factor in equation (2.25). The definition explicitly allows $c(E, |\psi\rangle)$ to depend on $|\psi\rangle$, but it turn outs that it does not (see section 2.5). It does depend on E, however.

To prove this, and in many other contexts, it is helpful to treat the decoding map as a unitary. This can be done via the Stinespring dilation, by purifying $\mathcal{D}$. To do so, we must add an ancilla register. Let the input ancilla to $\mathcal{D}$ be of dimension D and let the output ancilla have dimension D'. Then $\mathcal{H}_M \otimes \mathcal{H}_D \cong \mathcal{H}_K \otimes \mathcal{H}_{D'}$. $\mathcal{H}_{D'}$ plays the role of the error syndrome. Call the purification $V : \mathcal{H}_M \otimes \mathcal{H}_D \to \mathcal{H}_K \otimes \mathcal{H}_{D'}$. Now, $U|\psi\rangle = |\overline{\psi}\rangle$, and when the QECC corrects $E \in \mathcal{E}$,

$$V(E|\overline{\psi}\rangle_N \otimes |0\rangle_D) = \sqrt{c(E, |\psi\rangle)}\, |\psi\rangle_K \otimes |A(E, |\psi\rangle)\rangle_{D'} \tag{2.26}$$

I have put subscripts on the kets to help you keep track of which Hilbert spaces they belong to. There might be a phase on the RHS, but it can be absorbed into the ancilla state $|A(E, |\psi\rangle)\rangle$.

Proposition 2.2. *In definition 2.1, $c(E, |\psi\rangle)$ is independent of $|\psi\rangle$, as is $|A(E, |\psi\rangle)\rangle$ when we purify the decoder.*

Proof. Consider two codewords $|\overline{\psi}\rangle$ and $|\overline{\phi}\rangle$. Imagine we purify the decoder to the unitary V, so

$$VE|\overline{\psi}\rangle \otimes |0\rangle = \sqrt{c(E, |\psi\rangle)}\, |\psi\rangle \otimes |A(E, |\psi\rangle)\rangle \tag{2.27}$$

$$VE|\overline{\phi}\rangle \otimes |0\rangle = \sqrt{c(E, |\phi\rangle)}\, |\phi\rangle \otimes |A(E, |\phi\rangle)\rangle. \tag{2.28}$$

By linearity and the definition of a QECC applied to the superposition codeword $\alpha|\overline{\psi}\rangle + \beta|\overline{\phi}\rangle$,

$$VE(\alpha|\overline{\psi}\rangle + \beta|\overline{\phi}\rangle) \otimes |0\rangle = \alpha\sqrt{c(E, |\psi\rangle)}\, |\psi\rangle \otimes |A(E, |\psi\rangle)\rangle + \beta\sqrt{c(E, |\phi\rangle)}\, |\phi\rangle \otimes |A(E, |\phi\rangle)\rangle \tag{2.29}$$

$$= \sqrt{c(E, \alpha|\psi\rangle + \beta|\phi\rangle)}\, (\alpha|\psi\rangle + \beta|\phi\rangle) \otimes |A(E, \alpha|\psi\rangle + \beta|\phi\rangle)\rangle. \tag{2.30}$$

These two expressions can only be equal if $|A(E,|\psi\rangle)\rangle = |A(E,|\phi\rangle)\rangle = |A(E)\rangle$ (or the discarded ancilla will be entangled with the output state) and $c(E,|\psi\rangle) = c(E,|\phi\rangle) = c(E)$ (or the decoded state for the superposition will be wrong). The conceptual point here is that in order to preserve the coherent superposition in the decoder's output, there cannot be any information about the encoded state left in the ancilla, and the amplitudes (and thus norms) of different encoded states have to match, or the Hilbert space will get distorted. □

Note that the set of correctable errors is not a unique invariant property of an encoding map or code space, which is why I included it as part of the definition. The same code space could be used to correct different sets of errors. For instance, the 3-qubit phase correction code can correct the set $\mathcal{E} = \{I, Z_1, Z_2, Z_3\}$, but it also corrects the set $\mathcal{E} = \{I, Z_1Z_2, Z_1Z_3, Z_2Z_3\}$. In the former case, we interpret the non-zero error syndromes as caused by a single phase error, whereas in the latter case, we only consider two-qubit errors. Since Z_1 has the same error syndrome as Z_2Z_3 but they correspond to different logical states, we have to make a choice between them and cannot include both in the set of correctable errors at the same time. That is, $\mathcal{E} = \{I, Z_1, Z_2Z_3\}$ is *not* a possible set of correctable errors for the 3-qubit phase correction code.

There are many variations of the terms defined above. For instance, code space is sometimes coding space, code subspace, etc., and sometimes it is the encoded subspace. However, the encoded subspace also sometimes refers to $\mathcal{H}_K$, which can also be called the data or encoded data. As defined above, $\mathcal{D}$ incorporates both the error correction procedure and the "unencoding" process of returning the data to $\mathcal{H}_K$, but sometimes they are considered separately.

The decoder $\mathcal{D}$ incorporates whatever processing is necessary to correct and decode the state. For instance, it may incorporate a measurement of the error syndrome, and application of a correction operation conditional on the classical bits resulting from the syndrome measurement. The decoder $\mathcal{D}$ may not, however, be unique. Its behavior is completely determined on the subspace spanned by $E|\overline{\psi}\rangle$, where $E \in \mathcal{E}$ and $|\overline{\psi}\rangle$ is a codeword, but outside of that subspace, $\mathcal{D}$ can act arbitrarily, and the distinctions between different decoders can be important in a number of contexts, such as when studying the efficiency of the decoder, or its behavior on errors outside $\mathcal{E}$, or when considering fault tolerance.

Note that if we define a QECC just as a subspace instead of an encoding map, we haven't lost much information. In particular, the set of correctable errors is the same for all encoders:

Proposition 2.3. *Suppose we have two different encoders $U_1, U_2 : \mathcal{H}_K \to \mathcal{H}_N$ with* $\mathrm{Image}(U_1) = \mathrm{Image}(U_2)$. *Then $(U_1, \mathcal{E})$ is a QECC iff $(U_2, \mathcal{E})$ is a QECC.*

Proof. Because U_1 and U_2 are partial isometries with the same image, they can only differ by a unitary $V : \mathcal{H}_K \to \mathcal{H}_K$:

$$U_2 = U_1 V. \tag{2.31}$$

If we use decoder $\mathcal{D}_1$ for encoder U_1, then $V^\dagger \circ \mathcal{D}_1$ will serve as the decoding map for U_2:

$$\begin{aligned} V^\dagger \mathcal{D}_1(EU_2|\psi\rangle\langle\psi|U_2^\dagger E^\dagger)V &= V^\dagger \mathcal{D}_1(EU_1V|\psi\rangle\langle\psi|V^\dagger U_1^\dagger E^\dagger)V & (2.32)\\ &= V^\dagger(c(E,V|\psi\rangle)V|\psi\rangle\langle\psi|V^\dagger)V & (2.33)\\ &= c(E,|\psi\rangle)|\psi\rangle\langle\psi|. & (2.34)\end{aligned}$$

□

In the definition of a QECC, we have let the dimensions K, M, and N of the various Hilbert spaces involved be arbitrary, although $N \geq K$ or no QECC is possible. Nevertheless, there are some common restrictions. Frequently we assume $K = 2^k$ and $N = 2^n$, so there are k *logical qubits* (or *encoded qubits*) and n *physical qubits*. Sometimes we work with q-dimensional qudits instead of qubits, so $K = q^k$ and $N = q^n$. Occasionally we have even more general situations, where K and N might use different bases or not be powers at all. When there is a tensor factorization of the overall Hilbert space, but I don't want to be too specific about whether the individual factors are qubits, qudits, or maybe even different sizes from each other, I will refer to the *registers*, with each register being one tensor factor. A single error then affects a single register, whatever its size.

Most often $M = N$, so errors map the physical Hilbert space to itself. There is little lost by assuming this, since we can generally ignore any unused states in $\mathcal{H}_N$ or $\mathcal{H}_M$ without negative consequence. The main time when it matters is when errors occur repeatedly on the state before we perform error correction, as happens, for instance, in fault-tolerant quantum computation. In that case, you really need $M = N$, so that you can sensibly apply the same error over and over again.

One special case worth mentioning is that of erasure errors. Recall that an erasure error formally maps a qubit into a qutrit, so to treat erasure errors in the most precise way, we would take $N = 2^n$ and $M = 3^n$. However, most often we imagine that a "stop-leak" gate of some sort is performed before the decoder. The stop-leak gate will map the third $|\perp\rangle$ state of each qutrit to some state, perhaps a random one, of the corresponding qubit. Then we can consider an erasure error as mapping a qubit to a qubit, but with some additional classical information indicating that the qubit has undergone an error.

This is maybe a good place for an extremely important digression on the terminology of error correction. Specifically, I want to discuss the use of hyphens. Wait, did I say "important?" I meant "unimportant." But I am going to discuss hyphens anyway. In English, hyphens are occasionally used in compound nouns but not that frequently. Where they *are* used is to make compound adjectives. Thus, we have a hyphen in "error-correcting code" and "fault-tolerant computation" but not in "error correction" and "fault tolerance," unless one of the component words happens to get split between lines. (Notably, though, there is no hyphen for the "quantum" part, so "quantum error-correcting code.") Certainly, many scientists are not native speakers of English, so they have a good excuse for making this mistake. But you no longer have an excuse.

2.4.3 Linearity of Quantum Error Correction

Now we are ready to generalize the phenomenon we observed in section 2.4.1.

Theorem 2.4. *If $(U, \mathcal{E})$ is a QECC, then $(U, \mathcal{E}')$ is a QECC, where $\mathcal{E}' = \operatorname{span} \mathcal{E}$ is the linear span of $\mathcal{E}$.*

In other words, if a QECC can correct E and F, then it can also correct $\alpha E + \beta F$.

Proof. Basically, the idea is to duplicate the argument used in section 2.4.1 for the general case. We haven't defined the error syndrome for a general QECC, but we do have the decoding map to work with. We will purify it to V so that we can work with only unitary maps.

The QECC corrects $E, F \in \mathcal{E}$, so

$$V(E|\overline{\psi}\rangle_M \otimes |0\rangle_D) = c_E|\psi\rangle_K \otimes |s_E\rangle_{D'} \tag{2.35}$$

$$V(F|\overline{\psi}\rangle_M \otimes |0\rangle_D) = c_F|\psi\rangle_K \otimes |s_F\rangle_{D'}. \tag{2.36}$$

One difference from section 2.4.1 is that $|s_E\rangle$ and $|s_F\rangle$ don't need to be orthogonal. Also note that the c's from definition 2.1 would be the squares of c_E and c_F.

By linearity,

$$V[(\alpha E + \beta F)|\overline{\psi}\rangle_M \otimes |0\rangle_D] = \alpha c_E|\psi\rangle_K \otimes |s_E\rangle_{D'} + \beta c_F|\psi\rangle_K \otimes |s_F\rangle_{D'} \tag{2.37}$$

$$= |\psi\rangle_K \otimes (\alpha c_E|s_E\rangle_{D'} + \beta c_F|s_F\rangle_{D'}). \tag{2.38}$$

Tracing out $\mathcal{H}_{D'}$, we find that the decoder map is of the desired form, with

$$c(\alpha E + \beta F, |\psi\rangle) = \|\alpha c_E|s_E\rangle + \beta c_F|s_F\rangle\|^2. \tag{2.39}$$

□

2.4.4 Correcting Paulis Implies Correcting All Errors

As a corollary of theorem 2.4, we can simplify the condition for a code to correct t errors:

Corollary 2.5. *If a QECC set of correctable errors includes all tensor products of I, X, Y, and Z of weight t or less, it is a t-error correcting code.*

Proof. The set of all errors with support on a given set of t qubits is the set of $2^t \times 2^t$ matrices acting on those qubits. Tensor products of I, X, Y, and Z acting on those t qubits form a basis for this set of matrices. Therefore, an arbitrary weight t error can be written as the sum (with appropriate coefficients) of weight t tensor products of I, X, Y, and Z, and an arbitrary t-qubit error can be written as a sum weight t errors. The corollary then follows from theorem 2.4. □

As a consequence, we find that the nine-qubit code is a 1-error correcting code. Applying the definitions, we find that it encodes one logical qubit into nine physical qubits.

2.4.5 QECCs and Error Channels

We have finally answered all four objections to the existence of quantum error-correcting codes. However, it is worth thinking a little more carefully about what happens when we have a decoherent error that maps pure states to mixed states. For example, suppose that we have a dephasing channel with error probability p acting on qubit i. Then the pure codeword state $|\overline{\psi}\rangle$ becomes:

$$|\overline{\psi}\rangle\langle\overline{\psi}| \mapsto (1-p)I|\overline{\psi}\rangle\langle\overline{\psi}|I + pZ_i|\overline{\psi}\rangle\langle\overline{\psi}|Z_i, \tag{2.40}$$

which is a mixture of the pure states $|\overline{\psi}\rangle$ and $Z_i|\overline{\psi}\rangle$. Now, we know that the full error correction procedure corrects these two pure states:

$$|\overline{\psi}\rangle \mapsto |\overline{\psi}\rangle|I\rangle_{\text{syn}} \tag{2.41}$$

$$Z_i|\overline{\psi}\rangle \mapsto |\overline{\psi}\rangle|Z_i\rangle_{\text{syn}} \tag{2.42}$$

By the linearity of density matrices, that means that the final state after the dephasing channel followed by error correction is:

$$(1-p)|\overline{\psi}\rangle\langle\overline{\psi}| \otimes |I\rangle\langle I|_{\text{syn}} + p|\overline{\psi}\rangle\langle\overline{\psi}| \otimes |Z_i\rangle\langle Z_i|_{\text{syn}} = |\overline{\psi}\rangle\langle\overline{\psi}| \otimes \left[(1-p)|I\rangle\langle I|_{\text{syn}} + p|Z_i\rangle\langle Z_i|_{\text{syn}}\right]. \tag{2.43}$$

That is, the decoded logical qubit ends up as a pure state, but the overall state is still mixed: the randomness introduced by the error ends up in the error syndrome, or in the ancilla Hilbert space $\mathcal{H}_{D'}$ in the proof of theorem 2.4.

Recall that a t-qubit error channel is one for which there exists a Kraus decomposition where each Kraus operator is a sum of weight t linear operators. By theorem 2.4 and the argument above for the dephasing channel, any code that corrects arbitrary t-qubit linear errors will therefore also correct arbitrary t-qubit error channels.

What about independent channels?

Theorem 2.6. *Let $\mathcal{I}$ be the 1-qubit identity channel and $\mathcal{E} = \otimes_{i=1}^n \mathcal{E}_i$ be an n-qubit independent channel, with $\|\mathcal{E}_i - \mathcal{I}\|_\diamond < \epsilon \leq \frac{t+1}{n-t-1}$, and let U and $\mathcal{D}$ be the encoder and decoder for a QECC with n physical qubits that corrects t-qubit errors. Then*

$$\|\mathcal{D} \circ \mathcal{E} \circ U - \mathcal{I}\|_\diamond < 2\binom{n}{t+1}(e\epsilon)^{t+1}. \tag{2.44}$$

In other words, a QECC that can correct t-qubit errors also corrects small independent error channels up to a very good approximation. A similar statement immediately follows from theorem 1.1. This should thus be viewed as a specialization of theorem 1.1 with better constant factors and a much easier proof.

Proof. The strategy is straightforward: Expand $\mathcal{E}$ as a sum of a t-qubit error map (not normalized) and an additional small term. The t-qubit error map is corrected by the code due to linearity, and then we just have to bound the size of the additional term to prove the theorem.

Since $\|\mathcal{E}_i - \mathcal{I}\|_\diamond < \epsilon$, we can write $\mathcal{E}_i = \mathcal{I} + \delta\mathcal{E}_i$, with $\|\delta\mathcal{E}_i\|_\diamond < \epsilon$. Now,

$$\mathcal{E} = \mathcal{F} + \mathcal{G}, \tag{2.45}$$

where

$$\mathcal{F} = \sum_{r=0}^{t} \sum_{|S|=r} \bigotimes_{i \in S} \delta\mathcal{E}_i \tag{2.46}$$

$$\mathcal{G} = \sum_{r=t+1}^{n} \sum_{|S|=r} \bigotimes_{i \in S} \delta\mathcal{E}_i \tag{2.47}$$

are the sums over all tensor factors of $\leq t$ and $> t$ of the $\delta\mathcal{E}_i$s, respectively. (The tensor with $\mathcal{I}$ on other qubits is implicit.)

The first term $\mathcal{F}$ is a sum of terms acting on at most t qubits. Each term in the sum is not a completely positive map. In fact, $\delta\mathcal{E}_i$ is not even positive. However, $\delta\mathcal{E}_i$ is a difference of two completely positive maps, and therefore

$$\mathcal{F}(\rho) = \sum_k \alpha_k A_k \rho A_k^\dagger, \tag{2.48}$$

where the α_k coefficients can be either positive or negative real numbers and each A_k acts on at most k qubits. By theorem 2.4 (and the linearity of density matrices, as for the dephasing channel example), the QECC corrects $\mathcal{F}$.

By proposition 2.2, the scaling factor $c(E, |\psi\rangle)$ in definition 2.1, the definition of a QECC, does not depend on $|\psi\rangle$, which implies that

$$\mathcal{D} \circ \mathcal{F} \circ U = c\mathcal{I} \tag{2.49}$$

for some constant c.

Meanwhile,

$$\|\mathcal{G}\|_\diamond \leq \sum_{r=t+1}^{n} \sum_{|S|=r} \prod_{i \in S} \|\delta\mathcal{E}_i\|_\diamond \tag{2.50}$$

$$\leq \sum_{r=t+1}^{n} \binom{n}{r} \epsilon^r \tag{2.51}$$

$$\leq \binom{n}{t+1} (e\epsilon)^{t+1} = \delta \tag{2.52}$$

by lemma 1.2.

Now, $\mathcal{E}$, $\mathcal{D}$, and U are CPTP maps, so $\|\mathcal{D} \circ \mathcal{E} \circ U\|_\diamond = 1$. Thus,

$$1 - \delta \leq \|\mathcal{D} \circ \mathcal{E} \circ U\|_\diamond - \|\mathcal{D} \circ \mathcal{G} \circ U\|_\diamond \leq \|\mathcal{D} \circ \mathcal{F} \circ U\|_\diamond = \|c\mathcal{I}\|_\diamond = c \leq \|\mathcal{D} \circ \mathcal{E} \circ U\|_\diamond + \|\mathcal{D} \circ \mathcal{G} \circ U\|_\diamond \leq 1 + \delta. \tag{2.53}$$

That is, $|1 - c| \leq \delta$. Therefore,

$$\|\mathcal{D} \circ \mathcal{E} \circ U - \mathcal{I}\|_\diamond = \|\mathcal{D} \circ \mathcal{F} \circ U + \mathcal{D} \circ \mathcal{G} \circ U - \mathcal{I}\|_\diamond \tag{2.54}$$

$$= \|(c-1)\mathcal{I} + \mathcal{D} \circ \mathcal{G} \circ U\|_\diamond \tag{2.55}$$

$$\leq |c - 1| + \|\mathcal{G}\|_\diamond \tag{2.56}$$

$$\leq 2\delta. \tag{2.57}$$

□

2.5 The Quantum Error Correction Conditions

2.5.1 Sufficient Conditions (Non-degenerate Orthogonal Coding)

Recall the argument that we used to show that the nine-qubit code could correct phase errors. Very little about it was specific to the nine-qubit code. We said that phase errors generated subspaces which were orthogonal to the code space and to each other, and that therefore there was a measurement that distinguished

them. We can use this same argument to give a general sufficient condition to have a QECC: Suppose that $Q \subseteq \mathcal{H}_N$ is the code space, and let $E(Q)$ be the subspace formed by acting on Q with E. Then we want that for any pair of errors $E_1, E_2 \in \mathcal{E}$, the subspace $E_1(Q)$ is orthogonal to $E_2(Q)$. Then we will be able to make a measurement that identifies which subspace we are in and therefore identifies the error.

To be sure we have a QECC, we need a little bit more. After identifying the error, we need to be able to reverse it. That is only possible if $E|_Q$ is a unitary map from Q to $E(Q)$. $E|_Q$ is unitary iff $\langle\psi|E^\dagger E|\phi\rangle = \langle\psi|\phi\rangle$ for all $|\psi\rangle, |\phi\rangle \in Q$. Thus, we get the sufficient condition that $(Q, \mathcal{E})$ is a QECC if $\forall|\psi\rangle, |\phi\rangle \in Q$, $\forall E_a, E_b \in \mathcal{E}$,

$$\langle\psi|E_a^\dagger E_b|\phi\rangle = \delta_{ab}\langle\psi|\phi\rangle. \tag{2.58}$$

A code that satisfies this condition could be called a *non-degenerate orthogonal code*, although the term "orthogonal code" is not widely used.

2.5.2 The QECC Conditions

However, by looking at the nine-qubit code, we can already see that equation (2.58) is not a necessary condition. If $E_1 = Z_1$ and $E_2 = Z_2$, then equation (2.58) fails since $E_1|\overline{\psi}\rangle = E_2|\overline{\psi}\rangle$. We need to generalize slightly:

Theorem 2.7 (QECC Conditions)**.** *$(Q, \mathcal{E})$ is a QECC iff $\forall|\psi\rangle, |\phi\rangle \in Q$, $\forall E_a, E_b \in \mathcal{E}$,*

$$\langle\psi|E_a^\dagger E_b|\phi\rangle = C_{ab}\langle\psi|\phi\rangle. \tag{2.59}$$

Note that C_{ab} does not depend on $|\psi\rangle$ or $|\phi\rangle$.

Proof. $\Leftarrow$: Recalling theorem 2.4, we might as well pick a useful spanning set for $\mathcal{E}$ and restrict attention to that spanning set. Taking the adjoint of equation (2.59) and putting in $|\phi\rangle = |\psi\rangle$, we find that $C_{ab}^\dagger = C_{ba}^*$, i.e., the matrix C is Hermitian. Therefore C_{ab} is diagonalizable, and by choosing an appropriate spanning set $\{F_a\}$ for $\mathcal{E}$ we can actually diagonalize C_{ab}. (Note that it is not necessarily true that $F_a \in \mathcal{E}$, but that is not particularly relevant, it just means that the original set $\mathcal{E}$ is smaller than the maximal set of possible errors the code can correct.)

We have

$$\langle\psi|F_a^\dagger F_b|\phi\rangle = d_a\delta_{ab}\langle\psi|\phi\rangle. \tag{2.60}$$

This is almost equation (2.58), but the coefficient d_a can be different from 1. (d_a is an eigenvalue of C_{ab}, so it must be real.) However, it is still true that the different subspaces $F(Q)$ are orthogonal to each other, so we can make a measurement that determines which error F_a we have. It is not true that $F_a|_Q$ must be unitary, but if d_a is nonzero, $F_a|_Q$ can be written as a unitary followed by a uniform rescaling. The decoding then gives the original state rescaled by d_a. This is allowed by definition 2.1.

If d_a is zero, then $F_a|_Q$ cannot be inverted, since there is no state left. Formally, we are still OK, since in definition 2.1, we would just get that $c(F_a, |\psi\rangle) = 0$. Physically, this means that the error F_a *never occurs*. It has probability proportional to $\langle\psi|F_a^\dagger F_a|\psi\rangle$, which is 0.

Therefore, equation (2.59) gives sufficient conditions to have a QECC.

$\Rightarrow$: Recall that by proposition 2.2, $c(E, |\psi\rangle)$ and $|A(E, |\psi\rangle)\rangle$ don't depend on $|\psi\rangle$. More generally, if we purify the decoder to V, we get a unitary transformation, which preserves inner products:

$$\langle\overline{\psi}|E_a^\dagger E_b|\overline{\phi}\rangle = (\langle\overline{\psi}| \otimes \langle 0|)E_a^\dagger V^\dagger V E_b(|\overline{\phi}\rangle \otimes |0\rangle) \tag{2.61}$$

$$= \sqrt{c(E_a)c(E_b)}\,(\langle\psi| \otimes \langle A(E_a)|)(|\phi\rangle \otimes |A(E_b)\rangle) \tag{2.62}$$

$$= \sqrt{c(E_a)c(E_b)}\,\langle A(E_a)|A(E_b)\rangle\langle\psi|\phi\rangle. \tag{2.63}$$

This is equation (2.59) with $C_{ab} = \sqrt{c(E_a)c(E_b)}\langle A(E_a)|A(E_b)\rangle$.

□

2.5.3 Degenerate Codes

The difference between equation (2.58) and equation (2.59) consists in the fact that C_{ab} might not be δ_{ab}. When C_{ab} has maximum rank, this difference has no deep meaning, since we have just made a bad choice of basis errors. As in the proof of theorem 2.7, we can choose a different set of errors with the same span to diagonalize C_{ab}, and we can even rescale to make $C_{ab} = \delta_{ab}$.

When C_{ab} has non-maximal rank, we cannot do this. If the error set $\mathcal{E}$ is not linearly independent, C_{ab} cannot have maximal rank, since a linearly dependent set of errors will produce a linearly dependent set of rows in C_{ab}. The interesting case is when $\mathcal{E}$ is linearly independent. It is possible then to still have a QECC with non-maximal rank for C_{ab}. This is known as a degenerate code.

Definition 2.2. Suppose $(U, \mathcal{E})$ is a QECC and $\mathcal{E}$ is linearly independent. Then the code is *degenerate* if $\text{rank}(C_{ab}) < |\mathcal{E}|$. A code $(U, \mathcal{E})$ (for linearly dependent $\mathcal{E}$) is degenerate if $(U, \mathcal{E}')$ is degenerate, where $\mathcal{E}'$ is a minimal spanning set for $\mathcal{E}$. If a code is not degenerate, it is *non-degenerate*.

We have already seen an example of a degenerate code: the nine-qubit code. The essence of degeneracy is that different errors will produce the same result (or at least linearly dependent results) when acting on a codeword. In the nine-qubit code, any two Z errors acting on the same set of 3 qubits will produce the same result. For instance,

$$Z_1|\overline{0}\rangle = Z_2|\overline{0}\rangle = \frac{1}{2\sqrt{2}}(|000\rangle - |111\rangle)(|000\rangle + |111\rangle)(|000\rangle + |111\rangle) \tag{2.64}$$

$$Z_1|\overline{1}\rangle = Z_2|\overline{1}\rangle = \frac{1}{2\sqrt{2}}(|000\rangle + |111\rangle)(|000\rangle - |111\rangle)(|000\rangle - |111\rangle). \tag{2.65}$$

2.5.4 Distance

The most common situation when designing a QECC is to let the set of possible errors be all t-qubit errors. It is therefore worth examining theorem 2.7 more explicitly for a t-error correcting code. When E_a and E_b are both weight t errors, then $E_a^\dagger E_b$ is a weight $2t$ error.

Definition 2.3. Let $C \subseteq \mathcal{H}_N$ be a QECC. The *distance* of C is the minimum weight d of an error F such that

$$\langle\psi|F|\phi\rangle \neq c(F)\langle\psi|\phi\rangle, \tag{2.66}$$

where $|\psi\rangle$ and $|\phi\rangle$ run over all possible pairs of codewords of C.

Note that the notion of distance implies that the Hilbert space is broken up into qubits because weight is defined in terms of qubits. Naturally, we can easily generalize the distance to work with a code over qudits of dimension q by defining weight in terms of the number of qudits in the support of an operator.

We get the following corollary of theorem 2.7:

Corollary 2.8. *A distance d code corrects $\lfloor (d-1)/2 \rfloor$ errors.*

Inverting this formula, we find that to correct t errors, a code needs distance $2t + 1$. If the distance is even, the extra point is "wasted" for this application, but as we shall see in a moment, the extra point of distance can be helpful in alternative applications.

The three central properties of a QECC are the size of the logical subspace, the number of physical qubits, and the distance, so there is a notation encapsulating those properties.

Notation 2.4. A QECC which encodes $\mathcal{H}_K$ into $\mathcal{H}_{2^n}$ and has distance d is denoted as an $((n, K, d))$ code. If the physical Hilbert space is n qudits each of dimension q, it is an $((n, K, d))_q$ code. If the distance is unknown or irrelevant, it is an $((n, K))$ code or an $((n, K))_q$ code.

This notation is derived from an analogous one for classical error-correcting codes, which will be introduced in chapter 4. The classical notation uses single parentheses, and the double parenthesis indicates that we have a quantum error-correcting code.

Sometimes when we discuss a code we consider it to have a distance less than its true distance or to correct fewer errors than the maximum number it can correct. This is just a convenience indicating that we are ignoring some of the error-correcting capability of the code. However, the true distance of a QECC does give us the tightest estimate of the number of errors it can correct — that is, the converse of the corollary also holds. In order to get the definition of distance, we need F to run over all errors of weight $< d$, but the QECC conditions applied to a t-error correcting code only gives us equation (2.66) for errors of the form $E_a^\dagger E_b$, which does not include all possible weight $2t$ errors. However, equation (2.66) is linear in F, so it is sufficient to check the formula for a basis of the set of weight $2t$ errors, and the set of errors of the form $E_a^\dagger E_b$ does include such a basis.

By similar reasoning, you get the same notion of distance if you alter the definition of distance to use all d-qubit errors or just weight d tensor products of I, X, Y, and Z.

When describing a code without an explicit set of correctable errors, I said in the definition that you are supposed to determine the set of correctable errors by context. Typically, we will choose the error set to be the set of t-qubit errors, which makes the distance of a code one of its most critical properties. When the error set is given implicitly, we can define degeneracy in terms of the code's capability as a t-error correcting code.

Definition 2.5. Let $Q \subseteq \mathcal{H}_N$ be a QECC with distance $d = 2t + 1$. Then Q is *degenerate* if it is degenerate when the set of correctable errors is all t-qubit errors.

If the distance is even, $2t+2$, and there is no more structure, it makes the most sense to define degeneracy by also considering the set of correctable errors to be t-qubit errors, but for certain families of codes (such as stabilizer codes, discussed in chapter 3), we can do better.

2.5.5 Quantum Error Detection and Erasure Errors

The distance is useful information about a QECC partially because it tells you how many errors the code can correct, but also because it encapsulates some additional error-correction-related properties of the code. In particular, the distance also tells you about the code's ability to detect errors without correcting them and about the code's ability to correct erasure errors. These applications can take advantage of even distance codes, whereas correcting general t-qubit errors only needs odd distance codes.

Definition 2.6. An encoder U (defined as for a QECC) and a set of detectable errors $\mathcal{E}$ form a *quantum error-detecting code* if they have the following property: Let Π be the projector on the code space. Then $\Pi E|\psi\rangle = c(E, |\psi\rangle)|\psi\rangle$, for all $E \in \mathcal{E}$ and all codewords $|\psi\rangle$. The code space is defined as the image of U, as for a QECC, and we frequently refer to the code space as defining the error-detecting code instead of the encoder.

Based on the definition of an error-detecting code, the measurement $(\Pi, I - \Pi)$ will either project us back on the original state (with some probability $|c(E, |\psi\rangle)|^2$) or will identify that an error occurred. This condition can be easily rewritten in similar terms to the QECC conditions:

Theorem 2.9. *$(U, \mathcal{E})$ is a quantum error-detecting code iff*

$$\langle\psi|E|\phi\rangle = c(E)\langle\psi|\phi\rangle \tag{2.67}$$

for all codewords $|\psi\rangle$ and $|\phi\rangle$ and all $E \in \mathcal{E}$. A code with distance d detects arbitrary $(d-1)$-qubit errors.

Based on this theorem, we can understand the distance of the code as the minimum number of qubits on which we can act to produce an undetectable error, i.e., an error with a component taking a codeword to a *different* codeword. (An error taking a codeword to itself is considered "detectable" by the definition.)

Proof. $\Leftarrow$: We can write $\Pi = \sum |\psi_i\rangle \langle\psi_i|$ where the sum runs over a basis $|\psi_i\rangle$ for the code space Q. We can then calculate $\Pi E|\psi\rangle$ using equation (2.67) to see that we have a quantum error-detecting code.

$\Rightarrow$: Equation (2.67) follows immediately from the definition of a quantum error-detecting code if we can prove that $c(E, |\psi\rangle)$ does not depend on $|\psi\rangle$. This can be done by considering a superposition $\alpha|\psi_1\rangle + \beta|\psi_2\rangle$, as in proposition 2.2.

The definition of distance then shows that a distance d code detects $d-1$ errors. □

While I have presented quantum error-correcting codes and quantum error-detecting codes as different things, clearly there is a very close connection. A code able to correct t errors will also be able to detect $2t$ errors, and vice-versa. Detecting errors and correcting errors are really just two different applications for the same code, and henceforth, I won't make a distinction between a code designed to correct errors and one designed to detect errors. Both will be referred to as QECCs.

There is no unique maximal set of correctable errors for a QECC, but there is a unique maximal set of detectable errors. The quantum error-correction conditions involve a product of two errors, and there may be more than one set of errors that will run over all possibilities for the product. However, equation (2.67) is only linear in the error, so we can define a unique set of detectable errors.

Definition 2.7. Given a subspace Q, its set of *detectable errors* is

$$\{E \text{ s.t. } \langle\psi|E|\phi\rangle = c(E)\langle\psi|\phi\rangle \,\forall |\psi\rangle, |\phi\rangle \in Q\}. \tag{2.68}$$

If $\mathcal{E}_C$ is the set of correctable errors and $\mathcal{E}_D$ is the set of detectable errors for a code, then the QECC conditions can be rephrased as $\mathcal{E}_C^2 \subseteq \mathcal{E}_D$.

A code's ability to correct erasure errors can be understood just by applying the regular QECC conditions. The interesting twist in this case is that we can apply the side classical information we have about the location of the errors to correct twice as many errors:

Theorem 2.10. *A QECC with distance d can correct $d-1$ erasure errors.*

Proof. The best way to think about an erasure-correcting code is as a set of QECCs which all have the same encoder. Each code is associated with a different error set, depending on where the erasure errors took place. Since we don't know when doing the encoding where the errors are, we have to use the same encoder in all cases. When *decoding*, however, we know where the errors are (although not what kind of errors they are), so we can choose a decoder based on the actual error set — all possible errors on the actual set of qubits erased.

Therefore we need a single code space that satisfies the QECC conditions for any error set of the form $\mathcal{E}_S$, which is the set of all possible errors with support on the set S of at most t qubits. But $\mathcal{E}_S^2 = \mathcal{E}_S$ since the product of two errors with support on S still has support on S. For a distance d code, the set of detectable errors includes all $(d-1)$-qubit errors, so when $t \leq d-1$, all sets $\mathcal{E}_S$ are subsets of the set of detectable errors. □

2.5.6 Alternate Forms of the Quantum Error Correction Conditions

There are a number of variants of the QECC conditions which are useful in different contexts. You have just seen one relating a correctable set of errors with the set of detectable errors for a code. A few more appear in the following proposition:

Proposition 2.11. *The following are equivalent to the QECC conditions given in theorem 2.7:*

1. *For all codewords $|\psi\rangle$, all pairs $E_a, E_b \in \mathcal{E}$,*

$$\langle\psi|E_a^\dagger E_b|\psi\rangle = \mathrm{tr}(|\psi\rangle\langle\psi|E_a^\dagger E_b) = C_{ab}. \tag{2.69}$$

2. *If $\mathrm{span}(\mathcal{E}) = \mathcal{E}$: For any pair of codewords $|\psi\rangle, |\phi\rangle$, any error $E \in \mathcal{E}$,*

$$\langle\psi|E^\dagger E|\phi\rangle = C(E)\langle\psi|\phi\rangle. \tag{2.70}$$

3. If $\mathrm{span}(\mathcal{E}) = \mathcal{E}$*: For any codeword* $|\psi\rangle$*, any error* $E \in \mathcal{E}$*,*

$$\mathrm{tr}(|\psi\rangle\langle\psi|E^\dagger E) = C(E). \tag{2.71}$$

4. Let $\{|\psi_i\rangle\}$ *be a basis for the code space. For any* i *and* j *and for any pair* $E_a, E_b \in \mathcal{E}$*,*

$$\langle\psi_i|E_a^\dagger E_b|\psi_j\rangle = C_{ab}\delta_{ij}. \tag{2.72}$$

Proof. The QECC conditions from equation (2.59) immediately imply all four of these variant conditions, so we only need to show the reverse direction.

To show that the standard QECC conditions follow from the first variant above, pick arbitrary $|\phi_1\rangle$ and $|\phi_2\rangle$ in the code space and consider $|\psi\rangle = \alpha|\phi_1\rangle + \beta|\phi_2\rangle$ for different values of α and β. $|\phi_1\rangle$ and $|\phi_2\rangle$ may not be orthogonal, so we have

$$1 = |\alpha|^2 + |\beta|^2 + 2\,\mathrm{Re}(\alpha^*\beta\langle\phi_1|\phi_2\rangle). \tag{2.73}$$

Now, $|\psi\rangle$ is also in the code space, and

$$\begin{aligned} C_{ab} &= \langle\psi|E_a^\dagger E_b|\psi\rangle & (2.74)\\ &= |\alpha|^2\langle\phi_1|E_a^\dagger E_b|\phi_1\rangle + |\beta|^2\langle\phi_2|E_a^\dagger E_b|\phi_2\rangle + \alpha^*\beta\langle\phi_1|E_a^\dagger E_b|\phi_2\rangle + \alpha\beta^*\langle\phi_2|E_a^\dagger E_b|\phi_1\rangle & (2.75)\\ &= (|\alpha|^2 + |\beta|^2)C_{ab} + (\alpha^*\beta\langle\phi_1|E_a^\dagger E_b|\phi_2\rangle + \alpha\beta^*\langle\phi_2|E_a^\dagger E_b|\phi_1\rangle). & (2.76) \end{aligned}$$

If we first plug in $\alpha = \beta = 1/\sqrt{2(1 + \mathrm{Re}\langle\phi_1|\phi_2\rangle)}$, we find

$$\langle\phi_1|E_a^\dagger E_b|\phi_2\rangle + \langle\phi_2|E_a^\dagger E_b|\phi_1\rangle = C_{ab}(\langle\phi_1|\phi_2\rangle + \langle\phi_2|\phi_1\rangle). \tag{2.77}$$

Plugging in $\alpha = -i\beta = 1/\sqrt{2(1 - \mathrm{Im}\langle\phi_1|\phi_2\rangle)}$, we get

$$\langle\phi_1|E_a^\dagger E_b|\phi_2\rangle - \langle\phi_2|E_a^\dagger E_b|\phi_1\rangle = C_{ab}(\langle\phi_1|\phi_2\rangle - \langle\phi_2|\phi_1\rangle). \tag{2.78}$$

Putting these two equations together, we find

$$\langle\phi_1|E_a^\dagger E_b|\phi_2\rangle = C_{ab}\langle\phi_1|\phi_2\rangle, \tag{2.79}$$

as desired.

The second and third variants use a similar argument for E_a and E_b. The proof of the fourth version is left as an exercise. □

Applying a similar argument to definition of the set of detectable errors, we find that an operator E is detectable iff $\mathrm{tr}(\rho E)$ does not depend on the codeword ρ. This has an interesting interpretation: it says that an operator is detectable if and only if measurement of that operator reveals no information about the logical state of the code. Certainly, if measuring an operator does reveal information about the logical state, it will create errors in the encoded state. What is perhaps surprising is that first, that some element of that error cannot be detected, and second, that revealing encoded information is the *only* thing that prevents an error from being detectable.

Another version of this insight appears when we apply the QECC conditions specifically to erasure errors.

Proposition 2.12. *Let* $\mathcal{E}$ *be a set of erasure errors, each one corresponding to a set of erased qubits (or qudits). Then* Q *corrects* $\mathcal{E}$ *iff* ρ_S *is the same for all logical states* $|\psi\rangle$ *whenever* $S \in \mathcal{E}$*. Here* S *is a subset of qubits that can be erased and* ρ_S *is the encoded state restricted to* S*.*

Note that the proposition just says that Q can correct for erasures on the set S iff the codeword on S has no information about the encoded state. Another interesting feature of this is that the ability to correct for erasures only refers to single sets, not pairs of sets, whereas more generally the QECC conditions refer to pairs of errors. This means that any QECC has a unique maximal set of erasure errors that it can correct, whereas, as I mentioned before, there is not a unique set of general errors that can be corrected — there is some tradeoff between correcting different kinds of errors.

Proof. As in the proof of theorem 2.10, we think of the code as a set of QECCs labelled after-the-fact by the set of erased qubits. We specialize to a single set $S \in \mathcal{E}$ and want to show that Q corrects erasures on S iff ρ_S is independent of the encoded state. Once we fix S, correcting erasures on S is equivalent to correcting arbitrary linear operators supported on S.

Let $E_a = |k\rangle\langle j|$ and $E_b = |k\rangle\langle i|$ where i, j, and k are bitstrings labelling basis vectors for the qubits just in S. We use variant 1 of the QECC conditions from proposition 2.11. Since E_a and E_b are always supported on S,

$$\begin{aligned} \operatorname{tr}(|\psi\rangle\langle\psi| E_a^\dagger E_b) &= \langle i|\rho_S|j\rangle\langle k|k\rangle && (2.80) \\ &= (\rho_S)_{ij}. && (2.81) \end{aligned}$$

As we let i and j vary over all possible basis states for S, equation (2.69) says that ρ_S does not depend on which codeword $|\psi\rangle$ we have, proving the forward direction of the proposition.

The reverse direction follows easily from variant 1 or variant 3 of proposition 2.11: If ρ_S is independent of $|\psi\rangle$, then taking the trace of ρ with operators on S will also give something independent of $|\psi\rangle$. □

2.6 What Makes a Good QECC?

Before we get any further in our discussion of quantum error-correcting codes, it is worth stopping to think about what we want out of a code. The first thing that comes to mind is that we want it to be good at correcting errors, so we want the set $\mathcal{E}$ of correctable errors to be large. When we're dealing with a t-error channel, we want the distance of the QECC to be large.

In order to correct errors, we have to add some extra qubits. For instance, to make the 9-qubit code, we had to add 8 extra qubits to encode 1. That 9 : 1 ratio seems pretty bad, so we'd like to find codes that use less overhead. In other words, we want k, the number of encoded qubits, to be large, and n, the number of physical qubits, to be small. Certainly we need $n > k$, but we'd like the *rate* k/n to be as close to 1 as possible. It's not clear a priori whether it's better to have a code with large k or small k, so we'd like good examples of codes for any value of k.

In general, if we fix t (the number of errors corrected) and let k get larger, we can make k/n larger too. However, this is rarely a sensible thing to do. As n gets larger, there are more opportunities for errors, so the expected number of errors gets larger. A better plan is to fix t/n when we vary n. In that scenario, there is a definite limit on the rate k/n that can be achieved, and we'd like to get as close to it as possible. We'd also like to know, theoretically, what the best possible rate is so that we know what to shoot for when designing codes. A family of codes that has both k/n and t/n as constants when n gets arbitrarily large is known as a *good* family of codes, and such code families are known.

The three parameters n, k, and d encapsulate the most interesting properties of a code, but not the only interesting ones. In part II, we'll talk about fault-tolerant quantum computation, and we'd like codes that are good for doing fault-tolerant protocols. We'd like a code that has a nice succinct classical description — describing even a single state on n qubits could require specifying 2^n amplitudes, and to specify a QECC we might need to list all 2^k states in a basis for the code subspace. Preferably, the n and k values of the code will be in the correct size range for the application we have in mind — we wouldn't want to have to add lots of extra logical qubits in order to use an otherwise nice code.

We'd also like the encoding and decoding operations to be implementable with a reasonable number of quantum gates. Frequently, when there is a short description of the code, the encoder can be performed with a small circuit. For instance, this is the case for the stabilizer codes which will be introduced in chapter 3. Ideally, the encoder would run in time $O(n)$ for a code with n physical qubits, and that is harder to achieve. (An arbitrary stabilizer code takes time $\Omega(n^2/\log n)$ to encode.) We might want additional properties, for instance that the encoding circuit can be implemented easily in a two-dimensional layout.

Decoding is a much more sticky problem. Even codes with a simple description may have a very difficult decoder. For the 9-qubit code, we had an error syndrome which identified the error. The error syndrome is not well-defined for all QECCs, but even if we restrict attention to codes with an error syndrome, we are faced with the *syndrome decoding problem* of determining the actual error given the error syndrome.

The syndrome decoding problem is NP-hard for most broad classes of codes. (This is true even for classical error-correcting codes.) When we come up with new codes, we'd like ones for which the decoding problem is not nearly as hard. Again, the ideal solution would be a code for which the syndrome decoding problem can be solved in linear time. Such codes exist, but they are somewhat rare.

This is a rather long wish list of properties we'd like out of a code. It's actually possible to satisfy many of them at the same time, but we don't currently know of any QECCs that are perfect in all respects. In other words, choosing a code is a matter of trade-offs. We'll want to use one code for some purposes and a different code for other purposes, depending on which property is most desirable for our current goal.

One thing to bear in mind when looking for a new code is that codes which look different may have pretty much the same properties. For instance, if you switch the first and second qubits of a code, you probably have a different subspace than you did before, but it doesn't really deserve the name of a new code. We capture this intuition with the notion of equivalent codes:

Definition 2.8. Two QECCs C and C' on n physical qubits are *equivalent* if C' can be produced from C by performing some set of single-qubit unitaries and by permuting the qubits.

The notion of equivalence can of course be extended to codes on a q^n-dimensional Hilbert space as well. Two equivalent codes have the same basic properties.

Proposition 2.13. *If C and C' are equivalent, then C and C' have the same number of logical qubits and the same distance.*

It's not necessarily true that C and C' correct exactly the same set of errors, since the local unitaries and permutations of the qubits can scramble up the errors. However, it's always true that an equivalence will map errors into errors of the same weight. It's also true that a fast decoding procedure for C will give a fast decoding procedure for C', so the codes don't differ in computational complexity either.

When you're looking for new codes, if you find a code that's equivalent to a known code, you probably shouldn't consider it to be new. It's not always straightforward to tell if two codes are equivalent, however. The safest thing is to find codes with new (hopefully better) parameters $((n, K, d))$ than preexisting codes.

Chapter 3

Will The Real Codeword Please Stand Still?: Stabilizer Codes

The treatment of QECCs in the last chapter was very general, but a bit unwieldy, particularly when it comes to finding new codes. We'd like a better method of discussing, manipulating, and finding QECCs, even if it comes at the cost of restricting somewhat the codes we can talk about. This chapter introduces the class of stabilizer codes, which have some nice properties and are much more tractable for most purposes than a general QECC. Stabilizer codes are built around a group of symmetries of the code that can distinguish between correct codewords and states with errors: The incorrect states will change under the action of the symmetry, while the correct ones will stay put.

3.1 The 9-Qubit Code Revisited

3.1.1 Error Syndrome Measurement for the 9-Qubit Code

The 9-qubit code is a stabilizer code, so I'll introduce the basic idea of a stabilizer code with some further examination of the nine-qubit code. Let's consider more carefully exactly what we measure when we measure the error syndrome for the nine qubit code. There are two parts to it: We measure each set of three qubits to see if one is different in the standard basis, and then we compare the three phases of the three sets of three qubits to see if one of the phases is different from the other two.

Recall that to determine if one of three qubits is different, we make two measurements. One measurement determines the parity of the first two qubits, and the other tells us the parity of the second and third qubits. Let us put that in a more quantum language: Determining the parity of two qubits is the same as measuring the eigenvalue of $Z \otimes Z$ on those qubits.

$$Z \otimes Z = \begin{pmatrix} +1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & +1 \end{pmatrix} \begin{matrix} 00 \text{ (even)} \\ 01 \text{ (odd)} \\ 10 \text{ (odd)} \\ 11 \text{ (even)} \end{matrix} \tag{3.1}$$

Eigenvalue $+1$ corresponds to even parity (syndrome bit 0) and eigenvalue -1 corresponds to odd parity (syndrome bit 1). Thus, within each set of three, to figure out if one qubit is different in the standard basis, we should measure $Z \otimes Z \otimes I$ and $Z \otimes I \otimes Z$.

We also need to determine whether two sets of three have the same phase or opposite phase. We'd like to phrase this as measurement of an eigenvalue. It should probably involve X, since the eigenvalues of X tell us the phase in $|0\rangle \pm |1\rangle$, but we actually have an entangled state. To determine the phase of $|000\rangle \pm |111\rangle$ we should instead measure the eigenvalue of $X \otimes X \otimes X$. To determine whether two sets of three have the same phase or opposite phase, we should thus measure the eigenvalue of the tensor product of six X's on

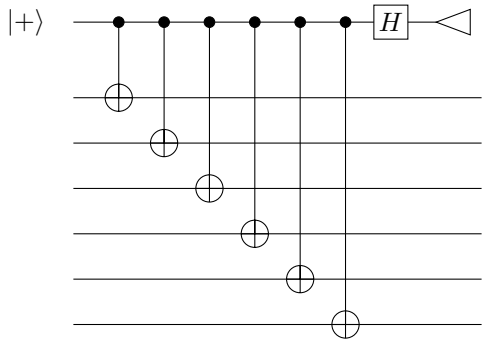


Figure 3.1: a) A non-fault-tolerant measurement procedure for $X \otimes X \otimes X \otimes X \otimes X \otimes X$.

Z	Z							
	Z	Z						
			Z	Z				
				Z	Z			
						Z	Z	
							Z	Z
X	X	X	X	X	X			
			X	X	X	X	X	X

Table 3.1: The generators of the stabilizer for the nine-qubit code. Blank spaces represent I operators.

the six relevant qubits. Remember, we want to learn whether the phases are the same or different without learning the actual value of each phase.

In case you're curious as to how one would actually measure this eigenvalue, you can look at figure 3.1. The question of how to measure the eigenvalue of a tensor product of arbitrary Paulis will be discussed in detail in section 12.1. Most of the focus of that section is on how to do the measurement fault-tolerantly, but it starts by discussing how to make a non-fault-tolerant measurement.

3.1.2 The Stabilizer for the 9-Qubit Code

The error syndrome for the 9 qubit code is 8 bits long: 2 bits for each set of three, identifying a bit flip error within that group of qubits, plus 2 additional bits telling us if and where a phase error has occurred. Putting together the list of all the operators whose eigenvalues give us those syndrome bits, we get the list in table 3.1. Each row corresponds to one operator, and the operator in column i indicates how that operator acts on qubit i. Blank spaces should be interpreted as I. Thus, the first row is the operator $Z \otimes Z \otimes I \otimes I \otimes I \otimes I \otimes I \otimes I \otimes I$. In other stabilizers, sometimes I will include the Is and sometimes omit them, whichever is most aesthetically appealing. (OK, maybe clarity plays a role too. And whimsy, definitely whimsy.)

As I mentioned in section 2.2, the choice of what qubits to measure for the error syndrome is not unique. For instance, we could have compared the parity of the first and third qubits. The parity would then be given by the eigenvalue of the operator $Z_1 Z_3$. Notice that $Z_1 Z_3$ is the product of the first two rows of table 3.1, and naturally the eigenvalue of the product will be the product of the eigenvalues. In general, we could use any product M of the operators listed in the table. The eigenvalue of M will tell us some information about the errors, but it won't be *new* information, since the eigenvalue of M can be determined from the eigenvalues of the operators used in the product.

Really, the operators we can measure to tell us about the error form a group, known as the *stabilizer* of

the nine-qubit code. The eight operators listed in table 3.1 are the *generators* of the stabilizer. Choosing a different way of defining the error syndrome of the code corresponds to choosing a different set of generators of the stabilizer.

Notice that we use tensor products of Zs to identify X (bit flip) errors, and we use tensor products of Xs to identify Z (phase flip) errors. The pattern here is that the errors anticommute with the stabilizer elements used to find them, and it is this property that makes stabilizers useful for discussing quantum error-correcting codes.

3.2 Pauli Group

I warned that you would be sick of the Pauli operators by the end of this book, and this chapter will get you started. Stabilizer codes are built around the Pauli operators, as you can perhaps see from the example of the nine-qubit code. It's therefore worth examining them in more detail.

Definition 3.1. The *Pauli group* P_n is composed of tensor products of I, X, Y, and Z on n qubits, with an overall phase of ± 1 or $\pm i$.

I will refer to elements of the n-qubit Pauli group as "Pauli operators," "Pauli errors," or just "Paulis" in the future; if I need to make a distinction between P_1 and P_n, I will do so explicitly.

The Pauli group, as its name suggests, is a group: It is closed under multiplication since $Y = iXZ$, and similarly, the product of any two of X, Y, and Z is equal to the third with an overall factor of $\pm i$. In addition, $X^2 = Y^2 = Z^2 = I$. Any tensor product of Paulis is its own inverse, i.e., it squares to I, and if there is an overall phase factor of $\pm i$, it instead squares to $-I$. (I use the one-qubit I and the n-qubit identity I interchangeably.)

The one-qubit Paulis anticommute with each other

$$XZ = -ZX \tag{3.2}$$

$$YZ = -ZY \tag{3.3}$$

$$XY = -YX. \tag{3.4}$$

Of course, I commutes with X, Y, and Z, and any Pauli commutes with itself. More general pairs (P, Q) of operators from P_n always either commute ($PQ = QP$) or anticommute ($PQ = -QP$). The difference from the single qubit case is that for P_1, only trivial pairs commute (when one Pauli is I or both are the same Pauli), but for $n > 1$, there are non-trivial commuting pairs. For instance, $X \otimes X$ commutes with $Z \otimes Z$. I will sometimes use the notation $[P, Q] = PQ - QP$ for the commutator and $\{P, Q\} = PQ + QP$ for the anticommutator.

P_n has 4^{n+1} elements since there are 4^n n-fold tensor products of I, X, Y, and Z, and 4 overall phases they could have. However, for many purposes, we can ignore the phase, giving us effectively 4^n distinct Paulis. If I wish to ignore the phase, I will refer to $\hat{\mathsf{P}}_n$. Thus, $\hat{\mathsf{P}}_n \cong \mathsf{P}_n / \{I, iI, -I, -iI\}$. The elements of $\hat{\mathsf{P}}_n$ are sets of 4 operators of the form $\{\pm P, \pm iP\}$, with P some n-qubit tensor product of I, X, Y, and Z. Each element of $\hat{\mathsf{P}}_n$ can thus be associated with an unsigned Pauli operator P, and it is more convenient by far to refer to elements of $\hat{\mathsf{P}}_n$ in terms of the associated Pauli rather than as a set of Paulis with signs.

Definition 3.2. Let $P \in \mathsf{P}_n$. Then $\hat{P} \in \hat{\mathsf{P}}_n$ is the element of $\hat{\mathsf{P}}_n$ corresponding to P. Similarly, if S is a subset of P_n, then $\hat{S}$ is the subset of $\hat{\mathsf{P}}_n$ consisting of $\hat{P}$ for all $P \in S$.

I have introduced these conventions separating P_n and $\hat{\mathsf{P}}_n$ in order to be mathematically precise. In some cases, we need to work with elements or subsets of P_n, and in some cases, to get the details completely correct, it is better to work with $\hat{\mathsf{P}}_n$. However, almost always, the distinction between the two is a small technical detail. My advice is to ignore the difference between P_n and $\hat{\mathsf{P}}_n$ unless you get confused.

Any Pauli operator has eigenvalues $+1$ and -1. Each eigenspace is exactly half the Hilbert space. That is, the $+1$ and -1 eigenspaces have dimension 2^{n-1}.

One feature of the Pauli group already mentioned is that P_n spans the space of all n-qubit linear operations. The weight t Paulis span the set of t-qubit errors. As discussed in corollary 2.5, this property means that to have a t-error correcting QECC, it suffices to correct all weight t Pauli operators.

This section has only discussed the most basic properties of the Pauli group. It has many more useful properties, some of which we will encounter in future sections of the book. Don't be surprised, however, if someday you are trying to solve a problem and discover a new property of the Paulis that is not mentioned in this book. In group theory, the Pauli group is an *extraspecial* group, and it truly deserves that name.

3.3 Stabilizer Codes

3.3.1 Definition and Basic Properties of the Stabilizer

Given a general quantum error-correcting code, we can define its stabilizer in more or less the same way as we did for the nine-qubit code:

Definition 3.3. Given a QECC $T \subseteq \mathcal{H}_{2^n}$, the *stabilizer* $\mathsf{S}(T)$ is

$$\mathsf{S}(T) = \{M \in \mathsf{P}_n | M|\psi\rangle = |\psi\rangle \ \forall |\psi\rangle \in T\}. \tag{3.5}$$

In other words, the stabilizer is composed of the Pauli operators for which all codewords are $+1$ eigenstates. In principle, one could define a stabilizer consisting of all unitaries which fix every codeword, but it turns out to be most useful to concentrate on the Pauli operators.

Proposition 3.1. *The stabilizer $\mathsf{S}(T)$ of a nonempty QECC T has three basic properties:*

a) $-I \notin \mathsf{S}(T)$

b) $\mathsf{S}(T)$ *is a group*

c) $\mathsf{S}(T)$ *is Abelian*

The property of being Abelian is the least obvious, but it makes sense since we want a set of operators which have the codewords as simultaneous eigenstates. Normally, for this to be possible, the operators should commute; in general, they only need to commute on a subspace, but for Pauli operators this is only possible if they truly commute.

Proof.

a) $-I$ has no $+1$ eigenstates, so it cannot be in the stabilizer.

b) If $M, N \in \mathsf{S}(T)$, it is clear that their product is also in $\mathsf{S}(T)$: For any $|\psi\rangle \in T$,

$$MN|\psi\rangle = M|\psi\rangle = |\psi\rangle. \tag{3.6}$$

c) By the argument for property 2, $MN|\psi\rangle = NM|\psi\rangle$ for any $|\psi\rangle \in T$, $M, N \in \mathsf{S}(T)$. Thus, $[M, N]$ annihilates the codewords. Paulis either commute or anticommute. If M and N anticommute, then $[M, N] = 2MN$, which is again an element of P_n, and therefore has no 0 eigenvalues. Therefore, the only option is that M and N commute.

□

It turns out to be most useful, when dealing with stabilizers, to work the other way around. We are given (or invent) a stabilizer with the properties we desire, and then use it deduce the code.

Definition 3.4. Let $\mathsf{S} \subseteq \mathsf{P}_n$ be an Abelian group, with $-I \notin \mathsf{S}$. Then define the code space corresponding to S by

$$\mathcal{T}(\mathsf{S}) = \{|\psi\rangle \text{ s.t. } M|\psi\rangle = |\psi\rangle \ \forall M \in \mathsf{S}\} \tag{3.7}$$

X	Z	Z	X	I
I	X	Z	Z	X
X	I	X	Z	Z
Z	X	I	X	Z

Table 3.2: The generators for the five-qubit code.

In general, $T \subseteq \mathcal{T}(\mathsf{S}(T))$, but when they are actually equal, the code has special properties.

Definition 3.5. If T is a QECC with $T = \mathcal{T}(\mathsf{S}(T))$, it is a *stabilizer code*.

Stabilizer codes are also sometimes known as *symplectic codes*, *additive codes*, or GF(4) *additive codes* for reasons that will be discussed in section 3.5 and section 5.2.

If we define a code from its stabilizer, then it is always a stabilizer code. In other words, a stabilizer code is one that could be defined just by giving its stabilizer. This is a consequence of the following proposition:

Proposition 3.2. $\mathsf{S} = \mathsf{S}(\mathcal{T}(\mathsf{S}))$.

I delay the proof until section 3.5 to keep you in suspense. Also, the proof will use a tool I won't introduce until then.

When dealing with stabilizer codes, I will frequently refer to the stabilizer S as the code instead of using the rather unwieldy phrase "stabilizer of the code $\mathcal{T}(\mathsf{S})$." When $M \in \mathsf{S}$, I will call M a "stabilizer element" or some variant of that phrase. However, many people seem to find that terminology unwieldy too, and just call M a "stabilizer." I do not approve of that usage, but I can't stop you if you want to say it.

Frequently we will want to pick a particular generating set $\{M_1, \ldots, M_r\}$ for the stabilizer, as we did for the nine-qubit code. In a minimal generating set no generator is a product of other generators. When we take a product of generators, the order does not matter since the generators commute. Also, note that since $-I \notin \mathsf{S}$, all elements of S must square to I, and any power of a generator greater than 1 can be reduced. Therefore, the elements of the stabilizer are uniquely determined by taking a product

$$M_{i_1,\ldots,i_r} = \prod_j M_j^{i_j}, \tag{3.8}$$

with $i_j \in \{0, 1\}$. This implies $|\mathsf{S}| = 2^r$.

Naturally, the set of generators is not unique, but when we do have a particular generating set to work with, I will talk about its elements as "stabilizer generators" or just "generators." When you describe a stabilizer, almost always the easiest way to do so is by listing one particular set of generators. The generators are enough to define the whole stabilizer, and it is much easier to list r generators than to list all 2^r elements of the stabilizer.

The code space of a stabilizer code is defined by requiring that all the eigenvalues be $+1$, but there is nothing magical about the $+1$ eigenstates. After all, a -1 eigenstate of M is a $+1$ eigenstate of $-M$, which is still in P_n. Given any set of generators $\{M_i\}$, we could define the code equally well by taking the -1 eigenstate, but switching M_i to $-M_i$. We *can't* arbitrarily change the eigenvalues associated with non-generators because the eigenvalues of the non-generators can be deduced from the eigenvalues of the generators. The generators are independent of each other, so we can pick their eigenvalues freely, but once we've done that, it defines the eigenvalues of all operators in the stabilizer. For more on this point, see section 3.5.

I'll conclude this subsection with a second example QECC to go with the nine-qubit code. This code has five physical qubits, and is, unsurprisingly, known as the "five-qubit code." The stabilizer for the five-qubit code is given in table 3.2. The code is cyclic: if you permute the qubits cyclically, you'll get the same stabilizer. You can almost see this from table 3.2. Shifting by one qubit for generators one through three gives you generators two through four. However, the fifth permutation $Z \otimes Z \otimes X \otimes I \otimes X$ is not one of the generators. Nevertheless, it is still in the stabilizer, as the product of all four generators given in the table.

Even though it's not a generator, it still has the same status in the code as the four operators that are listed. Indeed, we could have chosen any four of these five operators as a set of generators without changing the code.

3.3.2 Projection Operator on a Stabilizer Code Subspace

In order to convert back from the representation of a stabilizer code in terms of its stabilizer to one as a subspace of a larger physical Hilbert space, we have definition 3.4, but it is sometimes helpful to have something more concrete. In particular, we'd like a projection operator that defines the subspace.

The codewords are supposed to be $+1$ eigenvectors of every element of the stabilizer, although it is sufficient to check that they are $+1$ eigenvectors of the stabilizer generators. The projector on the $+1$ eigenspace of generator M_i is $(I + M_i)/2$. Therefore, the projector on the code space of S is

$$\Pi_{\mathsf{S}} = \frac{1}{2^r} \prod_{i=1}^{r} (I + M_i), \tag{3.9}$$

when the stabilizer has r generators. Only states that are codewords — $+1$ eigenvectors of all generators — will avoid being annihilated by Π_{S}, and any codeword will be left alone by Π_{S}, as desired. Note that the order of the generators in the product does not matter since they all commute.

We can rewrite the projector Π_{S} in an interesting way by multiplying out the product. We get a sum of terms, each of which consists of a distinct product of the generators $\{M_i\}$. Based on equation (3.8), the products of the generators give us all elements of the stabilizer:

$$\Pi_{\mathsf{S}} = \frac{1}{2^r} \sum_{M \in \mathsf{S}} M. \tag{3.10}$$

We can come up with actual codewords for the code by applying Π_{S} to states in the physical Hilbert space and renormalizing. We might get 0, if the state we started with is orthogonal to the code space, but frequently we'll get a real codeword. For instance, for the five-qubit code, we can apply the projector to $|00000\rangle$ and $|11111\rangle$ to get two orthogonal codewords which can serve as a basis for the code space:

$$\begin{aligned} |\overline{0}\rangle = {} & |00000\rangle + |10010\rangle + |01001\rangle - |11011\rangle + |10100\rangle - |00110\rangle - |11101\rangle - |01111\rangle \\ & + |01010\rangle - |11000\rangle - |00011\rangle - |10001\rangle - |11110\rangle - |01100\rangle - |10111\rangle + |00101\rangle \end{aligned} \tag{3.11}$$

$$\begin{aligned} |\overline{1}\rangle = {} & |11111\rangle + |01101\rangle + |10110\rangle - |00100\rangle + |01011\rangle - |11001\rangle - |00010\rangle - |10000\rangle \\ & + |10101\rangle - |00111\rangle - |11100\rangle - |01110\rangle - |00001\rangle - |10011\rangle - |01000\rangle + |11010\rangle. \end{aligned} \tag{3.12}$$

I got these codewords by working my way through all 16 elements of the stabilizer for the five-qubit code (products of the generators given in table 3.2) and applying them to the starting states. You need to be careful of the signs involved, but otherwise the process is straightforward if tedious.

3.3.3 Distance and Size of a Stabilizer Code

When a stabilizer code is given either as a subspace or via the stabilizer, the number of physical qubits n is immediately obvious. The other two central properties (number k of logical qubits and distance d) are not so obvious, but can be deduced directly from the stabilizer.

Proposition 3.3. *If the stabilizer S on n physical qubits has $|\mathsf{S}| = 2^r$ (i.e., S has r generators), then $\mathcal{T}(\mathsf{S})$ has dimension 2^{n-r}. That is, there are $k = n - r$ logical qubits.*

Intuitively the result is easy to understand. The first generator of the stabilizer divides the Hilbert space into a $+1$ eigenspace and a -1 eigenspace of equal size. The codewords live in the $+1$ eigenspace. Each additional generator divides the subspace available for codewords in half again, so the total available dimension is 2^{n-r}. Non-generators do not divide the space since their eigenvalues can be derived by looking

at the eigenvalues of the generators. This argument is not a proof, since we would need to show that the additional generators divide not just the full Hilbert space in half, but also all the smaller subspaces defined by the earlier generators. The actual proof is straightforward, but less intuitive:

Proof. The dimension of $\mathcal{T}(\mathsf{S})$ is equal to the trace of Π_{S}, the projection operator onto $\mathcal{T}(\mathsf{S})$. Now,

$$\operatorname{tr} \Pi_{\mathsf{S}} = \frac{1}{2^r} \sum_{M \in \mathsf{S}} \operatorname{tr} M. \tag{3.13}$$

However, $\operatorname{tr} P = 0$ for all Paulis P except for the identity. Thus,

$$\operatorname{tr} \Pi_{\mathsf{S}} = \frac{\operatorname{tr} I}{2^r} = \frac{2^n}{2^r}. \tag{3.14}$$

□

You can check the nine-qubit code in this formula: 9 physical qubits, 8 stabilizer generators, and 1 encoded qubit. For the five-qubit code, we have four generators, so again there should be 1 encoded qubit, with the basis codewords we saw above.

There's one special case which is not, strictly speaking, a QECC, but is interesting nonetheless. Yes, it's true, there are some things which are interesting other than quantum error correction. When the stabilizer has n generators on n qubits, proposition 3.3 would tell us we have 0 encoded qubits, which is a Hilbert space of dimension 1. That is, it is a single state, up to normalization.

Definition 3.6. A *stabilizer state* is the code space of a stabilizer with n generators on n qubits.

Since $r = n$ is the maximum number of generators you can have, a stabilizer state is an extreme limit of a QECC. Some constructions of QECCs will alter the number of encoded qubits from another code, and sometimes a stabilizer state can be the starting or ending point of such a construction. Also, stabilizer states are fairly common in the theory of quantum information, even discounting states arising from quantum error correction. For instance, the GHZ state $|000\rangle + |111\rangle$ and the Bell states $|00\rangle \pm |11\rangle$, $|01\rangle \pm |10\rangle$ are stabilizer states.

To understand how to deduce the distance of a stabilizer code, let us go back to the nine-qubit code and recall how it handled errors. The generators of the stabilizer were used to give us bits of the error syndrome. In particular, some generators were able to signal the presence of certain errors while missing other errors, but by looking at the full set of generators, we were able to identify all of the single-qubit errors.

I mentioned that the property responsible for determining whether a generator M is useful for an error E is anticommutation. Let us see how this works. Suppose $M \in \mathsf{S}$ and $E \in \mathsf{P}_n$ anticommutes with M. Then for any $|\psi\rangle \in \mathcal{T}(\mathsf{S})$,

$$M(E|\psi\rangle) = -EM|\psi\rangle = -E|\psi\rangle. \tag{3.15}$$

$|\psi\rangle$ was a $+1$ eigenvector of M — that is the definition of the code space — but $E|\psi\rangle$ is a -1 eigenvector.

Conversely, if E commutes with M then

$$M(E|\psi\rangle) = EM|\psi\rangle = E|\psi\rangle. \tag{3.16}$$

One advantage to dealing with the Pauli group is that these are the only choices. Either M and E commute or they anticommute. If we have an error that commutes with M, M retains a $+1$ eigenvalue, whereas if the error anticommutes with M, M's eigenvalue becomes -1, signaling that an error has occurred.

Definition 3.7. The *normalizer* $\mathsf{N}(\mathsf{S})$ of the stabilizer S is

$$\mathsf{N}(\mathsf{S}) = \{N \in \mathsf{P}_n | NM = MN \ \forall M \in \mathsf{S}\}. \tag{3.17}$$

If you know some group theory, you might recognize this as the definition of the *centralizer* of S (the set of things that commute with all elements of S) rather than the *normalizer* (the set of things that preserve S under conjugation), but because Paulis either commute or anticommute and $-I \notin \mathsf{S}$, they are the same thing for a stabilizer. I am choosing to call it the normalizer rather than the centralizer because the normalizer relates to logical operations, and this is an important function of $\mathsf{N}(\mathsf{S})$, as we shall see shortly in section 3.4.2.

Since the stabilizer is Abelian, $\mathsf{S} \subseteq \mathsf{N}(\mathsf{S})$ always. Indeed, $\mathsf{N}(\mathsf{S})$ also contains $-\mathsf{S}$ and $\pm i\mathsf{S}$. Since we worry about eigenvectors of S, changing the sign of an operator in the stabilizer is important, but $\mathsf{N}(\mathsf{S})$ is about errors, and global phase no longer matters. Therefore, we will usually want to work with $\hat{\mathsf{N}}(\mathsf{S})$.

The normalizer tells us which errors can be detected by the stabilizer code. If a Pauli E is outside the normalizer $\mathsf{N}(\mathsf{S})$, then $P|\psi\rangle$ has an eigenvalue -1 for some $M \in \mathsf{S}$, and thus is detected by measuring the eigenvalues of the stabilizer elements. Note that this is true for *any* codeword $|\psi\rangle$. Also note that it is sufficient to measure the generators of the stabilizer: If N commutes with M_1 and M_2, it also commutes with $M_1 M_2$. Thus, if N commutes with all generators of S, then $N \in \mathsf{N}(\mathsf{S})$.

When $E \in \mathsf{N}(\mathsf{S})$, then $E|\psi\rangle$ has eigenvalue $+1$ for all $M \in \mathsf{S}$, and therefore $E|\psi\rangle \in \mathcal{T}(\mathsf{S})$ for codewords $|\psi\rangle$. You might think that that means it is an undetectable error, but there is actually another class of Paulis that is "detectable" by definition 2.7. If $E \in \mathsf{S}$, then, while it's true that $E|\psi\rangle \in \mathcal{T}(\mathsf{S})$ for any codeword $|\psi\rangle$, it's also true that $E|\psi\rangle = |\psi\rangle$, so E is not actually an "error": it acts like the identity on the code space, leaving codewords unchanged. Ignoring global phases, we can say the same if $\hat{E} \in \hat{\mathsf{S}}$.

Putting this together, we get a characterization of the detectable errors for a stabilizer code.

Theorem 3.4. *The set of undetectable errors for a stabilizer code* S *is* $\hat{\mathsf{N}}(\mathsf{S}) \setminus \hat{\mathsf{S}}$. *The distance of* S *is* $\min\{\operatorname{wt} E | \hat{E} \in \hat{\mathsf{N}}(\mathsf{S}) \setminus \hat{\mathsf{S}}\}$.

The slanty line is a "set minus" operation. That is, $\hat{\mathsf{N}}(\mathsf{S}) \setminus \hat{\mathsf{S}}$ consists of those elements of $\hat{\mathsf{N}}(\mathsf{S})$ that are not in $\hat{\mathsf{S}}$.

Proof. We can prove the first statement directly from the definition of the set of detectable errors (definition 2.7). The second statement will then follow immediately from the definition of distance. We need to consider three cases for E. In cases 1 and 2, $|\psi\rangle$ and $|\phi\rangle$ are arbitrary codewords.

1. Case 1: $\hat{E} \in \hat{\mathsf{S}}$. Then for some choice of phase, $E \in \mathsf{S}$ and $E|\phi\rangle = |\phi\rangle$, so

$$\langle\psi|E|\phi\rangle = \langle\psi|\phi\rangle, \tag{3.18}$$

   and $c(E) = 1$. $\hat{E}$ is detectable.

2. Case 2: $\hat{E} \notin \hat{\mathsf{N}(\mathsf{S})}$. Then $\exists M \in \mathsf{S}$ such that $\{M, E\} = 0$ (for any choice of phase of E), so $M(E|\phi\rangle) = -E|\phi\rangle$, as per equation (3.15). Then

$$\langle\psi|E|\phi\rangle = \langle\psi|MEM|\phi\rangle = -\langle\psi|E|\phi\rangle = 0, \tag{3.19}$$

   since $M^2 = I$ for stabilizer elements. Thus $c(E) = 0$ and E is detectable.

3. Case 3: $\hat{E} \in \hat{\mathsf{N}}(\mathsf{S}) \setminus \hat{\mathsf{S}}$. Since $E \notin \mathsf{S}$, $\exists$ codeword $|\phi\rangle$ such that $E|\phi\rangle \neq |\phi\rangle$. Let $|\psi\rangle = E|\phi\rangle$. Since $E \in \mathsf{N}(\mathsf{S})$, $|\psi\rangle$ is also a codeword. However,

$$\langle\psi|E|\phi\rangle = 1 = \frac{1}{\langle\psi|\phi\rangle}\langle\psi|\phi\rangle, \tag{3.20}$$

   whereas

$$\langle\phi|E|\phi\rangle = \langle\phi|\psi\rangle = (\langle\phi|\psi\rangle)\langle\phi|\phi\rangle. \tag{3.21}$$

   Comparing equations (3.20) and (3.21) to definition 2.7 tells us that $\hat{E}$ is not detectable unless $|\langle\phi|\psi\rangle| = 1$, meaning $E|\phi\rangle = e^{i\theta}|\phi\rangle$. Furthermore, by equation (3.21), $\hat{E}$ is undetectable unless θ is the same for all $|\phi\rangle$. But that is not possible, since that would imply $\hat{E} \in \hat{\mathsf{S}}$.

X	X	X	X
Z	Z	Z	Z

Table 3.3: The stabilizer for the four-qubit code.

□

The key point here is that when $E \in \hat{\mathsf{N}}(\mathsf{S}) \setminus \hat{\mathsf{S}}$, then E maps some codewords to *different* codewords. That's the essence of an undetectable error, because the code has no way of knowing if a codeword is the original encoding of the state or the result of the action of the error on a different codeword.

Moving now to correcting errors, we find

Theorem 3.5. *The stabilizer code* S *corrects a set of errors* $\mathcal{E} \subseteq \mathsf{P}_n$ *iff* $\hat{E}^\dagger \hat{F} \notin \hat{\mathsf{N}}(\mathsf{S}) \setminus \hat{\mathsf{S}}$ *for all* $E, F \in \mathcal{E}$.

The theorem follows immediately from theorem 3.4 by comparing definition 2.7 with theorem 2.7. Recall that by the linearity of QECCs, and particularly corollary 2.5, it suffices to consider Pauli errors to understand the error-correcting capabilities of the code, at least when we are interested in correcting t-qubit errors.

For stabilizer codes, we have a slightly different notation than the more general $((n, K, d))$ notation that applies to arbitrary QECCs. Since the encoded subspace has a dimension that's always a power of 2, we write the code in terms of the number of encoded qubits k rather than the dimension $K = 2^k$ of the logical Hilbert space. We also use square brackets to indicate that we are dealing with a stabilizer code.

Notation 3.8. A stabilizer code with n physical qubits, k logical qubits, and distance d is denoted as an $[[n, k, d]]$ code. If the distance is unknown or irrelevant, it is an $[[n, k]]$ code.

We know that the nine-qubit code corrects a single-qubit error, so it must have distance at least 3. In fact, there are some 3-qubit Paulis (such as $X_1X_2X_3$) in $\hat{\mathsf{N}}(\mathsf{S}) \setminus \hat{\mathsf{S}}$ for the code, so the distance is exactly 3. Thus, the nine-qubit code is a $[[9, 1, 3]]$ code. The five-qubit code also turns out to have distance 3, so it is a $[[5, 1, 3]]$ code. Note that these codes are also $((9, 2, 3))$ and $((5, 2, 3))$ codes, but the more specific notation for a stabilizer code is generally used for them. As it happens, the five-qubit code is the smallest distance 3 code. You'll see the proof of that, as well as other techniques for proving limits on QECCs, in chapter 7.

You might wonder about distance 2 codes, since all the QECCs we have seen so far have distance 3. Distance 2 codes tend to be much simpler, but they can still be interesting. After all, a distance 2 code can detect one error or correct one erasure. The smallest distance 2 code is a $[[4, 2, 2]]$ code given in table 3.3. It is not hard to see that any one-qubit Pauli will anticommute with one or both of the two generators, but there are some two-qubit Paulis (e.g., $X \otimes X \otimes I \otimes I$) that commute with both. Thus, the code has distance 2.

3.3.4 Degeneracy and Stabilizer Codes

When is a stabilizer code degenerate? In the proof of theorem 3.4, we implicitly calculated the matrix C_{ab}. In particular, we found that $C_{ab} = 0$ if $E^\dagger F \notin \mathsf{N}(\mathsf{S})$ and $C_{ab} = 1$ if $E^\dagger F \in \mathsf{S}$. Any set of Paulis is linearly independent, so we just need to check if the resulting C_{ab} has maximum rank.

If $E_1^\dagger E_2 \in \mathsf{S}$, then $E_1^\dagger F \in \mathsf{S} \Leftrightarrow E_2^\dagger F \in \mathsf{S}$, so in this case the rows (or columns) of C_{ab} associated with E_1 and E_2 are the same. When $E_1^\dagger E_2 \notin \mathsf{S}$, then only one (or neither) of $E_1^\dagger F$ and $E_2^\dagger F$ can be in S, so in this case the rows associated with E_1 and E_2 cannot have 1s in the same column. Thus, we get the following proposition:

Proposition 3.6. *A stabilizer code* S *is degenerate for the error set* $\mathcal{E}$ *iff* $\exists E_1, E_2 \in \mathcal{E}$ *with* $E_1^\dagger E_2 \in \mathsf{S}$.

As with general QECCs, we can define degeneracy as a property of the code subspace without referring to specific set of errors by considering the distance. In the case of stabilizer codes, motivated by the above analysis, we can also sensibly define degeneracy when the code is used for error detection rather than error correction.

Definition 3.9. A stabilizer code S with distance d is *degenerate* if $\exists M \in \mathsf{S}$, $M \neq I$, with $\mathrm{wt}\, M < d$.

Note that this is slightly more general than the generic notion of a degenerate code. A stabilizer code with distance $d = 2t+2$ can be degenerate if S contains any elements of weight $2t+1$, but degeneracy wasn't defined for general non-stabilizer QECCs of even distance.

Looking once more at the stabilizer of the nine-qubit code (table 3.1), it is immediately obvious that it is a degenerate code. There are many generators of weight 2 and the code has distance 3. However, it is not always obvious from looking at the generators whether a stabilizer code is degenerate or not. It may be that the set of generators given to you all have high weight, but there is some product of the generators that does not. The five-qubit code is non-degenerate because all of the operators in the stabilizer have weight 4, but to see that you need to do some work.

3.4 Cosets and Error Syndromes

3.4.1 The Error Syndrome and the Stabilizer

For the nine-qubit code, we figured out the stabilizer by thinking about what we wanted to measure for the error syndrome. Each generator corresponded to a bit of the error syndrome. In fact, this is completely general:

Definition 3.10. Suppose S is a stabilizer code with generators $M_1, \ldots, M_r$ and $|\psi\rangle$ is an eigenvector (not necessarily with eigenvalue $+1$) of all $N \in \mathsf{S}$. The *error syndrome* of $|\psi\rangle$ is an r-bit string with ith bit 0 if $|\psi\rangle$ is a $+1$ eigenvector of M_i and ith bit 1 if $|\psi\rangle$ is a -1 eigenvector of M_i. The error syndrome $\sigma(E) : \mathsf{P}_n \to \mathbb{Z}_2^r$ is the error syndrome of $E|\phi\rangle$ for any codeword $|\phi\rangle$. When the stabilizer associated with a syndrome is in doubt, we will write $\sigma_{\mathsf{S}}(E)$.

If $P, Q \in \mathsf{P}_n$, then $c(P,Q) = 0$ if P and Q commute and $c(P,Q) = 1$ if they anticommute. ($c(P,Q) : \mathsf{P}_n \times \mathsf{P}_n \to \mathbb{Z}_2$.)

For a Pauli, bit i of the error syndrome is the same as $c(E, M_i)$. Note that the syndrome of E will be the same no matter what codeword $|\psi\rangle$ we choose to evaluate. This follows from equations (3.15) and (3.16) and means that Pauli errors map the code space, which is a subspace with $+1$ eigenvalue for all stabilizer elements, to a subspace that has a different set of eigenvalues, but for which each eigenvalue is still shared for all states in the subspace.

The subspaces derived this way are also error-correcting codes, but they are associated with different eigenvalues of the stabilizer. We can reinterpret them as traditional stabilizer codes (with all $+1$ eigenvalues) by replacing M_i with $-M_i$ whenever the ith syndrome bit is 1. Note that there are 2^r possible values of the error syndrome, and each subspace is isomorphic to the code space, which has dimension 2^k. When there are n physical qubits, $r = n - k$. Also, all the subspaces of different error syndromes are orthogonal to each other, since they have different eigenvalues. Thus, the full Hilbert space $\mathcal{H}_{2^n}$ decomposes as a direct sum of the 2^{n-k} subspaces associated with different syndromes.

Proposition 3.7. *The $c(P,Q)$ and $\sigma(E)$ functions have the following properties. In all cases, $+$ refers to binary addition.*

a) $c(P_1P_2, Q) = c(P_1, Q) + c(P_2, Q)$.

b) $c(P,Q) = c(Q,P)$.

c) $\sigma(EF) = \sigma(E) + \sigma(F)$.

These are straightforward to verify.

The various error syndromes are associated with different cosets of the normalizer in P_n.

Proposition 3.8. *Let $E, F \in \mathsf{P}_n$, and S be a stabilizer. Then F and E are in the same coset of $\mathsf{N}(\mathsf{S})$ iff E and F have the same error syndrome.*

Proof.

$\Rightarrow$: If E and F are in the same coset, then $F = EN$, with $N \in \mathsf{N}(\mathsf{S})$. Let $M \in \mathsf{S}$. Then $c(F, M) = c(E, M) + c(N, M)$. But $N \in \mathsf{N}(\mathsf{S})$, so $c(N, M) = 0$. Therefore the error syndromes of E and F are the same.

$\Leftarrow$: The argument works the other way too: Let $N = E^\dagger F$. If the error syndromes of E and F are the same, then $c(N, M) = 0$ for all $M \in \mathsf{S}$. Therefore, $N \in \mathsf{N}(\mathsf{S})$.

□

In much the same way as the full Hilbert space can be written as a direct sum of subspaces associated to different syndromes, the whole Pauli group can be partitioned into cosets of $\mathsf{N}(\mathsf{S})$, each associated with a different syndrome. All the cosets are the same size and there are 2^r of them. As a consequence, we know the size of the normalizer:

Proposition 3.9. $|\mathsf{N}(\mathsf{S})| = 4 \cdot 2^{n+k}$ *when* S *has* n *physical qubits and* k *logical qubits.*

3.4.2 Cosets inside the Normalizer

We can similarly analyze the cosets of S in $\mathsf{N}(\mathsf{S})$.

Proposition 3.10. *Let* $N_1, N_2 \in \mathsf{N}(\mathsf{S})$*. Then* N_1 *and* N_2 *are in the same coset of* S *iff* $N_1|\psi\rangle = N_2|\psi\rangle$ *for all codewords* $|\psi\rangle$*.*

Proof. $N_1|\psi\rangle = N_2|\psi\rangle$ iff $|\psi\rangle$ is a $+1$ eigenstate of $M = N_1^\dagger N_2$. This is true for all codewords $|\psi\rangle$ iff $M \in \mathsf{S}$. However, N_1 and N_2 are in the same coset of S iff $N_2 = N_1 M$ with $M \in \mathsf{S}$. □

Recall that when $N \in \mathsf{N}(\mathsf{S})$, then $N|\psi\rangle$ is a codeword for any input codeword $|\psi\rangle$. Thus, N is a *logical operation*: it always maps codewords to (possibly different) codewords. Since the different representatives of a coset of S act the same way on codewords, they are all different realizations of the *same* logical operation. Thus, the set $\mathsf{N}(\mathsf{S})/\mathsf{S}$ is of particular interest, since it consists of the distinct logical operations performed by the normalizer. (S is a normal subgroup of $\mathsf{N}(\mathsf{S})$ by definition, so we can take this quotient without any difficulties.)

Theorem 3.11. *Let* S *be a stabilizer with* n *physical qubits and* k *logical qubits. Then* $\mathsf{N}(\mathsf{S})/\mathsf{S} \cong \mathsf{P}_k$*.*

The quotient group $\mathsf{N}(\mathsf{S})/\mathsf{S}$ can be taken to be the *logical Pauli group*, performing Paulis on the encoded qubits. The proof of theorem 3.11 will be in section 3.5.

We can also look at cosets of S in the full Pauli group P_n, as in figure 3.2. Each coset of $\mathsf{N}(\mathsf{S})$ breaks up into cosets of S, so we can associate each coset of S in P_n with an error syndrome, inherited from the coset of $\mathsf{N}(\mathsf{S})$ which it sits within. If we pick a particular representative E of the coset of $\mathsf{N}(\mathsf{S})$, then we can again identify the coset of S with a logical operation $\overline{P} \in \mathsf{N}(\mathsf{S})/\mathsf{S}$; the interpretation of the coset is then a combination of the logical operation $\overline{P}$ and the error E.

When we perform decoding on a stabilizer code, we do just this. For each error syndrome s, we assign a particular error E_s with $\sigma(E_s) = s$. If our syndrome measurement gives s, we assume the error actually was E_s and correct that. If the error was actually some E' with the same syndrome, we hope that E_s and E' are in the same coset of S. If not, there has been a logical Pauli error, the one associated with the actual coset of S in which E' lies.

Theorem 3.12. *If* S *is a non-degenerate stabilizer code, the error syndromes of all errors in the correctable error set* $\mathcal{E}$ *are distinct. If* S *is a degenerate code,* E *and* F *have the same error syndrome iff* $\hat{E}^\dagger \hat{F} \in \hat{\mathsf{S}}$*.*

Proof. By proposition 3.8, two errors $E \neq F \in \mathcal{E}$ have the same error syndrome iff they are in the same coset of $\mathsf{N}(\mathsf{S})$, meaning $E^\dagger F \in \mathsf{N}(\mathsf{S})$. But both errors are correctable, and by theorem 3.5, $\hat{E}^\dagger \hat{F} \notin \hat{\mathsf{N}}(\mathsf{S}) \setminus \hat{\mathsf{S}}$, so $E^\dagger F \in \mathsf{N}(\mathsf{S})$ iff $\hat{E}^\dagger \hat{F} \in \hat{\mathsf{S}}$. Thus, the error syndromes of two correctable errors E and F are the same iff $\hat{E}^\dagger \hat{F} \in \hat{\mathsf{S}}$. If this ever occurs, the code is degenerate. □

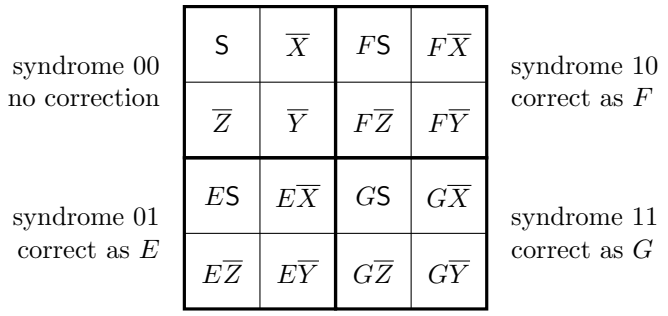


Figure 3.2: Cosets of the normalizer and stabilizer in the Pauli group. The errors E, F, and G are chosen as canonical errors for the syndromes 01, 10, and 11.

	X	Z	Z	X	I
	I	X	Z	Z	X
	X	I	X	Z	Z
	Z	X	I	X	Z
$\overline{X}$	X	X	X	X	X
$\overline{Z}$	Z	Z	Z	Z	Z

Table 3.4: The generators for the five-qubit code supplemented by representatives for logical Paulis.

Applying this theorem is one way to see that the five-qubit code has distance 3 and is non-degenerate. There are 15 possible one-qubit errors (X, Y, Z on each of five qubits), plus the identity. There are 4 generators, so there are 16 error syndromes. If you list the syndromes of the 16 errors, you'll find that each one is unique. There are no syndromes left over, so the five-qubit code is known as a *perfect* code.

It is frequently helpful to also pick representatives of the cosets of S in the normalizer. These representatives don't have an interpretation in error correction, but they do give us concrete realizations of the logical Pauli group, which is helpful in fault-tolerant computation. For now, they are simply convenient computational tools. We don't actually need to pick representatives for every coset of S. Since $\mathsf{N}(\mathsf{S})/\mathsf{S}$ has a group structure itself, it is sufficient to pick representatives of generators of $\mathsf{N}(\mathsf{S})/\mathsf{S}$ and let the generators of other cosets be determined by multiplication. In other words, we can pick representatives for $\overline{X}_i$ and $\overline{Z}_i$ for $i = 1, \ldots, k$. (The subscript i here means the operator acts on the ith logical qubit, not the ith physical qubit.) I have done this for the five-qubit code and the four-qubit code in tables 3.4 and 3.5. Then, for instance, you can get $\overline{Y}$ for the five qubit code to be $i\overline{X}\overline{Z} = Y \otimes Y \otimes Y \otimes Y \otimes Y$.

You need to be careful when assigning cosets to elements of the logical Pauli group. Remember, $\mathsf{N}(\mathsf{S})/\mathsf{S} \cong \mathsf{P}_k$, and we'd like to realize this through a choice of representatives for the generating cosets. This means that the commutation and anticommutation relationships of the logical Paulis must be realized through the coset representatives. For instance, the coset representatives for $\overline{X}_i$ and $\overline{Z}_i$ must anticommute, while the coset representative of $\overline{X}_i$ must commute with the representatives for $\overline{X}_j$ or $\overline{Z}_j$ ($j \neq i$).

We could in principle do the same thing for $\mathsf{P}_n/\mathsf{N}(\mathsf{S})$, choosing representative errors only for the basis error syndromes and deducing the other errors by multiplication. However, in most cases, this is not a good thing to do. If our goal is to correct t errors for the maximum possible t, what we'd like to do instead is choose the lowest weight error in each coset of $\mathsf{N}(\mathsf{S})$ as its representative. If we rely on multiplication to tell us the coset representatives, we'll frequently get errors of higher weight than necessary, and then there will be some lower weight errors that won't get corrected properly.

	X	X	X	X
	Z	Z	Z	Z
$\overline{X}_1$	X	X	I	I
$\overline{X}_2$	X	I	X	I
$\overline{Z}_1$	I	Z	I	Z
$\overline{Z}_2$	I	I	Z	Z

Table 3.5: The stabilizer for the four-qubit code supplemented by representatives for logical Paulis.

3.4.3 Maximum Likelihood Decoding

I'd like to depart briefly from the convention of most of this chapter, for which we had some fixed set $\mathcal{E}$ of errors that we'd like to correct, and we won't settle for anything less than correcting all of them. For the purposes of this subsection, assume we instead have some n-qubit Pauli channel, with $P \in \mathsf{P}_n$ occurring with probability p_P.

Given a stabilizer code S, we'd like to find a decoding procedure which minimizes the probability of error when the code goes through a Pauli channel. As discussed above, a decoding procedure can be understood as assigning to each error syndrome s a Pauli Q_s. When we measure s, we assume the actual error was Q_s and so perform the correction by inverting that error. If the actual error was something else, we may end up with the wrong state. In particular, if the true error P was in a different coset of S than Q_s was, we'll end up with some logical Pauli operation.

Thus, we can calculate the probability of logical error by summing over the cosets of S and $\mathsf{N}(\mathsf{S})$.

Proposition 3.13. *Let C_s be the coset of $\hat{\mathsf{N}}(\mathsf{S})$ with error syndrome s. $\hat{Q}_s\hat{\mathsf{S}}$ is the coset of $\hat{\mathsf{S}}$ containing the canonical error $\hat{Q}_s$ with syndrome s, so $\hat{Q}_s\hat{\mathsf{S}} \subseteq C_s$.*

a) The probability of error syndrome s (regardless of whether we correctly identify the error) is

$$p_s = \sum_{\hat{P} \in C_s} p_P. \tag{3.22}$$

b) The probability of having syndrome s and no logical error (i.e., error correction is successful) is

$$p_{s,\mathrm{OK}} = \sum_{\hat{P} \in \hat{Q}_s\hat{\mathsf{S}}} p_P. \tag{3.23}$$

c) The total probability of successfully decoding the state is

$$p_{\mathrm{OK}} = \sum_s \sum_{\hat{P} \in \hat{Q}_s\hat{\mathsf{S}}} p_P. \tag{3.24}$$

There is no interaction between the different error syndromes. If we change Q_s for one value of s, the only change to p_{OK} comes through the change to $p_{s,\mathrm{OK}}$. We can therefore treat each syndrome separately to maximize $p_{s,\mathrm{OK}}$. It doesn't matter which element of the coset $Q_s\mathsf{S}$ we choose, only which coset we choose within C_s.

In order to maximize the probability of successfully decoding, for each syndrome s, we should choose the coset $Q_s\mathsf{S}$ which maximizes $p_{s,\mathrm{OK}}$. As given by equation (3.23), that just means we look at each coset and add up the probability of all Paulis in the coset. Then we choose the coset with the highest value to be the representative coset for s.

The maximum likelihood decoder should be contrasted with the decoder that optimizes the distance, for which we choose the coset containing the lowest-weight Pauli. If we have an independent Pauli channel, the

two procedures frequently, but not always, give the same answer: When the probability of error per qubit is small, a specific low-weight error will have higher probability than a specific high-weight error. However, a coset which contains one low-weight error and a bunch of high-weight errors may be less likely than a coset which contains a large number of medium-weight errors.

Getting the *exactly* optimal decoder, in the sense of always picking the most likely coset, is usually a computationally challenging task. Often, therefore, we are willing to settle for a good approximate decoder that gets p_{OK} to be close to the optimal value but with a much smaller computational cost.

3.5 Binary Symplectic Representation

3.5.1 The Symplectic Representation of Paulis

One of the most useful techniques when dealing with stabilizer codes is to ignore the phases of Paulis and work with $\hat{\mathsf{P}}_n$ instead of P_n. We lose a couple of things by doing this. One is the ability to define the code space, which seems like a serious loss, but isn't really — as discussed above, the other eigenspaces of the stabilizer have the same error correction properties as the code space. The other missing thing is the ability to tell whether Paulis commute or anticommute, since $\hat{\mathsf{P}}_n$ is an Abelian group. This is a much greater loss, so we'll need to find a substitute.

The structure of $\hat{\mathsf{P}}_n$ is very straightforward. There are 4^n elements, and all pairs commute under multiplication. Therefore, $\hat{\mathsf{P}}_n \cong \mathbb{Z}_2^{2n}$. The group operation for $\mathbb{Z}_2^{2n}$ is usually written as addition. Conventionally, we choose to represent elements of $\hat{\mathsf{P}}_n$ as two n-bit binary vectors, one representing the "X" component and one representing the "Z" component.

Definition 3.11. The *binary symplectic representation* of $\hat{\mathsf{P}}_n$ is an isomorphism between $\hat{\mathsf{P}}_n$ and $\mathbb{Z}_2^n \times \mathbb{Z}_2^n$: $P \leftrightarrow v_P = (x_P|z_P)$. The ith component of x_P is 0 if P acts on qubit i as I or Z and 1 if P acts on qubit i as X or Y. The ith component of z_P is 0 if P acts on qubit i as I or X and 1 if P acts on qubit i as Y or Z. That is,

$$\begin{array}{cc} I & (0|0) \\ X & (1|0) \\ Y & (1|1) \\ Z & (0|1) \end{array} \quad \longleftrightarrow \quad \begin{array}{c} \\ \\ \\ \\ \end{array} \tag{3.25}$$

For instance, $X \otimes I \otimes Y \leftrightarrow (1\ 0\ 1|0\ 0\ 1)$. The "symplectic" refers to the substitute for commutativity/anticommutativity:

Definition 3.12. The *symplectic form* (or symplectic product) on $\mathbb{Z}_2^n \times \mathbb{Z}_2^n$ is $(x_1|z_1) \odot (x_2|z_2) = x_1 \cdot z_2 + x_2 \cdot z_1$, where the dot product is the usual scalar product in the vector space $\mathbb{Z}_2^n$ and addition is modulo 2.

Proposition 3.14. *$v_P \odot v_Q = c(P, Q)$ for $P, Q \in \mathsf{P}_n$.*

Proof. This can be checked exhaustively for a single qubit. For more than one qubit, note that both $\odot$ and $c(\cdot, \cdot)$ are equal to the parity of their single-qubit results. Therefore, since they are equal for single qubits, they are also equal for n qubits. □

Combining the symplectic form with the binary symplectic representation recovers the primary structure of the Pauli group. We can switch back and forth between the two representations to use whichever one is the most convenient at the moment. The conversion process is summarized in table 3.6. The one thing that is lost in the transition is the phase of a Pauli, so you must be careful whenever dealing with something that depends on that.

Some of the conversions need a little more explanation. There were three conditions for S to be a stabilizer: that $-I \notin \mathsf{S}$, that S is a group, and that S is Abelian. Since phase is lost when converting to the binary symplectic representation, the first condition is irrelevant. The second condition means that S becomes a linear subspace in $\mathbb{Z}_2^n \times \mathbb{Z}_2^n$, and the third means that $v \odot w = 0\ \forall v, w \in \mathsf{S}$. This analysis and the conversion of the normalizer prompts the following definition:

In the Pauli group	Binary symplectic representation
$P \in \mathsf{P}_n$	$v_P = (x_P \vert z_P) \in \mathbb{Z}_2^n \times \mathbb{Z}_2^n$
Multiplication	Addition
$c(P, Q)$	$v_P \odot v_Q$
Phase	No equivalent
Stabilizer S	Weakly self-dual subspace S
Normalizer $\mathsf{N}(\mathsf{S})$	Dual subspace $\mathsf{S}^\perp$ (under $\odot$)
Minimal set of generators for S	Basis for S

Table 3.6: Equivalence between the Pauli group and its binary symplectic representation.

$$\begin{array}{c} \\ \\ \\ \\ \overline{X} \\ \overline{Z} \end{array} \left(\begin{array}{ccccc|ccccc} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{array}\right)$$

Table 3.7: The binary symplectic representation of the stabilizer and logical Pauli operators for the five-qubit code.

Definition 3.13. Let V be a linear subspace of $\mathbb{Z}_2^n \times \mathbb{Z}_2^n$. The *dual* of V (with respect to $\odot$) is $V^\perp = \{w \in \mathbb{Z}_2^n \times \mathbb{Z}_2^n | w \odot v = 0 \ \forall v \in V\}$. A subspace is *self-dual* if $V^\perp = V$. A subspace is *weakly self-dual* if $V \subseteq V^\perp$.

Table 3.7 gives the five-qubit code in binary symplectic representation, including the logical $\overline{X}$ and $\overline{Z}$ operators.

Conversion can go the other way too. For instance, it is convenient to define a notion of a set of Paulis being independent based on the standard definition for binary vector spaces:

Definition 3.14. A set of Paulis $\{P_1, \ldots, P_m\}$ is *independent* iff the vectors $\{v_{P_1}, \ldots, v_{P_m}\}$ are linearly independent.

A set of Paulis is independent unless one of them is a product of others; this is equivalent to saying that a set of $2n$-bit binary vectors is independent unless one of them is a sum of others. Since the binary vector space is $2n$-dimensional, the maximum number of independent Paulis is $2n$, which is equal to the number of generators of the Pauli group.

3.5.2 Linear Algebra Lemma

The power of the binary symplectic representation is that we can use standard results from linear algebra to prove a number of properties of the Pauli group and stabilizers. In particular, we have the following lemma:

Lemma 3.15. *Let $\{P_1, \ldots, P_m\}$ be an independent set of Paulis on n qubits, and let s be an m-bit vector with components s_i. Then $\exists Q \in \mathsf{P}_n$ with $c(P_i, Q) = s_i$. In fact, there are 2^{2n-m} such Paulis.*

Proof. Converting to the binary symplectic representation gives us m linearly independent vectors v_i, and we wish to find vector w such that $v_i \odot w = s_i$. Each of these conditions is a linear equation, and since the vectors v_i are independent, the system of linear equations is non-singular. There are m equations in a $2n$-dimensional vector space, so the space of solutions has dimension $2n - m$. It is a binary vector space, so that corresponds to 2^{2n-m} solutions. □

An important consequence of the lemma is that it tells us about the decomposition of the full physical Hilbert space into subspaces associated with the different error syndromes. I've already discussed this decomposition, but the missing piece is given by the following corollary of lemma 3.15:

Proposition 3.16. *Let* S *be a stabilizer with generators* $\{M_i\}$*. Then for any error syndrome* s*,* $\exists P \in \mathsf{P}_n$ *with* $\sigma(P) = s$*.*

That is, not only is every Pauli outside $\mathsf{N}(\mathsf{S})$ associated with an error syndrome, but every possible error syndrome is associated with some Pauli.

3.5.3 Consequences of the Lemma

Now it's time for some of the proofs I owe you. There is proposition 3.2, which claims $\mathsf{S} = \mathsf{S}(\mathcal{T}(\mathsf{S}))$, and theorem 3.11, which says $\mathsf{N}(\mathsf{S})/\mathsf{S} \cong \mathsf{P}_k$.

First, here is the proof that $\mathsf{N}(\mathsf{S})/\mathsf{S} \cong \mathsf{P}_k$:

Proof of theorem 3.11. The most straightforward way to show this is by sequentially picking coset representatives for the cosets corresponding to $\overline{X}_i$ and $\overline{Z}_i$. The Pauli group P_k is determined, like any finitely-presented group, by its generators and relations between them. In the case of the Pauli group, the relations are

$$c(X_i, X_j) = 0 \tag{3.26}$$
$$c(Z_i, Z_j) = 0 \tag{3.27}$$
$$c(X_i, Z_j) = \delta_{ij}. \tag{3.28}$$

It is sufficient to consider only the generating cosets, since we can let the representatives of a product of cosets be the product of the representatives, as discussed in section 3.4.2. Provided we can choose $\overline{X}_i$ and $\overline{Z}_i$ with the correct commutation relations, that gives us an injective map of P_k into $\mathsf{N}(\mathsf{S})/\mathsf{S}$. We know that $|\mathsf{N}(\mathsf{S})| = 4 \cdot 2^{n+k} = |\mathsf{S}||\mathsf{P}_k|$, so an injection has to be isomorphism.

Suppose we've chosen some independent set of $\overline{X}_i$'s and $\overline{Z}_i$'s with the correct commutation relations, and we wish to choose one more. The new logical Pauli must be in $\mathsf{N}(\mathsf{S})$, so in particular, it must commute with all generators of the stabilizer. Second, it has a defined commutation relation with the already chosen logical Paulis. By lemma 3.15, there exists a Pauli that satisfies all of these constraints.

The only remaining thing to check is that the new logical Pauli can be chosen to be independent of the prior ones. To simplify the analysis, let us first pick all the $\overline{X}_i$'s, then the $\overline{Z}_i$'s. When we are picking $\overline{Z}_j$, it satisfies different commutation relations than all previously selected logical Paulis. In particular, $\overline{Z}_j$ anticommutes with $\overline{X}_j$, unlike all the previous logical Paulis and all the stabilizer generators. Thus, $\overline{Z}_j$ must be independent.

When we pick $\overline{X}_j$, this argument doesn't apply, since the new one will commute with all stabilizer generators and all previous logical Paulis. However, there are $n-k$ stabilizer generators and at most $k-1$ logical Paulis already chosen, for a total of at most $n-1$ constraints. By lemma 3.15, there are thus at least 2^{n+1} possible solutions. The group generated by the stabilizer and previous logical Paulis contains only 2^{n-1} operators, so there are possible choices for $\overline{X}_j$ that are independent of the previous choices.

□

Finally, we can prove proposition 3.2, showing that $\mathsf{S} = \mathsf{S}(\mathcal{T}(\mathsf{S}))$:

Proof of proposition 3.2. If $M \in \mathsf{S}$, then certainly $M|\psi\rangle = |\psi\rangle$ for any $|\psi\rangle \in \mathcal{T}(\mathsf{S})$, so $\mathsf{S} \subseteq \mathsf{S}(\mathcal{T}(\mathsf{S}))$. We need to show that if $N \notin \mathsf{S}$, then $N \notin \mathsf{S}(\mathcal{T}(\mathsf{S}))$. If $N \notin \mathsf{N}(\mathsf{S})$ then $N|\psi\rangle$ (for any codeword $|\psi\rangle$) has a different eigenvalue for some $M \in \mathsf{S}$, and is therefore orthogonal to the code space, which in turn means $N \notin \mathsf{S}(\mathcal{T}(\mathsf{S}))$.

If S has n generators, that is all we need. Otherwise, if $r < n$, we still need to show that if $N \in \mathsf{N}(\mathsf{S}) \setminus \mathsf{S}$, then $\exists|\psi\rangle \in \mathcal{T}(\mathsf{S})$ such that $N|\psi\rangle \neq |\psi\rangle$. By lemma 3.15, we can choose a $M \in \mathsf{N}(\mathsf{S})\setminus\mathsf{S}$ such that $\{M, N\} = 0$. Consider the modified stabilizer S' formed by adding M to S as a new generator. S' has $r+1$ generators, so its code space has dimension 2^{k-1} (by proposition 3.3). Since $k \geq 1$, there is at least one state $|\psi\rangle$ (up to normalization) in the code space of S'. $|\psi\rangle$ is left fixed by all elements of S', and in particular by the elements

of $\mathsf{S} \subset \mathsf{S}'$. Thus, $|\psi\rangle \in \mathcal{T}(\mathsf{S})$. However, N anticommutes with M, which is in the stabilizer of $|\psi\rangle$. Therefore, $N|\psi\rangle$ has eigenvalue -1 for M, and in particular is orthogonal to $|\psi\rangle$. It follows that $N \notin \mathsf{S}(\mathcal{T}(\mathsf{S}))$, which proves the proposition.

□

Chapter 4

Maybe I Should Have Started Here: Classical Error Correction

In chapter 3, we saw the formalism of stabilizer codes, but we didn't see how to find new stabilizer codes. Indeed, finding new codes is a tricky topic. Already in the theory of classical error correction, it is quite difficult to find good new codes. Luckily, there are a few ways of taking already-discovered classical codes and turning them into quantum codes. That is not the subject of this chapter.

This chapter is instead about the theory of classical error-correcting codes. Naturally, I won't have time to go into complete detail on classical error correction, so I will focus on the particular points of the theory of classical error correction which have most relevance to QECCs. One purpose is to give you the background for chapter 5, which *is* about converting classical codes into quantum codes. The other reason is that it can give you a deeper understanding of the theory of QECCs and of stabilizer codes, since there are many parallels — and some critical differences — between the theories of classical and quantum error correction.

For those who are familiar with the theory of classical error correction, this chapter is likely to be somewhat boring, but I hope it won't be a complete waste of time. In particular, I will try to point out the parallels between classical coding theory and those aspects of quantum coding theory that we've seen so far. Some you may have already noticed yourself, but perhaps there are some parallels you missed.

4.1 Classical Error Correction in General

4.1.1 Abstract Description of a Classical Code

A general classical error-correcting code can be defined, much as we did for a QECC, as an encoding map.

Definition 4.1. A classical *error-correcting code* $(e, \mathcal{E})$ is a map $e : [1 \dots K] \to [1 \dots N]$ with a set of correctable errors $\mathcal{E}$ (consisting of maps $E : [1 \dots N] \to [1 \dots M]$) with the following property: $\exists$ map $d : [1 \dots M] \to [1 \dots K]$ such that $\forall E \in \mathcal{E}$, $\forall x \in [1 \dots K]$,

$$d(E(e(x))) = x. \tag{4.1}$$

The map d is the *decoder* for the code and e is the *encoder* for the code. Frequently, the code is just referred to as C, the image of the encoder e in $[1 \dots N]$.

I have written this to be completely analogous to the definition I gave for a QECC. The only real difference is that quantum states live in a Hilbert space, and maps between them should be quantum operations, or at least linear maps, whereas classically, we can consider states from any set and arbitrary functions between the sets as the encoder, decoder, and errors. But remember that classical codes are actually just a special case of quantum codes. The apparent extra constraint of linearity is really an extra freedom to include superpositions of basis states, whereas classical codes can only contain basis states.

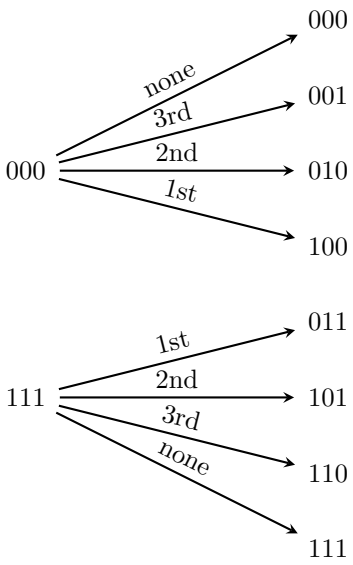


Figure 4.1: Codewords of the repetition code with errors on a single bit never get confused.

Ultimately, a classical error-correcting code is a just a set of objects (frequently bit strings), and the errors diversify those objects. The goal of the decoder is to get back to the original object. We have an uncorrectable error when two different logical objects x and y get confused: $E(e(x)) = F(e(y))$. If that never happens, a decoder exists: the sets $S_x = \{E(e(x))|E \in \mathcal{E}\}$ are all distinct, so we can define a function mapping all elements of S_x to x unambiguously.

This condition has no precise analogy for quantum codes. In some sense equation (2.58) is similar, if we interpret the correct quantum analogue of "different" as "orthogonal." Equation (2.58) says that orthogonal quantum states and different errors must produce orthogonal states, but for QECCs, that is only a sufficient condition, not a necessary one. From that point of view, QECCs seem somewhat more generous than classical error-correcting codes, since they don't require that it is possible to exactly determine the error. On the other hand, if you did exercise **??**, you saw a condition which can be broken up into two parts. When $i \neq j$, it says that we should never confuse different quantum states under an error; that seems analogous to the condition for classical error correction. But then there is an additional non-trivial condition for a QECC when $i = j$, which makes the quantum code seem more stringent than the classical one. The extra condition can be interpreted as saying that the environment should not learn anything about the logical subspace for a QECC. Classically, of course, it is harmless if the environment simply learns the encoded data, provided it doesn't change it. In the quantum case, learning about the data *necessarily* involves changing it.

4.1.2 Distance of a Classical Code

As with quantum codes, frequently we focus on codes where the encoded state N can be broken up into smaller pieces, $N = q^n$, and where $M = N$. We consider $[1 \dots N]$ as the direct product of n copies of $[1 \dots q]$. I will refer to each $[1 \dots q]$ factor as a *register*, a *component*, or a *coordinate*. In the common case where $q = 2$, I may just refer to the ith bit (or physical bit) of the code. Sometimes I may even forget myself and accidentally refer to the ith qubit. In the quantum case, I'll use these same terms, but I try to avoid component since it also can refer to the component in some direction in the overall Hilbert space. In the classical case, the terminology comes from treating $[1 \dots q^n]$ as an n-dimensional vector space over the finite field GF(q). We'll get back to that approach in section 4.2.

Also, we tend to focus on cases where $\mathcal{E}$ is composed of errors of weight t or less. The classical definitions

of *weight* and *support* are just the same as the quantum ones: the error E acts trivially on all but the coordinates in the support of E, and the weight of E is the size of the support. When we have a classical channel which causes errors independently on each coordinate with a small probability p, then the probability of having errors of weight $t+1$ or greater is $O(p^{t+1})$, much as in the quantum case. If we can find a code that simply corrects all errors of weight t or less, then with high probability, we'll get the correct state when we decode.

Definition 4.2. The *distance* of a classical error-correcting code $C \subseteq \mathbb{Z}_N$ (with $N = q^n$) is

$$\min\{\mathrm{wt}(E) | \exists x \neq y \in C, E(x) = y.\} \tag{4.2}$$

A classical code with distance d encoding K possible states in n registers of size q is an $(n, K, d)_q$ code. When $q = 2$, we just write (n, K, d).

In other words, the distance is the minimum number of coordinates that have to be changed to get from one codeword to another. The quantum distance can be understood in the same way, but there are more subtleties in the quantum case.

As with quantum codes, the distance tells us how many errors we can correct:

Proposition 4.1. *A classical code with distance d can:*

1. *correct general errors on up to $\lfloor (d-1)/2 \rfloor$ coordinates,*
2. *correct erasure errors on up to $d-1$ coordinates, or*
3. *detect errors on up to $d-1$ coordinates.*

4.2 Classical Linear Codes

General classical codes are unwieldy to deal with, just as are general quantum codes. The class of linear codes is very analogous to stabilizer codes — it is a much more tractable subclass of classical error-correcting codes, and many of the most interesting known codes are, in fact, linear codes.

For a linear code, we assume $N = q^n$, as above. For this section, I will further assume $q = 2$, and treat $[1 \ldots N]$ as an n-dimensional vector space over $\mathbb{Z}_2$. In section 4.4, I will return to the $q \neq 2$ case.

4.2.1 Generator Matrix and Parity Check Matrix

By considering $\mathbb{Z}_N = \mathbb{Z}_2^n$ as a vector space, we add an additional linear structure, and a linear code exploits that structure.

Definition 4.3. An error-correcting code $C \subseteq \mathbb{Z}_2^n$ is a *linear code* if $x, y \in C \Rightarrow x + y \in C$.

A linear code is thus a linear subspace of $\mathbb{Z}_2^n$. We can choose a basis $x_1, \ldots, x_k$ for the code, and all other vectors in the code will be linear combinations of the x_i's. Since $x + x = 0 \ \forall x \in \mathbb{Z}_2^n$, the only question is which subset of the basis vectors are added together to make a codeword. There are thus 2^k codewords in total and k encoded bits.

Definition 4.4. The *generator matrix* G_C of a linear code C is a matrix with row i equal to x_i.

The generator matrix can be used to define the encoder of C:

Proposition 4.2. *Let $C \subseteq \mathbb{Z}_2^n$ be a linear code with k encoded bits and generator matrix G. Then the linear map $v \in \mathbb{Z}_2^k \mapsto G^T v \in \mathbb{Z}_2^n$ is the encoder for C. In other words, x is a codeword of C iff $x = G^T v$ for some $v \in \mathbb{Z}_2^k$.*

In order to check for errors, we measure the parities of bits in the codeword.

Definition 4.5. Suppose the linear code C has generator G. A *parity check matrix* H_C for C is a matrix with row i equal to $y_i \in \mathbb{Z}_2^n$, where $Gy_i = 0\ \forall i$, and the set $\{y_i\}$ is a maximal linearly independent set with this property.

The parity check matrix of a code is not unique, but it is still frequently referred to as "the" parity check matrix. You can take any linear combinations of rows of the parity check matrix, and provided the new set of rows remains linearly independent, it will still function as a parity check matrix.

Theorem 4.3. *If C has k encoded bits and n physical bits, then G_C is a $k \times n$ matrix and H_C is an $(n-k)\times n$ matrix. $G_C {H_C}^T = 0$ and $H_C {G_C}^T = 0$.*

Proof. The only property which is not completely trivial is that H_C has $n-k$ rows. The constraints ${G_C}^T y_i = 0$ form a set of k non-singular linear equations on n bits, so the solution space has dimension $n-k$. Thus, a maximal set of $\{y_i\}$ will consist of $n-k$ of them. □

Using the linear structure of $\mathbb{Z}_2^n$, we can represent a set of bit flip errors as a vector $e \in \mathbb{Z}_2^n$, with 1 for those bits which have been flipped and 0 for those bits which have not been flipped. Under this convention, $\mathrm{wt}\, e$ is the weight of the error. If we start with bit string x and the error e occurs, we now have the bit string $x + e$.

The virtue of the parity check matrix is that it filters out the codewords and just tells us about the errors. Suppose $x \in C$ undergoes error e. Then, using linearity and proposition 4.2,

$$H_C(x+e) = H_C x + H_C e = H_C {G_C}^T v + H_C e = H_C e. \tag{4.3}$$

Definition 4.6. The *error syndrome* of an error e for linear code C is $H_C e$.

All of this probably sounds a bit familiar, since stabilizer codes and classical linear codes are closely related. The stabilizer of a stabilizer code is very analogous to the parity check matrix for a linear code, although the stabilizer also has some vague similarity to the generator matrix (in that the projector on the code space is formed from the stabilizer). In fact, linear codes are a special case of stabilizer codes, and the parity check matrix can be realized as the stabilizer:

Theorem 4.4. *Let C be a linear code with parity check matrix H, and let C correct the set of errors $\mathcal{E} \subseteq \mathbb{Z}_2^n$. Define a stabilizer code S to have stabilizer with binary symplectic representation $(0|H)$. Then S corrects the set of errors $\{(e|0)|e \in \mathcal{E}\}$ and for any $x \in \mathbb{Z}_2^n$, $x \in C$ iff $|x\rangle \in \mathsf{S}$.*

In other words, we form a stabilizer by replacing the 1s in H with Zs, with each row of the parity check matrix becoming a generator of the stabilizer. The resulting stabilizer code corrects the same errors as C and has the same basis codewords. The one difference is that the stabilizer code encodes a quantum state, so superpositions such as $|x_1\rangle + |x_2\rangle$ are also codewords of S, even though the combination is meaningless when considering a classical code. The proof of the theorem is straightforward, and is left as an exercise.

The non-uniqueness of the parity check matrix is just the same as the non-uniqueness of the generators of a stabilizer. Replacing a row of the parity check matrix with a linear combination of rows is equivalent to replacing a generator of the stabilizer with a product of generators.

The best analogy for the rows of the generator matrix are the logical Paulis in $\mathsf{N}(\mathsf{S})/\mathsf{S}$. For a stabilizer code, the coset representatives are non-unique, which is not an issue for classical codes. It is true that the generator matrix is not completely unique — we can take a different encoder, which corresponds to a different generator matrix. However, this is a different phenomenon, and corresponds to labeling the cosets in $\mathsf{N}(\mathsf{S})/\mathsf{S}$ with different logical Paulis rather than choosing different representatives of them.

4.2.2 Distance of a Linear Code

For a linear code, the distance has a somewhat nicer form than for a general classical error-correcting code.

Proposition 4.5. *The distance of a linear code C is* $\min\{\mathrm{wt}(e)|e \in C, e \neq 0\}$.

Proof. The general definition of distance is as $\min\{\operatorname{wt} e | x \neq y \in C, x+e=y\}$ (with the error rewritten from definition 4.2 to take advantage of linearity). But when $x, y \in C$, $e = x+y$ is also in C since C is linear. Any $e \in C$ can be realized this way by simply choosing any $x \in C$ and letting $y = x+e$, which will automatically be in C, again since C is linear. $x \neq y$ is equivalent to $e \neq 0$. □

Notation 4.7. An $(n, 2^k, d)$ linear code is denoted as an $[n, k, d]$ code.

For a linear code, the codewords themselves are the undetectable errors: adding a non-trivial codeword to another codeword results in a new codeword, and if you add a non-codeword to a codeword, you always get a non-codeword. For a set of errors to be correctable, any pair of errors must not add up to a codeword:

Theorem 4.6. *A linear code C corrects the error set $\mathcal{E}$ iff $e + f \notin C$ for all $e \neq f \in \mathcal{E}$. Equivalently, $H_C e \neq H_C f$ $\forall e \neq f \in \mathcal{E}$ (i.e., all errors in $\mathcal{E}$ have different error syndromes).*

Proof. If all errors in $\mathcal{E}$ have different error syndromes, than the code certainly can correct for $\mathcal{E}$ using the following procedure: Given an erroneous codeword $x + e$, apply the parity check matrix to get $H_C e$. Since the error syndromes are unique, we can identify e and then recover $x = (x + e) + e$.

If $e + f \in C$ then $H_C(e + f) = H_C e + H_C f = 0$. Conversely, if $e + f \notin C$, then $H_C(e + f) \neq 0$. This is true because the parity check matrix is formed from a *maximal* set of vectors annihilated by the generator matrix. When $e + f \notin C$, then the set of solutions of $G_C y_i = 0$ and of $(e + f) \cdot y_i = 0$ is only $n - k - 1$ dimensional, whereas the parity check matrix has $n - k$ rows. Thus, $e + f \notin C$ iff $H_C e \neq H_C f$.

Finally, if $e + f \in C$ for some pair $e \neq f \in \mathcal{E}$, then the code cannot correct $\mathcal{E}$. For instance, given $x \in C$, let $y = x + (e + f)$, which will also be in C. Then $x + e = y + f$, so if this string shows up, there is no way to tell whether it should be decoded to x (with error e) or y (with error f). □

The combination $e + f$ is reminiscent of the combination $E^\dagger F$ that appears in theorem 3.5 giving the set of correctable errors for a stabilizer code. Comparing further, you can see that for a classical code, $C \setminus \{0\}$ plays the role of $\hat{\mathsf{N}}(\mathsf{S}) \setminus \hat{\mathsf{S}}$ for a stabilizer code. Indeed, if you apply the transformation theorem 4.4 to code C to get stabilizer S, and then calculate $\hat{\mathsf{N}}(\mathsf{S}) \setminus \hat{\mathsf{S}}$, you find that it includes the conversion of $C \setminus \{0\}$. That's not all it contains, but it is a good exercise to work this out yourself.

Importantly, there is no good classical analogue to the concept of a degenerate quantum code. All classical codes are non-degenerate, and the presence of degenerate quantum codes complicates quantum coding theory.

4.2.3 Example: Hamming Codes

One example of a linear code is the repetition code $0 \mapsto 000$, $1 \mapsto 111$. It is linear because $000 + 000 = 000$, $000 + 111 = 111$, and $111 + 111 = 000$. The generator matrix is

$$G = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}, \tag{4.4}$$

and the parity check matrix is

$$H = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \tag{4.5}$$

The three-bit repetition code (a $[3, 1, 3]$ code) can of course be generalized to d-bit repetition codes, which have parameters $[d, 1, d]$, but there is another interesting generalization, starting from the observation that in the parity check matrix for the three-bit repetition code, every column is different.

Supposing we want to correct only 1-bit errors. If $\operatorname{wt} e = 1$, with the only 1 in the ith coordinate, then for any parity check matrix H, He — the error syndrome of e — will be the ith column of H. Thus, if we pick all columns of H to be different, the error syndromes of all 1-bit errors will be different. We should not pick an all 0s column since we want the no error case ($e = 0$) to have error syndrome 0.

If we fix the number of rows of H to be r, we can choose up to $2^r - 1$ distinct columns by letting them run over all nonzero r-bit strings. This gives a Hamming code:

Definition 4.8. The $[2^r - 1, 2^r - r - 1, 3]$ Hamming code is the code whose $r \times (2^r - 1)$ parity check matrix has as columns all possible r-bit strings.

Notice that here we have defined the Hamming code by choosing a parity check matrix, much as for a stabilizer code, we choose the stabilizer to define the code. In this case, you can derive the set of actual codewords as $\{x|Hx = 0\}$. There are r linear constraints on $2^r - 1$ bits, so the linear space of solutions is $(2^r - r - 1)$-dimensional, giving the number of encoded qubits in the definition.

As a concrete example, when $r = 3$, we get a $[7, 4, 3]$ code. It has parity check matrix

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}, \tag{4.6}$$

and can be taken to have generator matrix

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}. \tag{4.7}$$

4.2.4 Example: Reed-Muller Codes

Looking at the generator matrix for the 7-bit Hamming code, it has a somewhat nice form. The form gets even nicer if you add an extra bit which is 1 for the first row and 0 for the others:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}. \tag{4.8}$$

Now every codeword has weight 0, 4, or 8. This is an example of a Reed-Muller code, specifically $\mathcal{R}(1, 3)$. It is an $[8, 4, 4]$ code.

Definition 4.9. The *1st order Reed-Muller code* $\mathcal{R}(1, m)$ is a linear code with $n = 2^m$ physical bits. It has a generator matrix whose rows are the all-1s vector and the vectors v_i $(i = 1, \ldots, m)$, where the jth coordinate of v_i is equal to the ith bit of j, when j is expanded in binary (and assuming j runs from 0 to $n - 1$).

For instance, for $n = 4$, $v_1 = (0011)$ and $v_2 = (0101)$. The generator matrix is

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} \tag{4.9}$$

It is a $[4, 3, 2]$ code. Note that in equation (4.8), the order of the bits is reversed relative to this convention.

Theorem 4.7. *$\mathcal{R}(1, m)$ is a $[2^m, m + 1, 2^{m-1}]$ code.*

Proof. Each of the vectors v_i has weight exactly $n/2$ $(n = 2^m)$, as exactly half the numbers $0, \ldots, n - 1$ will be 1 in any given bit location. We can also easily show that any sum of $s > 0$ of the v_is will have weight exactly $n/2$: The sum will be 0 or 1 in the jth location iff the XOR of the corresponding bits of j is 0 or 1. For instance, for $v_1 + v_3$, take the XOR of the first and third bits of j. We then let j run over from $0, \ldots, n - 1$. As we do this, the set of bits we are looking at can take on every possible set of values, and furthermore, each set of values appears the same number of times, corresponding to every set of values for the other bits. In particular, any particular assignment of the s bits we are interested in shows up 2^{n-s} times. For half of these assignments, the XOR will be 0, and for the other half, the XOR will be 1. Thus the weight of the sum we are looking at is exactly $n/2$.

The all-1s vector has weight n, and the all-1s vector added to any vector of weight $n/2$ is again a vector of weight $n/2$. Thus, the code has distance $n/2 = 2^{m-1}$.

This argument also shows that all the rows of the generator matrix are independent, since no linear combination gives 0. Therefore, the code has $m + 1$ encoded bits, the same as the number of rows.

□

Definition 4.10. The rth order Reed-Muller code $\mathcal{R}(r, m)$ has as rows of its generator matrix all products of up to r of the vectors v_i given in definition 4.9, where product means the bitwise product (1 in a coordinate iff all vectors in the product are 1 at that coordinate) The all-1s generator is also included. (It can be considered as the product of 0 of the v_i's.)

For instance, $\mathcal{R}(2, 2)$ has the additional generator $v_1 v_2 = (0001)$, giving the generator matrix

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.10}$$

$\mathcal{R}(2, 2)$ is a $[4, 4, 1]$ code, which actually means it contains all 4-bit vectors. $\mathcal{R}(2, 3)$ is more interesting, with generator matrix

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.11}$$

$\mathcal{R}(2, 3)$ is an $[8, 7, 2]$ code.

Theorem 4.8. *$\mathcal{R}(r, m)$ is a $[2^m, N(r, m), 2^{m-r}]$ code, where*

$$N(r, m) = 1 + \binom{m}{1} + \binom{m}{2} + \cdots + \binom{m}{r}. \tag{4.12}$$

Proof. We must show that the code has $N(r, m)$ encoded bits and has distance 2^{m-r}.

Number of encoded bits: $\mathcal{R}(r, m)$ is spanned by a product of vectors for each subset of up to r elements of the numbers $1, \ldots, m$. Counting them gives the formula for $N(r, m)$. However, we do need to show that these are linearly independent to see that this is also the number of encoded bits.

Consider the extreme case of $\mathcal{R}(m, m)$. The generator matrix for $\mathcal{R}(r, m)$ is contained in the top rows of that for $\mathcal{R}(m, m)$, so it is enough to show that the generator matrix for $\mathcal{R}(m, m)$ has full rank. $N(m, m) = 2^m = n$, so the rows are linearly independent iff they span the whole vector space.

We can imagine arbitrary vectors on 2^m bits as functions from m bits to one bit, as follows: for the jth bit of the vector, interpret j as the input of the function, with the output of the function given by the jth bit. (For instance, the vector 0110 is the function $f(00) = 0$, $f(01) = 1$, $f(10) = 1$, $f(11) = 0$.) In this interpretation, v_i is the function "take the ith bit of the input."

The product of multiple vectors $v_{i_1}, \ldots, v_{i_s}$ is the function "take the AND of bits $i_1, \ldots, i_s$ of the input," and the sum of products is the XOR of these ANDs. We can write an arbitrary function from m bits to 1 bit as the XOR of ANDs of up to all m bits, so all possible vectors are in the code $\mathcal{R}(m, m)$. Therefore all products of the v_is are linearly independent.

Distance: It is easy to see that the product of any r vectors v_i has weight $n/2^r = 2^{m-r}$, as the product is 1 in the jth location iff the AND of the corresponding bits of j is 1, which happens for only a fraction $1/2^r$ of the possible values of j. We also have to check the distance, however, for the sums of these products, and it is not clear that taking the sum cannot cause the weight to decrease.

To show that the distance of $\mathcal{R}(r,m)$ is indeed 2^{m-r}, we perform induction on r and m. We have already calculated the distance for $\mathcal{R}(1,m)$ in theorem 4.7. As $m \geq r$, we also want as a base case to show it for a given r for the smallest possible value of m, namely $\mathcal{R}(r,r)$. In this case, there is nothing really to show, as the distance from the formula is 1, which is indeed the weight of the product of all r v_i vectors, and there is no possibility of having a shorter distance.

Claim 4.9. *Assume the distance of $\mathcal{R}(r-1,m)$ is 2^{m-r+1} and the distance of $\mathcal{R}(r,m)$ is 2^{m-r}. Then the distance of $\mathcal{R}(r,m+1)$ is 2^{m+1-r}.*

Proof of claim. We can consider any given sum of basis vectors, and break it up into a term where none of the products includes the vector v_1 and a term where all of the products include v_1. Now, if we restrict attention to the first 2^m coordinates, the second term is uniformly 0 and the first term is a vector from $\mathcal{R}(r,m)$, which we already know has weight at least 2^{m-r} unless it is the 0 vector.

If we restrict attention to the last 2^m coordinates, v_1 is always 1, so it can be ignored, and the second term is the sum of products of at most $r-1$ vectors, and is thus a vector from $\mathcal{R}(r-1,m)$. The first term is the same on the last 2^m coordinates as it was on the first 2^m coordinates, and is again a vector from $\mathcal{R}(r,m)$. But $\mathcal{R}(r-1,m) \subseteq \mathcal{R}(r,m)$, so the sum of a term from $\mathcal{R}(r-1,m)$ and a term from $\mathcal{R}(r,m)$ is in $\mathcal{R}(r,m)$, and therefore the last 2^m coordinates have weight at least 2^{m-r} by the inductive hypothesis unless the last 2^m coordinates are all 0.

If the first term is the 0 vector, we actually have a vector from $\mathcal{R}(r-1,m)$ on the last 2^m coordinates, which therefore has weight at least 2^{m-r+1} unless it is 0. To get 0 on the last 2^m coordinates, either both terms are 0 (in which case the whole vector is 0), or the first and second terms must cancel on the last 2^m coordinates. In order to cancel the second term, the first term must actually be a vector from $\mathcal{R}(r-1,m)$ on each half of the coordinates, which again means the vector on each half of the coordinates has weight 2^{m-r+1}.

That is, we have three cases (assuming the overall vector is not 0): In case one, the first term is 0, in which case the last 2^m coordinates have weight 2^{m-r+1}. In case two, the first and last 2^m coordinates will each have weight at least 2^{m-r}. In case three, the last 2^m coordinates have weight 0, but the first 2^m coordinates have weight 2^{m-r+1}. In all of these cases, we know that the overall vector has weight at least 2^{m+1-r}, completing the induction for the distance. □

If we have proven the formula for distance for $r-1$ and all m, then we can use induction on m. The base case is $\mathcal{R}(r,r)$ and we use the claim to prove the distance formula for this specific value of r and all m. This then allows us to use induction on r and the base case of $\mathcal{R}(1,m)$ to prove the distance formula for all values of r and m.

□

4.3 Dual Codes

4.3.1 Definition of a Dual Code

Definition 4.11. If C is a linear code, the *dual code* $C^\perp$ is

$$C^\perp = \{y \in \mathbb{Z}_2^n | x \cdot y = 0 \ \forall x \in C\}. \tag{4.13}$$

The dual code switches the role of the generator and parity check matrices: The generator matrix of $C^\perp$ is the parity check matrix of C and the parity check matrix of $C^\perp$ is the generator matrix of C. (Note that $(C^\perp)^\perp = C$.) It follows that the dual code of an $[n,k,d]$ code is an $[n,n-k,d']$ code. The distance d' does not in general have to be related to the distance d.

We also saw a definition of a dual code when discussing the binary symplectic representation of a stabilizer code. The dot product appears in this definition and the symplectic form appeared in that one, but otherwise they are the same.

We can also define *self-dual* codes and *weakly self-dual* codes in the same way as in section 3.5. That is, a self-dual code is equal to its dual, and a weakly self-dual code is contained in its dual.

4.3.2 Dual Codes for the Examples

For the 7-bit Hamming code, we've already worked out the generator and parity check matrices. Looking at the 8 vectors in the span of the rows of the parity check matrix of the 7-bit Hamming code, we see that all the nonzero vectors in the dual code have weight 4. The dual code of the $[7, 4, 3]$ is thus a $[7, 3, 4]$ code. We've already seen that the 7-bit Hamming code is related to a Reed-Muller code. The dual is too — if take $\mathcal{R}(1, 3)$ and drop the all-1s vector, one bit is always zero. If we then discard that bit, we get the dual of the 7-bit Hamming code.

This is true in general:

Theorem 4.10. *Take $\mathcal{R}(1, m)$, remove the all-1s vector, and then* puncture *it: drop the bit that is always 0. The resulting $[2^m - 1, m, 2^{m-1}]$ code is the dual of the $[2^m - 1, 2^m - m - 1, 3]$ Hamming code.*

Proof. First, let us check the parameters of the punctured Reed-Muller code. Dropping the all-1s vector removes one encoded bit, leaving m logical bits. Removing codewords does not decrease the distance, and since the remaining vectors all have weight 2^{m-1}, it does not increase it in this case either. Dropping a bit that is always 0 also does not change the distance, giving the parameters $[2^m - 1, m, 2^{m-1}]$.

We are left with a generator matrix which has m rows $v_1, \ldots, v_m$. The jth bit of v_i is the ith bit of the binary representation of j, so the jth column of the generator matrix is exactly the binary representation of v_i. This was how we constructed the parity check matrix of the $[2^m - 1, 2^m - m - 1, 3]$ code, which is the generator of the dual. □

Naturally, this also means that the duals of the Reed-Muller codes $\mathcal{R}(1, m)$ are related to the Hamming codes. In the special case of $\mathcal{R}(1, 3)$, it was related both to the 7-bit Hamming code *and* to its dual. That is because $\mathcal{R}(1, 3)$ is a self-dual code. Most Reed-Muller codes are not self-dual, but $\mathcal{R}(1, 3)$ is not the only one that is. More importantly, the dual of every Reed-Muller code is another Reed-Muller code.

Theorem 4.11. *The dual of $\mathcal{R}(r, m)$ is $\mathcal{R}(m - r - 1, m)$ ($r \leq m - 1$).*

Proof. Let us take the dot product of two basis vectors $w \in \mathcal{R}(r, m)$ and $w' \in \mathcal{R}(r', m)$. The dot product is the parity of the pointwise product ww'. But w is the pointwise product of up to r of the v_i vectors, and w' is the pointwise product of up to r' of the v_i vectors, so ww' is the pointwise product of up to $r + r'$ of the v_i vectors. Suppose we eliminate redundant v_is that appear twice in the product, leaving us with $s \leq r + r'$ vectors v_i that appear in at least one of the two products w and w'. We already know from theorem 4.8 that such a product has weight exactly 2^{m-s}. Therefore the dot product of w and w' (the parity of the pointwise product) is 0 unless $s = m$, which is only possible if $r + r' \geq m$. Therefore, $\mathcal{R}(m - r - 1, m)$ is orthogonal to $\mathcal{R}(r, m)$, and is contained in its dual.

Now, $\mathcal{R}(r, m)$ encodes $N(r, m) = \binom{m}{0} + \binom{m}{1} + \cdots + \binom{m}{r}$ bits by theorem 4.8, so its dual encodes $2^m - N(r, m)$ bits. But $\mathcal{R}(m - r - 1, m)$ encodes

$$\binom{m}{0} + \binom{m}{1} + \cdots + \binom{m}{m - r - 1} = \binom{m}{m} + \binom{m}{m - 1} + \cdots + \binom{m}{r + 1} \tag{4.14}$$

bits. Therefore, $N(r, m) + N(m - r - 1, m) = 2^m$, and $\mathcal{R}(m - r - 1, m)$ is not only contained in the dual of $\mathcal{R}(r, m)$, it is the same size as the dual, and therefore equals the dual. □

Corollary 4.12. *When $m = 2r + 1$, the Reed-Muller code $\mathcal{R}(r, m)$ is self-dual. When $m \geq 2r + 1$, $\mathcal{R}(r, m)$ is weakly self-dual.*

4.4 Non-Binary Linear Codes

While binary codes, with the number of physical states $N = 2^n$, are the most common sort of error-correcting code, non-binary codes, which have other values of N, are also used. Some non-binary codes are quite important in the theory of quantum error correction, so we'll quickly cover the concepts here. We won't get to non-binary *quantum* codes until chapter 8, but we'll use some facts about non-binary classical codes even when discussing QECCs over qubits.

4.4.1 Linear Codes Over Finite Fields

When using the binary generalization of linear codes, we usually assume that $N = q^n$, and $q = p^m$ is a prime power. We want q to be a prime power because then there is a finite field $\mathrm{GF}(q)$ of size q, and we can consider $[1 \dots N]$ to be a vector space of dimension n over $\mathrm{GF}(q)$. (See appendix C for an introduction to finite fields.)

Definition 4.12. An error-correcting code $C \subseteq \mathrm{GF}(q)^n$ is a *linear code* if $x, y \in C \Rightarrow \alpha x + \beta y \in C$ for any $\alpha, \beta \in \mathrm{GF}(q)$. C is an *additive code* if $x, y \in C \Rightarrow x + y \in C$.

For binary linear codes, we only needed to worry about adding together codewords, but for non-binary linear codes, multiplication by scalars from the field $\mathrm{GF}(q)$ must also keep us within the code. If adding codewords gives a codeword but multiplication by scalars does not necessarily do so, then the code is merely *additive*. For bits, additive implies linear because the only scalars are 0 and 1, but for larger fields, there are more options.

We can again consider the possible errors to be vectors in $\mathrm{GF}(q)^n$. If an error e acts on a string x, the resulting state is $x + e$, as before. Bear in mind, however, that there are $q - 1$ different errors that can affect each register.

Non-binary linear codes have generator and parity check matrices defined in the same way as for binary codes, and the distance is also defined in the same way. We use the notation $[n, k, d]_q$ for a non-binary linear code with n physical registers, each of size q. The only difference in the basic properties of the code is that we now need to be careful of the distinction between addition and subtraction, which are the same for a binary code. In particular,

Theorem 4.13. *A non-binary linear code C corrects the error set $\mathcal{E}$ iff $e - f \notin C$ for all $e \neq f \in \mathcal{E}$. Equivalently, $H_C e \neq H_C f$ $\forall e \neq f \in \mathcal{E}$ (i.e., all errors in $\mathcal{E}$ have different error syndromes).*

We can define the dual code for a non-binary code in just the same way as for a binary code using the dot product for a vector space over $\mathrm{GF}(q)$.

4.4.2 Example: Hamming Codes

The Hamming codes have a natural generalization to non-binary fields. Our goal in designing non-binary Hamming codes is again to choose the columns of the parity check matrix so that all single-qubit errors have different error syndromes. However, we need to be careful because there is now more than one possible single-bit error per register. If e_i is the error that is 1 in location i and 0 elsewhere, then e_i has syndrome equal to the ith column of the parity check matrix. When $\alpha \in \mathrm{GF}(q)$, αe_i has error syndrome equal to α times the ith column of the parity check matrix. Therefore, the correct generalization of the Hamming code is not to ensure just that all columns are different; instead, we want that no column of the parity check matrix is a scalar multiple of another for any scalar in $\mathrm{GF}(q)$.

For instance, GF(4) has 4 elements 0, 1, ω, and ω^2. Therefore, we have a $[5, 3, 3]$ Hamming code over GF(4) with the following parity check matrix:

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & \omega & \omega^2 \end{pmatrix} \tag{4.15}$$

Notice that we don't include columns $(0, \omega)$ or $(0, \omega^2)$ since they are scalar multiples of $(0, 1)$, but that we include all columns of the form $(1, \alpha)$, since no two of them are scalar multiples. We don't, however, then need to include any columns (ω, α) or (ω^2, α).

Following this logic, we find the non-binary Hamming codes:

Theorem 4.14. *There exists a $[(q^r - 1)/(q - 1), (q^r - 1)/(q - 1) - r, 3]_q$ code for any prime power q and any $r > 1$. These codes are known as* Hamming codes.

Proof. There are r rows in the parity check matrix. We wish to choose $(q^r-1)/(q-1)$ columns so that none is a scalar multiple of another. Note that

$$(q^r-1)/(q-1) = 1 + q + \cdots + q^{r-1}. \tag{4.16}$$

We will proceed by induction on r, the number of rows in the parity check matrix. If $r = 1$, we just have the parity check matrix (1), which gives a $[1, 0, 1]_q$ code. It is a special case.

Next, we assume that we have a parity check matrix for $r-1$. For the r-row matrix, we can choose the first column to be all 0s except for the last row, which is 1. Then any column for which the first $r-1$ entries are 0 will be a scalar multiple of this column. The remaining columns are of the form (v, α), where v is a column of the parity check matrix for the Hamming code for q and $r-1$, and α is any element of GF(q). When $v \neq v'$ are two different columns of the $r-1$ Hamming code parity check matrix, than $v \neq \beta v'$, so it's certainly true that $(v, \alpha) \neq \beta(v', \alpha')$. Thus, we don't get columns which are scalar multiples that have different entries in the first $r-1$ entries. We should also compare (v, α) with (v, α'), but since the first r entries are not all 0, the only possible scalar factor between them is 1, which implies that $\alpha = \alpha'$. Thus, this scheme gives independent columns, as desired. We can pick $(q^{r-1}-1)/(q-1)$ columns of the $r-1$ Hamming code, and q entries for the last row in the column, plus we have the first column $(0, 0, \ldots, 0, 1)$. The total number of columns is thus

$$1 + q[(q^{r-1}-1)/(q-1)] = 1 + q(1 + q + \cdots + q^{r-2}) = 1 + q + \cdots + q^{r-1} = (q^r-1)/(q-1), \tag{4.17}$$

as desired. □

4.4.3 Example: Reed-Solomon Codes

Reed-Solomon codes are a useful class of codes which are based on polynomials over finite fields.

Definition 4.13. Let $\alpha_1, \ldots, \alpha_n$ be n distinct elements of GF(q), and let $k \leq n$. A *Reed-Solomon code* is

$$\{(f(\alpha_1), \ldots, f(\alpha_n)) | f \text{ is a polynomial of degree } < k\}. \tag{4.18}$$

The encoder for this code equates the input $(\beta_0, \ldots, \beta_{k-1})$ to the polynomial

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_{k-1} x^{k-1}. \tag{4.19}$$

Reed-Solomon codes are widely used for practical purposes, for instance to encode DVDs to protect against minor damage and in QR codes in case some part of the code can't be read. They are useful because they have a large distance and are easy to encode and decode. But who cares about that stuff? It's all very 20th-Century. The *real* reason Reed-Solomon codes are interesting is that they are useful for making quantum codes, although we won't see how until we discuss non-binary QECCs in chapter 8.

Theorem 4.15. *A Reed-Solomon code with n points from* GF(q) *and polynomials of degree at most $k-1$ is an $[n, k, n-k+1]_q$ code.*

Proof. Directly from the definition we can see that there are n physical registers and k logical registers. The distance is less obvious. The most straightforward way to see it is to consider correcting erasure errors.

Suppose that t registers have been erased. We are then left with $n-t$ registers, which are the values of the polynomial f evaluated at $n-t$ different points. We know that f has degree at most $k-1$, and a degree $k-1$ polynomial is determined by its value at any k points. To be more concrete, given any k points, we have k different linear equations for the coefficients $\beta_0, \ldots, \beta_{k-1}$. The equations are defined by the Vandermonde matrix $V_{ij} = \alpha_i^{j-1}$. (I have assumed that the registers which are available correspond to the points $\alpha_1, \ldots, \alpha_k$ for notational simplicity.) The Vandermonde matrix is non-singular — it has determinant $\prod(\alpha_i - \alpha_j)$ — so the equations have a unique solution.

In short, provided $n - t \geq k$, we can reconstruct f and decode the code. Thus, the code corrects $n-k$ erasure errors, so it has distance $n-k+1$. The distance can't be higher than that, because if we have $n-k+1$ erasure errors, there are only $k-1$ points left, and there are multiple possible polynomials which go through those points since we could choose any value for a kth point and still reconstruct a valid polynomial. □

4.5 Hamming, Gilbert-Varshamov, and Singleton Bounds, MDS Codes

To conclude the discussion of classical codes, we'll discuss some basic limits on the existence of classical codes. I won't get to the quantum analogue of these bounds until chapter 7, but the bounds help to understand the example codes that we've discussed. As you'll see, the Hamming codes and Reed-Solomon codes are optimal codes, giving a maximal k and d for minimum n.

4.5.1 Hamming Bound, Perfect Codes

One can set a simple upper bound by taking advantage of the requirement that the states must be distinguishable after errors.

Theorem 4.16 (Hamming bound)**.** *An $(n, K, 2t+1)_q$ code must satisfy*

$$K\left(\sum_{j=0}^{t}(q-1)^j\binom{n}{j}\right) \leq q^n. \tag{4.20}$$

For large n,

$$(\log_q K)/n \leq 1 - (t/n)\log_q(q-1) - h_q(t/n), \tag{4.21}$$

where

$$h_q(x) = -x\log_q x - (1-x)\log_q(1-x). \tag{4.22}$$

Proof. Let x be a vector in the code C, and let S_x be the set of all vectors at distance at most t from x. (I.e., all strings that can be reached from x by altering up to t registers.) An error can take x to any string in S_x, so we need that $S_x \cap S_y = \emptyset$ when $x \neq y$. Otherwise, $z \in S_x \cap S_y$ can't be reliably decoded since it could have come from either x or y before the error.

Let us count the size of S_x. We break S_x into subsets $S_{x,j}$ which disagree with x on exactly $j \leq t$ registers. $S_{x,j}$ breaks down further into subsets which depend on which j registers disagree. There are $\binom{n}{j}$ possible sets of registers which disagree, and for a fixed set of registers, the strings in $S_{x,j}$ can take on any value except for the values in x. There are $q-1$ remaining choices for each of the j registers which disagree. Thus, the total size of $S_{x,j}$ is $(q-1)^j\binom{n}{j}$ and the total size of S_x is

$$\sum_{j=0}^{t}(q-1)^j\binom{n}{j}. \tag{4.23}$$

This is true for all x, and since $S_x \cap S_y = \emptyset$, the total number of strings in the union of all sets S_x is K times equation (4.23). This must be less than the total number of possible n-register strings, which is q^n, giving us equation (4.20).

To find the large n version of this equation, just take the logarithm base q. The h_q term comes from the log of the binomial coefficient (lemma 4.17), and only the largest value of $j = t$ contributes to the logarithm for large n. □

The formula for $h_q(x)$ is a fairly standard information-theoretic term, but it is sufficiently useful that I'll give a proof of it:

Lemma 4.17. *For large n,*

$$\log_q\binom{n}{j} = -j\log_q(j/n) - (n-j)\log_q(1-j/n) + o(n). \tag{4.24}$$

Proof.

$$\log \binom{n}{j} = \log(n!) - \log(j!) - \log[(n-j)!]. \tag{4.25}$$

(Assume everywhere that the base of the logarithm is q.)

Taking the logarithm of Stirling's formula and keeping only the terms at least linear in n, we have

$$\log(m!) = m \log m - m \log e. \tag{4.26}$$

Then

$$\log \binom{n}{j} = (n \log n - n \log e) - (j \log j - j \log e) - [(n-j)\log(n-j) - (n-j)\log e] \tag{4.27}$$

$$= j \log n + (n-j)\log n - j \log j - (n-j)\log(n-j) - [n - j - (n-j)]\log e \tag{4.28}$$

$$= -j \log(j/n) - (n-j)\log[(n-j)/n]. \tag{4.29}$$

□

The Hamming bound is also known as the *sphere-packing bound*. The name comes from the sets S_x, which can be viewed as "spheres" using the distance measure which counts the number of registers in which two strings differ. That metric is known as the *Hamming distance*. The spheres S_x are rather blocky for spheres, but that's what you get when you use a discrete distance.

The case when the Hamming bound is met exactly is somewhat interesting. First, it represents the best possible code you can have for a given n and $d = 2t + 1$. Second, when it is a linear code, it means that every error syndrome is used by an error of weight t or less.

Definition 4.14. A code which saturates the Hamming bound is *perfect*.

When $t = 1$, the condition for a perfect code is $K[1 + (q-1)n] = q^n$. Let us specialize to $K = q^k$. Then an $[n, k, 3]_q$ code is perfect if $(q-1)n = q^{n-k} - 1$. Letting $r = n - k$, we find that $n = (q^r - 1)/(q-1)$, $k = n - r$. These are exactly the parameters of the Hamming codes. The Hamming codes were designed to use up every error syndrome, so it's not surprising they are perfect codes, but now you can see that these parameters are the only ones possible for distance 3 perfect codes.

4.5.2 Gilbert-Varshamov Bound

The Hamming bound tells us that codes above a certain level of efficiency cannot exist — it is an *upper bound* on the efficiency of error-correcting codes. The Gilbert-Varshamov bound is a *lower bound*. It tells us that efficient codes do exist, provided we lower our standards a bit.

Theorem 4.18 (Gilbert-Varshamov bound)**.** *If n, K, and d satisfy*

$$K \left(\sum_{j=0}^{d-1} (q-1)^j \binom{n}{j} \right) \leq q^n, \tag{4.30}$$

then an $(n, K, d)_q$ code exists. For large n, a code exists if

$$(\log_q K)/n \leq 1 - (d/n)\log_q(q-1) - h_q(d/n), \tag{4.31}$$

with $h_q(x)$ given by equation (4.22).

The difference between the upper bound given by the Hamming bound and the lower bound given by the Gilbert-Varshamov bound is replacing t (the number of correctable errors) with $d - 1$ (the number of detectable errors), which is $2t$. The asymptotic (large n) versions of the two bounds for $q = 2$ are plotted in figure 4.2, along with the Singleton bound (section 4.5.3). Of course, there may exist codes whose efficiency is in the region between the Hamming and Gilbert-Varshamov bounds, but this is not the generic case.

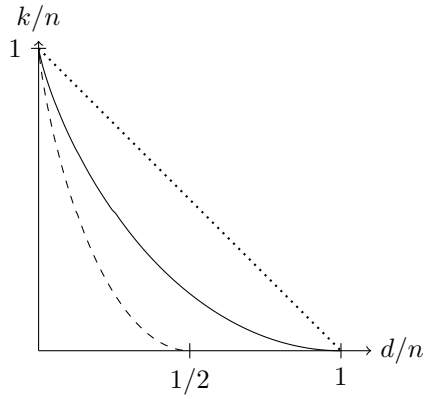


Figure 4.2: Classical Hamming bound (solid), Gilbert-Varshamov bound (dashed), and Singleton bound (dotted) for large n, $q = 2$

Proof. We will pick the codewords sequentially, thus showing the theorem by induction on K.

For the first codeword, pick any string x_1. Then exclude the Hamming sphere of radius $d-1$ around x_1, that is, the set S_{x_1} of all strings that are distance $d-1$ or less from x_1. Choose the second codeword x_2 to be any string outside S_{x_1}. x_2 has distance at least d from x_1, so the code $\{x_1, x_2\}$ has distance at least d.

In general, given any code $\{x_1, \ldots, x_{K-1}\}$ with distance at least d, exclude the radius $d-1$ Hamming spheres $S_{x_1}, \ldots, S_{x_{K-1}}$. Provided equation (4.30) is satisfied, there is at least one string not excluded. Choose it (at random if there is more than one) as the Kth codeword x_K. Then $\{x_1, \ldots, x_K\}$ also has distance at least d, since x_K is no closer than a Hamming distance d to any of the previous codewords. □

The proof can be strengthened to say not just that there exist codes with the parameters $(n, K, d)_q$, but actually that a *randomly chosen* (n, K) code has distance at least d (with high probability for large n). The Gilbert-Varshamov bound can also be improved to show the existence of codes with certain properties. For instance, when $K = 2^k$, there exists a linear code provided n, K, and d satisfy equation (4.30).

The proof of the Gilbert-Varshamov bound is sometimes referred to as *non-constructive*. Of course, in a literal sense it is constructive, since if we try all codes with the given parameters, we will eventually find one that works. In this context, the term means that it is not efficiently constructive in at least one of two ways. It can mean that picking a code according the algorithm implicitly given by the proof produces a code whose description is exponentially large in n. In this case, that is true, since $K = \exp(O(n))$ and each codeword has to be listed separately, but if we use the version of the Gilbert-Varshamov bound applying to linear codes, there is at least an efficient description of the resulting code in terms of the generator matrix. It could also mean that the algorithm for finding the code takes exponentially long in n. That is true both for the general version and linear code version of the Gilbert-Varshamov bound, since checking that a particular code has distance d takes exponentially long.

4.5.3 Singleton Bound

The Singleton bound is another upper bound, but it has a much simpler form than the Hamming bound. In particular, the form of the Singleton bound is the same for all register sizes q.

Theorem 4.19 (Singleton bound)**.** *If an $(n, q^k, d)_q$ code exists, then*

$$n - k \geq d - 1. \tag{4.32}$$

Proof. An $(n, q^K, d)_q$ code corrects $d-1$ erasure errors. For instance, if we discard the last $d-1$ registers, the remaining $n-(d-1)$ registers can still be used to reconstruct all k logical registers. This is only possible

if $n - (d-1) \geq k$ since otherwise two different logical strings will be represented by the same string with $n - (d-1)$ components. □

Codes which saturate the Singleton bound have interesting properties, and therefore get their own name.

Definition 4.15. If an $(n, q^k, d)_q$ code satisfies $n = k + d - 1$, it is a *MDS* code.

"MDS" stands for "maximum distance separable." You can probably figure out the "maximum distance" part of the name. The "separable" part means that k coordinates of the codewords can be taken to be the logical registers when no errors are present. This is not a property that is possible for a quantum code, since it would violate the no-cloning theorem (if k qubits contain the logical state, then the remaining qubits contain no information about it, and would not be useful for error correction). I will still use the term "quantum MDS codes" for codes satisfying the analogous quantum Singleton bound.

Checking the parameters, you'll find that the Reed-Solomon codes are MDS codes. The repetition codes $[n, 1, n]$ are too. The Hamming codes with $r \geq 3$ are not — even though they are optimal for the Hamming bound, they do not saturate the Singleton bound.

4.5.4 Properties of MDS Codes

The dual code of an $[n, k, d]$ code is an $[n, n-k]$ code, but its distance might be bad. A remarkable property of MDS codes, and one that is very useful for making quantum codes, is that their duals also have good distance.

Theorem 4.20. *The dual of an MDS code is also an MDS code. In particular, the dual of an* $[n, k, n-k+1]$ *code is an* $[n, n-k, k+1]$ *code.*

Proof. We wish to show that any codeword from the dual code has weight greater than k. This will show that the distance of the dual code is at least $k+1$, but it cannot be any higher than that by the Singleton bound.

In terms of equations, we wish to show that if C is the original code and y is some string with $0 < \operatorname{wt} y \leq k$, then $\exists x \in C$ such that $y \cdot x \neq 0$. Recalling the proof of the Singleton bound, let us erase $d-1$ registers from an MDS code, leaving only k remaining registers, which are chosen to include the support of y. Call R the set of k registers we are considering. Since the logical state can be reconstructed from R, they must take on every possible value. That is, for any string x' with support within R, $\exists x \in C$ such that $x|_R = x'$. Let $y|_R = y'$. Since $y' \neq 0$, there certainly exists some x' such that $y' \cdot x' \neq 0$, and then $y \cdot x \neq 0$ whenever y has support on R. Since R was an arbitrary set of size k, this shows that if $y \in C^\perp$, then $\operatorname{wt} y > k$. □

Chapter 5

Combining The Old And The New: Making Quantum Codes From Classical Codes

Now that we've covered the basics of both classical and quantum error-correcting codes, we're ready to try combining them. By borrowing some codes from the old theory of classical error-correcting codes, we'll be able to make brand new quantum error-correcting codes, thus marrying quantum and classical error correction. But don't be blue, I'm sure there are still plenty of interesting quantum codes left to be discovered.

5.1 CSS Codes

5.1.1 CSS Construction and Parameters of a CSS Code

The first construction we'll discuss gives us a class of codes known as CSS codes after their inventors Calderbank, Shor, and Steane. The idea of CSS codes is to build on the observation that classical linear codes are a special case of stabilizer codes. In theorem 4.4, we made a stabilizer code by taking the parity check matrix for a classical linear code and replacing each 1 with a Z. The resulting stabilizer code corrects the same set of bit flip errors as the original classical code. If you instead replace the 1's in the parity check matrix with X's, you get a stabilizer code that corrects phase flip errors. In a CSS code, we do both: some of the stabilizer generators come from one classical linear code C_1 with the parity check matrix converted to Z's, and some generators come from a second linear code C_2 which is used to correct phase errors.

Definition 5.1. A stabilizer code is a *CSS code* if there is a choice of generators for which the stabilizer's binary symplectic representation is of the form

$$\left(\begin{array}{c|c} 0 & A \\ B & 0 \end{array}\right), \tag{5.1}$$

where A is a $r_1 \times n$ matrix and B is a $r_2 \times n$ matrix for some r_1, r_2. The generators of the form $(b|0)$ are X *generators* and the generators of the form $(0|a)$ are Z *generators*.

In other words, some generators for a CSS code are tensor products of only X and I and some are tensor products of only Z and I. This statement is of course dependent on the exact choice of generators. If you pick a different set of generators by multiplying some of the X generators with some of the Z generators, you will usually get generators which involve more than one of X, Y, and Z in the same operator. That doesn't mean the code is not a CSS code, only that you've picked a strange set of generators.

	Z	Z	Z	Z	I	I	I
	Z	Z	I	I	Z	Z	I
	Z	I	Z	I	Z	I	Z
	X	X	X	X	I	I	I
	X	X	I	I	X	X	I
	X	I	X	I	X	I	X
$\overline{X}$	X	X	X	X	X	X	X
$\overline{Z}$	Z	Z	Z	Z	Z	Z	Z

Table 5.1: The stabilizer and logical Paulis for the 7-qubit code.

The CSS construction allows us to take two classical codes and make a quantum code. As an example, let's form a 7-qubit code from the 7-bit Hamming code discussed in section 4.2.3. The second code will also be the 7-bit Hamming code. Take the three rows in its parity check matrix and convert the 1's to Z's, getting three generators of the stabilizer. Then take the rows a second time and convert the 1's to X's, getting three more generators. The resulting stabilizer is shown in table 5.1.

What is the distance of the 7-qubit code? Using the Z generators of the stabilizer, we can detect and identify any single-qubit bit flip error, since those generators are derived from a classical code which can do so. The first three bits of the error syndrome tell us where a bit flip error is. Using the same logic, the last three bits tell us where any single-qubit phase error has occurred. If there is a Y error, or indeed an X error on one qubit and a Z error on another, then the error will show up in both the first three bits and the last three bits of the error syndrome, identifying it as an error combining X and Z. Thus, the code can correct any single-qubit X, Y, or Z error, and has distance 3.

How many encoded qubits do we have? Using proposition 3.3, we have $n = 7$ physical qubits and 6 generators, so there should be 1 encoded qubit. But hold on a minute. In order to have *any* encoded qubits, we can't just take an arbitrary set of Paulis and call them the generators of a stabilizer. In particular, to have a stabilizer, we need to check that the generators we've written down commute with each other. The Z generators automatically commute with each other and the X generators commute with each other, so all we need to check is that every Z generator commutes with each X generator. In the case of the 7-qubit code, they do. Therefore, the 7-qubit code is well-defined as a $[[7, 1, 3]]$ code. The 7-qubit code is also known as the *Steane code*, since Steane first proposed it.

In the case of the 7-qubit code, we derived both the X and Z generators from the same code, the $[7, 4, 3]$ Hamming code. For a more general CSS code, we don't need to do that. We can use C_1 to correct bit flip errors and C_2 to correct phase errors.

Theorem 5.1. *Let C_1 be an $[n, k_1, d_1]$ classical linear code with parity check matrix H_1 and C_2 be an $[n, k_2, d_2]$ classical linear code with parity check matrix H_2. Suppose ${C_1}^\perp \subseteq C_2$. Let S be the CSS code with stabilizer*

$$\left(\begin{array}{c|c} 0 & H_1 \\ H_2 & 0 \end{array}\right). \tag{5.2}$$

Then S is an $[[n, k, d]]$ quantum code with $k = k_1 + k_2 - n$ and $d \geq \min\{d_1, d_2\}$.

There are a couple of things to notice about the statement of the theorem. You might think that the condition ${C_1}^\perp \subseteq C_2$ implies an asymmetry between C_1 and C_2, but that is not the case. Actually, the theorem treats them on an equal basis because ${C_1}^\perp \subseteq C_2 \Leftrightarrow {C_2}^\perp \subseteq C_1$. Also, observe that if $n > k_1 + k_2$, the theorem would predict a stabilizer code with a negative number of encoded qubits. That can't be right, and of course, it isn't. When $n > k_1 + k_2$, it is not possible that ${C_1}^\perp \subseteq C_2$.

While the theorem is phrased as just one way to make a CSS code, by comparing the theorem and the definition of a CSS code, you can see that it is actually the *only* way to make a CSS code. In the future, I'll refer to the codes C_1 and C_2 of a general CSS codes, indicating the classical codes that produce the Z generators and X generators, respectively. The choice of whether C_1 has the X generators or the Z

generators is an arbitrary convention, and you may find the other choice in the literature. Also, sometimes people will use the name "C_1" when they mean what I am calling "${C_1}^\perp$." Again, this is somewhat an arbitrary convention.

Proof. The main thing we need to check is that the X generators commute with the Z generators. Any X generator is of the form $(x|0)$, where $x \in {C_2}^\perp$. It is in the dual since it is derived from a row of the parity check matrix of C_2. Any Z generator is derived from the parity check matrix of C_1, so it has the form $(0|z)$, with $z \in {C_1}^\perp$. Then

$$(x|0) \odot (0|z) = x \cdot z, \tag{5.3}$$

with the usual binary inner product on the right. Thus, the stabilizer is Abelian iff $x \cdot z = 0$ for all $x \in {C_2}^\perp$, $z \in {C_1}^\perp$. Equivalently, we could say that if $z \in {C_1}^\perp$, then $z \in ({C_2}^\perp)^\perp$. Since $({C_2}^\perp)^\perp = C_2$, that produces the condition ${C_1}^\perp \subseteq C_2$.

Now let us determine the parameters of S. H_1 has $n - k_1$ rows and H_2 has $n - k_2$ rows, so the stabilizer has $2n - k_1 - k_2$ generators. Therefore, it has $n - (2n - k_1 - k_2) = k_1 + k_2 - n$ logical qubits. The code can detect up to $d_1 - 1$ bit flip errors using the Z generators, and it can detect up to $d_2 - 1$ phase errors using the X generators, and detecting bit flip errors does not in any way interfere with detecting phase errors. Any Pauli of weight less than $\min\{d_1, d_2\}$ can be written as a product PQ, with P a tensor product of X and I with weight $< d_1$ and Q a tensor product of Z and I with weight $< d_2$. The Pauli is non-trivial if at least one of P and Q is non-trivial, in which case it can be detected by looking at the appropriate bits of the error syndrome. Thus, the distance is at least $\min\{d_1, d_2\}$. □

From theorem 5.1, you can see why I made such a big deal about determining the duals of codes in chapter 4. The $[7, 4, 3]$ Hamming code contains its own dual, which is why we can use two copies of it to make the 7-qubit code. We can get other CSS codes by using the other example classical codes. For instance, let $r > m - r - 1$. Then $\mathcal{R}(r, m)^\perp = \mathcal{R}(m - r - 1, m) \subset \mathcal{R}(r, m)$, and we can make a CSS code with $C_1 = C_2 = \mathcal{R}(r, m)$. For instance, $\mathcal{R}(2, 4)$ is a $[16, 11, 4]$ code, and using it for C_1 and C_2, we get a $[[16, 6, 4]]$ CSS code.

We don't have to use the same code twice. For instance, we can let $C_1 = \mathcal{R}(2, 4)$ and $C_2 = \mathcal{R}(3, 4)$, which is a $[16, 15, 2]$ code. ${C_2}^\perp = \mathcal{R}(0, 4)$, which is just the 16-bit repetition code. $\mathcal{R}(0, 4) \subset C_1$, so we can make a CSS code, getting a $[[16, 10, 2]]$ code. Of course, this particular construction is not ideal if we're interested in a distance 2 code, since by taking $C_1 = C_2 = \mathcal{R}(3, 4)$, we get a $[[16, 14, 2]$ code, which has the same distance but more encoded qubits. Still, it might be useful if we want a code that detects any single-qubit phase error but can actually correct a bit flip error.

5.1.2 Degeneracy and CSS Codes

Theorem 5.1 says that the distance of S is greater than or equal to $\min\{d_1, d_2\}$. Why not just equal to? After all, since code C_1 has distance d_1, there is a bit flip error of weight d_1 that cannot be detected by code C_1 and therefore the corresponding Pauli error has 0 error syndrome for S. Similarly, C_2 has distance d_2, so there is also a Pauli with weight d_2 which is a tensor product of Z and I and has 0 error syndrome. But for a quantum code, 0 error syndrome is not enough to make an error undetectable. 0 error syndrome means that the error is in $\mathsf{N}(\mathsf{S})$. An error is undetectable only if it is in $\hat{\mathsf{N}}(\mathsf{S}) \setminus \hat{\mathsf{S}}$. That is, when a CSS code is degenerate, its distance could be greater than one might expect simply by examining the two classical codes that make it up.

For instance, the 9-qubit code is a degenerate CSS code. C_1 is composed of three copies of the repetition code. It has distance 3. However, C_2 has the following parity check matrix:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}. \tag{5.4}$$

C_2 contains strings such as $(1, 1, 0, 0, 0, 0, 0, 0, 0)$ which have weight 2. Thus, C_2 has distance 2. (It is easy to check that any single-bit error will have nonzero error syndrome.) If we just took the distance to be $\min\{d_1, d_2\}$, we would think that the nine-qubit code had distance 2. However, the phase error formed from

the short codeword above is Z_1Z_2, which is in S. Similarly for all other weight 2 codewords of C_2. This is why the distance of the nine-qubit code can be 3.

5.1.3 Codewords of a CSS Code in Computational and Dual Basis

CSS codes also look nice when the codewords are written in the standard basis. The projector on the code space is a product of two projectors Π_1 and Π_2. Π_i is the projector on the $+1$ eigenspace of the generators derived from C_i. Π_1 thus projects on the subspace spanned by codewords of C_1. Π_2 can be written as a sum

$$\Pi_2 = \frac{1}{2^{n-k_2}} \left(\sum_{x \in {C_2}^\perp} P_x \right), \tag{5.5}$$

where P_x is the Pauli with binary symplectic representation $(x|0)$. The Paulis of this form (for $x \in {C_2}^\perp$) are the stabilizer elements formed from products of the generators derived from the parity check matrix of C_2.

Therefore, we can find the codewords of S by taking some codeword of C_1 (which is all that can pass Π_1) and applying Π_2. In general, we get something of the form

$$|u + {C_2}^\perp\rangle = \sum_{v \in {C_2}^\perp} |u + v\rangle, \tag{5.6}$$

for $u \in C_1$. Recall that ${C_2}^\perp \subseteq C_1$ and C_1 is linear, so $u + v \in C_1$.

When are two such codewords $|u + {C_2}^\perp\rangle$ and $|u' + {C_2}^\perp\rangle$ equal?

$$0 = |u + {C_2}^\perp\rangle - |u' + {C_2}^\perp\rangle = \sum_{v \in {C_2}^\perp} |u + v\rangle - \sum_{v' \in {C_2}^\perp} |u' + v'\rangle. \tag{5.7}$$

This is only possible if every term in the first sum is cancelled by a term in the second sum. That is, when $v \in {C_2}^\perp$, $u + v = u' + v'$ for some $v' \in {C_2}^\perp$. Thus, $u - u' = v' - v \in {C_2}^\perp$ since ${C_2}^\perp$ is linear. The states $|u + {C_2}^\perp\rangle$ only depend on cosets of ${C_2}^\perp$ within C_1, thus explaining the notation I chose to represent the basis states.

C_1 has 2^{k_1} codewords and ${C_2}^\perp$ has 2^{n-k_2} codewords, so $C_1/{C_2}^\perp$ has $2^{k_1+k_2-n}$ codewords, which is the same as the dimension of S. The codewords $|u + {C_2}^\perp\rangle$ form a basis for the code space of the CSS code.

I claimed that in the CSS construction, the two classical codes used were treated equally, but in this expansion, they certainly seem unequal. The solution lies in looking in the Hadamard rotated basis. The Hadamard transform switches the role of X and Z, so you might expect that it switches the role of C_1 and C_2. If you expected that, you are correct. Let's calculate it explicitly:

$$\begin{aligned}
H^{\otimes n}|u + {C_2}^\perp\rangle &= \sum_{v \in {C_2}^\perp} H^{\otimes n}|u + v\rangle && (5.8)\\
&= \sum_{v \in {C_2}^\perp} \sum_w (-1)^{(u+v)\cdot w}|w\rangle && (5.9)\\
&= \sum_w (-1)^{u\cdot w} \left(\sum_{v \in {C_2}^\perp} (-1)^{v\cdot w} \right) |w\rangle && (5.10)\\
&= \sum_{w \in C_2} (-1)^{u\cdot w}|w\rangle && (5.11)\\
&= \sum_{x \in C_2/{C_1}^\perp} (-1)^{u\cdot x} \sum_{y \in {C_1}^\perp} |x + y\rangle && (5.12)\\
&= \sum_{x \in C_2/{C_1}^\perp} (-1)^{u\cdot x}|x + {C_1}^\perp\rangle && (5.13)
\end{aligned}$$

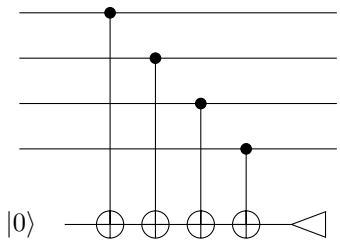


Figure 5.1: A circuit to measure the parity of 4 qubits.

To get the fourth line, observe that if $w \in C_2$, then $v \cdot w = 0$ always, but if $w \notin C_2$, then $v \cdot w = 0$ for half of the values of v and $v \cdot w = 1$ for the other half of the values of v. (It must be nonzero for some v since $w \notin C_2$. Call that v_0. Then if $v \in {C_2}^\perp$, so is $v + v_0$, but exactly one of $v \cdot w = 1$ and $(v + v_0) \cdot w = 1$. Thus we can pair the elements of ${C_2}^\perp$ such that within each pair one is orthogonal to w and one is not.)

In the next-to-last line, I have broken the sum over w into a sum over cosets of ${C_1}^\perp$ and a sum over elements of the cosets. This is a sensible thing to do because $u \cdot w$ only depends on which coset w lies in of ${C_1}^\perp$ in C_2: If $w' = w + y$, $y \in {C_1}^\perp$, then $u \cdot w' = u \cdot w + u \cdot y$, but $u \in C_1$, so $u \cdot y = 0$.

In particular, if we have associated the basis codewords $|u + {C_2}^\perp\rangle$ with logical basis codewords, then we recognize equation (5.13) as a Hadamard transform of the logical codewords, with the new basis codewords $|x + {C_1}^\perp\rangle$. In the standard basis, the CSS code consists of superpositions over cosets of ${C_2}^\perp$ in C_1, while in the Hadamard-rotated basis, it consists of superpositions over cosets of ${C_1}^\perp$ in C_2.

5.1.4 Error Correction for a CSS Code

CSS codes are stabilizer codes, so any general technique for performing error correction on stabilizer codes will work for CSS codes too. However, CSS codes have additional special structure which we can take advantage of to get better error correction procedures. In section 12.3, I'll discuss a fault-tolerant error-correction procedure that is designed for CSS codes, but for now, I'll just discuss a non-fault-tolerant scheme.

For a general stabilizer code, the bits of the error syndrome come from the eigenvalues of the stabilizer generators. For a CSS code, the stabilizer generators break up into two parts, one from the code C_1 and one from the code C_2. The first set of syndrome bits identifies bit flip errors, while the remaining syndrome bits identify phase errors.

In the standard basis, the codewords are superpositions of codewords of C_1, so we can measure the bit flip syndrome by measuring the parity checks of C_1. Of course, we must be careful to measure *only* the parity checks and nothing more, since we don't want to destroy any superposition of logical states. To measure a parity check, you can add a single ancilla qubit in the state $|0\rangle$ and perform CNOT gates from all the qubits involved in the parity check to the ancilla. Then measure the ancilla qubit and the outcome will be the desired parity. See figure 5.1 for an example.

The bit flip syndrome we get this way is exactly the syndrome of the classical code code C_1 if it underwent the same bit flip error. In order to decode the syndrome and learn the actual error, we can just invoke the classical procedure. If the code C_1 has an efficient syndrome decoding algorithm, then we get one for the bit flip errors of the CSS code also.

For the phase errors, we can simply rotate into the Hadamard basis. Now the codewords are superpositions of codewords of C_2, and the phase errors have become bit flip errors. If, after the Hadamard transform, we measure the parity checks of C_2, that is the same as measuring the phase error syndrome for the CSS code. The phase error syndrome is exactly the same as the error syndrome for C_2 if it underwent bit flip errors corresponding to the locations of the phase errors. Again, we can use the syndrome decoding algorithm for C_2 to identify the phase errors for the CSS code.

$+$	0	1	ω	ω^2
0	0	1	ω	ω^2
1	1	0	ω^2	ω
ω	ω	ω^2	0	1
ω^2	ω^2	ω	1	0

$\times$	0	1	ω	ω^2
0	0	0	0	0
1	0	1	ω	ω^2
ω	0	ω	ω^2	1
ω^2	0	ω^2	1	ω

Table 5.2: The addition and multiplication tables for GF(4).

5.2 GF(4) Codes and Stabilizer Codes

Now we'll return to more general stabilizer codes. CSS codes have many nice properties, but they are not perfect. For instance, general stabilizer codes can be more efficient than CSS codes — the 7-qubit code is the smallest CSS code that corrects 1 error, whereas the 5-qubit code, a stabilizer code, is the best QECC of any type that corrects 1 error. While there are some non-stabilizer codes known that are slightly better than any stabilizer code, the differences are small for codes correcting Pauli channels.

In this section, I'll therefore discuss a technique for adapting some classical codes to get stabilizer codes which are not CSS codes. In order to do so, we'll have to go to non-binary codes.

5.2.1 Correspondence Between GF(4) and the Pauli Group

The main "insight" of this particular technique is that the one-qubit Pauli group has 4 elements, and so does the finite field GF(4). Amazing, no?

Really, we're connecting $\hat{\mathsf{P}}_n$, the Pauli group sans phases, with an n-dimensional vector field over GF(4). For $n = 1$, we simply associate each element of the Pauli group with the elements of GF(4):

$$\begin{array}{ll} I \leftrightarrow 0 & X \leftrightarrow 1 \\ Z \leftrightarrow \omega & Y \leftrightarrow \omega^2 \end{array} \tag{5.14}$$

For n qubits, the ith tensor factor of $P \in \hat{\mathsf{P}}_n$ becomes the ith component of a vector over GF(4) using the rules in equation (5.14).

The conversion works much the same way as the conversion between $\hat{\mathsf{P}}_n$ and the binary symplectic representation of the Pauli group. Multiplication in $\hat{\mathsf{P}}_n$ becomes addition in GF(4). The full correspondence is summarized in table 5.3. It's also worth recalling the addition and multiplication tables for GF(4), which are given in table 5.2.

5.2.2 The GF(4) Symplectic Inner Product

As with the binary symplectic representation, we need to recover somehow the notation of commutation, and again we turn to a symplectic inner product. We want some map from GF(4) $\times$ GF(4) to $\mathbb{Z}_2$ which gives 0 when one input is 0 or both inputs are the same, and gives 1 otherwise. It turns out that the correct formula is $\mathrm{tr}(\overline{a}b)$. The $\overline{a}$ and trace operations are defined as follows for GF(4):

$$\begin{array}{llll} \overline{0} = 0 & \overline{1} = 1 & \mathrm{tr}\, 0 = 0 & \mathrm{tr}\, 1 = 0 \\ \overline{\omega} = \omega^2 & \overline{\omega^2} = \omega & \mathrm{tr}\, \omega = 1 & \mathrm{tr}\, \omega^2 = 1 \end{array} \tag{5.15}$$

You can check by trying all combinations that this formula works. The generalization to n-dimensional vectors is then straightforward:

$$a * b = \mathrm{tr}(\overline{a} \cdot b), \tag{5.16}$$

where $\cdot$ is just the usual dot product. In the classical coding literature, this inner product is sometimes known as the *trace-Hermitian inner product*.

In the Pauli group	In GF(4)
I	0
X	1
Z	ω
Y	ω^2
Multiplication	Addition
$c(P,Q)$	$a * b = \mathrm{tr}(\overline{a} \cdot b)$
Phase	No equivalent
No equivalent	Multiplication
Unitary T (see chapter 6)	Multiplication by ω
Stabilizer S	Weakly self-dual additive code S
Normalizer $\mathsf{N}(\mathsf{S})$	Dual $\mathsf{S}^\perp$ (under $*$)

Table 5.3: Equivalence between the Pauli group and GF(4).

5.2.3 Stabilizer Codes as GF(4) Codes

We can convert a stabilizer to sets of vectors over GF(4) just as we converted them to the binary symplectic representation. We'd like to interpret the resulting set as a classical error-correcting code over GF(4). Formally, there is no problem in doing so. However, what we get is not necessarily a linear code. Sometimes we do get a linear code, as in the example of the five-qubit code (converted to a GF(4) code in table 5.4), but it is not hard to come with stabilizer codes which don't produce linear GF(4) codes.

Because $P, Q \in \mathsf{S} \Rightarrow PQ \in \mathsf{S}$, the GF(4) code S which we get is additive (closed under addition). However, for it to be linear, S would also need to be closed under multiplication by ω, and that is not necessarily true. (It also needs to be closed under multiplication by ω^2, but that follows automatically if it is closed under multiplication by ω.)

The other condition we need to satisfy is that the stabilizer is Abelian. We can express this in terms of a dual with respect to the symplectic product $*$. The dual of an additive code is additive, and the dual of the GF(4) conversion of a stabilizer is the GF(4) conversion of the normalizer. An additive code corresponds to an Abelian group if it is weakly self-dual.

We now have the main components we need to convert GF(4) codes into stabilizer codes.

Theorem 5.2. *Suppose C is an additive code over* GF(4) *which is weakly self-dual under $*$. Then C can be converted to a stabilizer S with parameters $[[n, k, d]]$. n is the number of physical registers in C. If C contains $K = 2^r$ codewords, then $k = n - r$. The distance d of S is the smallest weight of a member of $C^\perp \setminus C$, and in particular, d is at least equal to the distance of $C^\perp$.*

Note that there are a number of peculiar things involved in the conversion. In some sense $C^\perp$ is more closely analogous to the quantum code we get, since the distance of S is at least equal to the distance of $C^\perp$. However, the number of encoded qubits is a formula that does not show up at all in classical coding theory (which is more interested in writing $K = 4^{r'}$ than $K = 2^r$), and of course the distance can be even larger than the distance of $C^\perp$ because of degeneracy. And it is important to bear in mind that the dual is taken with respect to the symplectic product $*$ rather than the usual inner product.

The upshot is that if we look at the most common classical GF(4) codes and try to convert them into quantum codes, we won't get the most general stabilizer code. But who cares about that? We only want good codes, and if we can get them by looking at existing GF(4) codes, that's good enough.

5.2.4 Linear GF(4) Codes

For linear GF(4) codes, the conditions in theorem 5.2 simplify somewhat. In particular, we have the following:

Proposition 5.3. *If C is a linear* GF(4) *code, then its dual with respect to the inner product $\overline{x} \cdot y$ is the same as the dual with respect to $*$.*

1	ω	ω	1	0
0	1	ω	ω	1
1	0	1	ω	ω
ω	1	0	1	ω

Table 5.4: The five-qubit code converted to a GF(4) code.

Proof. We wish to show that $\overline{x} \cdot y = 0 \ \forall y \in C$ iff $x * y = 0 \ \forall y \in C$. The forward direction is trivial, so we only need to show the backwards direction.

Suppose $\mathrm{tr}(\overline{x} \cdot y) = 0$, but $\overline{x} \cdot y = 1$. Then $\overline{x} \cdot (\omega y) = \omega$ and $x * (\omega y) = \mathrm{tr}(\overline{x} \cdot (\omega y)) = 1$. Therefore, since C is linear, if $x * y = 0 \ \forall y \in C$, then $\overline{x} \cdot y = 0 \ \forall y \in C$. □

This simplifies the procedure of checking for weakly self-dual codes because we can use the dual under the standard inner product and then take the conjugate rather than have to compute the dual under the unusual symplectic inner product $*$.

5.2.5 Example: Perfect Qubit Codes

Now let's look at some concrete examples of constructing stabilizer codes from GF(4) codes. It turns out that the GF(4) Hamming codes have the right properties, or rather their duals do.

Theorem 5.4. *The duals (with respect to the standard inner product) of the Hamming codes over* GF(4) *can be converted to stabilizer codes. The dual of the* $[(4^r - 1)/3, (4^r - 1)/3 - r, 3]_4$ *Hamming code becomes a* $[[(4^r - 1)/3, (4^r - 1)/3 - 2r, 3]]$ *qubit stabilizer code.*

Notice that the resulting stabilizer code encodes $(4^r - 1)/3 - 2r$ qubits whereas the Hamming code encodes $(4^r - 1)/3 - r$ GF(4) registers. This is a consequence of using a base of 2 instead of 4 to count registers, as mentioned in the discussion of theorem 5.2.

Proof. The Hamming codes are linear, so we need to show that a Hamming code contains its dual relative to $\overline{x} \cdot y$. Looking at the construction of the non-binary Hamming codes in the proof of theorem 4.14, we see that if we discard the last row of the parity check matrix, the first column is all 0, and then every other column is repeated four times. That means that the inner product of any two of these rows will be zero, since we end up adding the same thing four times.

Next, we should show that we also get 0 if we take the inner product of the last row with another row i. Since the first column of row i is 0, we can ignore the first column. The other columns break up into sets of four, within which row i has some value a repeated four times, while the last row runs over all the values in GF(4): 0, 1, ω, and ω^2. Whatever the value of a,

$$\overline{a}0 + \overline{a}1 + \overline{a}\omega + \overline{a}\omega^2 = \overline{a}(0 + 1 + \omega + \omega^2) = 0. \tag{5.17}$$

That proves that the parity check matrix of the Hamming code can be converted into a stabilizer. The number of encoded qubits follows from theorem 5.2.

The dual (with respect to $\overline{x} \cdot y$) of the dual (with respect to the standard inner product) of a Hamming code is just the conjugate of the Hamming code, produced by replacing x with $\overline{x}$ everywhere in the code. The conjugate of a code has the same distance as the code since taking the conjugate does not change the weight, and therefore, the stabilizer codes we have derived have distance at least 3. In fact, these codes are non-degenerate, so the distance is exactly 3. □

We get codes with parameters $[[5, 1, 3]]$, $[[21, 15, 3]]$, $[[85, 77, 3]]$, etc. The $[[5, 1, 3]]$ code produced this way (shown in table 5.5) is equivalent to the $[[5, 1, 3]]$ code we discussed before. Note that while the classical $[5, 3, 3]_4$ Hamming code has two rows in its parity check matrix, the quantum $[[5, 1, 3]]$ code has four generators of its stabilizer. The vectors v, ωv are considered linearly dependent for a code over GF(4), so we only need

					$I \quad X \quad X \quad X \quad X$
0	1	1	1	1	$I \quad Z \quad Z \quad Z \quad Z$
1	0	1	ω	ω^2	$X \quad I \quad X \quad Z \quad Y$
					$Z \quad I \quad Z \quad Y \quad X$

Table 5.5: The parity check matrix of the five-bit GF(4) Hamming code and the five-qubit code derived from it.

to include one of them in the parity check matrix, but they convert to Paulis which are independent when considered as operators on qubits, so we need to list them both in the stabilizer.

This family of quantum codes is interesting because, like the classical Hamming codes they are derived from, the codes in this family use up all of the error syndromes. The $[[(4^r-1)/3, (4^r-1)/3-2r, 3]]$ code has $2r$ stabilizer generators, so 4^r error syndromes. There are $3n+1 = 4^r$ zero and one-qubit errors, and the code is non-degenerate, so each syndrome is used exactly once.

It is unclear what the correct definition of a perfect degenerate QECC should be, but a non-degenerate quantum code is *perfect* if the number of correctable errors is exactly equal to the number of error syndromes. Thus, the QECCs derived from the GF(4) Hamming codes are perfect qubit codes.

Chapter 6

Symmetries Of Symmetries: The Clifford Group

When dealing with stabilizer codes, it is helpful to restrict attention to a set of quantum gates that is guaranteed to treat the code nicely. There exists a unitary operation that will take any subspace (for instance, a quantum error-correcting code) into any other subspace of the same dimension. Sometimes that's exactly what you want. However, if you've taken the effort to work with a code with a nice tractable description in terms of its stabilizer, you don't want your work ruined by using a poorly-thought-out unitary. In general, quantum gates will map the code space of a stabilizer code into some other code, which might not be a stabilizer code.

The Clifford group is a group of unitary gates that is specifically chosen so that it does not do this. If you start with a stabilizer code and perform a Clifford group gate, you will always have another stabilizer code. The key to this is using only unitary operations which can be thought of as permutations of the Pauli group. A Clifford group operation then just switches one stabilizer into another.

6.1 Definition of the Clifford Group

6.1.1 Motivation for the Clifford Group

Suppose we perform the unitary U on a state $|\psi\rangle$ from a stabilizer code S. What happens to the stabilizer? Suppose $M \in \mathsf{S}$, so $M|\psi\rangle = |\psi\rangle$. We want to find M' for which $U|\psi\rangle$ is a $+1$ eigenstate. It turns out the correct choice is $M' = UMU^\dagger$:

$$M'U|\psi\rangle = UMU^\dagger U|\psi\rangle = UM|\psi\rangle = U|\psi\rangle, \tag{6.1}$$

since U is unitary. Running over all M in the stabilizer, we find that $\mathsf{S}' = \{UMU^\dagger | M \in \mathsf{S}\}$ is a set of operators for which all states $U|\psi\rangle$ are $+1$ eigenstates (where $|\psi\rangle$ is any element of $\mathcal{T}(\mathsf{S})$).

We'd *like* to say that S' is the new stabilizer of the code, with code space $U(\mathcal{T}(\mathsf{S}))$. However, the catch is that without any additional constraint on U, S' might contain many non-Paulis, and the stabilizer is supposed to be a subset of P_n. In addition, the true stabilizer of the subspace $U(\mathcal{T}(\mathsf{S}))$ might contain additional Paulis that are not in S'. Furthermore, $U(\mathcal{T}(\mathsf{S}))$ might not be a stabilizer code — the Paulis in $\mathsf{S}(U(\mathcal{T}(\mathsf{S})))$ might be insufficient to specify the subspace. (Recall definition 3.4.)

The Clifford group is a set of unitaries that does not have these complications. It maps stabilizers to stabilizers. Fortunately, the Clifford group contains many interesting quantum gates; unfortunately, it is not enough for a universal quantum computer. The Clifford group is sufficient for encoding stabilizer codes, and gives a good start on fault-tolerant operations, but eventually we will need to go beyond it.

6.1.2 Definition of the Clifford Group and Variants

Definition 6.1. The *Clifford group* C_n on n qubits is the normalizer of P_n in the unitary group $\mathsf{U}(2^n)$. That is,

$$\mathsf{C}_n = \{U \in \mathsf{U}(2^n) | UPU^\dagger \in \mathsf{P}_n \ \forall P \in \mathsf{P}_n\}. \tag{6.2}$$

The name "Clifford group" is not particularly illuminating, perhaps. It is motivated by the idea that there might be a connection of some sort to Clifford algebras, but the connection is not very close. To add to the confusion, you might encounter the term "Clifford group" in the mathematics literature referring to a different group. In quantum information papers, Clifford group refers to definition 6.1 or one of its variants defined below. You might also encounter the terms "normalizer group," "symplectic group" (or operations), or sometimes "stabilizer operations" for C_n. None of these terms is completely satisfactory, and "Clifford group" is the most widespread, so I will use that.

The Clifford group contains all gates of the form $e^{i\theta}I$, and if $U \in \mathsf{C}_n$, then $e^{i\theta}U \in \mathsf{C}_n$. As with the Paulis, global phase is frequently not significant for Clifford group elements. Indeed, it is *less* likely to matter for the Clifford group. Anticommutation is less important in the Clifford group than it is in the Pauli group, so we will usually consider not the full Clifford group, but the Clifford group with phases removed.

By definition, the Pauli group P_n is a normal subgroup of C_n. Sometimes it is better to consider the Clifford group with the Pauli subgroup modded out. The Pauli group only contains the phases $\pm 1, \pm i$, so even once the Pauli group is gone, there are still additional phases to worry about. Typically, we will want to remove those as well.

Definition 6.2.

$$\hat{\mathsf{C}}_n = \mathsf{C}_n / \{e^{i\theta}I\} \tag{6.3}$$

$$\check{\mathsf{C}}_n = \hat{\mathsf{C}}_n / \hat{\mathsf{P}}_n. \tag{6.4}$$

I will refer to these variants as the "Clifford group," sometimes without specifying which one I mean. Usually it doesn't much matter, or can be deduced from context (or both).

6.1.3 Example Clifford Group Elements

Now let's look at some Clifford gates. We know the Pauli group is a subgroup of C_n, so that gives us one set of examples. The Cliffords are defined to act on P_n by conjugation, and, as you'll see shortly, the best way of characterizing an element of the Clifford group is usually by giving the action under conjugation. Suppose we have $P \in \mathsf{P}_n \subset \mathsf{C}_n$. What does it do to another Pauli Q under conjugation?

$$PQP^\dagger = (-1)^{c(P,Q)} QPP^\dagger = (-1)^{c(P,Q)} Q. \tag{6.5}$$

Thus, $Q \mapsto \pm Q$, with the sign determined by whether P and Q commute or anticommute. The effect of conjugating by a Pauli is to rearrange the signs of other Paulis without changing their identity. This is easy to square with what we know about stabilizers from chapter 3: Applying the error P will move us from the $+1$ eigenspace of Q to the -1 eigenspace of Q iff P and Q anticommute. Moving to the -1 eigenspace is equivalent to modifying the stabilizer to contain $-Q$ instead of Q, which is the way we understand the action of Clifford group elements.

Another gate in the Clifford group is the Hadamard transform H. As we've discussed, the Hadamard transform switches the role of X and Z. This can be made concrete by looking at the conjugation action of H. You can work it out by multiplying together the matrices, but I'll just tell you the answer:

$$HXH = Z \tag{6.6}$$

$$HYH = -Y \tag{6.7}$$

$$HZH = X. \tag{6.8}$$

$H^\dagger = H$, so I've skipped the adjoints in the above equations. As you can see, the action of H is indeed to switch X and Z. The effect on states is to switch Z eigenstates with X eigenstates:

$$|0\rangle \text{ (stabilizer } Z) \leftrightarrow |+\rangle = |0\rangle + |1\rangle \text{ (stabilizer } X) \tag{6.9}$$

$$|1\rangle \text{ (stabilizer } -Z) \leftrightarrow |-\rangle = |0\rangle - |1\rangle \text{ (stabilizer } -X) \tag{6.10}$$

While its action on Y is not as dramatic as its action on X and Z, H does not leave Y completely alone. Instead, it changes the sign of Y, so $+1$ eigenstates of Y get switched with -1 eigenstates of Y:

$$\frac{1}{\sqrt{2}}(|0\rangle + i|1\rangle) \leftrightarrow \frac{1}{2}[(1+i)|0\rangle + (1-i)|1\rangle] = \frac{e^{i\pi/4}}{\sqrt{2}}(|0\rangle - i|1\rangle). \tag{6.11}$$

The conjugation action doesn't tell us about the overall phase $e^{i\pi/4}$ introduced by H, but that has no physical significance anyway.

The other important thing about H's action on Y is that I didn't need to specify it. The action of H on Y can be deduced from its action on X and Z. Conjugation is a *group homomorphism* — the conjugate of the product is the product of the conjugates:

$$UPQU^\dagger = (UPU^\dagger)(UQU^\dagger). \tag{6.12}$$

In this case,

$$HYH = H(iXZ)H = i(HXH)(HZH) = iZX = -Y. \tag{6.13}$$

Equivalently, we could deduce the action of H on X from the action on Y and Z, or the action on Z from the action on X and Y. The action on I, of course, will always be trivial ($UIU^\dagger = I$).

The Hadamard switches X and Z, but only changes the phase of Y. There are also Clifford group elements that switch other pairs of Paulis. For instance, $R_{\pi/4} \in \mathsf{C}_n$:

$$R_{\pi/4} X R_{\pi/4}^\dagger = Y \tag{6.14}$$

$$R_{\pi/4} Y R_{\pi/4}^\dagger = -X \tag{6.15}$$

$$R_{\pi/4} Z R_{\pi/4}^\dagger = Z. \tag{6.16}$$

This is not 100% analogous to H, since Z really is left alone by $R_{\pi/4}$, but Y picks up a minus sign under H. However, by multiplying with a Pauli, we can take care of that sign issue:

$$(YR_{\pi/4}) X (YR_{\pi/4})^\dagger = Y \tag{6.17}$$

$$(YR_{\pi/4}) Y (YR_{\pi/4})^\dagger = X \tag{6.18}$$

$$(YR_{\pi/4}) Z (YR_{\pi/4})^\dagger = -Z. \tag{6.19}$$

You can deduce these equations by performing the conjugation action of $R_{\pi/4}$ and then following it with the conjugation action of Y, which switches the signs around.

You might wonder if we can completely get rid of the minus sign, rather than simply switching it to another location, by choosing an appropriate Pauli. The answer is no: any Pauli will commute with one Pauli and anticommute with two of them. Thus, it switches the sign of two of the three one-qubit Paulis X, Y, and Z. We can go from one minus sign to three minus signs, but never to zero or two. What we *can* do is change the signs of the generators of P_1 (or P_n when dealing with n qubits) in any way we like. For instance, to change $X \mapsto -X$, $Z \mapsto Z$, conjugate by Z. The change in the sign of Y is determined by the change in the signs of X and Z.

Among two-qubit Clifford group elements, the most famous gate is the CNOT gate, which has the following conjugation action on the Paulis:

$$X \otimes I \mapsto X \otimes X \tag{6.20}$$

$$Z \otimes I \mapsto Z \otimes I \tag{6.21}$$

$$I \otimes X \mapsto I \otimes X \tag{6.22}$$

$$I \otimes Z \mapsto Z \otimes Z. \tag{6.23}$$

We are dealing with the two-qubit Pauli group, and I have taken advantage of the homomorphism property of conjugation to just list the action of CNOT on a generating set of four Paulis for P_2. For any other Pauli, you can deduce the conjugation action of CNOT by multiplying as described above for the Hadamard.

6.1.4 Determining Clifford Group Element From Action on Generating Paulis

Describing the conjugation action of Clifford group elements certainly tells us a lot about the unitary we are dealing with, but you might worry that some information is being lost. Indeed, if $U' = e^{i\theta}U$, then

$$U'PU'^{\dagger} = e^{i\theta}UPU^{\dagger}e^{-i\theta} = UPU^{\dagger}, \tag{6.24}$$

so the conjugation action tells us nothing about the global phase of the unitary. Of course, the global phase doesn't have any physical significance, so this is not a great loss. But perhaps there are more important properties not captured by the conjugation action? The next theorem tells us that there are not:

Theorem 6.1. *Suppose U and V are unitaries which have the same action by conjugation on P_n:*

$$UPU^{\dagger} = VPV^{\dagger} \tag{6.25}$$

for all $P \in \mathsf{P}_n$. Then $U = e^{i\theta}V$ for some θ.

The theorem does not apply only to Clifford group elements; U and V can be *any* unitary gates. When applied to the Clifford group, theorem 6.1 tells us that the conjugation action uniquely identifies elements of $\hat{\mathsf{C}}_n$. Consequently, we will be able to work with elements of $\hat{\mathsf{C}}_n$ using just the conjugation action.

Proof. Note that $V^{\dagger}U$ acts on the Pauli group trivially by conjugation ($P \mapsto P$), so it will be sufficient to consider the case of $V = I$ and show that $U = e^{i\theta}I$.

When $UPU^{\dagger} = P$ for all $P \in \mathsf{P}_n$, that means that U maps any stabilizer state to a state with the same stabilizer. For instance, the basis state $|j\rangle$ is a stabilizer state with stabilizer generated by $(-1)^{j_i}Z_i$, $i = 1, \ldots, n$ (where j_i is the ith bit of j). The only states with this stabilizer are $e^{i\theta_j}|j\rangle$. We conclude that U is diagonal

$$U = \begin{pmatrix} e^{i\theta_1} & 0 & \cdots & 0 \\ 0 & e^{i\theta_2} & \cdots & 0 \\ \vdots & & \ddots & \\ 0 & & \cdots & e^{i\theta_{2^n}} \end{pmatrix} \tag{6.26}$$

We still need to show that all θ_j are the same. Now consider stabilizer states with stabilizer generators X_1 and $(-1)^{j_{i-1}}Z_i$ for $i = 2, \ldots, n$. The eigenstates of those stabilizers are of the form $e^{i\phi_j}|+\rangle|j\rangle$. Applying equation (6.26) to $|+\rangle|j\rangle$, we find

$$e^{i\theta_j} = e^{i\theta_{2^{n-1}+j}} = e^{i\phi_j}. \tag{6.27}$$

Similarly, looking at stabilizers with X_i in place of X_1, we find that

$$e^{i\theta_j} = e^{i\theta_{2^{n-i}+j}}, \tag{6.28}$$

for any j such that the ith bit is 0. Applying all these equalities, we find that all $e^{i\theta_j}$ terms are equal, proving the theorem. □

Which permutations of P_n are allowed for a conjugation action by a Clifford group element? We know that conjugation is a group homomorphism. That means that there is no hope that we can freely choose images for any elements of P_n except for a generating set. In addition, conjugation will always take $-I$ to $-I$. That means that conjugation must preserve commutation and anticommutation:

$$U(PQ)U^{\dagger} = U[(-1)^{c(P,Q)}QP]U^{\dagger} = (-1)^{c(P,Q)}(UQU^{\dagger})(UPU^{\dagger}) \tag{6.29}$$

$$= (UPU^{\dagger})(UQU^{\dagger}) = (-1)^{c(UPU^{\dagger},UQU^{\dagger})}(UQU^{\dagger})(UPU^{\dagger}). \tag{6.30}$$

Thus, $c(UPU^\dagger, UQU^\dagger) = c(P, Q)$.

The usual set of generators for $\hat{\mathsf{P}}_n$ is $\{X_i, Z_i\}$. To keep the right commutation relations, the images of X_i and Z_i must commute with the images of X_j and Z_j for $j \neq i$. However, the images of X_i and Z_i must anticommute with each other.

There is one additional constraint: Conjugation by U is an isomorphism, since it can be inverted by conjugating by $U^\dagger$. Therefore, the generators must map to an independent set of Paulis. However, this property actually follows from the commutation relations, so we don't need to impose it separately.

If these conditions are satisfied, a similar approach to the proof of theorem 6.1 gives a constructive method for finding the unitary corresponding to a given conjugation map. The same procedure applies even for conjugation by a non-Clifford unitary.

Procedure 6.1. Suppose the map $M : \mathsf{P}_n \to \mathsf{U}(2^n)$ is a group homomorphism, with $M(X_i) = \overline{X}_i$, $M(Z_i) = \overline{Z}_i$, such that

$$\begin{aligned} c(\overline{X}_i, \overline{X}_j) = c(\overline{Z}_i, \overline{Z}_j) = 0, \\ c(\overline{X}_i, \overline{Z}_j) = \delta_{ij}. \end{aligned} \tag{6.31}$$

(When $\overline{X}_i$ and $\overline{Z}_j$ are outside the Pauli group, the definition of $c(\cdot, \cdot)$ is the same, and $c(P, Q)$ is undefined here if P and Q neither commute or anticommute.)

The following procedure finds the matrix representation in the standard basis of a U for which conjugation by U performs M:

1. Find the state $|\psi_0\rangle$ which is a $+1$ eigenstate of $\overline{Z}_i$ for all $i = 1, \ldots, n$.
2. Let b be any number from 0 to $2^n - 1$, and let b_i be the ith bit of b. Let
$$\overline{X}(b) = \prod_i (\overline{X}_i)^{b_i}. \tag{6.32}$$
3. Let $|\psi_b\rangle = \overline{X}(b)|\psi_0\rangle$.
4. Let
$$U_{ab} = \langle a|\psi_b\rangle. \tag{6.33}$$

Proof of the validity of procedure 6.1. M is a group homomorphism and $Z_i = Z_i^\dagger$, so it follows that $\overline{Z}_i = \overline{Z}_i^\dagger$ as well. All of the $\overline{Z}_i$ commute with each other, so $\Pi = \frac{1}{2^n} \prod_i (I + \overline{Z}_i)$ is a projector with trace 1. Thus, there is a unique state $|\psi_0\rangle$ (up to global phase) which is a $+1$ eigenstate of all $\overline{Z}_i$, as needed for step 1. The remaining steps of the procedure give a linear map $U|b\rangle = |\psi_b\rangle$.

I claim that $|\psi_b\rangle$ is an orthonormal basis, so U is unitary:

$$\langle \psi_b | \psi_{b'} \rangle = \langle \psi_0 | (\overline{X}(b))^\dagger \overline{X}(b') | \psi_0 \rangle \tag{6.34}$$

$$= \langle \psi_0 | \prod_i (\overline{X}_i)^{b'_i - b_i} | \psi_0 \rangle \tag{6.35}$$

using the fact that the $\overline{X}_i$ commute with each other. As with the $\overline{Z}_i$s, $\overline{X}_i = \overline{X}_i^\dagger$. Then since $|\psi_0\rangle$ is a $+1$ eigenstate of $\overline{Z}_i$,

$$\langle \psi_b | \psi_{b'} \rangle = \langle \psi_0 | \prod_i (\overline{X}_i)^{b'_i + b_i} | \psi_0 \rangle \tag{6.36}$$

$$= \langle \psi_0 | \overline{Z}_j \prod_i (\overline{X}_i)^{b_i + b'_i} | \psi_0 \rangle \tag{6.37}$$

$$= \prod_i (-1)^{(b_i + b'_i) c(\overline{Z}_j, \overline{X}_i)} \langle \psi_0 | \prod_i (\overline{X}_i)^{b_i + b'_i} \overline{Z}_j | \psi_0 \rangle \tag{6.38}$$

$$= (-1)^{b_i + b'_i} \langle \psi_0 | \prod_i (\overline{X}_i)^{b_i + b'_i} | \psi_0 \rangle \tag{6.39}$$

$$= (-1)^{b_i + b'_i} \langle \psi_b | \psi_{b'} \rangle. \tag{6.40}$$

It follows that if, for any j, $b_j \neq b'_j$, then $\langle \psi_b | \psi_{b'} \rangle = 0$ as desired. The states $|\psi_b\rangle$ we get from procedure 6.1 are normalized, so we have an orthonormal basis.

The last step is to prove that U acts by conjugation on the Pauli group according to M. Let us compute $UZ_iU^\dagger$ and $UX_iU^\dagger$:

$$UZ_iU^\dagger|\psi_b\rangle = UZ_i|b\rangle \tag{6.41}$$
$$= (-1)^{b_i}|\psi_b\rangle. \tag{6.42}$$

Now,

$$\overline{Z}_i|\psi_b\rangle = \overline{Z}_i\overline{X}(b)|\psi_0\rangle \tag{6.43}$$
$$= (-1)^{b_i}\overline{X}(b)\overline{Z}_i|\psi_0\rangle \tag{6.44}$$
$$= (-1)^{b_i}|\psi_b\rangle. \tag{6.45}$$

Since the $|\psi_b\rangle$ form a basis, $UZ_iU^\dagger = \overline{Z}_i$.

Similarly,

$$UX_iU^\dagger|\psi_b\rangle = UX_i|b\rangle \tag{6.46}$$
$$= |\psi_{b'}\rangle, \tag{6.47}$$

where b' is b with the ith bit flipped ($b'_i = b_i + 1$, $b'_j = b_j$ for $j \neq i$).

$$\overline{X}_i|\psi_b\rangle = \overline{X}_i\overline{X}(b)|\psi_0\rangle \tag{6.48}$$
$$= \overline{X}(b')|\psi_0\rangle \tag{6.49}$$
$$= |\psi_{b'}\rangle, \tag{6.50}$$

so $UX_iU^\dagger = \overline{X}_i$. □

If you apply procedure 6.1 to a mapping which does not preserve commutation or anticommutation, you will still get a linear map, but it won't be unitary. Furthermore, it may not realize the conjugation action you wanted, so there is no real reason to apply procedure 6.1 unless the map you are working with satisfies equation (6.31).

6.1.5 The Clifford Group and the Symplectic Group

The conditions on the conjugation map performed by a Clifford group operation become somewhat more straightforward if we consider them in terms of the binary symplectic representation. A group homomorphism of the Paulis becomes a linear map on $2n$-dimensional binary vectors. The requirement that the generators map to independent Paulis just says that the linear map has maximum rank. The constraint that conjugation preserves the commutation relations just means that the linear map must preserve the symplectic inner product:

$$v \odot w = (Mv) \odot (Mw) \tag{6.51}$$
$$v^T J w = v^T M^T J M w, \tag{6.52}$$

with J the $2n \times 2n$ matrix

$$J = \left(\begin{array}{c|c} 0 & I \\ \hline I & 0 \end{array}\right). \tag{6.53}$$

Running over all values of v and w, we find

$$J = M^T J M. \tag{6.54}$$

M is a member of the symplectic group $\mathsf{Sp}(2n, \mathbb{Z}_2)$.

As usual, by going to the binary symplectic representation, we lose all information about the phases of Paulis in the stabilizer. An operation which only changes the phases of Paulis is always performed by conjugation by a Pauli:

Proposition 6.2. *If $UPU^\dagger = \pm P$ for all $P \in \mathsf{P}_n$, then $U = e^{i\theta} Q$ for $Q \in \mathsf{P}_n$ and some value of θ.*

Proof. I will show that any mapping $P \mapsto (-1)^{c_P} P$ which could possibly be performed by a unitary can also be performed by conjugation by some $Q \in \mathsf{P}_n$. Then the proposition follows from theorem 6.1.

For any unitary U, conjugation performs a group homomorphism, so

$$c_{PQ} = c_P + c_Q. \tag{6.55}$$

In particular, the mapping is completely determined by its action on X_i and Z_i for $i = 1, \ldots, n$. For any c_P consistent with equation (6.55), I claim there exists $Q \in \mathsf{P}_n$ that implements it. By equation (6.5), we need to find Q such that $c(Q, X_i) = c_{X_i}$ and $c(Q, Z_i) = c_{Z_i}$. By lemma 3.15, such a Q exists, though there is only one. Specifically,

$$Q = \bigotimes_{i=1}^{n} Q_i \tag{6.56}$$

with

$$Q_i = \begin{cases} I & \text{if } c_{X_i} = c_{Z_i} = 0 \\ X & \text{if } c_{X_i} = 0 \text{ and } c_{Z_i} = 1 \\ Y & \text{if } c_{X_i} = 1 \text{ and } c_{Z_i} = 1 \\ Z & \text{if } c_{X_i} = 1 \text{ and } c_{Z_i} = 0 \end{cases} \tag{6.57}$$

□

It follows from proposition 6.2 that elements of $\check{\mathsf{C}}_n = \hat{\mathsf{C}}_n / \hat{\mathsf{P}}_n$ correspond uniquely to symplectic operations. Indeed, $\check{\mathsf{C}}_n$ is *exactly* the symplectic group.

Theorem 6.3. $\check{\mathsf{C}}_n \cong \mathsf{Sp}(2n, \mathbb{Z}_2)$. *Equivalently, for every map $X_i \mapsto \overline{X}_i$, $Z_i \mapsto \overline{Z}_i$ with $\overline{X}_i, \overline{Z}_i \in \mathsf{P}_n$ and satisfying the correct commutation relations, there exists $U \in \mathsf{C}_n$ which performs that map under conjugation, and U is unique up to overall phase.*

Proof. Most of the pieces of this theorem are derived from theorem 6.1, equation (6.54), and proposition 6.2. The only remaining piece needed to complete the characterization is to show that every symplectic operation can be realized by a Clifford group operation. This is straightforward: Given any linear map from $\hat{\mathsf{P}}_n$ to $\hat{\mathsf{P}}_n$, we can lift it to a group homomorphism $M : \mathsf{P}_n \to \mathsf{P}_n$ by choosing arbitrary signs for the images of X_i and Z_i. When the original linear map is symplectic, M satisfies equation (6.31), so using procedure 6.1, we find a unitary U realizing M and thus realizing the symplectic map on the binary symplectic representation. U maps Paulis to Paulis under conjugation, so $U \in \mathsf{C}_n$. □

6.2 Classical Simulation of the Clifford Group

One of the most interesting and useful things about the Clifford group is also one of the most disappointing. If $U \in \mathsf{C}_n$, we can specify it by giving a global phase plus the images of $2n$ Paulis $X_1, Z_1, \ldots, X_n, Z_n$. Each image is also an n-qubit Pauli, so requires at most $2n + 1$ bits to specify. Thus, a Clifford group element can be specified using only $(2n + 1)(2n)$ bits plus a global phase. Contrast that with a general unitary $U \in \mathsf{U}(2^n)$, which needs 2^{2n} real parameters to specify. Clifford group elements have a much more succinct representation than general unitary gates.

Indeed, circuits made out of Clifford group gates have a much stronger property: they can be efficiently simulated on a classical computer. This is a very useful fact, since it makes working with even relatively large Clifford group circuits tractable. For instance, stabilizer codes are exactly those QECCs which can be

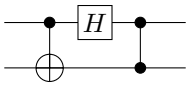


Figure 6.1: An example 2-qubit circuit made of Clifford group gates.

encoded using a circuit composed of Clifford group gates. The ability to easily describe a stabilizer code via its stabilizer is one aspect of the efficient simulatability of the Clifford group gates which form its encoder. We'll also see in part II that this property plays a big role in helping us find fault-tolerant implementations of Clifford group gates.

However, there is a price to be paid. Because Clifford group gates can classically be simulated, it means that a circuit composed only of Clifford group gates cannot access the full power of quantum computation. (Presumably; as with most statements about computational power, this one relies on some unproven complexity-theoretic assumptions, in this case the assumption that quantum computers are computationally more powerful than classical computers.) When we get to fault tolerance, that means that we'll need to venture outside the Clifford group in order to get a universal set of fault-tolerant gates.

6.2.1 Simulation of a Unitary Circuit of Clifford Group Gates

If we have a unitary circuit consisting only of Clifford group gates, the simulation procedure is quite straightforward. Note that if $U, V \in \mathsf{C}_n$, and $U : P \mapsto Q$, $V : Q \mapsto R$, then $(VU) : P \mapsto R$. That's really all there is to the simulation.

Procedure 6.2. You are given a circuit consisting of a product $U = \prod_{i=m}^{1} U_i$, with $U_i \in \mathsf{C}_n$. (I.e., U_1 is the first gate performed, and U_m is the last gate to be performed.) Each U_i can be specified by its action on the generators of the Pauli group, $U_i : X_j \mapsto U_i(X_j)$, $U_i : Z_j \mapsto U_i(Z_j)$. Then the action of the overall circuit U on the Pauli group can be determined as follows:

1. Initialize $\overline{X}_j = X_j$ and $\overline{Z}_j = Z_j$.

2. Starting with $i = 1$, and stepping through i up to the last gate m, repeat the following steps:

   (a) Calculate $U_i(\overline{X}_j)$. This can be done by writing $\overline{X}_j$ as a product of single-qubit Xs and Zs and applying equation (6.12).

   (b) Let the new value of $\overline{X}_j$ be $U_i(\overline{X}_j)$.

   (c) Similarly, replace $\overline{Z}_j$ by $U_i(\overline{Z}_j)$.

3. The overall circuit U has the action $X_j \mapsto \overline{X}_j$, $Z_j \mapsto \overline{Z}_j$, using the final values of $\overline{X}_j$ and $\overline{Z}_j$.

When the circuit consists of only two-qubit gates, updating an $\overline{X}$ or $\overline{Z}$ operator only requires updating two qubits in the decomposition, since all qubits not affected by the gate retain the same Pauli values. Therefore, each gate can be updated in a total time $O(n)$ (since we need to update $2n$ $\overline{X}$ and $\overline{Z}$ operators). The simulation of the full circuit thus takes a time $O(nm)$.

As an example, consider the simple circuit given in figure 6.1. Following the above procedure, we find that this circuit has the following action on Paulis:

$$\begin{array}{lccccccc} \overline{X}_1: & X\otimes I & \to & X\otimes X & \to & Z\otimes X & \to & I\otimes X \\ \overline{Z}_1: & Z\otimes I & \to & Z\otimes I & \to & X\otimes I & \to & X\otimes Z \\ \overline{X}_2: & I\otimes X & \to & I\otimes X & \to & I\otimes X & \to & Z\otimes X \\ \overline{Z}_2: & I\otimes Z & \to & Z\otimes Z & \to & X\otimes Z & \to & X\otimes I. \end{array} \tag{6.58}$$

6.2.2 Simulation of a Unitary Circuit on a Stabilizer Subspace

Another interesting case is when we have some constraints on the input state of the circuit. For instance, the circuit may involve some ancillas, or it may be intended to be performed on a state encoded in a QECC. It may even be that the full input state of the circuit is specified, and we wish to determine the exact output state of the circuit. When the input state is a stabilizer state or lies within a stabilizer code, there still exists an efficient simulation procedure.

If the initial state is completely specified as a stabilizer state, we can keep track of the behavior of the generators $M_1, \ldots, M_n$ of the stabilizer. As before, we step through gates, updating the stabilizer generators at each step just as we updated $\overline{X}$ and $\overline{Z}$ before. When M_j is a generator of the stabilizer before we perform gate U_i, $U_i(M_j)$ is a generator of the stabilizer after the gate, so this procedure allows us to learn the stabilizer of the output state of the circuit. The are only two small differences from the case without a constraint. First of all, we need keep track of only n generators instead of $2n$ Paulis. The other difference is that the generators of the stabilizer are not unique, so after performing any gate, if it is convenient to do so, we may choose a new set of generators of the current stabilizer. There is no requirement to do so; only do it if it is clearly going to simplify the computation.

When the input state is only partially specified by a stabilizer code, we keep track both of the stabilizer, which now has $n-k$ generators, and of $2k$ logical $\overline{X}$ and $\overline{Z}$ operators. The case with $k = n$ is the "no constraint" case of the previous subsection, and the case with $k = 0$ is the stabilizer state case just discussed. Again, the stabilizer generators are non-unique, so at any step, we may choose a new set of generators. The logical $\overline{X}$ and $\overline{Z}$ operators are also now non-unique, so at any step we may multiply them by elements of the stabilizer, which means we are choosing a different coset representative.

Procedure 6.3. You are given a circuit consisting of a product $\prod_{i=m}^{1} U_i$, with $U_i \in \mathsf{C}_n$, which is to be performed on an arbitrary input state from stabilizer code S. S encodes k qubits ($0 \leq k \leq n$), has generators $M_1, \ldots, M_{n-k}$, and has logical Pauli operators $\overline{X}_j$, $\overline{Z}_j$, $j = 1, \ldots, k$. Each U_i can be specified by its action on the generators of the Pauli group, $U_i : X_j \mapsto U_i(X_j)$, $U_i : Z_j \mapsto U_i(Z_j)$. Then the action of the overall circuit on the stabilizer code can be determined as follows:

1. Initialize variables $N_j = M_j$ ($j = 1, \ldots, n-k$), $\overline{X}'_j = \overline{X}_j$ and $\overline{Z}'_j = \overline{Z}_j$ ($j = 1, \ldots, k$) to be the values given by the stabilizer code input.

2. Starting with $i = 1$, and stepping through i up to the last gate m, repeat the following steps:

   (a) Calculate $U_i(\overline{X}'_j)$ for $j = 1, \ldots, k$. This can be done by writing $\overline{X}'_j$ as a product of single-qubit Xs and Zs and applying equation (6.12).

   (b) Let the new value of $\overline{X}'_j$ be $U_i(\overline{X}'_j)$ for $j = 1, \ldots, k$.

   (c) Similarly, replace $\overline{Z}'_j$ by $U_i(\overline{Z}'_j)$ for $j = 1, \ldots, k$.

   (d) Replace N_j by $U_i(N_j)$ for $j = 1, \ldots, n-k$.

   (e) The current stabilizer T is generated by $\langle N_1, \ldots, N_{n-k} \rangle$

   (f) If desired, choose a new set of generators for T.

   (g) If desired, rewrite $\overline{X}'_j = N\overline{X}'_j$ or $\overline{Z}'_j = N\overline{Z}'_j$ with $N \in \mathsf{T}$.

3. The output state lies in a stabilizer code with generators equal to the final values of N_j, $j = 1, \ldots, n-k$. The encoded state has undergone the Clifford group operation given by the transformation $\overline{X}_j \mapsto \overline{X}'_j$, $\overline{Z}_j \mapsto \overline{Z}'_j$.

An important special case is when some qubits are completely specified (e.g., to be $|0\rangle$), and the remaining qubits are completely unconstrained. In that case, the stabilizer is the stabilizer of the ancilla qubits, and the initial logical operators $\overline{X}$ and $\overline{Z}$ are the Paulis on the unconstrained input qubits.

As a example of the expanded procedure, let us consider the circuit in figure 6.1 again, but this time with the second qubit initially in the state $|0\rangle$. We now begin with $M_1 = I \otimes Z$, $\overline{X}_1 = X \otimes I$, and $\overline{Z}_1 = Z \otimes I$. Using the same analysis as before, we find that the final value of the stabilizer and logical operators are:

$$N_1 = X \otimes I \tag{6.59}$$

$$\overline{X}_1' = I \otimes X \tag{6.60}$$

$$\overline{Z}_1' = X \otimes Z. \tag{6.61}$$

We can choose a new coset representative for $\overline{Z}_1'$: $I \otimes Z = (X \otimes I)(X \otimes Z)$. Then we can see that the overall transformation performed is to move the input qubit from the first qubit to the second qubit. In the output state, the first qubit is fixed to be $|0\rangle + |1\rangle$.

6.2.3 Measurement of Paulis

The final step is to add measurements to our simulation. I will phrase this in the most general way, where we measure the eigenvalue of an arbitrary Pauli operator on n qubits, but it would be equivalent to just consider measurement of single qubits in the standard basis. This is because measurement of any Pauli can be implemented via single-qubit measurements plus Clifford group operations. For simplicity, we restrict attention to Paulis with overall phase ± 1, so the eigenvalues are ± 1.

The analysis of measurements is somewhat more complicated than the analysis of unitary gates. For one thing, there are three separate cases to consider. The first case is when the Pauli P we wish to measure is in the current stabilizer T up to a phase: $\pm P \in \mathsf{T}$. In that case, the measurement outcome is just ± 1 ($+1$ if $P \in \mathsf{T}$ and -1 if $-P \in \mathsf{T}$), and the state does not change, since the state being measured is an eigenstate of P.

The second case is when $P \in \mathsf{N}(\mathsf{T})$. Now, measurement of P constitutes a measurement of some of the encoded data. We must rewrite P as a product of the logical Paulis $\overline{X}$ and $\overline{Z}$, which determines that P corresponds to some logical Pauli $\overline{Q}$. The measurement output distribution of P is the same as the measurement output distribution of Q on the original input qubit. We must also update the encoded state for the effects of the measurement. Finally, the measurement has outcome ± 1, so we know that the post-measurement state is a $+1$ eigenstate of $\pm P$. In other words, $\pm P$ has joined the stabilizer.

The third case, when $P \notin \mathsf{N}(\mathsf{T})$, is the most complicated. Since $P \notin \mathsf{N}(\mathsf{T})$, $\exists M \in \mathsf{T}$ s.t. $\{P, M\} = 0$. Suppose $|\psi\rangle \in \mathcal{T}(\mathsf{T})$, so $M|\psi\rangle = |\psi\rangle$. The expectation value of a measurement of P on $|\psi\rangle$ is

$$\langle\psi|P|\psi\rangle = \langle\psi|PM|\psi\rangle \tag{6.62}$$

$$= \langle\psi|M(-P)|\psi\rangle \tag{6.63}$$

$$= -\langle\psi|P|\psi\rangle \tag{6.64}$$

$$= 0. \tag{6.65}$$

(since $M^2 = I$, meaning $M = M^\dagger$). Thus, the probability of a $+1$ outcome and a -1 outcome for the measurement are both $1/2$.

We can also calculate the residual state. If the measurement has outcome $(-1)^b$, the state afterwards is $|\psi'\rangle = (1/\sqrt{2})(I + (-1)^b P)|\psi\rangle$ (taking into account normalization). If $N \in \mathsf{T}$ commutes with P, then $|\psi'\rangle$ is still a $+1$ eigenstate of N:

$$N|\psi'\rangle = \frac{1}{\sqrt{2}} N(I + (-1)^b P)|\psi\rangle \tag{6.66}$$

$$= \frac{1}{\sqrt{2}}(I + (-1)^b P)N|\psi\rangle \tag{6.67}$$

$$= \frac{1}{\sqrt{2}}(I + (-1)^b P)|\psi\rangle = |\psi'\rangle. \tag{6.68}$$

$|\psi'\rangle$ is also a $+1$ eigenstate of $(-1)^b P$. That means that it is *not* an eigenstate of any M with $\{M, P\} = 0$, even if $M \in \mathsf{T}$. Define a stabilizer T' generated by $(-1)^b P$ plus all $N \in \mathsf{T}$ s.t. $[N, P] = 0$. The stabilizer of the remaining state after the measurement certainly contains T'. As we will see in a moment, that is all it contains.

How big is T'? To answer this, we need to know how many elements of T commute with P.

Proposition 6.4. *Let S be a stabilizer, and let $P \notin \mathsf{N}(\mathsf{S})$ be a Pauli that does not commute with every element of S. Then exactly half of the elements of S commute with P. Furthermore, for any coset $\overline{Q} \in \mathsf{N}(\mathsf{S})/\mathsf{S}$, exactly half the elements of $\overline{Q}$ commute with P.*

Proof. If $P \notin \mathsf{N}(\mathsf{S})$, let $M \in \mathsf{S}$ be an element of the stabilizer that anticommutes with P, $\{M, P\} = 0$. Then we can pair all elements of S:

$$N \leftrightarrow MN. \tag{6.69}$$

Note that $MN^2 = M$, so each element of S is a member of only one pair. The reason for doing this pairing is that N commutes with P iff MN anticommutes with P:

$$c(MN, P) = c(M, P) + c(N, P) = 1 + c(N, P). \tag{6.70}$$

Thus, the number of elements of S that commute with P is equal to the number of elements that anticommute with P.

Similarly, we can pair elements of the cosets representing logical Paulis, $N \leftrightarrow MN$, using the same $M \in \mathsf{S}$. Both N and MN are in the same coset $\overline{Q}$, and again, exactly one of them commutes with P and one anticommutes with P. Thus, half of the coset $\overline{Q}$ commutes with P. □

Corollary 6.5. $|\mathsf{T}'| = |\mathsf{T}|$.

The subspace of post-measurement states for a given measurement outcome can be no larger than the space of possible pre-measurement states, but it could potentially be smaller if multiple states collapse in the measurement to the same final state. However, as another corollary of proposition 6.4, we find that this cannot happen. If the code space of T contains k logical qubits, one possible basis for the code space is the set of 2^k codewords which are eigenstates of $\overline{Z}_1, \ldots, \overline{Z}_k$. Each of these codewords is a stabilizer state with stabilizer generated by $\mathsf{T}, \pm\overline{Z}_1, \ldots, \pm\overline{Z}_k$. By proposition 6.4 and the analysis preceding it, the stabilizer of the post-measurement state is generated by T' and by $\pm\overline{Z}_1, \ldots, \pm\overline{Z}_k$, with the same eigenvalues. However, we must make certain that each coset is represented by an element that commutes with P; such an element always exists by proposition 6.4. Since each of the 2^k basis states gets mapped to a distinct stabilizer state, the code space after the measurement also has dimension 2^k. This implies that T' is the actual stabilizer post-measurement, as claimed.

Furthermore, this logic also tells us that the T-coset of the logical $\overline{Z}_i$ operator gets replaced by the T'-coset which contains an element of the original $\overline{Z}_i$ that commutes with P. The same reasoning tells us how to find the new versions of the other logical Paulis. In short, we now know how to update both the stabilizer and the logical Paulis after measurement of a Pauli operator.

Theorem 6.6. *Suppose our system is a codeword of the stabilizer code T, with generators $N_1, \ldots, N_{n-k}$ and logical Paulis $\overline{X}_i$ and $\overline{Z}_i$ ($i = 1, \ldots, k$), and we measure the eigenvalue of Pauli $P \notin \mathsf{N}(\mathsf{T})$. Then, conditioned on outcome $(-1)^b$, the stabilizer and logical Paulis of the system after the measurement can be determined as follows:*

1. *Find generator $M \in \mathsf{T}$ such that $\{M, P\} = 0$. If necessary, reorder the generators so that $M = N_1$.*
2. *For each generator N_i, $i > 1$, if $[N_i, P] = 0$, let $N_i' = N_i$. If $\{N_i, P\} = 0$, let $N_i' = N_i M$.*
3. *Let $N_1' = (-1)^b P$.*
4. *The new stabilizer T' is generated by $N_1', \ldots, N_{n-k}'$.*

5. *Pick a representative $\overline{Z}_i$ for each of the logical Z operators for T. If $[\overline{Z}_i, P] = 0$, let $\overline{Z}'_i = \overline{Z}_i$; otherwise let $\overline{Z}'_i = \overline{Z}_i M$. $\overline{Z}'_i$ is a representative for the new logical Z_i operator.*

6. *Similarly, pick a representative $\overline{X}_i$ for each of the logical X operators for T. If $[\overline{X}_i, P] = 0$, let $\overline{X}'_i = \overline{X}_i$; otherwise let $\overline{X}'_i = \overline{X}_i M$. $\overline{X}'_i$ is a representative for the new logical X_i operator.*

One interesting twist on this system is that we can essentially control the measurement outcome. In particular, suppose we measure $P \notin \mathsf{N}(\mathsf{T})$ and get outcome -1. In theorem 6.6, we identify an element M from the old stabilizer T which anticommutes with P. However, M commutes with generators 2 through $n-k$ of T', since they are also elements of T. M also commutes with the standard coset representatives for $\overline{X}'_i$ and $\overline{Z}'_i$. Thus, if we perform M on the post-measurement state, we transform the stabilizer (according to procedure 6.3) only by changing $N'_1 = -P$ to $+P$. The logical operators are unchanged. This is exactly the same state as we would have gotten (according to theorem 6.6) if the original measurement outcome had been $+1$.

Putting everything together, we get the following procedure for simulating Clifford group gates and Pauli measurements on either a fixed initial stabilizer state or a partially specified state with some free (logical) qubits:

Procedure 6.4. You are given a circuit consisting of a m-element sequence of unitary Clifford group gates and Pauli measurements. At step number i, the circuit calls for either Clifford group gate $U_i \in \mathsf{C}_n$ (specified by its action on Paulis $U_i : Q \mapsto U_i(Q)$) or measurement of the eigenvalue of $P_i \in \mathsf{P}_n$ (assuming P_i has eigenvalues ± 1, not $\pm i$). The initial state of the circuit is given by stabilizer S, which encodes k qubits ($0 \leq k \leq n$), has generators $M_1, \ldots, M_{n-k}$, and has logical Pauli operators $\overline{X}_j$, $\overline{Z}_j$, $j = 1, \ldots, k$.

1. Initialize variables $k' = k$, $N_j = M_j$ ($j = 1, \ldots, n-k$), $\overline{X}'_j = \overline{X}_j$ and $\overline{Z}'_j = \overline{Z}_j$ ($j = 1, \ldots, k$) to be the values given by the stabilizer code input. Let the variable state $|\overline{\psi}\rangle$ be the initial encoded state of the system, corresponding to the logical state $|\psi\rangle$.

2. Perform the following procedure for each step i in order, for $i = 1, \ldots, m$:

   (a) If at step i, the circuit calls for Clifford gate U_i:

      i. Calculate $U_i(\overline{X}'_j)$ for $j = 1, \ldots, k'$. This can be done by writing $\overline{X}'_j$ as a product of single-qubit Xs and Zs and applying equation (6.12).

      ii. Let the new value of $\overline{X}'_j$ be $U_i(\overline{X}'_j)$ for $j = 1, \ldots, k'$.

      iii. Similarly, replace $\overline{Z}'_j$ by $U_i(\overline{Z}'_j)$ for $j = 1, \ldots, k'$.

      iv. Replace N_j by $U_i(N_j)$ for $j = 1, \ldots, n-k'$.

      v. The current stabilizer T is generated by $\langle N_1, \ldots, N_{n-k'} \rangle$

      vi. If desired, choose a new set of generators for T.

      vii. If desired, rewrite $\overline{X}'_j = N\overline{X}'_j$ or $\overline{Z}'_j = N\overline{Z}'_j$ with $N \in \mathsf{T}$.

   (b) If at step i, the circuit calls for measurement of P_i, determine whether $\pm P_i$ is in T, if $P_i \in \mathsf{N}(\mathsf{T}) \setminus \mathsf{T}$, or if P_i is not in $\mathsf{N}(\mathsf{T})$, as follows:

      i. Determine if P_i commutes with all generators N_j of the current stabilizer T. If not, then $P_i \notin \mathsf{N}(\mathsf{T})$.

      ii. If P_i does commute with all generators, determine if $\pm P_i \in \mathsf{T}$. This can be done by writing P_i and the generators N_j in their binary symplectic representations and seeing if v_{P_i} is in the linear span of the v_{N_j}.

      iii. If P_i commutes with all generators but is not in T, then $P_i \in \mathsf{N}(\mathsf{T}) \setminus \mathsf{T}$.

   (c) If at step i, the circuit calls for measurement of P_i with $\pm P_i \in \mathsf{T}$:

      i. Evaluate whether $+P_i \in \mathsf{T}$ or $-P_i \in \mathsf{T}$. This can be done using linear algebra and the binary symplectic representations of P_i and N_j to write $P_i = \pm \prod_{j=1}^{n-k'} N_j^{s_j}$.

ii. If $+P_i \in \mathsf{T}$, return measurement result $+1$. If $-P_i \in \mathsf{T}$, return measurement result -1.

(d) If at step i, the circuit calls for measurement of P_i with $P_i \in \mathsf{N}(\mathsf{T}) \setminus \mathsf{T}$:

i. Write $P_i = \prod_{j=1}^{n-k'} N_j^{s_j} \prod_{j'=1}^{k} \overline{X}_j'^{t_j} \overline{Z}_j'^{u_j}$, with $s_j, t_j, u_j \in \{0,1\}$. This can be done using linear algebra and the binary symplectic representations of the Paulis.

ii. Let $Q = \prod_{j'=1}^{k} X_j^{t_j} Z_j^{u_j}$.

iii. Perform a measurement of Q on the current logical state $|\psi\rangle$. Return the measurement result $(-1)^b$, and update $|\overline{\psi}\rangle$ accordingly. Note that this step may not be efficient, depending on the current state $|\psi\rangle$.

iv. Reduce k' by 1, and add $(-1)^b P_i$ to the stabilizer as a new generator $N_{n-k'+1}$. Update T accordingly, and if desired, choose a new set of generators for T.

v. Update the logical operators $\overline{X}_j'$ and $\overline{Z}_j'$ to relate to the new $|\overline{\psi}\rangle$. Again, this step may not be efficient.

(e) If at step i, the circuit calls for measurement of P_i with $P_i \notin \mathsf{N}(\mathsf{T})$:

i. Choose a uniformly random bit b and return measurement result $(-1)^b$.

ii. Find generator $M \in \mathsf{T}$ such that $\{M, P_i\} = 0$. If necessary, reorder the generators so that $M = N_1$.

iii. For each generator N_j, $j > 1$, if $[N_j, P_i] = 0$, leave N_j unchanged. If $\{N_j, P_0\} = 0$, replace N_j by $N_j M$.

iv. Replace N_1 by $(-1)^b P_i$.

v. Update the stabilizer T with the revised generators. If desired, choose new generators for the revised T.

vi. Pick a representative $\overline{Z}_j'$ for each of the logical Z operators for T. If $[\overline{Z}_j', P_i] = 0$, leave $\overline{Z}_j'$ unchanged; otherwise replace $\overline{Z}_j'$ by $\overline{Z}_j' M$. The updated $\overline{Z}_j'$ is a representative for the new logical Z_j operator; choose a new representative using the updated T if desired.

vii. Similarly, pick a representative $\overline{X}_j'$ for each of the logical X operators for T. If $[\overline{X}_j', P] = 0$, leave $\overline{X}_i'$ unchanged; otherwise replace $\overline{X}_j'$ by $\overline{X}_j' M$. The updated $\overline{X}_j'$ is a representative for the new logical X_j operator; choose a new representative using the updated T if desired.

3. The output state lies in a stabilizer code with generators equal to the final values of N_j, $j = 1, \ldots, n-k'$. The encoded state has had $k - k'$ logical qubits measured, and has undergone the transformation $\overline{X}_j \mapsto \overline{X}_j'$, $\overline{Z}_j \mapsto \overline{Z}_j'$ on the remaining qubits.

Generally, for this sort of simulation, we only consider circuits where measurement of $P_i \in \mathsf{N}(\mathsf{T}) \setminus \mathsf{T}$ does not occur, so that the simulation is an efficient one. For instance, if the circuit contains no measurements, this is not an issue, or equally if it contains no logical qubits (so $\mathsf{N}(\mathsf{T}) = \mathsf{T}$). In part II, we will see circuits that have measurements and some logical qubits, but they have been specially designed to avoid the troublesome case.

Theorem 6.7 (Gottesman-Knill)**.** *Given a circuit starting from an initial stabilizer state, followed by a sequence of Clifford group operations and Pauli measurements, which may depend on classical computations performed on previous measurement results, there is an efficient classical simulation of the circuit.*

If you follow procedure 6.4, you will find a time of $O(n^3 m)$ is needed to do the simulation, taking into account all the linear algebra manipulations needed to process measurement. However, by keeping track of slightly more information, this can be reduced to $O(n^2 m)$.

To illustrate the measurement procedure, let us consider a variation of the example circuit from figure 6.1 which has some measurements in it. The revised circuit is given in figure 6.2. The first few steps are as before. After the Hadamard, we have the following stabilizer and logical Paulis:

$$\begin{array}{lccc} N_1: & I \otimes Z & \rightarrow & X \otimes Z \\ \overline{X}: & X \otimes I & \rightarrow & Z \otimes X \\ \overline{Z}: & Z \otimes I & \rightarrow & X \otimes I. \end{array} \tag{6.71}$$

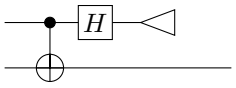


Figure 6.2: One-qubit teleportation: an example of a 2-qubit circuit made of Clifford group gates and measurement.

Then we measure $P = Z \otimes I$. In this case, there is only one element of the stabilizer, and it anticommutes with P. Suppose we get measurement outcome $+1$. By theorem 6.6, replace N_1 with P. $\overline{X}$ commutes with P, so its representative need not change. However, we can choose a new coset representative to take advantage of the new stabilizer: $(Z \otimes X)(Z \otimes I) = I \otimes X$. $\overline{Z}$ anticommutes with P, so we *must* choose a new coset representative

$$\overline{Z}N_1 = (X \otimes I)(X \otimes Z) = I \otimes Z. \tag{6.72}$$

After the measurement, the stabilizer and logical Paulis are thus

$$N_1 : Z \otimes I \tag{6.73}$$
$$\overline{X} : I \otimes X \tag{6.74}$$
$$\overline{Z} : I \otimes Z \tag{6.75}$$

The input qubit has been moved to the second qubit for the output, and the output value of the first qubit is $|0\rangle$.

In the case where the measurement result is -1, the stabilizer is $-P$ instead of P. The new representative for $\overline{X}$ also acquires this minus sign, so the final state is as follows:

$$N_1 : -Z \otimes I \tag{6.76}$$
$$\overline{X} : -I \otimes X \tag{6.77}$$
$$\overline{Z} : I \otimes Z \tag{6.78}$$

The output state of the first qubit is $|1\rangle$, and the output state of the second qubit is the input data qubit, but with a phase flip (Z) performed on it.

6.3 Generators of the Clifford Group

In section 6.1.3, we saw three common gates which are also elements of the Clifford group: H, $R_{\pi/4}$ and CNOT. In a sense, they are the *only* elements of the Clifford group, because they generate the Clifford group. That is:

Theorem 6.8. *Any gate in the n-qubit Clifford group C_n can be written as a product of $e^{i\theta}I$, H_i, $R_{\pi/4,i}$, and $\mathrm{CNOT}_{i,j}$, with $i, j = 1, \ldots, n$ and $\theta \in [0, 2\pi)$.*

$\mathrm{CNOT}_{i,j}$ means CNOT with qubit i as the control and qubit j as the target.

Proof. First, note that the Pauli group is inside the group generated by H and $R_{\pi/4}$: $Z = (R_{\pi/4})^2$, and $X = HZH$. Second, global phases are covered by the gates $e^{i\theta}I$. To finish the proof, we thus need only to prove that the symplectic representations of H, $R_{\pi/4}$ and CNOT generate $\check{\mathsf{C}}_n$.

It will be helpful to also use two additional gates: SWAP and $\mathrm{C}-\mathrm{Z}$. Both are in the Clifford group, and can easily be performed as a product of H, $R_{\pi/4}$, and CNOT, so we can use them freely without explicitly adding them to the generating set. In particular, SWAP is the product of 3 CNOT gates with alternating directions, and $\mathrm{C}-\mathrm{Z} = (I \otimes H)\mathrm{CNOT}(I \otimes H)$.

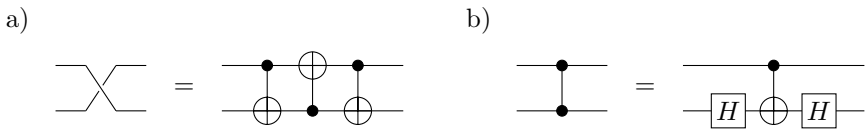


Figure 6.3: a) Clifford group circuit for the SWAP gate. b) Clifford group circuit for the $\mathrm{C}-\mathrm{Z}$ gate.

We can take advantage of theorem 6.3. The symplectic representation of H is

$$\left(\begin{array}{c|c} 0 & 1 \\ \hline 1 & 0 \end{array}\right), \tag{6.79}$$

the symplectic representation of $R_{\pi/4}$ is

$$\left(\begin{array}{c|c} 1 & 0 \\ \hline 1 & 1 \end{array}\right), \tag{6.80}$$

and the symplectic representation of CNOT is

$$\left(\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ \hline 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{array}\right). \tag{6.81}$$

Additional qubits will modify these matrices by adding a direct sum with the identity matrix. For instance, when we have 3 qubits, $\mathrm{CNOT}_{2,3}$ is

$$\left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right). \tag{6.82}$$

In general, if we have a symplectic matrix

$$U = \left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right), \tag{6.83}$$

representing the Clifford group gate U, then left multiplication by another symplectic matrix representing the gate V gives us the matrix for VU whereas right multiplication gives us UV. The effects of left and right multiplication by H, $R_{\pi/4}$, CNOT, $\mathrm{C}-\mathrm{Z}$, and SWAP are summarized in table 6.1. As you can see, left multiplication gives us a set of row operations and right multiplication gives us a set of column operations. We can't exactly do standard Gaussian elimination, but we can do something very similar. In particular, given an arbitrary symplectic matrix U, we will find a sequence of H, $R_{\pi/4}$, and CNOT to multiply by on the left and right to transform U into the identity:

$$\left(\prod_{k=1}^{g_L} V_{L,k}\right) U \left(\prod_{k=1}^{g_R} V_{R,k}\right) = I, \tag{6.84}$$

with $V_{L,k}$ and $V_{R,k}$ drawn from $\{H_i, R_{\pi/4,i}, \mathrm{CNOT}_{i,j}\}$. Then

$$U = \left(\prod_{k=g_L}^{1} V_{L,k}^\dagger\right)\left(\prod_{k=g_R}^{1} V_{R,k}^\dagger\right), \tag{6.85}$$

Gate	Left multiplication	Right multiplication
H_i	Switches the ith rows of $(A\|B)$ and $(C\|D)$	Switches the ith columns of $\left(\frac{A}{C}\right)$ and $\left(\frac{B}{D}\right)$
$R_{\pi/4,i}$	Adds the ith row of $(A\|B)$ to the ith row of $(C\|D)$	Adds the ith column of $\left(\frac{B}{D}\right)$ to the ith column of $\left(\frac{A}{C}\right)$
$\mathrm{CNOT}_{i,j}$	Adds the ith row of $(A\|B)$ to the jth row of $(A\|B)$, and adds the jth row of $(C\|D)$ to the ith row of $(C\|D)$	Adds the jth column of $\left(\frac{A}{C}\right)$ to the ith column of $\left(\frac{A}{C}\right)$, and adds the ith column of $\left(\frac{B}{D}\right)$ to the jth column of $\left(\frac{B}{D}\right)$
$\mathrm{C-Z}_{i,j}$	Adds the ith row of $(A\|B)$ to the jth row of $(C\|D)$, and adds the jth row of $(A\|B)$ to the ith row of $(C\|D)$	Adds the jth column of $\left(\frac{B}{D}\right)$ to the ith column of $\left(\frac{A}{C}\right)$, and adds the ith column of $\left(\frac{B}{D}\right)$ to the jth column of $\left(\frac{A}{C}\right)$
$\mathrm{SWAP}_{i,j}$	Swaps the ith rows of A, B, C, and D with the jth rows of A, B, C, and D	Swaps the ith columns of A, B, C, and D with the jth columns of A, B, C, and D

Table 6.1: The effects of left and right multiplication by H, $R_{\pi/4}$, CNOT, $\mathrm{C-Z}$, and SWAP on a symplectic matrix.

proving the theorem.

In equation (6.83), let a_i, b_i, c_i, and d_i be the ith columns of A, B, C, and D, respectively, and a_{ij}, b_{ij}, c_{ij}, and d_{ij} be the i,jth entries of A, B, C, and D. Because U is symplectic, we can apply equation (6.54) to get the following properties:

1. U has full rank.
2. $C^T A + A^T C = 0$; i.e., $(a_i|c_i) \odot (a_j|c_j) = 0$.
3. $C^T B + A^T D = I$; i.e., $(a_i|c_i) \odot (b_j|d_j) = \delta_{ij}$.
4. $D^T A + B^T C = I$, which is the same as the previous property.
5. $D^T B + B^T D = 0$; i.e., $(b_i|d_i) \odot (b_j|d_j) = 0$.

These properties all remain true if we multiply U on the left or right by other symplectic matrices.

To find a sequence of the form equation (6.84), we can use the following steps:

Procedure 6.5.

1. Since U has maximum rank, somewhere in the first column of A or C is a 1. Using left multiplication by H and SWAP, we can put this 1 in the upper left corner.
2. Do column reduction on the first column of A: Use left multiplication by CNOT to add the 1 in the upper left corner of A to any other row of A with a 1 in the first column.
3. Do row reduction on the first row of A: Use right multiplication by CNOT to add the 1 in the upper left corner of A to any other column of A with a 1 in the first row.
4. Using left multiplication by $R_{\pi/4}$ and/or $\mathrm{C-Z}$, we can similarly make the first column of C to be all 0s.
5. At this point, we find that the first row of C has also become all 0s. This must be the case since
$$0 = (a_1|c_1) \odot (a_j|c_j) = a_1 \cdot c_j + c_1 \cdot a_j = c_{1j} + 0. \tag{6.86}$$
6. Repeat steps 1 to 5 on the second row and column of A and C to make them all 0 except for a_{22}, which is 1. Continue with all other rows and columns in sequence, until we have $A = I$ and $C = 0$.
7. At this point it follows that $D = I$:
$$\delta_{ij} = (a_i|c_i) \odot (b_j|d_j) = a_i \cdot d_j + c_i \cdot b_j = d_{ij}. \tag{6.87}$$
Also, B is symmetric:
$$0 = D^T B + B^T D = B + B^T. \tag{6.88}$$
8. Right multiply by H for every qubit, switching A and B, and switching C and D, so now $B = C = I$, $D = 0$, and A is symmetric.
9. Right multiply by $R_{\pi/4}$ as needed to eliminate the diagonal of A. Right multiply by $\mathrm{C-Z}$ to eliminate all other elements of A, leaving $A = 0$. Since $D = 0$, these operations do not change C.
10. Right multiply by H for every qubit, switching the left and right halves back again. We are left with $A = D = I$, $B = C = 0$.

□

It is worth counting just how many gates from the generating set are needed in this procedure. To eliminate all the 1s in a single row and column of A and C takes $O(n)$ gates. Doing this for all n rows and columns thus requires $O(n^2)$ gates. Steps 8 and 10 only require $O(n)$ gates, but step 8 could require one gate for each element of A, which is again $O(n^2)$. Thus, the overall number of gates used in the procedure is $O(n^2)$. It turns out that this can be slightly improved, allowing an arbitrary element of the Clifford group to be written as a product of $O(n^2/\log n)$ generators.

6.4 Encoding Circuits for Stabilizer Codes

Any stabilizer code can be encoded with a Clifford group gate. Therefore, techniques useful for decomposing a Clifford group unitary into a detailed quantum circuit are also useful for deriving encoding circuits for stabilizer codes. The main difference between encoding a stabilizer code and finding a circuit for a Clifford group operation is that there is more freedom in choosing the encoder for a stabilizer code. When you are given a full Clifford group operation, it tells you exactly what unitary you must perform (up to global phase, perhaps), whereas for a stabilizer code, you have an additional freedom to perform unitaries which leave the code space unchanged.

The procedures discussed in this section are boring but necessary. However, it is sometimes possible to come up with clever codes with more exciting encoding circuits. The most general stabilizer code uses $O(n^2)$ gates in its encoding circuit, which is not too bad, but for a big code, we'd really like an encoding circuit with only $O(n)$ gates. Some such codes exist. It's not possible to do better than that for any sensible code, since with $o(n)$ gates, you can't even touch every qubit in the code with a gate. Some qubits will end up either unprotected or unused, maybe even unloved.

The true bottleneck, however, is the *decoding* circuit, and in particular, the syndrome decoding problem. Recall that the error syndromes correspond to cosets of $\mathsf{N}(\mathsf{S})$ in P_n, and that we assign a coset representative Q_s to each error syndrome s to serve as the "correct" way to correct a state with syndrome s. However, in general, it is quite difficult (NP-hard) to compute Q_s as a function of s. Efficient codes for which syndrome decoding can be performed efficiently are precious things, and one of the main points of coding theory is to find them. Even better is an efficient code for which encoding and syndrome decoding can both be done in time $O(n)$. In general, syndrome decoding, efficient or not, will use gates outside of the Clifford group, but is most often done on a classical computer since it is basically a classical problem.

6.4.1 Encoding a Stabilizer Code with Unspecified Logical Paulis

The most straightforward case is when the stabilizer is given, but not the logical Paulis. In this case, there is an additional freedom to perform unitaries within the code space, provided we do not mix codewords with non-codewords. In other words, we can use any set of logical Paulis that is convenient for us.

If the original state, pre-encoding, is $|0\rangle^{\otimes(n-k)} \otimes |\psi\rangle$ (where $|\psi\rangle$ is a k-qubit state), its initial stabilizer is generated by $Z_1, Z_2, \ldots, Z_{n-k}$. The final stabilizer S, post-encoding, also has $n-k$ generators $M_1, M_2, \ldots, M_{n-k}$. The choice of generators is not unique, of course, so we may pick any set of generators for S that we like. Then we must simply find some Clifford group circuit that maps $Z_i \mapsto M_i$ for $i = 1, \ldots, n-k$.

We can do this using a simplified version of procedure 6.5. There are two ways we can simplify: First, we don't really care what happens to Z_i for $i > n-k$ or to X_i. Thus, we need only concern ourselves with the first $n-k$ columns of A and C. Second, the choice of generators of the stabilizer is not unique. Therefore, we may permute columns and add columns to each other within A and C without using any actual gates. We are left with the following procedure:

Procedure 6.6. Generate a list of gates using the steps given below.

1. Write the generators $M_1, \ldots, M_{n-k}$ of S as binary symplectic vectors. Produce the first $n-k$ columns of a symplectic matrix as follows: The ith column of A is $a_i = x_{M_i}$. The ith column of C is $c_i = z_{M_i}$.

2. $(a_1|c_1)$ is a non-zero vector, so there is a 1 somewhere in the first column. Using left multiplication by H and SWAP, we can put this 1 in the upper left corner.

3. Do column reduction on the first column of A: Use left multiplication by CNOT to add the 1 in the upper left corner of A to any other row of A with a 1 in the first column.

4. Using left multiplication by $R_{\pi/4}$ and/or $\mathrm{C-Z}$, we can similarly make the first column of C to be all 0s.

5. Do row reduction on the first row of A: Replace generators of the stabilizer to add the 1 in the upper left corner of A to any other column of A with a 1 in the first row. (It does not matter much if this step is performed before or after the previous step.)

6. At this point, we find that the first row of C has also become all 0s. This must be the case since

$$0 = (a_1|c_1) \odot (a_j|c_j) = a_1 \cdot c_j + c_1 \cdot a_j = c_{1j} + 0. \tag{6.89}$$

7. Repeat the previous steps on the second row and column of A and C to make them all 0 except for a_{22}, which is 1. Continue with all other rows and columns in sequence up to column number $n - k$.

8. Perform H on qubits 1 through $n - k$ to switch A and C.

Take the inverse of this product of gates. The resulting Clifford group circuit will encode in some coset of S; that is, the stabilizer will be almost correct, except that we may have some error syndrome different from the correct trivial syndrome. (Also note that the choice of stabilizer generators may be different from that which was originally given to us.) Calculate the action of the encoding circuit on Z_i for $i = 1, \ldots, n - k$. If for any i, the resulting generator M_i has the wrong sign, add X_i to the beginning of the encoding circuit.

As an example, let us find an encoding circuit for the five-qubit code as given in table 3.2. We begin (step 1) with the matrix

$$\begin{array}{cccc} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{array}. \tag{6.90}$$

Step 2 is unnecessary, as there is already a 1 for a_{11}. We can perform steps 3 and 4 using $\mathrm{CNOT}_{1,4} \cdot \mathrm{C-Z}_{1,2} \cdot \mathrm{C-Z}_{1,3}$. Be careful: you must perform the correct transformations on the full rows, not just the first column. Then, in step 5, we can clear out the first row of A by adding the first column to the third column; this corresponds to replacing the third generator M_3 with $M_1 M_3$. We now have the following matrix:

$$\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{array}. \tag{6.91}$$

We can clear out the second column using $\mathrm{CNOT}_{2,5} \cdot \mathrm{C-Z}_{2,3} \cdot \mathrm{C-Z}_{2,4}$, then reduce the second row of A by replacing M_4 with $M_2 M_4$. The third column requires slightly more care. First column reduce the third column of A using $\mathrm{CNOT}_{3,4}$. Then use $\mathrm{C-Z}_{3,4} \cdot \mathrm{C-Z}_{3,5}$ to eliminate the third column of C. The caution is necessary because $\mathrm{CNOT}_{3,4}$ and $\mathrm{C-Z}_{3,4}$ do not commute; in this case, however, the difference is only Z, which will not affect the binary symplectic representation. Then we finish with the fourth column, using

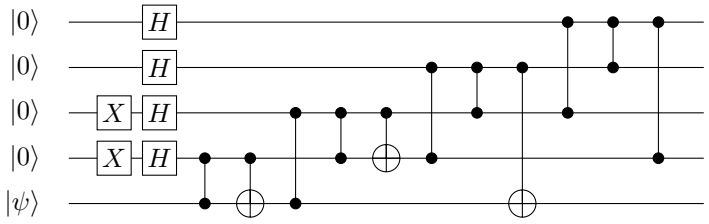


Figure 6.4: Encoding circuit for the five-qubit code derived in the text.

$\mathrm{CNOT}_{4,5}$ followed by $\mathrm{C}-\mathrm{Z}_{4,5}$. We then conclude by performing H on qubits 1 through 4. These gates give us the following sequence of matrices:

$$\begin{array}{cccc}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&1&1\\0&0&0&1\\\hline 0&0&0&0\\0&0&0&0\\0&0&1&1\\0&0&1&1\\0&0&1&1\end{array}\rightarrow\begin{array}{cccc}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\\0&0&0&1\\\hline 0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&1\\0&0&0&1\end{array}\rightarrow\begin{array}{cccc}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\\0&0&0&0\\\hline 0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{array}\rightarrow\begin{array}{cccc}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\\\hline 1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\\0&0&0&0\end{array}\tag{6.92}$$

Taking the inverses of these gates in reverse order, we get the following sequence of gates for the encoding circuit:

$$\begin{aligned}&\mathrm{CNOT}_{1,4}\cdot\mathrm{C}-\mathrm{Z}_{1,2}\cdot\mathrm{C}-\mathrm{Z}_{1,3}\cdot\mathrm{CNOT}_{2,5}\cdot\mathrm{C}-\mathrm{Z}_{2,3}\cdot\mathrm{C}-\mathrm{Z}_{2,4}\cdot\mathrm{CNOT}_{3,4}\\&\cdot\mathrm{C}-\mathrm{Z}_{3,4}\cdot\mathrm{C}-\mathrm{Z}_{3,5}\cdot\mathrm{CNOT}_{4,5}\cdot\mathrm{C}-\mathrm{Z}_{4,5}\cdot H_1\cdot H_2\cdot H_3\cdot H_4\end{aligned}\tag{6.93}$$

Now calculate the effect of these gates on $Z_1,\ldots,Z_4$:

$$Z_1\rightarrow X\otimes Z\otimes Z\otimes X\otimes I\tag{6.94}$$
$$Z_2\rightarrow I\otimes X\otimes Z\otimes Z\otimes X\tag{6.95}$$
$$Z_3\rightarrow -I\otimes Z\otimes Y\otimes Y\otimes Z\tag{6.96}$$
$$Z_4\rightarrow -Z\otimes I\otimes Z\otimes Y\otimes Y.\tag{6.97}$$

Since

$$I\otimes Z\otimes Y\otimes Y\otimes Z=+(X\otimes Z\otimes Z\otimes X\otimes I)(X\otimes I\otimes X\otimes Z\otimes Z)\tag{6.98}$$
$$Z\otimes I\otimes Z\otimes Y\otimes Y=+(I\otimes X\otimes Z\otimes Z\otimes X)(Z\otimes X\otimes I\otimes X\otimes Z),\tag{6.99}$$

the signs of Z_3 and Z_4 need to be corrected, so we start the encoding circuit with X_3X_4. The final encoding circuit is given in figure 6.4.

6.4.2 Encoding a Stabilizer Code with Specified Logical Paulis

If we are given a specific choice of logical Paulis, encoding becomes slightly more complicated. However, the same sort of techniques will work. There are two straightforward options.

Option 1 is to just use procedure 6.5, with fewer simplifications. The desired Clifford group operation maps $Z_i\rightarrow M_i$ for $i=1,\ldots,n-k$, $Z_{n-k+j}\rightarrow\overline{Z}_j$ for $j=1,\ldots,k$, and $X_{n-k+j}\rightarrow\overline{X}_j$ for $j=1,\ldots,k$. Again,

it is more convenient to perform the Hadamard on every qubit so that A and C of the desired symplectic matrix are full $n \times n$ matrices. Column operations which add the first $n-k$ columns of A and C to something can be done just by changing generators of the stabilizer or representatives of the logical Pauli cosets, but column operations involving the last k columns in either the right or left must be done with real gates.

Option 2 is to take advantage of the simplified procedure procedure 6.6 to find an encoding circuit for a stabilizer without specified logical Paulis and see what actual logical Paulis $\overline{P}'$ it produces. Determine the logical Clifford group operation that maps $\overline{P}'$ to the desired logical Pauli $\overline{P}$. Use procedure 6.5 on k qubits to find a circuit performing that Clifford group operation. Perform this circuit on qubits $n-k+1, \ldots, n$, followed by the encoding circuit given by procedure 6.6. The resulting circuit will then encode the code with the correct logical Paulis.

For instance, in the previous subsection, we found an encoding circuit for the five-qubit code. Let us determine what logical Paulis it gives:

$$X_5 \to Z \otimes I \otimes I \otimes Z \otimes X \tag{6.100}$$

$$Z_5 \to Z \otimes Z \otimes Z \otimes Z \otimes Z. \tag{6.101}$$

In this encoding, $\overline{Z}$ is correct. However,

$$\overline{X} = Z \otimes I \otimes I \otimes Z \otimes X = -(X \otimes I \otimes X \otimes Z \otimes Z)(Z \otimes X \otimes I \otimes X \otimes Z)(X \otimes X \otimes X \otimes X \otimes X). \tag{6.102}$$

Therefore, we need the Clifford $X \to -X$, $Z \to Z$. This is just the gate Z, so we can correct the circuit of figure 6.4 by beginning with a Z_5 gate.

6.5 Extending the Clifford Group to a Universal Gate Set

Since the Clifford group can be simulated classically, to do any really interesting quantum algorithms, we will need some gates outside the Clifford group. How many gates and which ones suffice? It turns out *any* single gate will do it.

Theorem 6.9. *Let $G \subseteq \mathsf{SU}(2^n)$ be a group of quantum gates such that $\hat{\mathsf{C}}_n \subset G$ but $G \neq \hat{\mathsf{C}}_n$. Then G is dense in $\mathsf{SU}(2^n)$.*

That is, for any $U \in \mathsf{SU}(2^n)$ and any $\epsilon > 0$, $\exists V \in G$ such that $\|U - V\| < \epsilon$. Thus, we can approximate all quantum gates to any desired degree of accuracy using G, even if G is generated by just the Clifford group, plus any additional gate.

People frequently pick a somewhat nice gate to use as the extra gate. Two popular choices are the Toffoli gate Tof (also known as the controlled-controlled-NOT gate) $|a\rangle|b\rangle|c\rangle \mapsto |a\rangle|b\rangle|c \oplus ab\rangle$ and the $\pi/8$ phase rotation $R_{\pi/8}$. These gates are useful extra gates because they have comparatively straightforward fault-tolerant implementations using common QECCs, as you'll see in chapter 13.

The proof of theorem 6.9 is complicated and involves some more advanced concepts. I hope to include an accessible version of it in a later draft of this book.

Chapter 7

Tighter, Please: Upper And Lower Bounds On Quantum Codes

As I've told you, the three most important properties of a quantum code are the number n of logical qubits (or qudits), the dimension K of the encoded subspace, and the distance d of the code. Together, the parameters $((n, K, d))$ tell us about the trade-off between the rate at which we can send quantum information and the tolerance we gain against errors. Ideally, we'd like to know for any given set $((n, K, d))$ whether a QECC exists with those parameters.

Unfortunately, we can't answer that question. It seems to be extremely hard. However, we can set some bounds, which set limits on where we can hope to find interesting QECCs. On the one hand, there are lower bounds, saying that codes definitely exist with certain parameters. They can be *constructive* (specifying a particular code) or *non-constructive* (proving that codes with the parameters exist without giving an efficient method of specifying one). Then we have the upper bounds. There are many different methods used for proving upper bounds, but they are uniformly destructive: they tell us there are no codes with certain parameters.

The most attractive bounds are the tightest, with the upper bounds hugging as closely as possible to the lower bounds. Occasionally we can actually make the upper and lower bounds touch, but more often there is some space between. Unfortunately, in many instances, our bounds are rather baggy and shapeless, leaving a lot of room, which may or may not contain actual QECCs, between the upper and lower bounds.

7.1 The Quantum Gilbert-Varshamov Bound

We'll start with a lower bound, the quantum Gilbert-Varshamov bound. As you might guess from the name, it is a quantum analogue of the classical Gilbert-Varshamov lower bound discussed in section 4.5. The main difference from the classical Gilbert-Varshamov bound is that there are more quantum errors that occur on a qubit than there are classical errors that can occur on a bit.

Theorem 7.1 (Quantum Gilbert-Varshamov bound)**.** *Suppose*

$$2^k \left(\sum_{j=0}^{d-1} 3^j \binom{n}{j} \right) \leq 2^n. \tag{7.1}$$

Then there exists a $((n, 2^k, d))$ QECC. For large n, a code exists if

$$k/n \leq 1 - (d/n) \log 3 - h(d/n), \tag{7.2}$$

with $h(x) = h_2(x)$ given by equation (4.22).

Actually, the theorem will show that a stabilizer code with these parameters $[[n, k, d]]$ exists. It even can be made a little bit tighter than the statement of the theorem indicates, but it is usually phrased this way to be the closest analogue of the classical Gilbert-Varshamov bound. The proof is non-constructive. In this case, as with the classical Gilbert-Varshamov bound, that means that, while the proof specifies a procedure to find a code with these parameters, the procedure is exponentially long as a function of n, so it's not useful in practice. Since we get a stabilizer code, the code we produce can be described efficiently, but that is small consolation if we can't get all the way through the procedure to find it.

One can also prove a version of the quantum Gilbert-Varshamov bound for qudits, at least for prime power dimensions. The proof is analogous, using the idea of a qudit stabilizer code, which I'll discuss in chapter 8.

Proof. Imagine making a long list of all $[[n, k]]$ stabilizer codes. We are going to run through all possible errors of weight up to $d-1$. For each error E, we can check which codes can detect that error and which cannot, and cross off the list any code that can't detect E. Once we finish running through all the errors, we know that any code that remains must have distance at least d. We'll prove that there must be at least one code to survive, giving us the desired $[[n, k, d]]$ code.

For any given error E, we need to count how many codes cannot detect it. For any two errors $E, F \in \mathsf{P}_n \setminus \{I\}$, there exists some Clifford group element U with $U(E) = F$. Furthermore, if stabilizer code S is a code that doesn't detect E, then $U(\mathsf{S})$ is a stabilizer code that doesn't detect F. That is, U will permute the list of stabilizer codes and their sets of undetectable errors. Thus, the number of codes that cannot detect E must be the same as the number of codes that cannot detect F. That is, all non-identity errors in the Pauli group are undetectable for the same number C of $[[n, k]]$ stabilizer codes.

Suppose there are N stabilizer codes with parameters $[[n, k]]$. (We can evaluate N exactly if we like, but it is not necessary for this argument.) The code S detects the errors outside $\hat{\mathsf{N}}(\mathsf{S}) \setminus \hat{\mathsf{S}}$, which contains $2^{n+k} - 2^{n-k}$ elements. If we consider a bigger list of pairs (S, E) for any pair for which $E \in \hat{\mathsf{N}}(\mathsf{S}) \setminus \hat{\mathsf{S}}$, we can count the number of elements on the list two ways: as the number of errors times the number of codes that cannot detect each error, or as the number of codes times the number of errors that each code cannot detect. There are $4^n - 1$ non-identity errors, and I never appears in $\hat{\mathsf{N}}(\mathsf{S}) \setminus \hat{\mathsf{S}}$, so we have that

$$(4^n - 1)C = N(2^{n+k} - 2^{n-k}). \tag{7.3}$$

Now let us go back to crossing codes off our list of stabilizer codes. We work our way through the set of errors of weight up to $d-1$. For any non-identity error E with weight less than d, there are C codes which do not detect it, so we can cross those codes off the list. Some of them may have already been eliminated because they fail to detect a previous error, but certainly we cannot eliminate more than C new codes from the list of $[[n, k]]$ QECCs. There are a total of $B - 1$ non-identity errors of weight less than d, with

$$B = \sum_{j=0}^{d-1} 3^j \binom{n}{j} \tag{7.4}$$

(not including the identity since all codes detect it). By the time we finish going through all errors of weight less than d, we have therefore crossed off at most $(B-1)C$ codes. If $(B-1)C < N$, then at least one code remains.

Plugging in equation (7.3) to eliminate C, the condition we get is

$$\frac{2^{n+k} - 2^{n-k}}{4^n - 1}(B-1)N < N, \tag{7.5}$$

or

$$2^{n-k}(B-1) < \frac{4^n - 1}{4^k - 1}. \tag{7.6}$$

This is the tightest version of the quantum Gilbert-Varshamov bound, but notice that

$$\frac{4^n - 1}{4^k - 1} > \frac{4^n}{4^k} = 4^{n-k}, \tag{7.7}$$

so if

$$2^{n-k}B \leq 4^{n-k}, \tag{7.8}$$

then equation (7.6) is satisfied too. Equation (7.8) is just what we wanted to prove. □

7.2 The Quantum Hamming Bound

We'll now move on to an upper bound of sorts. It is an analogue of the classical Hamming bound, so is called the quantum Hamming bound.

Theorem 7.2 (Quantum Hamming bound)**.** *If a non-degenerate* $((n, K, 2t+1))_q$ *code exists, then*

$$K\left(\sum_{j=0}^{t}(q^2-1)^j\binom{n}{j}\right) \leq q^n. \tag{7.9}$$

The big catch in the statement of the quantum Hamming bound is that while it provides an upper bound, the upper bound *only holds for non-degenerate codes*. Here is where we start to see very concretely how the existence of degenerate codes makes the quantum case more complicated than the classical case. Insisting that a code be degenerate is potentially a big limitation, yet it is easier to prove bounds on the existence of non-degenerate codes. Indeed, while we don't know how to prove that the quantum Hamming bound applies to degenerate codes as well as non-degenerate codes, we also don't know any $((n, K, 2t+1))$ codes that violate it.

Proof. The proof is a straightforward exercise in counting dimensions, and is very analogous to the proof of the classical Hamming bound. Because the code is non-degenerate, we know that in the QECC conditions, the matrix

$$C_{ab} = \langle\psi|E_a^\dagger E_b|\psi\rangle \tag{7.10}$$

(for any codeword $|\psi\rangle$) has maximum rank when $E_a^\dagger$ and E_b run over any basis for the space of $\leq t$-qubit errors. In particular, this means that the states $E_a|\psi\rangle$ are all linearly independent. Furthermore, if $|\psi\rangle$ and $|\phi\rangle$ are two orthogonal codewords, then the QECC conditions tell us that

$$\langle\psi|E_a^\dagger E_b|\phi\rangle = 0, \tag{7.11}$$

so $E_a|\psi\rangle$ and $E_b|\phi\rangle$ are orthogonal for any a, b.

Thus, if we let $\{|\psi_i\rangle\}$ be a basis for the code space, the set of states $\{E_a|\psi_i\rangle\}$ (over all a, i) are linearly independent. The codespace has dimension K and there are q^2-1 one-qudit non-identity errors, so there are a total of

$$B = \sum_{j=0}^{t}(q^2-1)^j\binom{n}{j} \tag{7.12}$$

linearly independent errors of weight up to t. Thus, we have a set of KB linearly independent vectors, which we must fit into a Hilbert space of n qudits. Therefore,

$$KB \leq q^n \tag{7.13}$$

□

Looking at this proof, you'll see that the quantum Hamming bound does not make essential use of the qudit structure of the space, or the fact that we are dealing with t-qudit errors. In general, if we have a non-degenerate QECC with a K-dimensional code space living in an N-dimensional physical Hilbert space, and the code can correct $\mathcal{E}$, a set of linearly independent errors, then $K|\mathcal{E}| \leq N$.

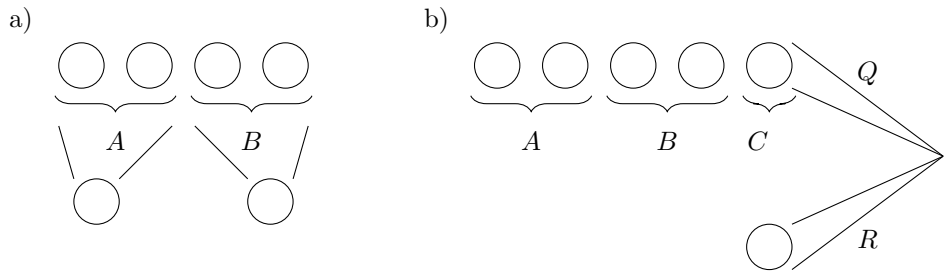


Figure 7.1: a) Dividing n registers into one set A of size $d-1$ and one set B of size $n-(d-1)$ for the proof of lemma 7.4. If $n-(d-1) \leq d-1$, we can decode each set separately to clone the encoded qubit. b) System Q is maximally entangled with system R. Q is then encoded into n registers, which are split into three sets for the proof of the quantum Singleton bound: sets A and B of size $d-1$ and set C of size $n-2(d-1)$.

7.3 The Quantum Singleton Bound

The quantum Singleton bound has a similar form to the classical Singleton bound, but is significantly harder to prove. The quantum Singleton bound is also known as the Knill-Laflamme bound, after the first people to state it. The quantum Singleton bound is a real upper bound, applying to non-degenerate codes as well as degenerate ones, and to both stabilizer and non-stabilizer codes over qubits or qudits.

Theorem 7.3 (Quantum Singleton bound)**.** *If an $((n, q^k, d))_q$ QECC exists, then*

$$n - k \geq 2(d-1). \tag{7.14}$$

As you can see by comparing with theorem 4.19, the quantum bound is more restrictive on the distance by a factor of 2. The reason for this factor of 2 turns out to be the No-Cloning Theorem! You may recall that our very first objection in section 2.1 to the possibility of a quantum error-correcting code was that classical codes seemed to use repetition of information as a key component. We circumvented that difficulty by using entanglement to spread out quantum information without repeating it. The quantum Singleton bound can be viewed as a way of codifying just how well information can be spread around without accidentally repeating anything. For instance, the bound immediately tells us that there can be no complete quantum analogue of the classical $(3, 2, 3)$ repetition code, since a $((3, 2, 3))$ QECC would violate the quantum Singleton bound.

Proof. We'll start by studying the $k = 1$ case, which is an immediate consequence of the No-Cloning Theorem:

Lemma 7.4. *If an $((n, q, d))_q$ QECC exists, then*

$$n - 1 \geq 2(d-1). \tag{7.15}$$

Proof of lemma. If a code has distance d, then it can correct $d-1$ erasure errors. We can imagine dividing the n registers of the code into two sets, a set A with $d-1$ registers and a set B with $n-(d-1)$ registers, as pictured in figure 7.1a. Set B has experienced $d-1$ erasures, so using just those registers, it is possible to reconstruct the original encoded state $|\psi\rangle$.

Now suppose that $n-1 < 2(d-1)$. Then $n-(d-1) \leq d-1$, and therefore set A has also experienced at most $d-1$ erasure errors. We would then be able to reconstruct a second copy of $|\psi\rangle$ using just the registers in set A. We could then use this code to clone the state $|\psi\rangle$ via the following procedure: Encode $|\psi\rangle$ using the QECC, split up the registers into the sets A and B, and then reconstruct $|\psi\rangle$ independently for each set. We know this isn't possible, leading to a contradiction and proving the lemma.

□

Now we can move on to the general case. When $k > 1$, we split the n registers into three sets, as pictured in figure 7.1b. Sets A and B will each have $d-1$ registers, and set C will have $n-2(d-1)$ registers. If an

$((n, q^k, d))_q$ code exists for $k \geq 1$, then a $((n, q, d))_q$ code also exists, as we can just ignore the extra logical codewords. We therefore know from the lemma that $n > 2(d-1)$, and set C is non-empty.

Now imagine taking a $2k$-qudit maximally entangled state between two registers R and Q, each with dimension q^k, and encode Q in the QECC. Then divide up the registers of the QECC into the sets A, B, and C, giving us a total of 4 sets (R, A, B, C). Globally, we now have a pure state, so $S(RABC) = 0$. If we split our sets into two groups, the entropy of the two groups is equal, e.g.

$$S(RA) = S(BC) \tag{7.16}$$
$$S(RB) = S(AC). \tag{7.17}$$

We also know that the code has distance d, so we can detect any erasure error restricted to just set A or to just set B (but not necessarily errors involving both sets). Applying the alternate QECC conditions of section 2.5.6, we see that this means that any operator we measure on set A will give us the same result for all possible logical codewords. Thus, the density matrix ρ_A of set A is the same for all logical codewords, and we find that the density matrix for R and A combined is of the form

$$\sum_{i=0}^{q^k-1} |i\rangle \langle i|_R \otimes \rho_A. \tag{7.18}$$

R and A are in a tensor product state, so $S(RA) = S(R) + S(A)$. Similarly, $S(RB) = S(R) + S(B)$. We also note that $S(R) = k \log q$.

Now, $S(RA) = S(BC)$, and by subadditivity of the entropy, $S(BC) \leq S(B) + S(C)$. Therefore,

$$k \log q + S(A) = S(RA) = S(BC) \leq S(B) + S(C), \tag{7.19}$$

or

$$k \log q \leq S(C) + [S(B) - S(A)]. \tag{7.20}$$

Applying subadditivity to $S(AC)$, we also find that

$$k \log q \leq S(C) + [S(A) - S(B)]. \tag{7.21}$$

Adding together these last two equations, we find that

$$k \log q \leq S(C). \tag{7.22}$$

However, $S(C) \leq \log \dim(C) = [n - 2(d-1)] \log q$. Therefore, we find

$$k \leq n - 2(d-1). \tag{7.23}$$

□

As an immediate application of the quantum Singleton bound, we find that the five-qubit code is optimal. Not only can there be no $((3, 2, 3))$ code, but there can also be no $((4, 2, 3))$ QECC. The five-qubit code exactly saturates the quantum Singleton bound; it is a *quantum MDS code*. We can also saturate the quantum Singleton bound with $d = 2$. There are $[[n, n-2, 2]]$ stabilizer codes for any even n (but not for odd n). As we go to more encoded qubits or to greater distances, it is no longer possible to exactly saturate the quantum Singleton bound with qubit codes. A similar phenomenon occurs classically, and as in the classical case, we get more MDS codes as we let the dimension of the registers increase. In section 8.3, we'll see a large family of such codes.

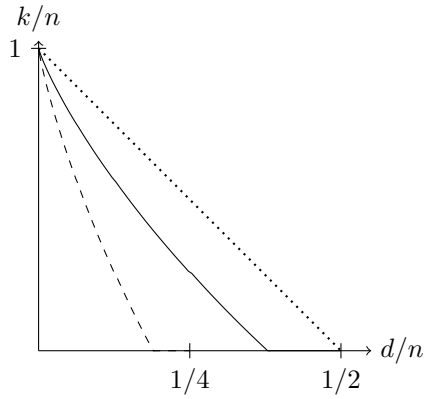


Figure 7.2: Quantum Hamming bound (solid), quantum Gilbert-Varshamov bound (dashed), and quantum Singleton bound (dotted) for large n, $q = 2$

7.4 Linear Programming Bounds

The quantum Singleton bound does provide us with some definite limits on the existence of QECCs since, unlike the quantum Hamming bound, it applies to degenerate codes too. Still, it doesn't tell us much. Figure 7.2 shows the lower bound provided by the quantum Gilbert-Varshamov bound along with the upper bound given by the quantum Singleton bound, both plotted for codes using qubits. I've also shown the quantum Hamming bound so you can see that even if we were able to prove it applied to degenerate codes, there would still be a lot of room where we don't know about the existence of codes. The quantum Singleton bound is much worse. There's no way we can be satisfied with that.

In order to do better, we'll need more powerful techniques. The main ones are known as *linear programming bounds*. A linear programming bound provides a set of linear inequalities that a QECC with a given set of parameters $((n, K, d))_q$ must satisfy. If we can prove that the set of inequalities has no solutions for a given n, K, and d, then we know that a QECC with those parameters cannot exist. If there *is* a solution, then a QECC may or may not exist, but the linear programming solution gives us a number of constraints about the structure of any possible code with those parameters, so provides a good hint as to what to look at to find a code.

In this section, I'll discuss the main linear programming bounds, but only for codes over qubits. These bounds can be generalized to apply to qudits, and it's possible to come up with more linear programming bounds, which could narrow down the set of possible codes even further.

7.4.1 Weight Enumerator and Dual Weight Enumerator

The primary tools used in coming up with linear programming bounds are known as *weight enumerators*. There are many different kinds of weight enumerators, but we'll start with the simplest, which gets no additional adjectives in its name. Weight enumerators are polynomials that encode information about the properties of a QECC.

To motivate the definition of weight enumerator, let's first consider the case of a stabilizer code. The stabilizer S consists of Pauli operators. Some of those Pauli operators are large, with high weight, and some are small, with low weight. Indeed, S always contains I, which has weight 0. We'll count up the number of elements of each weight; let A_j be the number of Paulis $M \in S$ with weight j. The A_js are integers which tell us about the structure of the stabilizer, although they don't tell us everything. How can we get similar information about more general QECCs?

Definition 7.1. Let Π be the projector onto the code subspace of some $((n, K))$ QECC Q. Let

$$A_j = \frac{1}{K^2} \sum_{P \in \hat{\mathsf{P}}_n \,|\, \mathrm{wt}(P)=j} |\mathrm{tr}(P\Pi)|^2 . \tag{7.24}$$

The *weight enumerator* of Q is the degree n polynomial

$$A(x) = \sum_{j=0}^{n} A_j x^j . \tag{7.25}$$

The motivation for defining the weight enumerator as a polynomial instead of just working with the individual coefficients will become clearer in the next subsection, where I'll discuss the quantum MacWilliams identity, which deals with the polynomial as a whole.

Proposition 7.5. *When Q is a stabilizer code with stabilizer S,*

$$A_j = |\{M \in \mathsf{S} |\, \mathrm{wt}\, M = j\}| . \tag{7.26}$$

Proof. For a stabilizer code, $K = 2^k$, and we have an explicit description of the projector onto the code:

$$\Pi = \frac{1}{2^{n-k}} \sum_{M \in \mathsf{S}} M. \tag{7.27}$$

Now, $\mathrm{tr}\, Q = 0$ if $Q \in \hat{\mathsf{P}}_n \setminus \{I\}$ and $\mathrm{tr}\, I = 2^n$. Thus,

$$|\,\mathrm{tr}(PM)| = \begin{cases} 2^n & \text{if } P = M \\ 0 & \text{if } P \neq M. \end{cases} \tag{7.28}$$

We will consider Paulis in $\hat{\mathsf{P}}_n$, so depending on what representatives we pick, it could actually be that $\mathrm{tr}(PM) = -2^n$ or $\pm i 2^n$. However, since we immediately take the absolute value, all of these give the same result.

Applying the general definition of A_j, we find

$$A_j = \frac{1}{2^{2k}} \sum_{P \in \hat{\mathsf{P}}_n \,|\, \mathrm{wt}(P)=j} \left| \frac{1}{2^{n-k}} \sum_{M \in \mathsf{S}} \mathrm{tr}(PM) \right|^2 \tag{7.29}$$

$$= \frac{1}{2^{2k}} \sum_{P \in \hat{\mathsf{P}}_n \,|\, \mathrm{wt}(P)=j} \left| \frac{1}{2^{n-k}} \sum_{M \in \mathsf{S}} 2^n \delta_{P,M} \right|^2 \tag{7.30}$$

$$= \frac{1}{2^{2k}} \sum_{P \in \hat{\mathsf{P}}_n \,|\, \mathrm{wt}(P)=j} 2^{2k} \chi_{\mathsf{S}}(P) \tag{7.31}$$

$$= |\{M \in \mathsf{S} |\, \mathrm{wt}\, M = j\}| . \tag{7.32}$$

In the third line, $\chi_{\mathsf{S}}(P)$ is the indicator function, which is 1 when $P \in \mathsf{S}$ and 0 when $P \notin \mathsf{S}$.

□

Similarly, we can define a dual weight enumerator. For a stabilizer code, it is built out of B_js which give the weight distribution of the *normalizer* of S. In general, we have the definition

Definition 7.2. Let Π be the projector onto the code subspace of some $((n, K))$ QECC Q. Let

$$B_j = \frac{1}{K} \sum_{P \in \hat{\mathsf{P}}_n \,|\, \mathrm{wt}(P)=j} \mathrm{tr}(P\Pi P^\dagger \Pi). \tag{7.33}$$

The *dual weight enumerator* of Q is the degree n polynomial

$$B(x) = \sum_{j=0}^{n} B_j x^j. \tag{7.34}$$

Again, the general definition reduces to the description given for stabilizer codes:

Proposition 7.6. *When Q is a stabilizer code with stabilizer S,*

$$B_j = \left|\{N \in \mathsf{N}(\mathsf{S}) | \operatorname{wt} N = j\}\right|. \tag{7.35}$$

Proof. Again, we have

$$\Pi = \frac{1}{2^{n-k}} \sum_{M \in \mathsf{S}} M, \tag{7.36}$$

and we see that

$$P\Pi P^\dagger = \frac{1}{2^{n-k}} \sum_{M \in \mathsf{S}} (-1)^{c(P,M)} M. \tag{7.37}$$

Note that when $P \in \mathsf{N}(\mathsf{S})$, then $P\Pi P^\dagger = \Pi$. However, when $P \notin \mathsf{N}(\mathsf{S})$, then $P\Pi P^\dagger$ will be the projector on the subspace given by a different error syndrome of S. That subspace is orthogonal to the code space, so we have that $P\Pi P^\dagger \Pi = 0$ when $P \notin \mathsf{N}(\mathsf{S})$. This is also straightforward to prove through direct computation:

$$\begin{aligned} P\Pi P^\dagger \Pi &= \frac{1}{2^{2(n-k)}} \sum_{M,N \in \mathsf{S}} (-1)^{c(P,M)} MN && (7.38)\\ &= \frac{1}{2^{2(n-k)}} \sum_{N' \in \mathsf{S}} \left(\sum_{M \in \mathsf{S}} (-1)^{c(P,M)} \right) N' && (7.39)\\ &= \frac{1}{2^{2(n-k)}} \sum_{N' \in \mathsf{S}} 2^{n-k} \chi_{\mathsf{N}(\mathsf{S})}(P)\, N' && (7.40)\\ &= \chi_{\mathsf{N}(\mathsf{S})}(P)\, \Pi, && (7.41) \end{aligned}$$

where $\chi_{\mathsf{N}(\mathsf{S})}(P)$ is the indicator function for $\mathsf{N}(\mathsf{S})$ and line three follows because if $P \notin \mathsf{N}(\mathsf{S})$, then P anticommutes with exactly half of S.

Thus, applying the general definition of B_j, we find

$$\begin{aligned} B_j &= \frac{1}{2^k} \sum_{P \in \hat{\mathsf{P}}_n | \operatorname{wt}(P) = j} \operatorname{tr}(\Pi) \chi_{\mathsf{N}(\mathsf{S})}(P) && (7.42)\\ &= \left|\{N \in \mathsf{N}(\mathsf{S}) | \operatorname{wt} N = j\}\right|. && (7.43) \end{aligned}$$

□

The coefficients A_j and B_j of the weight enumerator and dual weight enumerator are always integers for a stabilizer code, but can be non-integer for general QECCs.

For a stabilizer code, the dual weight enumerator tells us about the structure of the normalizer, which in turn tells us something about which errors we can detect. If B_j is non-zero, it means there are some Paulis of weight j in the normalizer, which worries us if j is small. Actually, though, we are interested in the number of elements of $\hat{\mathsf{N}}(\mathsf{S}) \setminus \hat{\mathsf{S}}$, which is given by $B_j - A_j$. If *that* is non-zero, then we actually have some errors of weight j that are undetectable, giving us a bound on the distance of the code. While this argument does not hold for general QECCs, the intuition and result about A_j and B_j does carry over:

Theorem 7.7. *Let Q be a QECC with weight enumerator $A(x) = \sum A_j x^j$ and dual weight enumerator $B(x) = \sum B_j x^j$. Then*

a) $A_0 = B_0 = 1$

b) $B_j \geq A_j \geq 0$

c) Q has distance d iff $A_j = B_j$ *for* $j < d$.

Proof.

a) There is only one Pauli of weight 0: the identity. Thus,

$$A_0 = \frac{1}{K^2} \left|\mathrm{tr}(\Pi)\right|^2 = 1 \tag{7.44}$$

$$B_0 = \frac{1}{K} \mathrm{tr}(\Pi^2) = \frac{1}{K} \mathrm{tr}(\Pi) = 1. \tag{7.45}$$

b) From the definition of A_j, we can immediately see that $A_j \geq 0$.

Let $\{|a\rangle\}$ be a basis for the code space of Q. Then

$$\Pi = \sum_a |a\rangle \langle a|, \tag{7.46}$$

so

$$A_j = \frac{1}{K^2} \sum_{P|\,\mathrm{wt}(P)=j} \left| \sum_a \langle a|P|a\rangle \right|^2, \tag{7.47}$$

$$B_j = \frac{1}{K} \sum_{P|\,\mathrm{wt}(P)=j} \sum_{a,b} \left|\langle a|P|b\rangle\right|^2. \tag{7.48}$$

Our next step is to apply the Cauchy-Schwarz inequality, which says that for two complex vectors $\vec{x}$ and $\vec{y}$,

$$|\vec{x} \cdot \vec{y}|^2 \leq |\vec{x}|^2 |\vec{y}|^2. \tag{7.49}$$

Let $\vec{x}$ be the K^2-dimensional complex vector with entries $\langle a|P|b\rangle$ (running over pairs (a,b)), and let $\vec{y}$ be the K^2-dimensional vector with entries equal to $(1/K)\delta_{ab}$ (again running over pairs (a,b)). Then we have

$$|\vec{x} \cdot \vec{y}|^2 = \left| \frac{1}{K} \sum_{a,b} \langle a|P|b\rangle \delta_{ab} \right|^2 \tag{7.50}$$

$$= \frac{1}{K^2} \left| \sum_a \langle a|P|a\rangle \right|^2 \tag{7.51}$$

$$\leq |\vec{x}|^2 |\vec{y}|^2 \tag{7.52}$$

$$= \left(\sum_{a,b} \left|\langle a|P|b\rangle\right|^2 \right) \left(\sum_{ab} \frac{1}{K^2} \delta_{ab} \right) \tag{7.53}$$

$$= \frac{1}{K} \sum_{a,b} \left|\langle a|P|b\rangle\right|^2. \tag{7.54}$$

Plugging into equations (7.47) and (7.48), we find that $A_j \leq B_j$.

c) From the definition of distance, equation (2.66), if the code has distance d, then for $\mathrm{wt}(E) < d$,

$$\langle \psi|E|\phi\rangle = c(E) \langle \psi|\phi\rangle \tag{7.55}$$

for any codewords $|\psi\rangle$ and $|\phi\rangle$. Applying equations (7.47) and (7.48) for $j < d$, we get

$$A_j = \frac{1}{K^2} \sum_{E|\,\mathrm{wt}\,E=j} K^2 |c(E)|^2 \tag{7.56}$$

$$B_j = \frac{1}{K} \sum_{E|\,\mathrm{wt}\,E=j} K |c(E)|^2, \tag{7.57}$$

and $A_j = B_j$.

For the converse, looking at the proof of part b, the equality condition for the Cauchy-Schwarz inequality is that $\vec{x}$ is proportional to $\vec{y}$. Thus, $A_j = B_j$ implies that

$$\langle a|E|b\rangle = C(E)\delta_{ab} \tag{7.58}$$

whenever $\mathrm{wt}\,E = j$. This is one of the alternate forms of the QECC conditions.

□

The classical theory of weight enumerators is similar, except that for classical codes, $A_j = B_j = 0$ for $j < d$. The difference is essentially a manifestation of the phenomenon of degeneracy. Let's think about the special case of stabilizer codes. In a non-degenerate stabilizer code S, there are no elements of $\mathsf{N}(\mathsf{S})$ with weight less than d. Therefore $B_j = 0$ for $j < d$. In a degenerate stabilizer code, $\mathsf{N}(\mathsf{S})$ can have elements with weight less than d, but those elements must also be in S. Thus, $B_j = A_j > 0$ for some $j < d$.

This intuition almost carries over to general QECCs, but it turns out to be a slightly different concept.

Definition 7.3. An $((n, K, d))$ QECC with weight enumerators $A(x)$ and $B(x)$ is *pure* if $A_j = B_j = 0$ for $j < d$. A code that is not pure is *impure*.

The argument above shows that for stabilizer codes, pure is the same as non-degenerate. However, for more general codes, the property of being pure is more stringent than the property of being non-degenerate. In other words, a code can be impure without being degenerate, but if it's degenerate, it is definitely impure.

The possibility of impure codes complicates the application of the linear programming bounds to quantum codes. Some results which are known for classical codes have not yet been adapted for quantum codes because of the additional difficulty created by dealing with impure codes.

7.4.2 Quantum MacWilliams Identity

Theorem 7.7 gives us one set of inequalities constraining the coefficients A_j and B_j. However, these inequalities by themselves are not yet enough to rule out any codes, since for any set of parameters $((n, K, d))$, we can easily come up with a set of A_j and B_j that satisfy theorem 7.7. In order to get actual upper bounds on codes this way, we'll need a stronger relation between $A(x)$ and $B(x)$.

In the case of a stabilizer code, the weight enumerator $A(x)$ encapsulates properties of the stabilizer S and the dual weight enumerator $B(x)$ captures properties of $\mathsf{N}(\mathsf{S})$. Since $\mathsf{N}(\mathsf{S})$ is completely determined by S, it's perhaps not too surprising that there is some relationship between $A(x)$ and $B(x)$ as well. What *is* remarkable is that even though the full description of S contains a lot more information than is contained in $A(x)$, the weight enumerator by itself is enough to completely determine $B(x)$. Moreover, this relationship also holds for non-stabilizer codes.

Theorem 7.8 (Quantum MacWilliams identity)**.** *Suppose we have an $((n, K))$ QECC with weight enumerator $A(x)$ and dual weight enumerator $B(x)$. Then*

$$B(x) = \frac{K}{2^n}(1+3x)^n A\left(\frac{1-x}{1+3x}\right) \tag{7.59}$$

The quantum MacWilliams identity above is phrased in terms of codes over qubits, but like the other bounds we've discussed, it can also be generalized to codes over qudits.

Proof. Let's eliminate the projector Π that appears in the definitions of A_j and B_j in favor of Paulis. We can do this by using the fact that $\hat{\mathsf{P}}_n$ gives a basis for the space of $2^n \times 2^n$ matrices, so

$$\Pi = \sum_{Q \in \hat{\mathsf{P}}_n} c_Q Q. \tag{7.60}$$

The coefficients $c_Q = \operatorname{tr}(Q^\dagger \Pi)/2^n$.

Then

$$A_j = \frac{1}{K^2} \sum_{P \in \hat{\mathsf{P}}_n \,|\, \operatorname{wt}(P)=j} \left| \sum_{Q \in \hat{\mathsf{P}}_n} c_Q \operatorname{tr}(PQ) \right|^2 \tag{7.61}$$

$$= \frac{4^n}{K^2} \sum_{P \in \hat{\mathsf{P}}_n \,|\, \operatorname{wt}(P)=j} |c_P|^2, \tag{7.62}$$

and

$$B_j = \frac{1}{K} \sum_{P \in \hat{\mathsf{P}}_n \,|\, \operatorname{wt}(P)=j} \sum_{Q,R \in \hat{\mathsf{P}}_n} c_Q c_R^* \operatorname{tr}(PQP^\dagger R^\dagger) \tag{7.63}$$

$$= \frac{1}{K} \sum_{P \in \hat{\mathsf{P}}_n \,|\, \operatorname{wt}(P)=j} \sum_{Q,R \in \hat{\mathsf{P}}_n} c_Q c_R^* (-1)^{c(P,Q)} \delta_{Q,R} 2^n \tag{7.64}$$

$$= \frac{2^n}{K} \sum_{Q \in \hat{\mathsf{P}}_n} \left[\sum_{P \in \hat{\mathsf{P}}_n \,|\, \operatorname{wt}(P)=j} (-1)^{c(P,Q)} \right] |c_Q|^2. \tag{7.65}$$

In the first line for B_j, I've used $\Pi^\dagger$ in one place instead of Π (they are the same, after all) to finesse issues about using Paulis with the overall sign modded out. We can see that A_j and B_j both involve a sum over $|c_Q|^2$, but with two differences: In A_j, we only sum over Paulis of weight j whereas for B_j we sum over all Paulis, and in B_j we have an additional sum with alternating signs which depend on the commutation relations.

Let's take some Q with weight i and try to count how many Paulis of weight j commute with it, which we'll call C_{ij}, and how many anticommute, A_{ij}. Then we will have

$$B_j = \frac{2^n}{K} \sum_{Q \in \hat{\mathsf{P}}_n} (C_{\operatorname{wt}(Q),j} - A_{\operatorname{wt}(Q),j}) |c_Q|^2. \tag{7.66}$$

First of all, we know that there are a total of

$$3^j \binom{n}{j} \tag{7.67}$$

Paulis of weight j. Also note that the answer won't depend on exactly what Q we take, only its weight i, because single-qubit Clifford group operations and permutations of the qubits can relate any two Paulis of weight i, and those operations won't change the commutation relations or weights of Paulis.

Let's break up the counting problem into counting Paulis P that overlap with Q on exactly m qubits (i.e., the intersection of their supports is m qubits), and determining how many commute (C_{ij}^m) and how many anticommute (A_{ij}^m). P has total weight j, so it acts on exactly $j-m$ qubits outside the support of Q. There are thus

$$3^{j-m} \binom{n-i}{j-m} \tag{7.68}$$

ways of choosing the action of P outside the support of Q. Within the support of Q, there are $\binom{i}{m}$ ways of choosing which qubits P acts on non-trivially.

We now wish to find how many weight m Paulis commute and anticommute with a fixed Pauli of weight m, which can assume without loss of generality to be $Z^{\otimes m}$. That is, it suffices to find C^m_{mm} and A^m_{mm}, and then we know that

$$C^m_{ij} = 3^{j-m}\binom{n-i}{j-m}\binom{i}{m}C^m_{mm} \tag{7.69}$$

$$A^m_{ij} = 3^{j-m}\binom{n-i}{j-m}\binom{i}{m}A^m_{mm}. \tag{7.70}$$

Now, $C^1_{11} = 1$ and $A^1_{11} = 2$. We can use induction to determine C^m_{mm} and A^m_{mm} for all m:

$$C^m_{mm} = C^{m-1}_{(m-1)(m-1)} + 2A^{m-1}_{(m-1)(m-1)} \tag{7.71}$$

$$A^m_{mm} = 2C^{m-1}_{(m-1)(m-1)} + A^{m-1}_{(m-1)(m-1)}. \tag{7.72}$$

Actually, what we're really interested in is $C^m_{ij} - A^m_{ij}$, and we see that

$$C^m_{mm} - A^m_{mm} = -C^{m-1}_{(m-1)(m-1)} + A^{m-1}_{(m-1)(m-1)} = (-1)^{m-1}\left(C^1_{11} - A^1_{11}\right) = (-1)^m. \tag{7.73}$$

Putting all of this together, we find that

$$B_j = \frac{2^n}{K}\sum_{Q\in\hat{\mathsf{P}}_n}(C_{\mathrm{wt}(Q),j} - A_{\mathrm{wt}(Q),j})|c_Q|^2 \tag{7.74}$$

$$= \frac{2^n}{K}\sum_{i=0}^{n}\sum_{Q\in\hat{\mathsf{P}}_n\,|\,\mathrm{wt}\,Q=i}\left[\sum_{m=0}^{\min(i,j)}\left(C^m_{ij} - A^m_{ij}\right)\right]|c_Q|^2 \tag{7.75}$$

$$= \frac{2^n}{K}\sum_{i=0}^{n}\sum_{Q\in\hat{\mathsf{P}}_n\,|\,\mathrm{wt}\,Q=i}\left[\sum_{m=0}^{\min(i,j)}(-1)^m 3^{j-m}\binom{n-i}{j-m}\binom{i}{m}\right]|c_Q|^2 \tag{7.76}$$

$$= \frac{K}{2^n}\sum_{i=0}^{n}\left[\sum_{m=0}^{\min(i,j)}(-1)^m 3^{j-m}\binom{n-i}{j-m}\binom{i}{m}\right]A_i. \tag{7.77}$$

$$\tag{7.78}$$

The function

$$P_j(z;n) = \sum_{m=0}^{\min(z,j)}(-1)^m 3^{j-m}\binom{n-z}{j-m}\binom{z}{m} \tag{7.79}$$

is known as a Krawtchouk polynomial, and

$$(1+3x)^{n-z}(1-x)^z = \sum_j P_j(z;n)x^j, \tag{7.80}$$

as can be verified by expanding the left-hand side. That tells us

$$B(x) = \sum_{j=0}^{n} B_j x^j \tag{7.81}$$

$$= \frac{K}{2^n} \sum_{i=0}^{n} \sum_{j=0}^{n} P_j(i;n) A_i x^j \tag{7.82}$$

$$= \frac{K}{2^n} \sum_{i=0}^{n} (1+3x)^{n-i} (1-x)^i A_i \tag{7.83}$$

$$= \frac{K}{2^n} (1+3x)^n \sum_{i=0}^{n} \left(\frac{1-x}{1+3x} \right)^i A_i \tag{7.84}$$

$$= \frac{K}{2^n} (1+3x)^n A\left(\frac{1-x}{1+3x} \right) \tag{7.85}$$

□

The quantum MacWilliams identity combined with theorem 7.7 puts a number of restrictions on the possible values of $((n, K, d))$ for a code, enough to rule out many more possibilities than the quantum Singleton bound. However, we can set even tighter bounds by adding another weight enumerator known as the quantum shadow enumerator.

7.4.3 Quantum Shadow Enumerator

The quantum shadow enumerator has a somewhat stranger definition than the weight enumerator and dual weight enumerator. The weight enumerator and dual weight enumerator treat all Paulis of a given weight on an equal footing. The quantum shadow enumerator does not:

Definition 7.4. Suppose Π is the projector onto the code subspace of some $((n, K))$ QECC Q. Let

$$Sh_j = \frac{1}{K} \sum_{P \in \hat{\mathsf{P}}_n \,|\, \mathrm{wt}\, P = j} \mathrm{tr}(P \Pi P^\dagger Y^{\otimes n} \Pi^* Y^{\otimes n}). \tag{7.86}$$

The *shadow enumerator* of Q is

$$Sh(x) = \sum_{j=0}^{n} Sh_j x^j. \tag{7.87}$$

Π^* is the complex conjugate of the projector Π. The definition of the shadow enumerator is heavily basis-dependent, both for defining the Paulis and for defining the complex conjugate of an operator.

Theorem 7.9. *Suppose we have an $((n, K))$ QECC with weight enumerator $A(x)$ and shadow enumerator $Sh(x) = \sum Sh_j x^j$. Then*

a) $Sh_j \geq 0 \ \forall j$

b) $Sh(x)$ can be determined from $A(x)$:

$$Sh(x) = \frac{K}{2^n} (1+3x)^n A\left(\frac{x-1}{1+3x} \right). \tag{7.88}$$

Notice that the relation between $A(x)$ and $Sh(x)$ differs from the quantum MacWilliams identity only in that there is an $x - 1$ in the numerator of the argument of A instead of $1 - x$.

Proof.

a) We can write $\Pi = \sum_a |a\rangle\langle a|$, for $|a\rangle$ a basis of the code space. Note that this basis might be different from the basis in which the shadow enumerator is defined, so we let $|a^*\rangle$ be the complex conjugate of $|a\rangle$ in the defining basis. Then

$$Sh_j = \frac{1}{K} \sum_{P \in \hat{\mathsf{P}}_n | \, \mathrm{wt}\, P = j} \sum_{a,b} \mathrm{tr}(P|a\rangle \langle a|P^\dagger Y^{\otimes n}|b^*\rangle \langle b^*|Y^{\otimes n}) \tag{7.89}$$

$$= \frac{1}{K} \sum_{P \in \hat{\mathsf{P}}_n | \, \mathrm{wt}\, P = j} \sum_{a,b} \langle b^*|Y^{\otimes n} P|a\rangle \langle a|P^\dagger Y^{\otimes n}|b^*\rangle \tag{7.90}$$

$$\geq 0 \tag{7.91}$$

since $Y = Y^\dagger$.

b) The proof proceeds similarly to that for the quantum MacWilliams identity. We again write

$$\Pi = \sum_{Q \in \hat{\mathsf{P}}_n} c_Q Q, \tag{7.92}$$

and find

$$Sh_j = \frac{1}{K} \sum_{P \in \hat{\mathsf{P}}_n | \, \mathrm{wt}(P) = j} \sum_{Q, R \in \hat{\mathsf{P}}_n} c_Q c_R^* \, \mathrm{tr}(PQP^\dagger Y^{\otimes n} R^* Y^{\otimes n}) \tag{7.93}$$

$$= \frac{1}{K} \sum_{P \in \hat{\mathsf{P}}_n | \, \mathrm{wt}(P) = j} \sum_{Q, R \in \hat{\mathsf{P}}_n} c_Q c_R^* (-1)^{c(P,Q)} \, \mathrm{tr}(QY^{\otimes n} R^* Y^{\otimes n}). \tag{7.94}$$

Now, $X^* = X$ and $Z^* = Z$, but $Y^* = -Y$, so $R^* = \pm R$, with the sign determined by whether there are an even or an odd number of Ys in the tensor decomposition of R. Furthermore,

$$Y^{\otimes n} R^* Y^{\otimes n} = (-1)^{c(R, Y^{\otimes n})} R^*. \tag{7.95}$$

$c(R, Y^{\otimes n})$ is 1 if the number of Xs plus Zs in the tensor decomposition of R is odd, and 0 if it is even. Thus,

$$\mathrm{tr}(QY^{\otimes n} R^* Y^{\otimes n}) = (-1)^{\mathrm{wt}(R)} \delta_{Q,R} 2^n. \tag{7.96}$$

The sign follows because if the number of Ys and the number of Xs plus Zs in R are both even or both odd, we get an overall factor of $+1$, whereas if one of them is odd and one is even, we get a factor of -1.

We therefore have

$$Sh_j = \frac{1}{K} \sum_{P \in \hat{\mathsf{P}}_n | \, \mathrm{wt}(P) = j} \sum_{Q, R \in \hat{\mathsf{P}}_n} c_Q c_R^* (-1)^{c(P,Q) + \mathrm{wt}(R)} \delta_{Q,R} 2^n \tag{7.97}$$

$$= \frac{2^n}{K} \sum_{Q \in \hat{\mathsf{P}}_n} \left[\sum_{P \in \hat{\mathsf{P}}_n | \, \mathrm{wt}(P) = j} (-1)^{c(P,Q)} \right] (-1)^{\mathrm{wt}(Q)} |c_Q|^2. \tag{7.98}$$

The evaluation of the sum over P is just the same as before. We again get a Krawtchouk polynomial, to find

$$Sh(x) = \frac{K}{2^n} \sum_{j=0}^{n} \sum_{i=0}^{n} (-1)^i P_j(i; n) A_i x^j \tag{7.99}$$

$$= \frac{K}{2^n} \sum_{i=0}^{n} (1+3x)^{n-i} (1-x)^i (-1)^i A_i \tag{7.100}$$

$$= \frac{K}{2^n} (1+3x)^n \sum_{i=0}^{n} \left(\frac{x-1}{1+3x} \right)^i A_i \tag{7.101}$$

$$= \frac{K}{2^n} (1+3x)^n A \left(\frac{x-1}{1+3x} \right). \tag{7.102}$$

□

7.4.4 Example: There Is No $((3,2,2))$ Code

OK. We have these three weight enumerators, but what do we do with them? How can we put together the constraints we have to find out if certain parameters of codes are possible or not? One approach is to make approximations and use properties of the Krawtchouk polynomials to prove non-existence of certain solutions. This approach gives another method of proving the quantum Singleton bound, and can also give tighter bounds on the existence of QECCs.

Another approach is to apply the linear programming method. The procedure is to treat the parameters A_j, B_j, and Sh_j as unknown variables, and write down all the equations we have above:

$$A_0 = 1 \tag{7.103}$$
$$B_0 = 1 \tag{7.104}$$
$$B_j = A_j \text{ (for } j < d) \tag{7.105}$$
$$B_j \geq A_j \text{ (for } j \geq d) \tag{7.106}$$
$$A_j \geq 0 \tag{7.107}$$
$$Sh_j \geq 0 \tag{7.108}$$
$$B(x) = \frac{K}{2^n}(1+3x)^n A\left(\frac{1-x}{1+3x}\right) \tag{7.109}$$
$$Sh(x) = \frac{K}{2^n}(1+3x)^n A\left(\frac{x-1}{1+3x}\right) \tag{7.110}$$

Of course, many of the lines actually represent groups of equations, running over values of j. Notice that all the equations, even the last two groups, are linear in A_j, B_j, and Sh_j. Some are equalities and some are inequalities. The question of whether a solution exists or not is a linear programming problem, which is a common task. There are standard computer packages for solving linear programming problems, and so the usual method is to put the equations corresponding to the desired parameters $((n,K,d))$ of a code into such a package, and see whether it says a solution is possible. If not, we know no code can exist with those parameters. If so, we learn the solution(s) of the linear programming problem, which doesn't tell us whether a code exists or not, but does at least tell us possible values for the code's weight enumerator if it does exist, and therefore something about the structure of the code.

For large parameters, this is certainly best done by a computer, but to show you how the procedure works, I'll do a small example here explicitly, showing that there is no $((3,2,2))$ QECC. It is easy to show that there is no $[[3,1,2]]$ stabilizer code, but for more general codes, we need a more powerful method, and the linear programming bounds will suffice. The code is allowed by the quantum Singleton bound, and indeed, you'll see a $((3,3,2))_3$ code over qutrits in chapter 8.

Let us start by writing down the equations produced by the quantum MacWilliams identity:

$$B(x) = \frac{2}{2^3}(1+3x)^3 A\left(\frac{1-x}{1+3x}\right) \tag{7.111}$$
$$= \frac{1}{4}\left[(1+3x)^3 A_0 + (1+3x)^2(1-x)A_1 + (1+3x)(1-x)^2 A_2 + (1-x)^3 A_3\right] \tag{7.112}$$

Collecting the coefficients of x^j for $j = 0,\ldots,3$, we find

$$4B_0 = A_0 + A_1 + A_2 + A_3 \tag{7.113}$$
$$4B_1 = 9A_0 + 5A_1 + A_2 - 3A_3 \tag{7.114}$$
$$4B_2 = 27A_0 + 3A_1 - 5A_2 + 3A_3 \tag{7.115}$$
$$4B_3 = 27A_0 - 9A_1 + 3A_2 - A_3. \tag{7.116}$$

We also know that $A_0 = A_1 = 1$, $B_1 = A_1 \geq 0$, $B_2 \geq A_2 \geq 0$, and $B_3 \geq A_3 \geq 0$. Combining these equations, we find

$$A_1 + A_2 + A_3 = 3 \tag{7.117}$$
$$A_1 + A_2 - 3A_3 = -9 \tag{7.118}$$
$$3A_1 - 9A_2 + 3A_3 \geq -27 \tag{7.119}$$
$$-9A_1 + 3A_2 - 5A_3 \geq -27 \tag{7.120}$$

From equations (7.117) and (7.118), we find $A_3 = 3$, so $A_1 + A_2 = 0$. Since $A_1, A_2 \geq 0$, this means that $A_1 = A_2 = 0$. This solution also satisfies equations (7.119) and (7.120). Using only the quantum MacWilliams identity, we cannot rule out a $((3, 2, 2))$ code, but we know that if one exists, it must have weight enumerator $A(x) = 1 + 3x^3$ and dual weight enumerator $B(x) = 1 + 9x^2 + 6x^3$.

We now invoke the shadow enumerator. Let us write down the relation between $A(x)$ and $Sh(x)$, and plug in the above value for $A(x)$:

$$Sh(x) = \frac{1}{4}\left[(1+3x)^3 A_0 + (1+3x)^2(x-1)A_1 + (1+3x)(x-1)^2 A_2 + (x-1)^3 A_3\right] \tag{7.121}$$
$$= \frac{1}{4}\left[(1+3x)^3 + 3(x-1)^3\right] \tag{7.122}$$
$$= \frac{1}{4}\left(-2 + 18x + 18x^2 + 30x^3\right). \tag{7.123}$$

However, we see that $Sh_0 < 0$, which is not allowed. Thus, a $((3, 2, 2))$ code is not possible.

Chapter 8

Bigger Can Be Better: Qudit Codes

Computers seem to like base 2, even when they are quantum computers. Consequently, it may seem natural to stick to bits and qubits when dealing with error correcting codes, classical or quantum. Indeed, I've gone to a lot of effort in the previous chapters to build the mathematical structure of stabilizer codes and the Clifford group, all of which depends on the basic registers of our quantum computer being qubits. The problem is that there are lots of interesting codes that don't quite fit into this structure. Instead, they fit into similar structures associated to higher-dimensional registers — qudits.

The phenomenon also occurs in the theory of classical error correction, where it's helpful to work with linear codes over arbitrary finite fields rather than just binary linear codes. That's why I introduced non-binary codes in chapter 4. Now we'll see how to do the same for quantum codes. Quantum codes with bigger registers can generally correct more errors and send quantum data with a higher rate than qubit codes.

Based on the last sentence, this may appear a clear-cut case of bigger being better, but it's not quite that straightforward. You can't compare an error on a single qubit with an error on a 32-dimensional qudit on an even basis. I wouldn't say you are comparing apples to oranges; rather, you are comparing apples to boxes of apples. A 32-dimensional qudit could be written using 5 qubits, so a single error on a 32-dimensional error could be 5 single-qubit errors, and a single-qudit gate in 32 dimensions might need many one- and two-qubit gates to achieve the same transformation. Nevertheless, some qudit codes are sufficiently interesting, in terms of efficiency or other properties, to be worthwhile even once you take the size difference of the registers into account.

8.1 Qudit Pauli Group

A *qudit* is a generic term for a q-dimensional Hilbert space, considered as a fundamental unit of the overall Hilbert space of dimension q^n in much the same way that a qubit is a fundamental unit of a 2^n-dimensional Hilbert space. A 2^n-dimensional Hilbert space has many possible factorizations into two-dimensional tensor factors, but one factorization is considered physically favored, and that factorization gives us the physical qubits making up the full Hilbert space. (The *logical* qubits encoded in a quantum error-correcting code make up part of a different factorization.) Similarly, a q^n-dimensional Hilbert space is generally assumed to have some physically favored tensor product decomposition, producing the physical qudits comprising the code.

The word "qudit" for a q-dimensional Hilbert space seems to have mostly won out in the literature, but in some of the older literature you can find a variety of other terms, such as "qupit" (which usually assumes the dimension is a prime p). You can also find, in papers young and old, occasional terms referring to a qudit for a dimension of a specific size. "Qutrit" is the most common, for $D = 3$. Beyond that, you may have to delve into the word's Latin or Greek roots to figure out the dimension. For instance, a 4-dimensional qudit might be a "ququart", a 5-dimensional one might be a "ququint" or a "qupent" or some such, and for $D = 6$, you might even get lucky with a "qusext."

The first step in building quantum error-correcting codes for qudits, by whatever name, is to come up with the correct generalization of the Pauli group. As for qubits, the qudit Pauli group will play a role both in describing the types of errors we face and as a structural element for the qudit generalization of stabilizer codes. We're going to make heavy use of the machinery of finite fields in this chapter, so if you're not already familiar with it, you may want to go through appendix C first.

8.1.1 Qudit Pauli Group for Prime Dimension

The case where the qudit dimension p is prime is the simplest, so we'll start there.

Definition 8.1. The single-qudit *Pauli group* $\mathsf{P}_1(p)$ for prime $p > 2$ consists of elements $\{\omega^a X^b Z^c\}$, where

$$\omega = e^{2\pi i/p} \tag{8.1}$$

$$X|j\rangle = |(j+1) \bmod p\rangle \tag{8.2}$$

$$Z|j\rangle = \omega^j |j\rangle, \tag{8.3}$$

and a, b, c can be anywhere from 0 to $p-1$. The n-qudit *Pauli group* $\mathsf{P}_n(p) = \mathsf{P}_1(p)^{\otimes n}$; it consists of tensor products of n terms, each of the form $X^b Z^c$, with an overall phase factor of ω^a. $\hat{\mathsf{P}}_n(p) = \mathsf{P}_n(p)/\{\omega^a I\}$ is the qudit Pauli group without phases.

That is, ω is a pth root of unity, X is "add one mod p", and Z is a phase shift by a pth root of unity. We can write X and Z as matrices, of course:

$$X = \begin{pmatrix} 0 & 0 & \dots & 0 & 1 \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{pmatrix} \qquad Z = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & \omega & \dots & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \dots & \omega^{p-2} & 0 \\ 0 & 0 & \dots & 0 & \omega^{p-1} \end{pmatrix} \tag{8.4}$$

You can see that X and Z are still unitary but are no longer Hermitian. I'd still like to talk about measuring X or Z; for any traditionalists reading this who prefer to measure only Hermitian operators, just imagine measuring a Hermitian operator that has the same eigenbasis as X or Z. For instance, $\sum_j j|j\rangle\langle j|$ works instead of Z, and outcome j corresponds to the ω^j eigenvalue of Z.

We can calculate the commutation relations between elements of the Pauli group. We find

$$ZX = \omega XZ. \tag{8.5}$$

It follows that

$$(X^a Z^b)(X^c Z^d) = \omega^{bc-ad}(X^c Z^d)(X^a Z^b). \tag{8.6}$$

Since ω is a pth root of unity, we might as well calculate $bc - ad$ using arithmetic modulo p. We can define a new function $c(P,Q) : \mathsf{P}_n(p) \times \mathsf{P}_n(p) \to \mathbb{Z}_p$ by

$$PQ = \omega^{c(P,Q)} QP. \tag{8.7}$$

We no longer are choosing between commuting and anti-commuting Paulis; now commutation is measured by an integer modulo p, which describes the phase factor that appears when we move P past Q as a power of ω.

A very important difference when p is an odd prime rather than 2 is that we only need the phases to be powers of ω; we don't get the annoying factor of i that shows up for qubits. This is because all elements of

$\mathsf{P}_n(p)$ have order p, regardless of overall phase:

$$(\omega^a X^b Z^c)^p = \omega^{ap}(X^b Z^c)(X^b Z^c)^{p-1} \tag{8.8}$$
$$= \omega^{bc} X^{2b} Z^{2c} (X^b Z^c)^{p-2} \tag{8.9}$$
$$= \omega^{bc+2bc} X^{3b} Z^{3c} (X^b Z^c)^{p-3} \tag{8.10}$$
$$\ldots = \omega^{bc+2bc+\cdots(p-1)bc} X^p Z^p \tag{8.11}$$
$$= \omega^{bcp(p-1)/2} \tag{8.12}$$
$$= 1, \tag{8.13}$$

since p is odd. This also means that the eigenvalues of all elements of the qudit Pauli group are of the form ω^j for integer j. While the qudit Pauli group is more complicated than the qubit Pauli group in a variety ways, in this one way, at least, it is simpler.

There are alternate representations of the qudit Pauli group, just as for the qubit Pauli group. For instance, we can define a $\mathbb{Z}_p$ symplectic representation of the qudit Pauli group by

$$P = \bigotimes_{j=1}^{n} X^{a_j} Z^{b_j} \mapsto v_P = (x_P | z_P), \tag{8.14}$$

with x_P and z_P being n-component vectors over $\mathbb{Z}_p$. x_P has entries a_j, and z_P has entries b_j. As with qubits, the overall phase of P is lost when moving to the symplectic representation.

Suppose we have v_P as defined in equation (8.14) and v_Q, with

$$Q = \bigotimes_{j=1}^{n} X^{c_j} Z^{d_j}. \tag{8.15}$$

We can define a symplectic inner product between v_P and v_Q as

$$v_P \odot v_Q = \sum_j (b_j c_j - a_j d_j), \tag{8.16}$$

with arithmetic taken mod p. With this definition, we have the following result:

Proposition 8.1. *Let $P, Q \in \mathsf{P}_n(p)$. Then $c(P,Q) = v_P \odot v_Q$.*

In other words, commutation in the qudit Pauli group corresponds to the symplectic inner product.

We can also map the qudit Pauli group to $\mathrm{GF}(p^2)$. However, the procedure is a bit more complicated than for qubits. We should think of $\mathrm{GF}(p^2)$ as a 2-dimensional vector space over $\mathrm{GF}(p)$ with basis $\{1, \alpha\}$. α can be any element of $\mathrm{GF}(p^2)$ that is not in the $\mathrm{GF}(p)$ subfield. We can write arbitrary $\beta \in \mathrm{GF}(p^2)$ as

$$\beta = a + b\alpha, \tag{8.17}$$

with $a, b \in \mathrm{GF}(p)$. We then map

$$X^a Z^b \mapsto a + b\alpha. \tag{8.18}$$

So far, so good. All is the same as for qubits. We lose the overall phase just as before. If we start with an element of $\hat{\mathsf{P}}_n(p)$, we end up with an n-component vector over $\mathrm{GF}(p^2)$ (with each element written as a 2-component vector over $\mathrm{GF}(p)$).

The complication comes in when we look at the generalization of the symplectic inner product. We wish to define it using multiplication within $\mathrm{GF}(p^2)$, and we want it to output an element of $\mathrm{GF}(p)$. The latter condition could be achieved by using the trace function (see appendix C for a discussion), but it turns out we don't want to do that in this case. For the former condition, the answer is not at all obvious, and historically took some time to find. It turns out that the inner product we want is given by the following definition:

Definition 8.2. Suppose a, b are n-dimensional vectors over $\mathrm{GF}(p^2)$. Let

$$a * b = \frac{a \cdot b^p - b \cdot a^p}{\alpha - \alpha^p}, \tag{8.19}$$

where a^p and b^p are the n-dimensional vectors whose entries are the pth power of the entries of a and b, and $\cdot$ represents the usual dot product between vectors.

Notice that the symplectic product we define (also called a *trace-alternating* inner product) depends explicitly on the specific element $\alpha \in \mathrm{GF}(p^2)$ that we used to give the mapping from $\hat{\mathsf{P}}_1(p)$ to $\mathrm{GF}(p^2)$. That wasn't the case for qubits because there were only two elements of GF(4) which are outside GF(2), namely ω and ω^2, but in $\mathrm{GF}(p^2)$ there are more choices for the pair (α, α^p).

I don't know of a better way to motivate this formula than by just calculating, but it turns out that it does work:

Theorem 8.2. *Suppose $P, Q \in \hat{\mathsf{P}}_n(p)$ correspond to vectors a, b over $\mathrm{GF}(p^2)$ according to equation (8.18). Then*

$$a * b = c(P, Q). \tag{8.20}$$

Proof. Let us calculate. We have

$$P \mapsto a = x + \alpha z \tag{8.21}$$
$$Q \mapsto b = x' + \alpha z'. \tag{8.22}$$

Now,

$$(x + \alpha z)^p = x^p + \alpha^p z^p = x + \alpha^p z, \tag{8.23}$$

since the field has characteristic p and x, z are vectors over $\mathrm{GF}(p)$. Then

$$a \cdot b^p = (x + \alpha z) \cdot (x' + \alpha z')^p \tag{8.24}$$
$$= x \cdot x' + \alpha z \cdot x' + \alpha^p x \cdot z' + \alpha^{p+1} z \cdot z' \tag{8.25}$$
$$b \cdot a^p = x \cdot x' + \alpha z' \cdot x + \alpha^p x' \cdot z + \alpha^{p+1} z \cdot z'. \tag{8.26}$$

Thus,

$$a \cdot b^p - b \cdot a^p = \alpha(z \cdot x' - x \cdot z') + \alpha^p (x \cdot z' - z \cdot x') \tag{8.27}$$
$$= (\alpha - \alpha^p)(x|z) \odot (x'|z') \tag{8.28}$$
$$= (\alpha - \alpha^p) c(P, Q). \tag{8.29}$$

□

8.1.2 Qudit Pauli Group for Prime Power Dimensions

The next most complicated case is when each qudit has dimension $q = p^m$, with p a prime. For these dimensions, the most straightforward thing to do is to let $\mathsf{P}_n(q) = \mathsf{P}_m(p)^{\otimes n}$. In other words, we consider each q-dimensional qudit as broken up into m p-dimensional qudits.

That's fine, as far as it goes, but we'd like to put more structure on the Hilbert space. In particular, we'd like to consider each q-dimensional qudit as an element of $\mathrm{GF}(q)$. To fully take advantage of that structure, we'd like to also involve $\mathrm{GF}(q)$ in our understanding of $\mathsf{P}_n(q)$. We can do so as follows:

Definition 8.3. Suppose $\beta \in \mathrm{GF}(q)$. Let X^β and Z^β be defined as

$$X^\beta |\gamma\rangle = |\gamma + \beta\rangle \tag{8.30}$$
$$Z^\beta |\gamma\rangle = \omega^{\mathrm{tr}(\beta\gamma)} |\gamma\rangle, \tag{8.31}$$

where $\gamma \in \mathrm{GF}(q)$, $\omega = \exp(2\pi i/p)$ is a pth root of unity and tr is the $\mathrm{GF}(q)$ trace function which maps elements of $\mathrm{GF}(q)$ to elements of $\mathrm{GF}(p)$. Then $\mathsf{P}_n(q)$ consists of elements of the form

$$\eta\omega^a \bigotimes_{j=1}^{n} X^{\beta_j} Z^{\gamma_j}, \tag{8.32}$$

where $a \in \mathrm{GF}(p)$, $\beta_j, \gamma_j \in \mathrm{GF}(q)$. For odd q, η is always 1. For even q, η can be 1 or i. As usual, $\hat{\mathsf{P}}_n(q) = \mathsf{P}_n(q)/\{\omega^a I\}$ (for odd p) or $\hat{\mathsf{P}}_n(q) = \mathsf{P}_n(q)/\{i^a I\}$ (for $p = 2$).

Note that

$$Z^\gamma X^\beta = \omega^{\mathrm{tr}(\gamma\beta)} X^\beta Z^\gamma. \tag{8.33}$$

Now let us examine how this definition plays out given a particular way of breaking up each q-dimensional register into m p-dimensional ones. Suppose we consider $\mathrm{GF}(q)$ as an m-dimensional vector space over $\mathrm{GF}(p)$, so

$$\gamma = \sum_{i=0}^{m-1} a_i \alpha_i, \tag{8.34}$$

with $\gamma \in \mathrm{GF}(q)$, $a_i \in \mathrm{GF}(p)$, and the α_i's elements of $\mathrm{GF}(q)$ which are linearly independent vectors when considered over $\mathrm{GF}(p)$. It is most common to take $\alpha_i = \alpha^i$, with α some primitive element of $\mathrm{GF}(q)$. Note that with this choice, $\alpha_0 = 1$.

We can immediately see how equation (8.34) corresponds to breaking the qudit up into pieces: $|\gamma\rangle \leftrightarrow |a_0\rangle \otimes |a_1\rangle \otimes \cdots \otimes |a_{m-1}\rangle$. X^{α_i} then has a natural interpretation as the p-dimensional X applied to the ith tensor factor:

$$X^{\alpha_i}|\gamma\rangle = |\gamma + \alpha_i\rangle = |\sum_j (a_j + \delta_{ij})\alpha_i\rangle. \tag{8.35}$$

If we apply $X^{\sum b_i \alpha_i}$ instead, that corresponds to performing X^{b_i} in the ith tensor factor. That is, to understand the action of X^β, we expand β in the same basis used for the standard basis decomposition, and apply the appropriate power of X on each factor.

The interpretation of Z^β is a little trickier. It hinges on the notion of a *dual basis* (sometimes called a *complementary basis*): The set $\{\alpha_i\}$ forms a basis for $\mathrm{GF}(q)$ considered as an m-dimensional vector space over $\mathrm{GF}(p)$, and for any basis, there exists a dual basis $\{\beta_j\}$ with the property:

$$\mathrm{tr}(\alpha_i \beta_j) = \delta_{ij}. \tag{8.36}$$

Then

$$Z^{\beta_j}|\gamma\rangle = \omega^{\mathrm{tr}(\beta_j \sum_i a_i \alpha_i)} \bigotimes_{i=0}^{m-1} |a_i\rangle = \omega^{a_j} \bigotimes_{i=0}^{m-1} |a_i\rangle, \tag{8.37}$$

since trace is linear over $\mathrm{GF}(p)$. That is, Z^{β_j} corresponds to performing the p-dimensional Z on the jth tensor factor. To understand the action of Z^β in general, we simply then expand β in the dual basis to the one used for the standard basis. The choice of the dual bases $\{\alpha_i\}$ and $\{\beta_j\}$ thus specifies an isomorphism $\mathsf{P}_n(q) \cong \mathsf{P}_{mn}(p)$.

For some fields (and in particular for $q = 2^m$, which is the most interesting case), it is possible to simplify the decomposition by choosing the basis $\{\alpha_i\}$ to be *self-dual*, i.e., $\beta_i = \alpha_i$. In that case, X^{α_i} and Z^{α_i} simply represent the Pauli matrices acting on the ith tensor factor of the q-dimensional register. Not all finite fields have self-dual bases, unfortunately, so for some values of q, we have to pick different decompositions for the exponents of X and Z. Alternatively, we could abandon definition 8.3, but the advantages of that notation greatly outweigh the inconvenience of having a slightly more complicated decomposition into p-dimensional qudits.

For either odd or even characteristic, when we drop phases from the Pauli group, we get vectors on a symplectic space:

$$P = \bigotimes_{j=1}^{n} X^{\eta_j} Z^{\gamma_j} \mapsto v_P = (x_P|z_P), \tag{8.38}$$

with x_P an n-dimensional vector over GF(q) with entries η_j and z_P an n-dimensional vector with entries γ_j. The symplectic inner product between v_P and v_Q (with $Q = \bigotimes X^{\eta'_j} Z^{\gamma'_j}$) is

$$v_P \odot v_Q = \sum_j \mathrm{tr}(\gamma_j \eta'_j - \eta_j \gamma'_j). \tag{8.39}$$

Multiplication now is the GF(q) multiplication rule, and we take the trace to end up with an element of GF(p). Once more proposition 8.1 applies.

You probably can guess the next step: We wish to map the q-dimensional Pauli group into GF(q^2). The procedure is similar to that for GF(p); we pick an element $\alpha \in \mathrm{GF}(q^2) \setminus \mathrm{GF}(q)$, and map

$$(x|z) \mapsto x + \alpha z. \tag{8.40}$$

The correct symplectic inner product in this case is

$$a * b = \mathrm{tr}_{q/p}\left(\frac{a \cdot b^q - b \cdot a^q}{\alpha - \alpha^q}\right). \tag{8.41}$$

There are two differences from equation (8.19): We use the exponent q instead of p, and we use the trace function to give us an element of GF(p) for the answer. Note that we are using the trace of GF(q) over GF(p), *not* the trace of GF(q^2). This is because the term in parentheses already gives an element of GF(q).

Theorem 8.3. *Suppose $P, Q \in \hat{\mathsf{P}}_n(q)$ correspond to vectors a, b over* GF(q^2) *according to equation (8.40). Then*

$$a * b = c(P, Q). \tag{8.42}$$

The proof is essentially the same as theorem 8.2. We just need to add one final step where we take the trace to get the GF(q) symplectic product.

8.1.3 Qudit Pauli Group for Other Dimensions

If the dimension q of the register is not a prime power, there are two sensible ways to generalize the above approaches to define a Pauli group. One idea is to break q up into its prime factorization $q = \prod p_i^{m_i}$, and treat each prime power factor as a separate sub-register of size $p_i^{m_i}$ with its own Pauli group.

The second approach is to directly generalize the Pauli group used for prime dimensions, by letting

$$\omega = e^{2\pi i/q} \tag{8.43}$$

$$X|j\rangle = |(j+1) \bmod q\rangle \tag{8.44}$$

$$Z|j\rangle = \omega^j |j\rangle. \tag{8.45}$$

As usual, the n-qubit Pauli group consists of products of the form $\omega^a \bigotimes X^b Z^c$ for odd q. For even q, we must include a possible overall factor of i as well. This version of the Pauli group is also known as the *Heisenberg-Weyl group*.

It is also possible to use the Heisenberg-Weyl group in place of the usual Pauli group for prime power dimensions. These groups are different: For instance, in the $q = 9$ Heisenberg-Weyl group, X has order 9, whereas all elements of the usual $q = 9$ Pauli group have order 3. There are some applications where this is a sensible thing to do, but the cost of using the Heisenberg-Weyl group is that we lose the field structure we normally have in prime power dimensions.

Because the mathematical structure of the Heisenberg-Weyl group is more complicated than the Pauli groups for prime dimension or prime-power dimension, the standard techniques of coding theory don't work as well. The basic structure of a stabilizer code still exists, but codes based on the Heisbenberg-Weyl group lack some of the usual properties of stabilizer codes.

8.1.4 Nice Error Bases

Indeed, we can generalize even further and still have a stabilizer code structure. The most important features are that the elements of our generalized Pauli group form a group, are independent, and span the set of possible errors. This is codified by the definition of a nice error basis:

Definition 8.4. In a q-dimensional Hilbert space, let $\mathcal{E} = \{E_1, \ldots, E_{q^2}\}$ be a set of unitary operators satisfying $E_1 = I$ and $\operatorname{tr} E_i^\dagger E_j = q\delta_{ij}$. The set $\mathcal{E}$ is a *nice error basis* if $E_i E_j = \omega_{ij} E_{f(i,j)}$ for all i, j and some phases ω_{ij}.

For instance, for a single qubit, the usual Pauli operators $\{E_1 = I, E_2 = X, E_3 = Y, E_4 = Z\}$ form a nice error basis, with $\omega_{ij} \in \{\pm 1, \pm i\}$. To get a generalized Pauli group from a nice error basis, we take elements of the form ωE_i, ω is a product of ω_{ij}s.

Because there are exactly q^2 independent elements in a nice error basis, they form a basis for the space of $q \times q$ matrices. In this sense, they can act like the Pauli matrices — we can take any error on the q-dimensional register and expand it as a sum of elements from the nice error basis. It is thus sufficient for a QECC to correct all errors in a nice error basis to correct arbitrary errors on the register. This justifies the "error basis" part of the name.

Proposition 8.4. *The indices i of the errors in a nice error basis form a group under multiplication given by the binary operation $f(i, j)$.*

Proof. By the definition of a nice error basis, the set of indices is closed under the group operation. Associativity follows from the associativity of operator multiplication:

$$E_i E_j E_k = \omega_{ij} E_{f(i,j)} E_k = \omega_{ij} \omega_{f(i,j)k} E_{f(f(i,j),k)} \tag{8.46}$$

$$= \omega_{jk} E_i E_{f(j,k)} = \omega_{jk} \omega_{if(j,k)} E_{f(i,f(j,k))}. \tag{8.47}$$

Since the E_is are orthogonal, it follows that $f(f(i, j), k) = f(i, f(j, k))$, the statement of associativity.

The group identity is $i = 1$ since $E_1 = I$:

$$\omega_{1i} E_{f(1,i)} = I E_i = E_i. \tag{8.48}$$

To establish the existence of inverses, first note that if $f(i, j) = f(i, j')$, then

$$E_i E_{j'} = \omega_{ij'} \omega_{ij}^{-1} E_i E_j, \tag{8.49}$$

implying $E_{j'} = \omega E_j$ for some phase ω. Since the E_is are orthogonal, it follows that $j = j'$. Therefore, the function $j \mapsto f(i, j)$ is one-to-one, and since the domain and range both have size q^2, it must be onto as well. In particular, for all i, there must exist i^{-1} such that $f(i, i^{-1}) = 1$, and i^{-1} is the inverse of i. □

Definition 8.5. The group of indices $\{i\}$ under group operation $f(i, j)$ is called the *index group* of the nice error basis.

The index group can also be obtained by taking the Pauli group generated by the nice error group and modding out overall phase. For the usual qubit Pauli group, the index group is therefore isomorphic to $\mathbb{Z}_2 \times \mathbb{Z}_2$. For the Heisenberg-Weyl group in dimension q, the index group is $\mathbb{Z}_q \times \mathbb{Z}_q$, and for the prime power Pauli group, the index group is $\mathrm{GF}(q) \times \mathrm{GF}(q)$ under addition. All of these are fairly straightforward Abelian groups, but for large q, there exist error groups with more exotic index groups, including non-Abelian ones.

8.2 Qudit Stabilizer Codes

Now we are ready to talk about actual QECCs using qudits as registers. After going to all that effort to find qudit analogs of the Pauli group, you might guess that I'm now going to define a qudit analog of stabilizer codes. If you guessed that, you'd be correct.

8.2.1 Definition and Properties of Qudit Stabilizer Codes

The basic definition of a stabilizer and a stabilizer code is the same as for qubits. When q is a non-prime-power or when we are using a nice error group more exotic than the standard $\mathsf{P}_n(q)$, it is still possible to define a stabilizer code, but there are some complications to the theory. I'll just stick to the simpler cases.

The definition of a stabilizer is the same as for qubits:

Definition 8.6. Let $q = p^m$ be a prime power, $\mathsf{P} = \mathsf{P}_n(q)$, and let Q be a subspace of the Hilbert space $\mathcal{H}_q^{\otimes n}$ (i.e., consisting of n qudits). The GF(q) *stabilizer of* Q is the set

$$\mathsf{S}(Q) = \{M \in \mathsf{P} || \psi\rangle \text{ is an eigenvector of } M \text{ with eigenvalue } +1 \ \forall |\psi\rangle \in Q\}. \tag{8.50}$$

Let S be a subgroup of P. We say S is a GF(q) *stabilizer*, or just stabilizer when the GF(q) is clear from context, if it is Abelian and if $e^{i\phi} I \notin \mathsf{S}$ for any phase $\phi \neq 0$. The *code space* of a GF(q) stabilizer S is the subspace

$$\mathcal{T}(\mathsf{S}) = \{|\psi\rangle \text{ s.t. } M|\psi\rangle = |\psi\rangle \ \forall M \in \mathsf{S}\}. \tag{8.51}$$

The code Q is a GF(q) *stabilizer code* iff $Q = \mathcal{T}(\mathsf{S}(Q))$. The *normalizer* $\mathsf{N}(\mathsf{S})$ of the stabilizer S is

$$\mathsf{N}(\mathsf{S}) = \{N \in \mathsf{P} | NM = MN \ \forall M \in \mathsf{S}\}. \tag{8.52}$$

The only real differences from the qubit definition are the use of $\mathsf{P}_n(q)$ instead of P_n and forbidding $e^{i\phi}$ for all non-zero phases ϕ, which is needed because there are more phases in $\mathsf{P}_n(q)$ than just $\pm i, \pm 1$.

Note that a GF(q) stabilizer automatically gives a GF(p) stabilizer when $q = p^m$ when we reinterpret $\mathsf{P}_n(q)$ as $\mathsf{P}_{mn}(p)$ as discussed insection 8.1.2. Indeed, using the same isomorphism, we can interpret a GF(p) stabilizer on mn qudits as a GF(p^m) stabilizer on n qudits. The difference between them is the definition of weight (and thus distance of a code), which counts the number of non-trivial p-dimensional qudits in an operator for a GF(p) stabilizer but the number of non-trivial q-dimensional qudits for a GF(q) stabilizer. Thus, a Pauli which has weight t for GF(q) might have weight up to mt for GF(p), but there are also Paulis which have the same weight for GF(p) and GF(q).

Usually, although this is not required by definition 8.6, we deal with GF(q) stabilizer codes that have an additional property:

Definition 8.7. Let $P \in \mathsf{P}_n(q)$ have GF(q) symplectic representation $(x_P|z_P)$. Let S be a GF(q) stabilizer, with symplectic representation $\hat{\mathsf{S}}$. Then we say S is a *true* GF(q) *stabilizer* if $(x_P|z_P) \in \hat{\mathsf{S}}$ implies $(\gamma x_P|\gamma z_P) \in \hat{\mathsf{S}}$ as well for all $\gamma \in$ GF(q).

That is, for a true GF(q) stabilizer code, the symplectic representation of the stabilizer is a GF(q)-linear space. Note that if q is prime, any GF(q) stabilizer code is a true GF(q) stabilizer code since $P \in \mathsf{S}$ implies $P^i \in \mathsf{S}$ as well, and

$$(x_{P^i}|z_{P^i}) = i(x_P|z_P). \tag{8.53}$$

However, when $q = p^m$ with $m > 1$, then there are some elements of GF(q) which are not numbers, so it is possible to have stabilizer codes which are not true GF(q) stabilizer codes.

GF(q) stabilizer codes have the same properties we are familiar with from qubit stabilizer codes.

Proposition 8.5. *If Q is a non-trivial subspace of the Hilbert space, then $\mathsf{S}(Q)$ is a* GF(q) *stabilizer (not necessarily a true* GF(q) *stabilizer). If S is a* GF(q) *stabilizer, then $\mathsf{S}(\mathcal{T}(\mathsf{S})) = \mathsf{S}$.*

The proof is almost identical to the qubit case. For prime qudit dimension p, we also get analogues of proposition 3.3 and theorem 3.4.

Theorem 8.6. *Let p be prime and let S be a* GF(p) *stabilizer for the code $\mathcal{T}(\mathsf{S})$, which has n physical qudits. If $|\mathsf{S}| = p^r$ (i.e., S has r generators), then $\dim \mathcal{T}(\mathsf{S}) = p^{n-r}$, so $\mathcal{T}(\mathsf{S})$ encodes $k = n - r$ physical qudits. The set of undetectable errors for S is $\hat{\mathsf{N}}(\mathsf{S}) \setminus \hat{\mathsf{S}}$. The distance of S is* $\min\{\text{wt}\, E | E \in \hat{\mathsf{N}}(\mathsf{S}) \setminus \hat{\mathsf{S}}\}$.

The proofs are closely analogous to the qubit case, so I omit the details. The main notable difference is that $\frac{1}{2}(I+M)$ is *not* the projection operator on the $+1$ eigenspace of M. Instead, the projector on the $+1$ eigenspace of M is $\frac{1}{p}\sum_{j=0}^{p-1} M^j$. The projector on the codespace can be written as

$$\Pi_{\mathsf{S}} = \frac{1}{p^r} \sum_{M \in \mathsf{S}} M. \tag{8.54}$$

(Note that the normalization is p^{-r} instead of 2^{-r}.) The other difference is that in the case where the error $E \notin \mathsf{N}(\mathsf{S})$, there is a phase ω^a instead of -1, but that does not really alter the proof.

For prime power dimensions, there is a complication due to the fact that all Paulis have order p rather than q. It is most helpful to think of a GF(q) staibilizer of n qudits as a GF(p) stabilizer on mn qudits, in which case we can apply theorem 8.6. Considered as a GF(p) stabilizer, a stabilizer with r generators has p^r elements and encodes a p^{mn-r}-dimensional Hilbert space, the straightforward analog of the qubit result. If you insist on thinking of it as a GF(q) stabilizer, the same stabilizer still has r generators and p^r elements and encodes a Hilbert space of dimension $p^{mn-r} = q^{n-r/m}$ (it is, after all, the same code). In particular, a GF(q) stabilizer does not need to encode an integer number of q-dimensional qudits. However, a *true* GF(q) stabilizer code always has a number of generators which is a multiple of m, so does encode an integer number of q-dimensional qudits.

The other subtlety in prime power dimensions is the definition of distance, and indeed, this is the only property for which it really makes a difference whether we think of the code as a GF(p) code or a GF(q) code. It is still the case, of course, that the set of undetectable errors is $\hat{\mathsf{N}}(\mathsf{S}) \setminus \hat{\mathsf{S}}$. The distance is again the minimum weight of a Pauli in $\hat{\mathsf{N}}(\mathsf{S}) \setminus \hat{\mathsf{S}}$. However, when we think of it is a GF(q) code, we should use weight as defined by the decomposition into q-dimensional qudits (i.e., the number of q-dimensional registers with non-trivial Paulis). If we instead want to think of the code as a GF(p) code, the weight would be the number of non-trivial Paulis in the decomposition into p-dimensional qudits. The GF(p) weight of a Pauli could be equal to the GF(q) weight, but it could also be as high as m times the GF(q) weight. Also note that the lowest-weight operator in $\hat{\mathsf{N}}(\mathsf{S}) \setminus \hat{\mathsf{S}}$ using the GF(q) weight might even be a different operator than the lowest-weight operator using the GF(p) weight.

Notation 8.8. A stabilizer code with n physical qudits of dimension q, k logical qudits (also of dimension q), and distance d is denoted as an $[[n,k,d]]_q$ code.

Compare the notation $((n,K,d))_q$ for a qudit code that is not necessarily a stabilizer code and $[[n,k,d]]$ for a qubit stabilizer code. This way of listing the properties of a qudit stabilizer code is an obvious hybrid.

8.2.2 Examples: Distance 2 Code, 5-Qudit Code

Let us start to explore the world of qudit stabilizer codes by looking at qudit versions of some specific qubit codes. First, we can derive distance 2 codes. Again, we will choose one generator to detect Z errors on any qudit and one generator to detect X errors on any qudit. For prime qudit dimension p, this leads to a stabilizer of the form

$$M_1 = \bigotimes_{i=1}^{n} X^{a_i} \tag{8.55}$$

$$M_2 = \bigotimes_{i=1}^{n} Z^{b_i}. \tag{8.56}$$

We need the generators to commute, so we must pick the exponents so that $\sum a_i b_i = 0$ using GF(p) arithmetic. For instance, when n is even, it suffices if $a_i = 1$ (for all i) and $b_i = 1$ for half of the values of i and $b_i = -1$ for the other half. Note that it does *not* work to take all powers equal to 1 unless $n = 0 \bmod p$. It has distance 2 if all a_i and b_i are non-zero because

$$c(X_i^c Z_i^d, M_1) = d a_i \tag{8.57}$$

$$c(X_i^c Z_i^d, M_2) = -c b_i. \tag{8.58}$$

$M_1 =$	X	Z	Z^{-1}	X^{-1}	I
$M_2 =$	I	X	Z	Z^{-1}	X^{-1}
$M_3 =$	X^{-1}	I	X	Z	Z^{-1}
$M_4 =$	Z^{-1}	X^{-1}	I	X	Z

Table 8.1: The generators for the five-qudit code.

On the other hand, $X_1^{b_2} \otimes X_2^{-b_1}$ commutes with both generators, so we can see that the code is only distance 2. Notice also that, whereas for qubits, the smallest distance 2 code has 4 physical qubits, for larger qudits, it is possible to do it with just 3 qudits. For instance, for $p = 3$ (qutrits), there is the straightforward $[[3, 1, 2]]_3$ code with generators $X \otimes X \otimes X$ and $Z \otimes Z \otimes Z$.

For prime power qudits, the above construction does not give a distance 2 code. For instance,

$$c(Z_i^\delta, M_1) = \operatorname{tr} \delta a_i, \tag{8.59}$$

and for any a_i, there exists a γ for which $\operatorname{tr} \gamma a_i \neq 0$. The problem here is that X on a single dimension-q qudit does not detect Z^δ for all δ, basically because a single qudit is really m separate p-dimensional qudits. No matter what power X^α we choose, we will have the same problem, since there will be some decomposition (not necessarily the standard one) which will lead X^α to act only on a single p-dimensional tensor factor of the full q-dimensional qudit.

Luckily, we can easily fix this by making the minimal modification needed to get a true GF(q) stabilizer code, repeating the same operators on all the tensor factors of a qudit. Let S be the smallest stabilizer containing

$$M_1(\gamma) = \bigotimes_{i=1}^{n} X^{\gamma\alpha_i} \tag{8.60}$$

$$M_2(\gamma) = \bigotimes_{i=1}^{n} Z^{\gamma\beta_i}, \tag{8.61}$$

for all γ and any particular choice of (α_i, β_i) such that $\sum \alpha_i\beta_i = 0$ in GF(q). Note that $M_1(1)$ and $M_2(1)$ commute if $\sum \operatorname{tr}(\alpha_i\beta_i) = 0$, but that is not sufficient to make sure that all $M_1(\gamma)$ commute with all $M_2(\gamma)$. With these extra elements added to the stabilizer, the code is now distance 2. For instance, consider again the example Z_i^δ:

$$c(Z_i^\delta, M_1(\gamma)) = \operatorname{tr}(\delta\gamma\alpha_i). \tag{8.62}$$

While this will certainly be 0 for some specific γ, $\operatorname{tr}(\delta\gamma\alpha_i)$ can only be 0 for all γ if $\delta\alpha_i = 0$ in GF(q).

The operators $M_1(\gamma)$ are not independent for all γ. Since the number of generators for S is best determined by thinking of it as a GF(p) code, we should represent each γ as an m-dimensional vector over GF(p). (It is m-dimensional since $q = p^m$.) It's not hard to see that $M_1(\gamma)M_1(\eta) = M_1(\gamma+\eta)$ and $[M_1(\gamma)]^a = M_1(a\gamma)$ for $a \in$ GF(p). A set $\{M_1(\gamma)\}$ is thus independent for a subset of possible γ's if the corresponding GF(p) vectors are linearly independent, which means that S has a total of m generators of the form $M_1(\gamma)$. Similarly, it has m generators of the form $M_2(\gamma)$. When we go back to thinking of it as a GF(q) code, we get an $[[n, n-2, 2]]_q$ code. As a GF(p) code, this would be a $[[mn, m(n-2), 2]]_p$ code; this is a case where the distance is the same over GF(p) and GF(q).

For the next example, I will look at a 5-qudit code which is the qudit generalization of the 5-qubit code. Again beginning with prime dimension, consider the stabilizer given in table 8.1. You can check directly that it is Abelian. It is a bit harder (though still not that hard) to see that it has distance 3, but trust me, it does. Thus, this is a $[[5, 1, 3]]_p$ code. We could also have used the same stabilizer as for the qubit version of the code (table 3.2), but this version has the minor advantage that it is cyclic, just like the 5-qubit code, whereas table 3.2 would give us a non-cyclic $[[5, 1, 3]]_p$ code. While the 5-qubit code is unique up to a tensor product of single-qubit unitary rotations, there are multiple inequivalent 5-qudit codes for qudit dimension

$p \geq 3$. Another difference is that the 5-qubit code is perfect (number of errors equals number of syndromes), whereas the 5-qudit codes are not (more syndromes than single-qudit errors).

Moving to prime power qudits, this 5-qudit code has the same problem as the distance 2 codes — it does not detect or correct errors on the additional tensor factors of a qudit. We can solve it in the same way. Add to the stabilizer all operators of the form $M_i(\gamma)$, with each Pauli raised to the power $\pm\gamma$ instead of ± 1. For instance, $M_1(\gamma) = X^\gamma \otimes Z^\gamma \otimes Z^{-\gamma} \otimes X^{-\gamma} \otimes I$. The result is a true GF(q) stabilizer code with parameters $[[5, 1, 3]]_q$.

8.3 Qudit CSS Codes

We can use some of the same methods to make qudit codes that we used for qubit codes. As discussed in section 8.1, we can map the q-dimensional qudit Pauli group (for either prime or prime power dimension) to vectors over GF(q^2). This lets us interpret qudit stabilizer codes as GF(q^2) additive codes using the symplectic inner product (8.19) or (8.41) to determine commutation. Note that GF(q^2) linear codes which are weakly self-dual under the sympletic inner product are equivalent to true GF(q) stabilizer codes with an additional symmetry, just as linear GF(4) codes give qubit stabilizer codes with an extra symmetry (see exercise **??**).

Alternatively, by writing Paulis in the GF(q) symplectic form (an n-qudit Pauli written as a $2n$-component vector over GF(q)), we can use the CSS construction. You'll notice that the example distance 2 codes in the last section had some generators that were all Z's and some generators that were all X's, whereas the 5-qudit code had a mix of X and Z in each generator. This is because those distance 2 codes are qudit CSS codes, whereas the 5-qudit code is not.

8.3.1 The CSS Construction for Qudits

For prime dimension, the CSS construction is basically the same as for qubits. Generate a stabilizer code of the form

$$\left(\begin{array}{c|c} 0 & H_1 \\ H_2 & 0 \end{array}\right), \tag{8.63}$$

from the parity check matrices of two classical linear codes C_1 and C_2. In order to define a stabilizer code, the stabilizer must commute, so use the symplectic inner product (8.16) to test that. Again we find the condition that, if $x \in C_2^\perp$ and $z \in C_1^\perp$, $x \cdot z = 0$, so in order to get a valid stabilizer code, it must be the case that ${C_1}^\perp \subseteq C_2$. The distance and number of encoded qudits are given by the same formulas as for qubits. The only difference from the qubit case is that the arithmetic to determine commutation is mod p instead of mod 2.

For prime power qudits, we'd like to do the same construction, but we must be a bit more careful. Given two classical linear GF(q) codes C_1 and C_2, let's define the stabilizer S to be the smallest group containing all $\bigotimes_j Z^{\gamma_j}$ and $\bigotimes_j X^{\eta_j}$, where γ runs over elements of $C_1^\perp$ and η runs over elements of $C_2^\perp$. Note that it is *not* sufficient to look at the group generated in this way for basis vectors γ and η of $C_1^\perp$ and $C_2^\perp$. This is because if $\gamma \in C_1^\perp$, then GF(q) linearity implies that $\xi\gamma \in C_1^\perp$ as well for any $\xi \in$ GF(q), but $\bigotimes_j Z^{\gamma_j} \in \mathsf{S}$ is not sufficient to imply that $\bigotimes_j Z^{\xi\gamma_j} \in \mathsf{S}$. This contrasts with the GF(p) case, where

$$\bigotimes_j Z^{ab_j} = \Big(\bigotimes_j Z^{b_j}\Big)^a, \tag{8.64}$$

and since S must be closed under multiplication, it must also be closed under integer exponentiation. Exponentiation by $\xi \in$ GF(q) doesn't make any sense, and this is responsible for the difference between GF(p) and GF(q) stabilizer codes (including CSS codes).

Theorem 8.7. *Let C_1 and C_2 be two classical linear codes over* GF(q) *with parameters $[n, k_1, d_1]_q$ and $[n, k_2, d_2]_q$ and satisfying $C_1^\perp \subseteq C_2$. Then there exists a true* GF(q) *stabilizer code with stabilizer given as above with parameters $[[n, k_1 + k_2 - n, d]]_q$, $d \geq \min(d_1, d_2)$.*

Proof. The first consideration is whether the stabilizer given above is well-defined. We need to check that $M = \bigotimes_j Z^{\gamma_j}$ and $N = \bigotimes_j X^{\eta_j}$ commute when $\gamma \in C_1^\perp$ and $\eta \in C_2^\perp$. By equation (8.39),

$$c(M, N) = \sum_j \operatorname{tr} \gamma_j \eta_j = \operatorname{tr} \gamma \cdot \eta \tag{8.65}$$

(using the GF(q) dot product). Since $C_1^\perp \subseteq C_2$, $\gamma \in C_2$, so $\gamma \cdot \eta = 0$, and $c(M, N) = 0$ as desired.

Therefore, we have a well-defined GF(q) stabilizer code. The elements of the stabilizer have symplectic representations of the form $(\eta|\gamma)$ for $\eta \in C_2^\perp$ and $\gamma \in C_1^\perp$. Since C_1 and C_2 are GF(q) linear, so are $C_1^\perp$ and $C_2^\perp$. Thus, the code is a true GF(q) stabilizer code.

The next question is to determine how many logical qudits there are in this code. Each basis vector of $C_1^\perp$ or $C_2^\perp$ gives us m independent elements of S ($q = p^m$ as usual) for the reasons noted above (exponentiation by $\xi \in$ GF(q) does not make sense). Thus, there are $m(n - k_1)$ generators of S derived from C_1 and $m(n - k_2)$ generators derived from C_2. Thought of as a GF(p) code, the number of logical qudits is thus $mn - m(n - k_1) - m(n - k_2) = m(k_1 + k_2 - n)$. Thought of as a GF($q$) code again, we have $k_1 + k_2 - n$ logical qudits.

Finally, the distance can be determined just as for a usual binary CSS code. □

It might appear at first sight that it is possible to have GF(q) CSS codes that don't satisfy the condition $C_1^\perp \subseteq C_2$ since we actually only need $\operatorname{tr} \gamma \cdot \eta = 0$ for all γ and η. However, because C_1 and C_2 are *linear* GF(q) codes, $\operatorname{tr} \gamma \cdot \eta = 0$ for all γ and η iff $\gamma \cdot \eta = 0$ is.

As with binary CSS codes, the basis codewords have a straightforward form when written out in the standard basis:

$$|\gamma + {C_2}^\perp\rangle = \sum_{\eta \in {C_2}^\perp} |\gamma + \eta\rangle, \tag{8.66}$$

for $\gamma \in C_1$. This works for both prime dimension and prime power dimension. Indeed, we could just take it as the definition of a CSS code in any dimension.

8.3.2 Polynomial Codes

One particularly interesting family of qudit CSS codes is the family of *polynomial codes*, which are CSS codes derived from classical Reed-Solomon codes or their variants. Let us pick the code C_1 as a Reed-Solomon code by choosing n distinct points $(\alpha_1, \ldots, \alpha_n)$ from GF(q) $\setminus \{0\}$ ($n < q$) and a number $k_1 \leq n$. We'll use polynomials with degree up to $k_1 - 1$. We also pick a second number $0 \leq k_2 \leq k_1$ which will determine the size of C_2.

Definition 8.9. The basis codewords of the polynomial code over GF(q) with (n, k_1, k_2) are

$$|\overline{\beta_0, \ldots, \beta_{k_2-1}}\rangle = \sum_{(\beta_{k_2}, \ldots, \beta_{k_1-1}) \in \mathrm{GF}(q)} \bigotimes_{i=1}^{n} |\beta_0 + \beta_1 \alpha_i + \beta_2 \alpha_i^2 + \cdots + \beta_{k_1-1} \alpha_i^{k_1-1}\rangle. \tag{8.67}$$

The polynomial code is the span of these basis codewords.

That is, the basis codewords are indexed by k_2 values, the lowest-order coefficients of the polynomials being used. We then take the superposition over polynomials for all possible coefficients of x^{k_2} and higher, up to the maximum degree x^{k_1-1}. A particularly interesting special case is when $k_2 = 1$ so there is just one encoded qudit. Note that if $k_2 = 0$, the polynomial code can still be defined, but it is only a single state.

Clearly a polynomial code is a qudit CSS code, since the basis codewords have the correct form. However, instead of choosing both C_1 and C_2 to be Reed-Solomon codes (which wouldn't work, since they are not precisely dual to each other), we have chosen ${C_2}^\perp$ to be a modified Reed-Solomon code. In particular, ${C_2}^\perp$ is the code made up of vectors $(f(\alpha_1), \ldots, f(\alpha_n))$, where $f(x)$ runs over degree $k_1 - 1$ or lower polynomials for which the lowest nonzero term is the x^{k_2} power or higher. With this choice, it is manifestly true that ${C_2}^\perp \subseteq C_1$ and that the code encodes k_2 qudits. What is not clear is the distance of the resulting polynomial code.

Theorem 8.8. *The* GF(q) *polynomial code with parameters* (n, k_1, k_2) *is a non-degenerate true* $[[n, k_2, d]]_q$ *stabilizer code with* $d = \min(n - k_1 + 1, k_1 - k_2 + 1)$.

Proof. The code C_1 is used to correct bit flip errors. By theorem 4.15, the distance of C_1 is $n - k_1 + 1$. To prove the formula for distance, we thus need to show that C_2 has distance $k_1 - k_2 + 1$ and that the code is non-degenerate. The fact that it is a true GF(q) code follows from theorem 8.7.

The dual code ${C_2}^\perp$ has basis vectors $(\alpha_1^j, \ldots, \alpha_n^j)$ for $j = (k_2, \ldots, k_1 - 1)$. These form the rows of the parity check matrix H_2. A vector orthogonal to all rows of H_2 (i.e., a vector in C_2) corresponds to a linear dependence among the columns of H_2, so the distance of C_2 is the minimum number of columns of H_2 that are linearly dependent. There are $k_1 - k_2$ rows, so certainly no more than $k_1 - k_2$ columns can be linearly independent.

Let us look at the matrix formed by any set of $k = k_1 - k_2$ columns, say the first k. The matrix entries are $V_{ij} = \alpha_i^{k_2+j-1}$. This is not a Vandermonde matrix, but is clearly closely related. The columns of the matrix are linearly independent iff the rows are, and we can think of a linear combination of the rows as a polynomial with degree $k_1 - 1$ and all coefficients below degree k_2 being 0; that is, an element of ${C_2}^\perp$. The ith entry of the linear combination is the polynomial evaluated at α_i. A linear dependence of the rows is thus a polynomial that evaluates to 0 on all the points $\alpha_1, \ldots, \alpha_k$. Because the lowest degree term of the polynomial is x^{k_2}, the polynomial also has a 0 at $x = 0$ with multiplicity k_2. That gives a total of $k_2 + k = k_1$ zeros for the polynomial, which is too many for a degree $k_1 - 1$ polynomial; thus, for this to be true, the polynomial must be uniformly 0 everywhere (not just on $\alpha_1, \ldots, \alpha_k$). In other words, there is no linear dependence of the rows and the matrix V_{ij} is non-singular. Thus, any k columns are independent. Since any $k + 1$ columns are linearly dependent, the code C_2 has distance $k + 1$.

The only remaining thing to show is that the code is non-degenerate, from which it follows that it has distance exactly $\min(n - k_1 + 1, k_1 - k_2 + 1)$ and not a greater distance. We wish to show that there exists some non-trivial logical Pauli that has this weight. I will show that there is a logical $\overline{X}$ with weight exactly $n - k_1 + 1$ and a logical $\overline{Z}$ with weight exactly $k_1 - k_2 + 1$.

For a CSS code (over either qubits or qudits), the logical $\overline{X}$ operators can be taken to be products of physical Xs, $\overline{X} = \bigotimes X^{\eta_i}$. Moreover, the vectors (η_i) must be in C_1 to commute with the Z stabilizer generators. Since the distance of C_1 is exactly $n - k_1 + 1$, there is a vector of this weight. That is, there is a non-trivial polynomial f of degree $k_1 - 1$ or less such that $\eta_i = f(\alpha_i)$ is 0 for exactly $k_1 - 1$ values of α_i. Thus $\operatorname{wt} \overline{X} = n - k_1 + 1$. But since f has degree $k_1 - 1$, if it has more than $k_1 - 1$ zeros, then it will be uniformly zero. In particular, $f(0) \neq 0$. Now,

$$\overline{X}|\overline{0}\rangle = \sum_{g \in {C_2}^\perp} \bigotimes_{i=1}^{n} |(f + g)(\alpha_i)\rangle. \tag{8.68}$$

But the x^0 coefficient of g is 0 (since $g \in {C_2}^\perp$) and the x^0 coefficient of $f + g$ is *not* zero, so $\overline{X}|\overline{0}\rangle \neq |\overline{0}\rangle$ and $\overline{X} \notin \mathsf{S}$.

Similarly, the logical $\overline{Z}$ operators are products of physical Zs, $\overline{Z} = \bigotimes_i Z^{\gamma_i}$. The vectors (γ_i) must be in C_2, which means there is such a vector with weight exactly $k_1 - k_2 + 1$. This gives us $\overline{Z}$ with $\operatorname{wt} \overline{Z} = k_1 - k_2 + 1$. Now,

$$\overline{Z}|\overline{f}\rangle = \overline{Z} \sum_{g \in {C_2}^\perp} \bigotimes_{i=1}^{k+1} |(f + g)(\alpha_i)\rangle \tag{8.69}$$

$$= \sum_{g \in {C_2}^\perp} \omega^{\sum_i \operatorname{tr} \gamma_i (f+g)(\alpha_i)} \bigotimes_i |(f + g)(\alpha_i)\rangle \tag{8.70}$$

$$= \sum_{g \in {C_2}^\perp} \omega^{\operatorname{tr} \sum_i \gamma_i f(\alpha_i) + \operatorname{tr} \sum_i \gamma_i g(\alpha_i)} \bigotimes_i |(f + g)(\alpha_i)\rangle. \tag{8.71}$$

Since $(\gamma_i) \in C_2$ and $g \in {C_2}^\perp$, $\sum_i \gamma_i g(\alpha_i) = 0$.

But I claim there exists some $f \in C_1$ such that $\operatorname{tr} \sum_i \gamma_i f(\alpha_i) \neq 0$. We can assume without loss of generality that γ_i is non-zero only for $i = 1, \ldots, k_1 - k_2 + 1$. Consider polynomials $f_j(x) = x^j$ for $j = 0, \ldots, k_1 - k_2$. The matrix $V_{ij} = \alpha_i^j = g_j(\alpha_i)$ *is* a Vandermonde matrix, so the vectors $(f_j(\alpha_1), \ldots, f_j(\alpha_{k_1-k_2+1}))$ are linearly independent. They are vectors in a $(k_1 - k_2 + 1)$-dimensional vector space, so there is no non-zero vector that is orthogonal to all of them. In particular, the vector (γ_i) must have non-trivial overlap with at least one of the vectors $(f_j(\alpha_i))$: $\sum_i \gamma_i f_j(\alpha_i) = \xi \neq 0$. It is possible that $\operatorname{tr} \xi = 0$, but if we instead use the polynomial $f'(x) = \eta f_j(x)$, then $\sum_i \gamma_i f_j(\alpha_i) = \xi\eta$. η is arbitrary, so we just need to pick some η such that $\operatorname{tr}(\xi\eta) \neq 0$ to prove the claim.

Consequently, $\overline{Z}|\overline{0}\rangle = |\overline{0}\rangle$ but $\overline{Z}|\overline{f}'\rangle \neq |\overline{f}'\rangle$. Thus, $\overline{Z}$, which has weight $k_1 - k_2 + 1$, is a non-trivial logical operation. This proves the polynomial code is non-degenerate and thus the distance is exactly $\min(n - k_1 + 1, k_1 - k_2 + 1)$. □

If we choose the distances of the bit flip code and the phase code to be equal, then $n - k_1 = k_1 - k_2$, or $2k_1 = n - k_2$. In this case, $2d = n - k_1 + 1 + k_1 - k_2 + 1 = n - k_2 + 2$. Since k_2 is the number of encoded qudits, a polynomial code with these parameters saturates the quantum Singleton bound and is a quantum MDS code. For instance, we can have a $[[5, 1, 3]]_q$ polynomial code for $q > 5$. (There is also a variant with $q = 5$.) Note that this code is not equivalent to the $[[5, 1, 3]]_q$ stabilizer in table 8.1. This code is a CSS code, unlike the previous one, but it does not work for $q = 2$, 3, or 4, whereas the code of table 8.1 works for any prime or prime power dimension.

8.4 Qudit Clifford Group

Although I've spent a number of pages discussing qudit stabilizer codes, ultimately they are not that different than qubit stabilizer codes. They're not identical, to be sure, which is why I've gone through them in detail, but the differences are small. When we get to the qudit Clifford group, however, larger differences start to appear. The same general principles hold as for the qubit case, but there is a significant divergence at the level of details. I will focus on the case of prime dimension; for prime powers, simply consider the $q = p^m$-dimensional qudit as m p-dimensional qudits.

We start the same way as for qubits:

Definition 8.10. Let p be a prime. The *qudit Clifford group* is

$$\mathsf{C}_n(p) = \{U \in \mathsf{U}(p^n) | UPU^\dagger \in \mathsf{P}_n(p)\ \forall P \in \mathsf{P}_n(p)\}. \tag{8.72}$$

As with qubits, $\mathsf{P}_n(p)$ is a normal subgroup of $\mathsf{C}_n(p)$. We can also define

$$\hat{\mathsf{C}}_n(p) = \mathsf{C}_n(p)/\{e^{i\phi} I\} \tag{8.73}$$

$$\check{\mathsf{C}}_n(p) = \hat{\mathsf{C}}_n(p)/\hat{\mathsf{P}}_n(p). \tag{8.74}$$

It is still the case that $\check{\mathsf{C}}_n(p) \cong \mathsf{Sp}(2n, \mathbb{Z}_p)$ and that there exists a unitary in $\mathsf{C}_n(p)$ that performs any transformation of the Paulis that preserves commutation relations. It is also possible to efficiently simulate the behavior of the qudit Clifford group, including Pauli measurements, with a classical computer using essentially procedure 6.4. The main difference in the simulation is when measuring P_i with $P_i \notin \mathsf{N}(\mathsf{T})$. The measurement outcome is a uniform random value b from 0 to $p - 1$ rather than 0 or 1. We still find a generator M of the stabilizer to replace with $\omega^b P_i$, but we choose M that has a factor of ω when commuting past P_i rather than anticommuting with it. By taking the appropriate power r of M, we can ensure that NM^r commutes with P for stabilizer generators or logical operators N; we replace the original N with the appropriate NM^r.

The biggest difference with the qubit Pauli groups comes when comparing the actual elements of the Clifford group. Some of the elements are very closely analogous. For instance, the discrete Fourier transform over $\mathbb{Z}_p$ is the generalization of the Hadamard transform (which is the discrete Fourier transform over $\mathbb{Z}_2$):

$$\mathcal{F}|a\rangle = \frac{1}{\sqrt{p}} \sum_b \omega^{ab} |b\rangle. \tag{8.75}$$

Working out the conjugation action on the Paulis, we find that Fourier does not precisely swap X and Z, but does so adding an inverse:

$$X \mapsto Z \tag{8.76}$$
$$Z \mapsto X^{-1}. \tag{8.77}$$

There is also a two-qubit SUM gate which is the direct analogue of the qubit CNOT gate:

$$\text{SUM}|a\rangle|b\rangle = |a\rangle|a+b \bmod p\rangle \tag{8.78}$$
$$X \otimes I \mapsto X \otimes X \tag{8.79}$$
$$Z \otimes I \mapsto Z \otimes I \tag{8.80}$$
$$I \otimes X \mapsto I \otimes X \tag{8.81}$$
$$I \otimes Z \mapsto Z^{-1} \otimes Z. \tag{8.82}$$

Again there is an extra inverse in one of the images in the conjugation action. The inverse powers are needed to ensure that the Clifford group gate preserves commutation relations among the Paulis. For instance, $Z \otimes Z$ does not commute with $X \otimes X$, but $Z^{-1} \otimes Z$ does.

From here, though, we start to see a bigger divergence. There are operators in $\mathsf{C}_n(p)$ which don't have any qubit analog, such as scalar multiplication by $c \in \mathbb{Z}_p \setminus \{0\}$:

$$S_c|a\rangle = |ca\rangle \tag{8.83}$$
$$X \mapsto X^c \tag{8.84}$$
$$Z \mapsto Z^{c^{-1}}. \tag{8.85}$$

The power of Z is c^{-1}, the multiplicative inverse of c in $\mathbb{Z}_p$. For instance, for $p = 7$, $2^{-1} = 4$ since $2 \cdot 4 = 8 \equiv 1 \bmod 7$. There is also no precise qudit analog of $R_{\pi/4}$. The closest is a quadratic phase gate which has a similar action on Paulis:

$$B|a\rangle = \omega^{a(a-1)/2}|a\rangle \tag{8.86}$$
$$X \mapsto XZ \tag{8.87}$$
$$Z \mapsto Z. \tag{8.88}$$

Theorem 8.9. *The gates $\mathcal{F}$, B, and* SUM *along with global phase $e^{i\theta}I$ generate $\mathsf{C}_n(p)$.*

Proof. First, we can show that $\mathcal{F}$, B, SUM, S_c (for all $c \neq 0$), $\mathsf{P}_n(p)$ and $e^{i\theta}I$ generate $\mathsf{C}_n(p)$ using essentially procedure 6.5. In the algorithm, we can replace H with $\mathcal{F}$, CNOT with SUM, and $R_{\pi/4}$ with B. SWAP can be realized as

$$\text{SWAP}_{i,j} = S_{-1\,i}\text{SUM}_{i,j}\text{SUM}^{-1}{}_{j,i}\text{SUM}_{i,j}. \tag{8.89}$$

The analog of $\text{C}-\text{Z}$ is $(I \otimes \mathcal{F})\text{SUM}(I \otimes \mathcal{F}^{-1})$. The symplectic matrices of these gates are quite similar to the qubit versions, but there are some additional minus signs that appear (when the power is an inverse). In the procedure, we will have to use some of them multiple times in order to cancel out terms. For instance, in step 2, we wish to eliminate entries in the first column of A. To do so, use left multiplication by SUM $p - a_{1j}$ to eliminate entry a_{1j}. We also add in S_c, which has the effect of multiplying rows or columns by c or c^{-1}. This is useful in step 1: we can use SWAP to move a non-zero entry into the upper left corner, but we then need S_c to make that entry 1.

The next step is to show that we don't need S_c. I claim that gates of this form can be generated using only $\mathcal{F}$, B, and $\mathsf{P}_1(p)$. Let $Q = \mathcal{F}B\mathcal{F}^{-1}$. Then

$$Q : X \mapsto X \tag{8.90}$$
$$Z \mapsto ZX^{-1}. \tag{8.91}$$

The action of $B^r Q^s B^m Q^n$ is then

$$X \mapsto \omega^a X^{1-ms} Z^{m+r-rms} \tag{8.92}$$

$$Z \mapsto \omega^b X^{-n-s+mns} Z^{1-(m+r)n+(mn-1)rs} \tag{8.93}$$

The powers a and b don't matter because we can get rid of the powers of ω using an appropriate element of $\mathsf{P}_1(p)$. Let $r = -c^{-1}$, $s = 1 - c$, $m = 1$, and $n = 1 - c^{-1}$, so $rs = rms = n = m + r$. Then

$$X \mapsto X^c \tag{8.94}$$

$$Z \mapsto Z^{c^{-1}}. \tag{8.95}$$

We also don't need the Paulis. $\mathcal{F}^2 = S_{-1}$, i.e. $\mathcal{F}^2|a\rangle = |-a\rangle$. Then

$$\mathcal{F}^2 B \mathcal{F}^2 B^{-1}|a\rangle = \mathcal{F}^2 B \omega^{-a(a-1)/2}|-a\rangle = \omega^{-a(-a-1)/2-a(a-1)/2}|a\rangle = \omega^a|a\rangle. \tag{8.96}$$

Thus, $Z = \mathcal{F}^2 B \mathcal{F}^2 B^{-1}$, and $X = \mathcal{F}^{-1} Z \mathcal{F}$, which then can be used to generate the full Pauli group. □

Chapter 9

Now, What Did I Leave Out?: Other Things You Should Know About Quantum Error Correction

By now, you know (or should, if you've been paying attention) quite a lot about quantum error-correcting codes. However, there's still a lot more to learn. Some topics, such as topological codes or channel capacity, deserve a chapter or more of their own, and those will be covered in part III. However, there are plenty of other things about QECCs which I don't want to get into at length, but are still interesting or important to know. Since many of them don't fit well into the structure of the previous chapters, I've put them here. Consequently, this chapter is a grab-bag of stuff about QECCs left out of the first 8 chapters. These topics are not really dependent on each other, and only a few of them are essential to later parts of the book. Concatenated codes (section 9.1) will play an important role in the discussion of fault tolerance in part II, and the coherent information (section 9.3) will be need in chapter 18. The rest of this chapter is optional, consisting of various stand-alone topics.

9.1 Concatenated Codes

9.1.1 Basic Properties

If one quantum error-correcting code is good, two codes must be better. That's the philosophy behind a concatenated code. To make a concatenated code, take your quantum data and encode using one code Q, then take each of the physical registers of Q and encode it again using R, as depicted in figure 9.1. The resulting concatenated code generally has a greater distance than either Q or R, albeit at the cost of more physical qudits for the same number of logical qudits. If concatenating once is not enough, you can add a third layer of concatenation, or a fourth, or however many it takes to satisfy your hunger for error correction.

More precisely, a concatenated code can be defined as follows:

Definition 9.1. Let Q be an $((n_1, K, d_1))_{q_1}$ code with encoder $\mathcal{E}_1$ and R be a $((n_2, q_1, d_2))_{q_2}$ code with encoder $\mathcal{E}_2$. The *concatenated code* formed from Q and R is an $((n_1 n_2, K))_{q_2}$ code with encoder $\mathcal{E}_2^{\otimes n_1} \circ \mathcal{E}_1$. Q is known as the *inner code* and R is the *outer code*.

Notice that a concatenated code gets the size of its logical Hilbert space from Q and the size of the physical registers from R, whereas the number of registers n is the product of n_1 and n_2. The register size from Q must match the logical Hilbert space size for R so that $\mathcal{E}_2$ can be applied to each register of Q.

It is easy to compute a bound on the distance of a concatenated code:

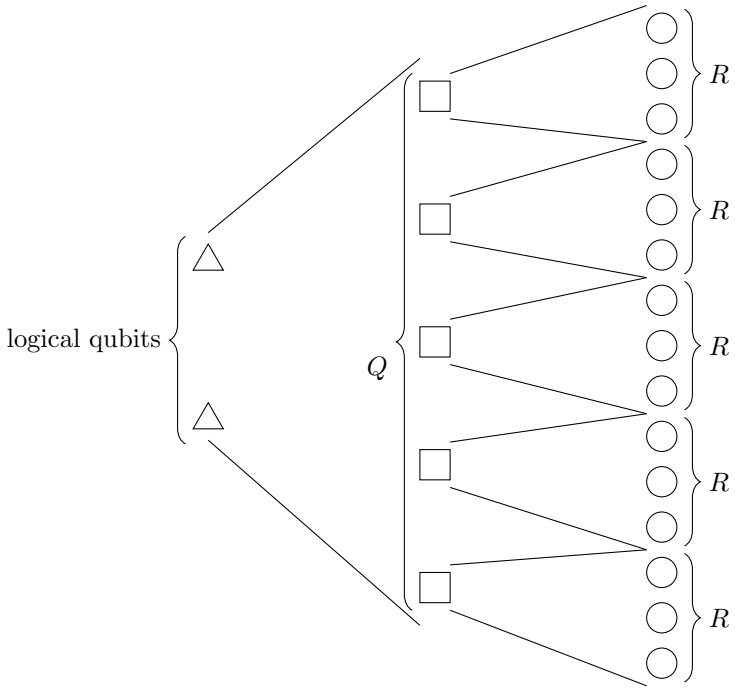


Figure 9.1: In a concatenated code, two codes are combined to produce a larger code. k_1 logical qubits (triangles) are encoded in the n_1 physical qubits of the inner code (squares), which are in turn individually encoded into the outer code, each using n_2 physical qubits (circles).

Proposition 9.1. *The concatenated code formed from a distance d_1 inner code and a distance d_2 outer code has distance $d \geq d_1 d_2$.*

Proof. Consider detecting errors with the code. If there are fewer than d_2 errors on any block of R involved in the concatenated code, we will detect the error using that that code alone. With at least d_2 errors on a single block, however, it may be possible to have an error that is undetectable by R alone. Formally, if $\operatorname{wt} E < d_2$, then $\Pi_R E|\psi\rangle = c(E)|\psi\rangle$ for all codewords $|\psi\rangle$, where Π_R is the projector on R. On the other hand, there must exist some E with $\operatorname{wt} E = d_2$ and some codeword $|\psi\rangle$, such that $\Pi_R E|\psi\rangle \neq c(E)|\psi\rangle$.

But that process only changes a single register of Q, which by itself would be detected using the properties of Q (provided $d_1 > 1$). In particular, let $F = \mathcal{D}_2 \Pi_R E$ be an error acting on a register of Q, where $\mathcal{D}_2$ is the decoder for R. From above, if $\operatorname{wt} E < d_2$, then $F \propto I$, whereas if $\operatorname{wt} E \geq d_2$, it is possible that $F \not\propto I$.

Suppose now that we have some error of weight w acting on the concatenated code as a whole. Break it up into a tensor product of errors, with E_i acting on the ith block of R. (We may assume the overall error is a tensor product or even a Pauli, by corollary 2.5.) Let $F_i = \mathcal{D}_2 \Pi_R E_i$. No matter how we divide the errors between blocks of R, there can be at most $\lfloor w/d_2 \rfloor$ blocks with d_2 or more errors per block, so at most $\lfloor w/d_2 \rfloor$ of the F_i are not proportional to the identity. That is, the error $F = \bigotimes_{i=1}^{n_1} F_i$ has weight at most $\lfloor w/d_2 \rfloor$

F is an error acting on Q when it is considered without concatenation. If $\operatorname{wt} F < d_1$, then Q by itself can detect F. In order to fool Q, therefore, we need an error acting on the concatenated code with $w/d_2 \geq d_1$. Therefore, the concatenated code can detect any error of weight $d_1 d_2 - 1$ or less, so by theorem 2.9, the distance of the concatenated code is at least $d_1 d_2$. □

Note that it is possible for the concatenated code to have distance greater than one would expect from proposition 9.1. For instance, the 9-qubit code can be thought of as the concatenation of an inner 3-qubit code correcting phase errors with an outer 3-qubit code correcting bit flip errors. Each of the 3-qubit codes by itself has distance 1 since it can only detect or correct one kind of error, but together they form a 9-qubit code with distance 3.

A common situation is when the inner code and the outer code are the same, which requires that the register size equal the encoded Hilbert space size. Therefore, we can concatenate a $((n, q, d))_q$ code with itself to get a $((n^2, q, d^2))_q$ code. We can repeat this process to get bigger and bigger codes.

Definition 9.2. Let $Q_1 = Q$ be a $((n, q, d))_q$ code, and let Q_k be the $((n^k, q, d^k))_q$ code obtained by concatenating Q_{k-1} as the inner code with Q as the outer code. We say that Q_k is a code involving k *levels* of concatenation. The physical qudits form *level* 0 qudits, the outer code for Q_k is *level* 1 of concatenation, and its logical qudits are *level* 1 qudits. The logical qudits for the outer code of the Q_ℓ that appears in the recursive definition of Q_k are *level* $k - \ell + 1$ qudits and the logical qudits for the overall code are *level* k qudits.

9.1.2 Concatenated Stabilizer Codes

If we concatenate two qubit stabilizer codes, the resulting code is a stabilizer code as well, with the stabilizer constructed in a particular way.

Procedure 9.1. Let the inner code be an $[[n_1, 1, d_1]]$ code with stabilizer S_1, and let the outer code be an $[[n_2, k, d_2]]$ code with stabilizer S_2 and logical Paulis $\overline{P}$. Then the stabilizer S of the concatenated code formed from these two codes is given as follows:

1. For each generator $M \in \mathsf{S}_2$, include in S the Pauli M_i consisting of M acting on the ith block of n_2 qubits in the code tensored with the identity on all other blocks.

2. For each generator $M \in \mathsf{S}_1$, replace each single-qubit Pauli P_i in its tensor product decomposition (i.e., acting on the ith physical qubit) with $\overline{P}_i$, the corresponding logical Pauli from the *outer* code acting on the ith block of n_2 qubits in the concatenated code. Take the tensor product of all the $\overline{P}_i$ operators for a single $M \in \mathsf{S}_1$ and include that in S.

	X	Z	Z	X	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
	I	X	Z	Z	X	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
	X	I	X	Z	Z	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
	Z	X	I	X	Z	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
	I	I	I	I	I	X	Z	Z	X	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
	I	I	I	I	I	I	X	Z	Z	X	I	I	I	I	I	I	I	I	I	I	I	I	I	I
	I	I	I	I	I	X	I	X	Z	Z	I	I	I	I	I	I	I	I	I	I	I	I	I	I
	I	I	I	I	I	Z	X	I	X	Z	I	I	I	I	I	I	I	I	I	I	I	I	I	I
	I	I	I	I	I	I	I	I	I	I	X	Z	Z	X	I	I	I	I	I	I	I	I	I	I
	I	I	I	I	I	I	I	I	I	I	I	X	Z	Z	X	I	I	I	I	I	I	I	I	I
	I	I	I	I	I	I	I	I	I	I	X	I	X	Z	Z	I	I	I	I	I	I	I	I	I
	I	I	I	I	I	I	I	I	I	I	Z	X	I	X	Z	I	I	I	I	I	I	I	I	I
	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	X	Z	Z	X	I	I	I	I	I
	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	X	Z	Z	X	I	I	I	I	I
	I	I	I	I	I	I	I	I	I	I	I	I	I	I	X	I	X	Z	Z	I	I	I	I	I
	I	I	I	I	I	I	I	I	I	I	I	I	I	I	Z	X	I	X	Z	I	I	I	I	I
	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	X	Z	Z	X	I
	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	X	Z	Z	X
	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	X	I	X	Z	Z
	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	Z	X	I	X	Z
	X	X	X	X	X	Z	Z	Z	Z	Z	Z	Z	Z	Z	X	X	X	X	X	I	I	I	I	I
	I	I	I	I	I	X	X	X	X	X	Z	Z	Z	Z	Z	Z	Z	Z	Z	X	X	X	X	X
	X	X	X	X	X	I	I	I	I	I	X	X	X	X	X	Z	Z	Z	Z	Z	Z	Z	Z	Z
	Z	Z	Z	Z	Z	X	X	X	X	X	I	I	I	I	I	X	X	X	X	X	Z	Z	Z	Z	Z
$\overline{X}$	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
$\overline{Z}$	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z

Table 9.1: The stabilizer for the $[[25, 1, 9]]$ code formed by concatenating the 5-qubit code with itself.

3. Form the group generated by the elements described in the previous steps.

One can also determine the logical Paulis for the concatenated code by the same process as in step 2: take a logical Pauli for S_1 and replace each Pauli in its tensor product decomposition with the corresponding logical Pauli for S_2.

An example of a concatenated stabilizer code is the $[[25, 1, 9]]$ code formed by concatenating the 5-qubit code with itself. The resulting stabilizer is given in table 9.1. Note that the 25-qubit code is degenerate. Indeed, a quick consideration of procedure 9.1 should convince you that concatenated stabilizer codes will frequently be degenerate: The stabilizer of a concatenated code contains many elements that are drawn from a single copy of the stabilizer of the outer code. These elements of S have weight bounded by n_2, which can easily be lower than the distance $d_1 d_2$ of the concatenated code.

Another interesting case is when the outer code is a qubit code, because then the resulting concatenated code is a qubit code as well. For instance, suppose the outer code is an $[[n_2, k_2, d_2]]_2$ code. Then the inner code should work with registers of size $q_1 = 2^{k_2}$. This is a prime power, so we are dealing with a code over $\mathrm{GF}(2^{k_2})$. For instance, we can use an $[[n_1, k_1, d_1]]_{2^{k_2}}$ code to get a $[[n_1 n_2, k_1 k_2, d_1 d_2]]_2$ code. (It has k_1 logical qudits of size 2^{k_2}, which corresponds to $k_1 k_2$ logical qubits.) This can be a useful way to find large qubit stabilizer codes with reasonable parameters.

9.1.3 Decoding Concatenated Codes

Given the structure of a concatenated code, one natural approach to decoding is to treat the inner code and the outer code separately. For each block of the outer code R, perform error correction via whichever

algorithm is appropriate to that code. Now each block of n_2 qudits is a valid codeword for R, but the logical state of each block may have been changed by the errors. If we decode each block, we are left with Q with some errors on it, and can perform the usual decoding procedure for it.

The advantage of this procedure is that it is efficient if the decoding algorithms used for Q and R are. It's also quite straightforward. The disadvantage is that it doesn't take advantage of the full error correction capability of the concatenated code. For instance, consider the 125-qubit code formed by concatenating the 25-qubit code of table 9.1 with the 5-qubit code. If a block of the outer 5-qubit code has 2 errors on it, the correction procedure on that block will fail, leaving an error on the logical qubit. However, the inner code can only correct four errors, so if five blocks of the outer code are wrong, which can happen with 10 errors in total, than this decoding procedure will make a mistake. The 125-qubit code has distance 27, so it should be able to correct 13 errors. The difference arises because when we correct the inner code and outer codes separately, we throw away some information that would be useful. For instance, if a single block of the five-qubit outer code has non-trivial syndrome, it is more likely the corresponding qubit of the inner code has an error on it. This means that if there are only two errors in any given 5-qubit block, the errors on the inner code act like *erasure* errors, allowing us to correct more of them. Of course, it's also possible that there are 3 errors on a 5-qubit code block, which could produce a logical error with no error syndrome advertising it, so the decoding process is not necessarily straightforward.

Notice that as we concatenate the same code many times, the fractional distance d/n decreases — with k levels of concatenation of an $[[n_0, 1, d_0]]$ code, $n = n_0^k$ and $d = d_0^k$. Thus, in a very meaningful sense, the code gets worse at correcting errors as we concatenate many times. However, this is only true if we insist on correcting the *worst case* error. For typical errors, the concatenated code does well. For instance, suppose we consider a depolarizing channel with total error probability p for each qubit (i.e., $p/3$ chance of each of X, Y, and Z errors). The probability that a single block of the $[[n_0, 1, d_0]]$ code fails and has a logical error is roughly

$$p_1 \approx \binom{n_0}{t+1} p^{t+1} = p_T (p/p_T)^{t+1}, \tag{9.1}$$

where $t = \lfloor (d-1)/2 \rfloor$, $p_T = \binom{n_0}{t+1}^{-1/t}$, and I have assumed that p is small, so that we can neglect terms of order p^{t+2} and higher, and that the code always fails when there are $t+1$ errors. If the code can correct some weight $t+1$ errors, the logical error rate will be lower.

Now, if we concatenate the $[[n_0, 1, d_0]]$ code with itself, and decode using the simple level-by-level scheme (which, as noted above, is sub-optimal), the probability of the concatenated code having a logical error is

$$p_2 \approx p_T (p/p_T)^{(t+1)^2}, \tag{9.2}$$

and we can show by induction that with k levels of concatenation, the probability of having a logical error is

$$p_k \approx p_T (p/p_T)^{(t+1)^k}. \tag{9.3}$$

When $p < p_T$, the logical error rate rapidly converges to 0 with k. Note that the typical case now has $pn_0^k \gg t^k$ errors occurring in the code. There will be cases where a small number of errors can cause the code to fail, but those are very rare.

9.2 Convolutional Codes

A *convolutional code*, as opposed to a *block code*, does not involve a fixed number of physical qubits. Instead, it is a family of arbitrarily large codes with the property that new logical qubits can be added on to the end. Convolutional codes frequently have a periodic structure, with the same stabilizer generators repeated shifted by a few qubits, although this is not required.

The main motivation for a classical convolutional code is to handle data provided in a streaming fashion — that is, logical bits appear one at a time and should be encoded promptly, without too much delay, and then the corresponding physical bits can be sent on their way. However, convolutional codes often have

other substantial advantages. In particular, the size of the encoding circuit for a convolutional code is usually linear in the number of logical bits, and there is a natural linear-time decoding algorithm as well for many convolutional codes.

In return, convolutional codes generally give up a bit of error protection, as logical bits are effectively localized, meaning an error affecting a small number of bits can change a logical bit. However, a well-designed convolutional code should also have the property that a local error only changes a few logical bits. This is in contrast to block codes, where all the logical bits are typically spread out over all the physical bits. This means a large physical error is necessary to cause a logical error, but when a logical error does occur, there is no protection against the error changing all of the logical bits.

The situation for quantum convolutional codes is more complicated. Unfortunately, it turns out not to be possible to make quantum convolutional codes with all the desireable properties of classical convolutional codes. Nevertheless, we can achieve some of the nice properties described above.

9.2.1 Basics of Quantum Convolutional Codes

A classical convolutional code is simply one in which the encoding of each logical bit depends only on itself and the previous bits. In particular, it does not depend on the subsequent bits. However, a quick consideration reveals that this definition does not make sense for quantum codes: Any non-trivial two-qubit gate alters *both* of the qubits, whereas a gate between bits can alter one bit conditionally while leaving the other unchanged.

However, most often, classical convolutional codes have an additional constraint — namely, a finite memory. Thus, each physical bit depends on only t bits of information about the previous logical bits, no matter how many bits have been encoded. This is a property that carries over sensibly to the quantum regime. We can thus define a quantum convolutional code as follows:

Definition 9.3. A qubit *quantum convolutional code* is a family of $((n_i, 2^{k_i}))$ quantum error-correcting codes Q_i ($i \in \mathbb{Z}^+$) with the properties

1. r, s, t are positive integers with $s \leq r \leq t$,
2. $n_i = n_{i-1} + r$, $k_i = k_{i-1} + s$ for $i > 1$,
3. There exist t-qubit unitaries U_i and V_i ($i \geq 2$) and n_1-qubit unitary W such that the encoding circuit for Q_i is $V_i(\prod_{j=i}^{2} U_j)W$. Here, U_i acts on qubits $n_i - t + 1$ to n_i and takes as input the last $t - r$ outputs of U_{i-1}, the s logical qubits $k_{i-1} + 1$ to k_i, and $r - s$ $|0\rangle$ ancilla qubits. V_i also acts on qubits $n_i - t + 1$ to n_i and takes as input the output qubits of U_i, and W acts on qubits 1 through n_1 and takes as input the logical qubits 1 to k_1 and $n_1 - k_1$ $|0\rangle$ ancilla qubits.

The *rate* of a quantum convolutional code is s/r.

The unitaries V_i and W are used to start and end the code block, to make the stream of qubits of finite length. The real meat of the convolutional code is in the unitaries U_i, which accumulate as the code gets longer and longer. Figure 9.2 illustrates the structure of the encoding circuit for a convolutional code, and it is evident from the figure that the encoder only relies on a finite memory for past qubits. Any qubits beyond the most recent t (including new logical qubits and ancillas) can be sent down the channel before encoding step U_i is performed, and are never needed again in the encoder. The rate of the convolutional code is the limiting rate k_i/n_i as $i \to \infty$. For finite i, the rate of the specific code Q_i might slightly differ from the asymptotic rate, but convolutional codes are usually considered in the limit of large i, so the asymptotic behavior is the dominant one.

Most often, the unitaries U_i and V_i are the same up to shifts. That is, $U_i = I^{\otimes(n_i - t)} \otimes U$ and $V_i = I^{\otimes(n_i - t)} \otimes V$. Moreover, we usually deal with stabilizer convolutional codes, for which U_i, V_i, and W are all Clifford group operations. It is also possible, of course, to define qudit quantum convolutional codes in the same way.

The stabilizer of a convolutional code consists of sets $\mathcal{S}_i$ of $r - s$ generators. The generators in $\mathcal{S}_i$ are of the form $I^{\otimes(n_i - t)} \otimes M_{i,j}$, with $j = 1, \ldots, r - s$. When the encoder is shift-invariant, so are the generators, so

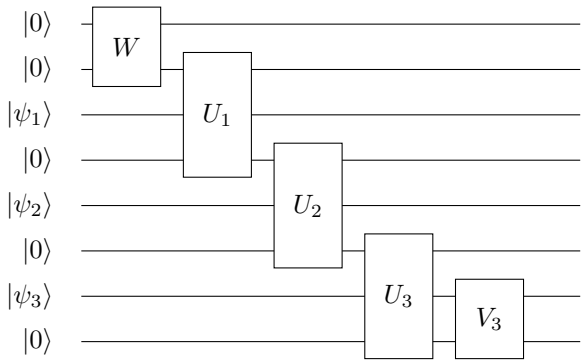


Figure 9.2: Encoding circuit for a convolutional code.

$M_{i,j} = M_j$ except perhaps for the largest and smallest values of i (due to the terminating unitaries V_i and W). Moreover, in many examples of convolutional codes (though not all), the weight of M_j is finite. Thus, the stabilizer of a large member of the convolutional code family can be described simply via $r - s$ finite-size Paulis which are then shifted systematically to give the full stabilizer of the code. For the remainder of this subsection, I will specialize to codes that have all of these properties (shift-invariant stabilizer convolutional codes with finite-weight stabilizer generators). I will also largely ignore the termination complication on the last few qubits, focusing instead on the main part of the code far from the end.

Since the shift transformation is so important in the description of convolutional codes, it is useful to have some specific notation for it. I will write D for the operation "shift by r qubits". Then, for instance, $M_{i,j} = D^{i-1} M_{1,j}$. We can also form polynomials out of D, of the form $\sum_i \alpha_i D^i$ with $\alpha \in \mathbb{Z}_2$. We can then define

$$\left(\sum_i \alpha_i D^i \right) (M_{1,j}) = \prod_i M_{i+1,j}^{\alpha_i}. \tag{9.4}$$

This is also an element of the stabilizer, since it is a product of generators. Indeed, we don't have to restrict to just finite-degree polynomials. Even an infinite-degree polynomial in D, also known as a formal *Laurent series*, gives us an element of the stabilizer in the limiting case where the code continues forever and never terminates.

Two examples of convolutional codes are given in tables 9.2 and 9.3. The first is based on the 5-qubit code, and has $r = 5$ and $s = 1$. Its rate is $1/5$. The second example is not related to any standard block code, and has $r = 3$ and $s = 1$, with rate $1/3$. By looking at the logical Paulis, you can see that both codes have distance 3. However, note that if there is only one error in each block of 5 (for the code of table 9.2) or 6 (for the code of table 9.3), then the syndrome will uniquely identify the error since it uniquely identifies that part of the error on each block. The generators M_0 and (for the second code) M_0' are produced by W and are not part of the repeated generators $\mathcal{S}_i$.

Since both of these codes have a finite extent to both the stabilizer generators and logical Paulis, we can deduce t by looking at the number of qubits involved in each block. For instance, in the case of the code in table 9.2, if we pick $t = 7$, we have enough room to make the 4 new generators and logical Paulis on the 1 new logical qubit have the correct commutation relations with each other and the old generators and logical

$$
\begin{array}{lcccccccccc}
M_0 & Z & X & & & & & & & & \\
M_1 & X & Z & Z & X & & & & & & \\
M_2 & & X & Z & Z & X & & & & & \\
M_3 & & & X & Z & Z & X & & & & \\
M_4 & & & & X & Z & Z & X & & & \\
DM_1 & & & & & X & Z & Z & X & & \\
DM_2 & & & & & & X & Z & Z & X & \\
DM_3 & & & & & & & X & Z & Z & X \\
DM_4 & & & & & & & & X & Z & Z & X \\
\vdots & & & & & & & & & & & \ddots \\
 & & & & & & & & & & & \\
\overline{X}_1 & & X & X & X & X & X & & & & & \\
\overline{Z}_1 & & X & & Z & & X & & & & & \\
\overline{X}_2 & & & & & & & X & X & X & X & X \\
\overline{Z}_2 & & & & & & & X & & Z & & X \\
\vdots & & & & & & & & & & & \ddots
\end{array}
$$

Table 9.2: A convolutional code with period 5 and rate 1/5.

$$
\begin{array}{lcccccccccc}
M_0 & X & X & Z & Y & & & & & & \\
M_0' & Z & Z & Y & X & & & & & & \\
M_1 & & X & X & X & X & Z & Y & & & \\
M_2 & & Z & Z & Z & Z & Y & X & & & \\
DM_1 & & & & & X & X & X & X & Z & Y \\
DM_2 & & & & & Z & Z & Z & Z & Y & X \\
\vdots & & & & & & & & & \ddots & \\
 & & & & & & & & & & \\
\overline{X}_1 & & X & Y & Z & & & & & & \\
\overline{Z}_1 & & Z & X & Y & & & & & & \\
\overline{X}_2 & & & & & X & Y & Z & & & \\
\overline{Z}_2 & & & & & Z & X & Y & & & \\
\vdots & & & & & & & & & \ddots &
\end{array}
$$

Table 9.3: A convolutional code with period 3 and rate 1/3.

Paulis. In particular, we can let U_i transform Paulis as follows:

$$X \otimes I \otimes I \otimes I \otimes I \otimes I \otimes I \mapsto X \otimes I \otimes I \otimes I \otimes I \otimes I \otimes I \tag{9.5}$$
$$Z \otimes X \otimes I \otimes I \otimes I \otimes I \otimes I \mapsto Z \otimes X \otimes I \otimes I \otimes I \otimes I \otimes I \tag{9.6}$$
$$I \otimes I \otimes I \otimes X \otimes I \otimes I \otimes I \mapsto I \otimes X \otimes X \otimes X \otimes X \otimes X \otimes I \tag{9.7}$$
$$I \otimes I \otimes I \otimes Z \otimes I \otimes I \otimes I \mapsto I \otimes X \otimes I \otimes Z \otimes I \otimes X \otimes I \tag{9.8}$$
$$I \otimes I \otimes I \otimes Z \otimes I \otimes I \otimes I \mapsto X \otimes Z \otimes Z \otimes X \otimes I \otimes I \otimes I \tag{9.9}$$
$$I \otimes I \otimes I \otimes I \otimes Z \otimes I \otimes I \mapsto I \otimes X \otimes Z \otimes Z \otimes X \otimes I \otimes I \tag{9.10}$$
$$I \otimes I \otimes I \otimes I \otimes I \otimes Z \otimes I \mapsto I \otimes I \otimes X \otimes Z \otimes Z \otimes X \otimes I \tag{9.11}$$
$$I \otimes I \otimes I \otimes I \otimes I \otimes I \otimes Z \mapsto I \otimes I \otimes I \otimes X \otimes Z \otimes Z \otimes X \tag{9.12}$$

The action on the other independent Paulis can be anything with the right commutation relations. This Clifford will then leave the generators and logical Paulis created by the previous U_{i-1} unchanged while encoding the new $\overline{X}$, $\overline{Z}$, and 4 new stabilizer generators into their final form. Similarly, we can pick $t = 6$ for the code of table 9.3.

As you can see from the examples, the stabilizer generators typically act on a stretch of qubits longer than r, but we can still describe each stabilizer element with a shorthand over r "qubits" by using D. In particular, a generator can be expressed as a tensor product of r polynomials in D with coefficients from $\{I, X, Y, Z\}$. I will denote the tensor product of polyonomials corresponding to P by $P(D)$. Thus, in the code of table 9.2, the polynomials for the generators of $\mathcal{S}_1$ are

$$M_1(D) = X \otimes Z \otimes Z \otimes X \otimes I \tag{9.13}$$
$$M_2(D) = I \otimes X \otimes Z \otimes Z \otimes X \tag{9.14}$$
$$M_3(D) = XD \otimes I \otimes X \otimes Z \otimes Z \tag{9.15}$$
$$M_4(D) = ZD \otimes XD \otimes I \otimes X \otimes Z \tag{9.16}$$

The polynomials for the generators of $\mathcal{S}_1$ for the code in table 9.3 are

$$M_1(D) = YD \otimes X \otimes X \otimes X \otimes X \otimes Z \tag{9.17}$$
$$M_2(D) = XD \otimes Z \otimes Z \otimes Z \otimes Y \otimes X. \tag{9.18}$$

To find the versions of these generators shifted by r, just multiply by D.

When the set of stabilizer generators acts on more than r qubits, it will overlap with the same set of generators shifted by D. Thus, in order to determine if a purported stabilizer for a quantum convolutional code actually commutes, it is necessary to compare not just generators within the same set $\mathcal{S}_i$, but also ones in different sets. Using the shorthand provides a natural criterion:

Proposition 9.2. *D^iP and D^jQ commute for all i, j iff*

$$v_{P(D)} \odot v_{Q(D^{-1})} = x_{P(D)} \cdot z_{Q(D^{-1})} + z_{P(D)} \cdot x_{Q(D^{-1})} = 0. \tag{9.19}$$

Proof. Suppose we write $v_{P(D)} = \sum_i v_{P_i} D^i$ and $v_{Q(D^{-1})} = \sum_j v_{Q_j} D^{-j}$. Then

$$v_{P(D)} \odot v_{Q(D^{-1})} = \sum_{i,j} v_{P_i} \odot v_{Q_j} D^{i-j} \tag{9.20}$$
$$= \sum_{k=-\infty}^{+\infty} \sum_i v_{P_i} \odot v_{Q_{i-k}} D^k. \tag{9.21}$$

The coefficient of D^0, $\sum_i v_{P_i} \odot v_{Q_i}$, we can recognize immediately as $v_P \odot v_Q = c(P, Q)$. Similarly, the coefficient of D^k is

$$\sum_i v_{P_i} \odot v_{Q_{i-k}} = c(P, D^k Q). \tag{9.22}$$

Thus, $v_{P(D)} \odot v_{Q(D^{-1})} = 0$ as polynomials iff P and D^kQ commute for all k. Since $c(D^iP, D^jQ) = c(P, D^{j-i}Q)$, this proves the proposition. □

9.2.2 Decoding of Convolutional Codes

One advantage of quantum convolutional codes is that by definition they have linear-time encoding algorithms. Note that this is true even for non-stabilizer codes, since each U_j, V_i, and W act on a constant number of qubits. Even for a stabilizer convolutional code, an $O(n)$ encoder is a signficant improvement over the $O(n^2/\log n)$ size of the encoding circuit for a generic stabilizer code.

Even more dramatic, and not as obvious, is that convolutional codes often have a linear-time decoding algorithm as well. Since general stabilizer codes could take exponential time to decode, this is a big deal. The algorithm is known as the *quantum Viterbi algorithm*. The Viterbi algorithm (classical or quantum) is an example of *dynamic programming*. The trick to it is to work in chunks of size r and try to figure out the most likely error up to qubit n_i. What we'd like to do then is to move to the next block of r qubits and try to figure out the most likely error up to qubit $n_{i+1} = n_i + r$. Ideally, we would just say, "Oh, the most likely error on n_{i+1} qubits is just the most likely error on the first n_i qubits followed by the most likely error on the next r qubits." Then we could just systematically work our way through each set of r qubits picking the most likely errors and get a decoding algorithm running in $O(n)$ time. Well, we could *say* that, but sometimes we'd be wrong. The problem is that (for a stabilizer code), there may be stabilizer generators that act on the first n_i qubits but also act on later qubits. These generators are not as helpful as they might be when determining the most likely error on the first n_i qubits because there will be multiple different choices of error on the next r qubits that get the error syndrome right. The problem is that some choices of error on the next r qubits might be extremely unlikely, but to know that, we would need to look at the error syndrome on more generators, which means involving even more qubits, and so on.

The solution is not to pick the *single* most likely error, but instead the most likely error ending in each possible way. That is, we need to consider all the different ways the error on the first n_i qubits might affect any stabilizer generators that continue into the next set of r qubits. Here's where we need to make an assumption: specifically, we will assume we have a convolutional code with finite-weight stabilizer generators, all bounded by a constant w. (Note that w might or might not be the same as t in general.) That means that the stabilizer generators in $\mathcal{S}_{i+1}$ only touch the last $w-r$ qubits out of the first n_i qubits. That's important, because it means that if we want to find an error that satisfies generators only in sets $\mathcal{S}_1$ through $\mathcal{S}_i$, there are only a constant number of ways, specifically 4^{w-r}, that the error can end. To extend to the most likely error satisfying the syndrome for $\mathcal{S}_{i+1}$ and ending in a particular way, we try out all the different ways an error on n_i qubits can end matched with all the compatible errors on the next r qubits and pick the most likely combination.

Putting this together, we get the following algorithm to find the most likely or lowest-weight error consistent with the measured error syndrome for all generators:

Algorithm 9.2 (quantum Viterbi algorithm)**.** Let $\{Q_i\}$ be a stabilizer quantum convolutional code with stabilizer generators $I^{\otimes(n_i-w)} \otimes M_{i,j}$, with $j = 1, \ldots, r-s$, $\mathrm{wt}\, M_{i,j} \leq w$, $r \leq w$. Given an error syndrome for some particular instance of the code with n total physical qubits, use the following algorithm to determine the most likely (or lowest-weight) error:

1. Create a table of all possible Pauli errors on $w-r$ qubits, with two entries (N_P, p_P) for each Pauli P. Initialize by simply letting $N_P = P$ and putting the probability (or weight) of P as p_P. If there are any stabilizer generators with support on just the first $w-r$ qubits, set the probability of any Pauli incompatible with the error syndrome to 0, or set the weight to ∞.

2. Set $n_0 = w - r$ and $i = 0$.

3. Until $n_i \geq n$, repeat the following steps:

   (a) For each Pauli P on qubits $n_{i+1} - (w-r) + 1$ to n_{i+1}, run through all possibilities for Q on qubits $n_i - (w-r) + 1$ to n_i and R on qubits $n_i + 1$ through $n_{i+1} - (w-r)$. (If $w - r \geq r$, we don't need to run over R and we only consider Q which is consistent with P on qubits $n_{i+1} - (w-r) + 1$ through n_i.) If $N_Q \otimes R \otimes P$ has the correct error syndrome for all generators $M_{i+1,j}$ $(j = 1, \ldots, r-s)$, calculate the probability (or weight) of $N_Q \otimes R \otimes P$ by multiplying the probability p_Q times the

probability of $R \otimes P$ (or by adding the weights p_Q and wt R+wt P). Choose Q and R such that the probability is highest or the weight is the lowest. Create an updated table with $N'_P = N_Q \otimes R \otimes P$ and p_P is the probability or weight of N'_P. If no Q and R gives the correct error syndrome, set p'_P to probability 0 or weight ∞.

(b) Switch the updated table (N'_P, p'_P) into the main table (N_P, p_P), increase i by 1 and let $n_{i+1} = n_i + r$.

4. Look through the table, running over all P, to find the value for which the probability p_P is a maximum or the weight is a minimum. Output that corresponding entry N_P as the most likely (or lowest-weight) error for this error syndrome.

Convolutional codes are looking pretty good right now: they have a non-zero asymptotic rate, a linear-time encoder, and can also have a linear-time decoder. Unfortunately, there is one big problem: they cannot have more than a constant distance. This is true for *any* quantum convolutional code, regardless of whether the U_i repeat, or whether it is a stabilizer code, or if the stabilizer generators have finite extent. This issue arises because qubits are not spread out enough by the encoding circuit. In particular, there is a small bottleneck, the finite memory of the encoder, which prevents too much information about old logical qubits from being involved in the later physical qubits.

Proposition 9.3. *Let $\{Q_j\}$ be a quantum convolutional code (not necessarily a stabilizer code). Then the distance of Q_j is less than c for all j, where c is the constant $c = r\lceil (3t + r)/s \rceil$.*

Proof. When the unitaries U_j and V_j are t-qubit unitaries, choose i to be an arbitrary value at least $\lceil t/r \rceil$, so U_{i+1} does not act on the first n_1 qubits. Also, U_i does not act on qubit $n_i + 1$ or any later qubit. Let $i' > i$ be such that $k_{i'} - k_i > 3t + r$, and let $i'' = i' + \lceil t/r \rceil$, so $U_{i''+1}$ does not act on the first $n_{i'}$ qubits. Finally, pick any $i''' \geq i''$. We can write the encoder as $U_C U_B U_A$, where $U_A = (\prod_{j=i}^{1} U_j)W$, $U_B = \prod_{j=i''}^{i+1} U_j$, $U_C = V_k(\prod_{j=i'''}^{i''+1} U_j)$.

Consider code $Q_{i'''}$. It has encoding circuit $U_C U_B U_A$. We will imagine fixing all logical qubits less than or equal to k_i and greater than $k_{i'}$ while varying the logical qubits $k_i + 1$ to $k_{i'}$. For instance, we can let the fixed qubits be all $|0\rangle$. Since U_A does not act on physical qubits $n_i + 1$ or later, it does not involve the varying logical qubits. Let $|\psi\rangle = U_A|0 \dots 0\rangle$. Now let us consider erasure errors E acting only on the physical qubits from $n_i + 1$ to $n_{i'}$. For the QECC conditions to hold, we must have, for $x \neq y$ basis states for logical qubits $k_i + 1$ through $k_{i'}$,

$$0 = \langle \overline{0 \dots 0x0 \dots 0} | E | \overline{0 \dots 0y0 \dots 0} \rangle \tag{9.23}$$

$$= \langle 0 \dots 0x0 \dots 0 | U_A^\dagger U_B^\dagger U_C^\dagger E U_C U_B U_A | 0 \dots 0y0 \dots 0 \rangle \tag{9.24}$$

$$= \langle 0 \dots 0x0 \dots 0 | U_A^\dagger U_B^\dagger E U_B U_A | 0 \dots 0y0 \dots 0 \rangle \tag{9.25}$$

$$= (\langle \psi | \otimes \langle x0 \dots 0 |) U_B^\dagger E U_B (|\psi\rangle \otimes |y0 \dots 0\rangle), \tag{9.26}$$

since U_C acts only on qubits not involved in E and U_A acts only logical qubits before k_i. Now, $|\psi\rangle$ could be an entangled state between those qubits which are acted on by U_B and those which are not, but even if it is maximally entangled, the Schmidt rank is at most t, since that is the largest number of qubits that U_B could act on. In particular, there exists unitary V acting only on qubits not acted on by U_B such that $V|\psi\rangle = |0 \dots 0\rangle \otimes |\psi'\rangle$, with $|\psi'\rangle$ a $2t$-qubit state, and with U_B not acting on the first tensor factor. We thus have

$$0 = (\langle \psi | \otimes \langle x0 \dots 0 |) U_B^\dagger E U_B (|\psi\rangle \otimes |y0 \dots 0\rangle) \tag{9.27}$$

$$= (\langle \psi | \otimes \langle x0 \dots 0 |) V^\dagger U_B^\dagger E U_B V (|\psi\rangle \otimes |y0 \dots 0\rangle) \tag{9.28}$$

$$= (\langle \psi' | \otimes \langle x0 \dots 0 |) U_B^\dagger E U_B (|\psi'\rangle \otimes |y0 \dots 0\rangle). \tag{9.29}$$

Moreover, only qubits up to $n_{i''}$ can be involved here too, since U_B cannot act on qubits $n_{i''} + 1$ or greater. Thus, equation (9.29) only involves $2t + r(i'' - i)$ qubits.

Finally, consider the case where $E = |0\ldots0\rangle\langle 0\ldots0|$ on qubits $n_i + 1$ through $n_{i'}$, a total of $n_{i'} - n_i = r(i' - i)$ physical qubits. Equation (9.29) then becomes an inner product

$$\langle \phi_x | \phi_y \rangle = 0, \tag{9.30}$$

where $|\phi_x\rangle$ is defined by $|\phi_x\rangle \otimes |0\ldots0\rangle = |0\ldots0\rangle\langle 0\ldots0|U_B(|\psi'\rangle \otimes |x0\ldots0\rangle)$.

$|\phi_x\rangle$ involves $2t + r(i'' - i') = 2t + r\lceil t/r \rceil < 3t + r$ qubits, and all the different $|\phi_x\rangle$ must be orthogonal. But the number of different such states is $k_{i'} - k_i > 3t + r$. This is a contradiction, so we have found an erasure error of weight $r(i' - i)$ for which the QECC conditions are not satisfied. Since $k_{i'} - k_i = s(i' - i)$, we have shown that the distance of the convolutional code is at most $r\lceil (3t + r)/s \rceil$. □

Basically, convolutional codes can only correct errors that do not cluster too much. Even randomly placed errors will have occasional clusters of size larger than c, so convolutional codes also are not good against them. Nevertheless, convolutional codes can still be useful if we're willing to settle for something less than correcting all errors. Typically, we restrict attention to codes which are *non-catastrophic*, meaning that a uncorrectable error confined to a region only affects logical qubits near that region, while logical qubits far away remain OK. Non-catastrophic convolutional codes do allow errors to happen, but they successfully quarantine them, limiting the damage.

Definition 9.4. Suppose the decoder for a stabilizer convolutional code $\{Q_i\}$ assigns Pauli P_σ to syndrome σ. Then for any error Q, $R_Q = P^\dagger_{\sigma(P)}Q$ is a logical Pauli operator. The decoder is *catastrophic* if there exists some finite-weight Pauli Q such that R_Q has unbounded weight as $i \to \infty$. A stabilizer convolutional code is catastrophic if all decoders for it are catastrophic.

9.3 Information-Theoretic Approach to QECCs

The QECC conditions (theorem 2.7) and their variants give us a nice algebraic set of conditions for determining if a subspace is a QECC. It is helpful to also have a different criterion based on information theory, which will basically say that the amount of quantum information in the encoded state stays constant.

One big difference in the information-theoretic approach compared to the algebraic approach of theorem 2.7 is that the information-theoretic approach requires us to specialize to a specific quantum channel for the noise, whereas the QECC conditions instead work with a list of possible errors. The advantage of the latter approach is that we don't need to know precisely what is going on with the noise, just the list of possible things that could happen to our data. The information-theoretic approach, on the other hand, is much more natural when we know that some types of error are substantially more likely than others. In the algebraic approach, our only option was to say "these errors are negligible, so we will ignore them." The information-theoretic approach lets us weight each error by its actual prevalence.

The algebraic approach is also useful for proving more algebraic properties of codes, such as the fact that the distance tells us about the code's ability to correct both general errors (corollary 2.8) and erasure errors (theorem 2.10) or the Eastin-Knill theorem (theorem 11.7) in fault tolerance. These types of things are tougher in the information-theoretic approach, which is more natural when dealing with information-theoretic properties like the quantum channel capacity, which I will discuss in chapter 18.

The big payoff of the information-theoretic approach is that it generalizes very naturally to approximate quantum error-correcting codes (section **??**), which don't recover the state perfectly but only to some good approximation. The algebraic QECC conditions don't work well for approximate codes, with a large (dimension-dependent) factor between the necessary and sufficient conditions, whereas the obvious generalization of the information-theoretic condition we will derive works straightforwardly. This is another reason to use the information-theoretic approach with the channel capacity, since it deals with the case of infinitely-many qubits, where the error is non-zero but asymptotes to zero.

9.3.1 Coherent Information

So, we want to quantify the amount of quantum information in a system Q. Information should be *about* something, which we can represent by an additional system R, known as a *reference system*. To make the

information quantum, we should allow for the possibility of entanglement between Q and R. However, the joint system of Q and R might not be pure, so we could also consider an environment E, as in the final state of figure 9.3. We might as well consider the whole universe outside of Q and R to be the environment; that is, we can let the joint state of RQE be a pure state. We imagine that R (being an idealized system) has remained isolated so that all its entanglement is with Q, but Q has not, so it may be entangled with E.

If Q and R are maximally entangled, it makes sense to say that the amount of quantum information is n when each system has n qubits, so Q has n qubits of quantum information "about" R. If ρ_{QR} is an arbitrary pure state, the amount of quantum information in Q about R should then just be the entanglement entropy between the two, $S(Q)$ (which I am using as shorthand for $S(\rho_Q)$).

Another case where the answer is obvious is when Q can be decomposed into a tensor product between a subsystem Q_1 which is entangled only with R and a second subsystem Q_2 which is entangled only with E. In that case, the amount of quantum information in Q about R is just

$$S(Q_1) = S(Q) - S(E). \tag{9.31}$$

This will turn out to be the correct answer more generally, and it makes sense. $S(E)$ is the amount of information that E "knows" about Q and if the environment knows about the state of a system, it can't be quantum. (For instance, because a measurement on the environment could collapse the logical state.)

We'd like to phrase our formula without talking about E so that it is just a property of the state of Q and R. Since the global state is pure, $S(E) = S(RQ)$. Then we have the following definition to quantify the amount of quantum information in a system.

Definition 9.5. Given a quantum system Q entangled with a reference system R in the state ρ_{RQ}, the *coherent information* in Q is

$$I_c(\rho_{RQ}) = S(\rho_Q) - S(\rho_{RQ}). \tag{9.32}$$

Let $\mathcal{E}$ be a quantum channel taking system L into Q. The initial state of L and R is a pure state ρ. Then the coherent information of $\mathcal{E}$ for ρ is

$$I_c(\mathcal{E}, \rho) = S(\mathcal{E}(\rho_L)) - S((\mathcal{I} \otimes \mathcal{E})(\rho)). \tag{9.33}$$

The channel set-up realizes the situation we discussed above: R remains isolated and does not go through the channel $\mathcal{E}$ or interact with the environment E, but Q does and therefore may lose some coherence to E. The coherent information of $\mathcal{E}$ on ρ then quantifies (as we will see shortly) how much quantum information about R remains in Q.

This suggests that it is helpful to consider not just the state of Q after the channel, but also the state of E. The channel that maps L to E is one that tells us a lot about error correction properties of $\mathcal{E}$:

Definition 9.6. Let $\mathcal{E}$ be a quantum channel taking system L into Q which can be purified into U using an environment E. Then the channel produced by performing U on L and discarding Q maps L to E. It is written $\hat{\mathcal{E}}$ and is called the *complementary channel*.

If you're familiar with classical information-theoretic quantities, you may recognize the formula for coherent information as the negative of a conditional entropy $I_c(\rho_{RQ}) = -S(R|Q)$. The conditional entropy is a quantification of the amount of uncertainty remaining in R once Q is specified. From a classical point of view, it is very peculiar to consider a negative conditional entropy: The coherent information can never be positive for a classical system! But from a quantum perspective, negative uncertainty actually makes a certain kind of perverse sense — negative uncertainty means that you are *more* certain about the state than would be allowed classically, which doesn't seem so dissimilar to entanglement providing stronger correlations than are possible classically, as happens with Bell inequalities.

One property of coherent information is that putting the state through a channel cannot increase it. This makes it suitable for the various applications we have in mind for it, since quantum information, once lost, should be gone for good. (Error correction does not replace lost quantum information, it protects it and spreads it out so that it is not destroyed in the first place.)

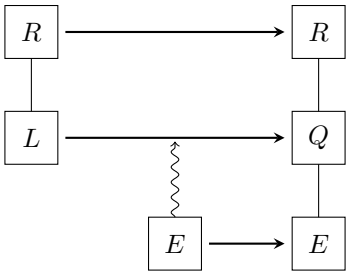


Figure 9.3: System L is entangled with reference system R. L goes through a noisy channel to become system Q after interacting with environment E.

Theorem 9.4 (Quantum data processing inequality)**.** *Let ρ be a mixed state on reference system R and quantum system L, and let $\mathcal{E}$ be a quantum channel mapping L to Q. Then*

$$I_c(\mathcal{E}, \rho) \leq I_c(\rho). \tag{9.34}$$

Equivalently, if $\mathcal{E}$ maps L to Q and $\mathcal{F}$ maps Q to Q', then

$$I_c(\mathcal{F} \circ \mathcal{E}, \rho) \leq I_c(\mathcal{E}, \rho). \tag{9.35}$$

Proof. I will only prove the first version, since the second is essentially identical. Let E be the environment used to purify the initial state ρ. $\mathcal{E}$ does not act on E; instead, it uses a new environment E'. $\mathcal{E}$ also does not act on R. Then

$$I_c(\rho) = S(L) - S(RL) = S(RE) - S(E) \tag{9.36}$$
$$I_c(\mathcal{E}, \rho) = S(Q) - S(RQ) = S(REE') - S(EE'). \tag{9.37}$$

Applying the strong subadditivity property of entropy to R, E, and E', we find

$$S(REE') + S(E) \leq S(RE) + S(EE') \tag{9.38}$$
$$I_c(\mathcal{E}, \rho) \leq I_c(\rho). \tag{9.39}$$

□

9.3.2 Coherent Information and QECCs

The data processing inequality immediately tells us that in order for a QECC to function, the coherent information must not decrease under the action of the error, since otherwise we cannot recover the lost coherent information and therefore cannot possibly get back to where we started. Thus, coherent information staying constant is a necessary condition for quantum error correction. It turns out that it is also a sufficient condition.

What is the condition, precisely? The coherent information is state-dependent, and we need it to be constant for any input codeword. It's simpler to phrase the condition by simply saying the coherent information remains constant for any input state, but then apply the condition not to the noise channel $\mathcal{E}$ by itself but to the composition of the encoder U followed by $\mathcal{E}$.

Theorem 9.5. *Let $\mathcal{F} : L \to Q$ be a quantum channel. Let R be a reference system, which must have a dimension at least as large as L. There exists a decoding CPTP map $\mathcal{G}$ with the property $\mathcal{G} \circ \mathcal{F} = \mathcal{I}$ iff $I_c(\mathcal{F}, \rho) = S(\rho_L)$ for all pure states ρ on $R \otimes L$.*

When we let $\mathcal{F} = \mathcal{E} \circ U$, where U is a partial isometry from the logical Hilbert space L to a larger physical Hilbert space, then the condition that $\exists\, \mathcal{G}$ such that $\mathcal{G} \circ \mathcal{F} = \mathcal{I}$ is immediately equivalent to the definition of a QECC for the case where the set of possible errors is just the set of Kraus operators of the channel $\mathcal{E}$. The normalization gets taken care of automatically because all of the channels U, $\mathcal{E}$, and $\mathcal{G}$ are trace preserving. Therefore, theorem 9.5 gives us the desired information-theoretic version of the QECC conditions.

Proof. Note that $I_c(\rho) = S(\rho_L)$ since ρ is a pure state, so there is no initial environment. Since $I_c(\mathcal{I}, \rho) = I_c(\rho)$, the forward direction follows immediately from the data processing inequality, as noted above.

Now suppose $I_c(\mathcal{F}, \rho) = S(\rho_L)$ for all ρ. Let us consider the purification of $\mathcal{F}$ to V, mapping L to $Q \otimes E$, and let $\sigma = \mathcal{I} \otimes V(\rho)$. Since the global state σ of RQE is a pure state, that means that

$$I_c(\mathcal{F}, \rho) = S(Q) - S(RQ) = S(RE) - S(E) \tag{9.40}$$

and since the initial state ρ is a pure state, $S(L) = S(R)$. Therefore,

$$S(RE) = S(E) + S(R), \tag{9.41}$$

which is true only when $\sigma_{RE} = \sigma_R \otimes \sigma_E$. This is true for all ρ, and in particular it is true when ρ is a maximally entangled state for RL.

When ρ is a maximally entangled state, we can prepare arbitrary pure states $|\psi\rangle_L$ on L by performing different projections on R. Since V doesn't act on R, projections on R commute with the channel. If we perform a projection on R for the final state σ, the state σ_E is unaffected, since $\sigma_{RE} = \sigma_R \otimes \sigma_E$. Thus the state of E is σ_E regardless of which pure state $|\psi\rangle_L$ was initially prepared for L.

Now, the effect of $\mathcal{F}$ is to do V, getting state σ, and then to discard E. Let us think of V as the encoding map of a QECC and discarding E as an erasure error that happens to this code. One version of the QECC conditions (proposition 2.12) says that we can correct for the erasure of E iff the state σ_E is independent of $|\psi\rangle_L$. That is true in this case, so there exists a recovery operation $\mathcal{G}$ for which $\mathcal{G} \circ \mathcal{F} = \mathcal{I}$. (Here, $\mathcal{F}$ combines both the encoding V and the error, an erasure of E, and recall that normalization automatically follows because $\mathcal{F}$ is trace preserving.) □

It's worth noting that in the proof, all we really need is for $I_c(\mathcal{F}, \rho) = S(\rho_L)$ for a maximally entangled state ρ. We don't actually need to check this condition for all ρ, but it follows from the proof that if it's true for a maximally entangled state, then it is true for all ρ.

Part II

Fault-Tolerant Quantum Computation

Chapter 10

Everyone Makes Mistakes: Basics Of Fault Tolerance

We've now discussed extensively the notion of quantum error correction. However, there's a flaw in the basic model we've been considering. Or rather, there is a *lack* of flaws: We've been assuming that the encoding and decoding circuits needed for error correction can be performed perfectly, and that errors only occur between these two steps. This might not be a bad idealization if you're mainly concerned about transmission over a noisy communications channel or storage for long amounts of time. These are cases where the errors introduced by mistakes in the encoding and decoding steps may be much rarer than errors occurring in the period in between. However, in practice, errors in gates are far from negligible and are likely to remain so for the forseeable future, so neglecting errors during the encoding and decoding circuits is a poor approximation. If we want to put together millions or more gates to perform a large quantum computation, the prospect of getting through the whole circuit without errors is vanishingly unlikely.

To protect quantum computations against errors, we need something more than a quantum error-correcting code, we need a *fault-tolerant protocol*, which allows us to perform unitary gates on encoded states despite errors occurring during the computation. Part II is devoted to developing the theory of fault-tolerant quantum computation, culminating in the *threshold theorem*, which says that arbitrarily long quantum computations are possible provided the error rate per physical gate and time step is below some constant threshold value. In this chapter, I'll introduce the basic concepts of fault tolerance. We'll learn what it means to compute in a world where no component can be taken for granted, where everyone — every qubit, every gate — can make a mistake.

10.1 The Fault-Tolerant Scenario

As discussed in chapter 1, there are many possible errors that can afflict a quantum state. In part I, the errors only had one shot at the quantum state: we encode, the errors do their thing (although we might wish they didn't), and then we decode and correct the state. For fault tolerance, we will need to deal with more pervasive errors. Errors can occur between gates or during gates, and even worse, the errors keep happening. Every time we do another gate, or even if we simply leave a qubit by itself for a little while, a new error can occur. Dealing with the full range of possible errors is a daunting task, so for the purposes of developing the theory and analyzing fault-tolerant protocols, we normally work with a simplified model, which I'll discuss in this section. We'll actually need to add another complication to this model before proving the threshold theorem in chapter 14, and we'll discuss fault tolerance for even more general error models in chapter 15.

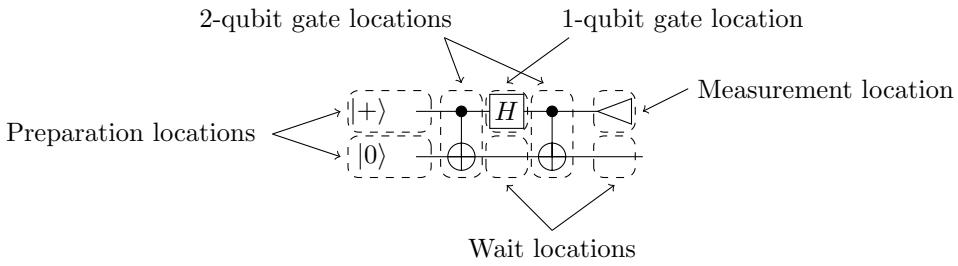


Figure 10.1: A sample circuit broken down into locations. Note that a 2-qubit gate location is a single location on 2 qubits. Thus, this sample circuit has a total of 8 locations.

10.1.1 Basic Model for Fault Tolerance (Independent Stochastic Errors)

Any quantum circuit, fault-tolerant or not, is composed of the same basic components. We perform gates on qubits, we may prepare new qubits and introduce them to the circuit, and sometimes we measure qubits and perhaps perform a classical computation on the outcome. Sometimes measurement occurs in the middle of the computation, in which case later actions may depend on the measurement outcomes, and sometimes measurement is delayed until the end of the computation. Each basic action that we can perform is called a *location*. We'll consider circuits which can perform gates in parallel, since we'll want to simultaneously correct errors in different parts of the computer.

Definition 10.1. Consider a quantum circuit, arranged into time steps. Each qubit can be involved in at most one action per time step, but multiple actions can be performed in a single time, provided they involve different qubits. A *location* is a single indivisible action in the circuit. We may consider the following different types of locations:

1. A *preparation location*: The preparation of a single qubit in a particular state, frequently $|0\rangle$. Sometimes we distinguish between different types of preparation locations based on the state prepared. Occasionally we may want a preparation location to encompass the creation of a multiple-qubit entangled state.

2. A *gate location*: A single gate from the universal gate set used to describe the circuit. Sometimes we may want to distinguish different types of gate location based on the kind of gate appearing (e.g., a CNOT location or a H location). If the gate is a multiple-qubit gate, the location encompasses all the qubits involved in the gate, but still just counts as a single location. Usually, we only consider unitary gates to produce gate locations, but one could also consider gate locations for more general quantum channels.

3. A *wait location* or *storage location*: A single qubit is stored for a unit time with no gate performed on it.

4. A *measurement location*: A single qubit is measured with a von Neumann measurement, usually in the standard basis $|0\rangle$, $|1\rangle$. The measurement outcome is captured as classical information. Occasionally we may want locations corresponding to other kinds of measurements, including possibly multiple-qubit entangled measurements.

5. A *classical computation location*: In the basic model for fault tolerance, we omit locations corresponding to classical computations. We assume that classical computation can be performed instantaneously (and flawlessly) on the outcomes of measurements, including combining the results of multiple measurement locations. The outcomes of the classical computations may later be used to control any other locations later in the circuit.

Remember, we are just developing the basic model for fault tolerance most often used in analyzing protocols. Because we're currently concerned with the task of manipulating quantum states, even giving ourselves unlimited classical computational power doesn't make this task trivial. Of course, we shouldn't abuse the privilege. For instance, it is definitely cheating to scrap the quantum circuit altogether, and solve the problem we are interested in using an inefficient classical algorithm — but note that even such a blatant cheat doesn't let us store an unknown quantum state indefinitely in the presence of noise. We'll discuss the assumption of perfect classical computation along with other deviations from the basic model in chapter 15.

Next, we want to add errors. In the basic model for fault tolerance, we consider that errors at different locations are independent, and furthermore consider a simplified model for the errors. We still want to include the possibility that any location can go bad.

Definition 10.2. Consider a quantum circuit C divided into locations, as above. An *independent error model* on the quantum circuit C is a circuit $\widetilde{C}$ where every preparation location, gate location, storage location, and measurement location is replaced by a separate quantum channel, arranged in the same way as the original circuit. In an *independent stochastic error model*, the quantum channel corresponding to a particular location L of C performs the correct action for L with probability $1 - p_L$ and does something else with the same input and output Hilbert space dimensions as L with probability p_L. I.e., the channel for a $|0\rangle$ preparation location L will prepare $|0\rangle$ with probability $1 - p_L$ and prepare some other single-qubit state (possibly mixed) with probability p_L. The quantum channel corresponding to L is still called a *location* and may be referred to as $\widetilde{L}$, or just L if the distinction between the two is not needed. p_L is called the *error probability* or *error rate* of location L.

The circuit $\widetilde{C}$ is called a *noisy* implementation of C. In any particular realization of a noisy circuit $\widetilde{C}$ with an independent stochastic error model, for some locations the correct action L is performed, while for other locations the wrong action is performed. The locations where the wrong thing happens are called *faulty*, or they are said to have a *fault*. We frequently say of a faulty location that a preparation error, gate error, storage error, or measurement error has occurred, depending on the type of location.

In an *independent Pauli error model*, a faulty location always performs the correct action followed by a Pauli channel. In a Pauli error model, a faulty measurement location will flip the measurement outcome bit with some probability.

A *fault path* is a set of locations in C. A *Pauli fault path* is a fault path with a Pauli error associated with each location in the fault path.

There are a number of things to point out about the independent stochastic error model. The error model is "independent" because the quantum channels for different locations are separate. In the stochastic version, we can consider that errors occur with a certain probability p_L. Then the probability of having faults at two specific locations L and M is $p_L p_M$.

Note, however, that (unless we are dealing with a Pauli error model) we make no prescription for what happens at faulty locations. In particular, multiple-qubit locations, such as a CNOT gate location, can have errors which involve both qubits, including errors which entangle the qubits in the wrong way. This still is considered to have probability p_L. The independence only applies to separate locations. This is an essential part of the error model, since errors that occur during the performance of a gate will generally affect all the qubits involved in the gate.

Another perhaps suprising observation is that, according to the definition, a faulty location might behave correctly! The identity is a possible matrix, even for a Pauli channel, so that "error" might turn out not to be one. In this case, we are essentially overcounting the number of faults, which generally means we are just being more conservative than we need to, and our conclusions will be correct. (There are occasional cases where having an identity "error" is actually worse than a real error, for instance if the error would cancel another previous error, but that just means that we are correct in thinking of the identity "error" location as a fault.)

The notion of fault path is useful when discussing and computing error rates, particularly (though not exclusively) for independent stochastic channels. The fault path represents the locations where errors occur in a particular realization of the circuit. A Pauli fault path adds in additional information about the type of error occurring at each location, under the assumption that the errors are Pauli errors.

Definition 10.3. A *basic model for fault tolerance* is as follows: An ideal circuit C is physically realized as a noisy circuit $\widetilde{C}$, according to an independent stochastic error model. The error model has the property that all locations of a given type have the same error probability. In other respects, the quantum channels used in the error model are arbitrary, and may be incompletely specified. A basic error model can therefore be summarized in terms of four error probabilities: p_P (the error probability for a preparation location), p_G (the error probability for a gate location), p_S (the error probability for a storage location), and p_M (the error probability for a measurement location).

Frequently the basic model is simplified even further, either by using an independent Pauli error model or by making some or all of the four error probabilities equal, or both. Sometimes the basic model is slightly elaborated by considering different error rates for different types of gate, for instance by letting the error rate for two-qubit gates be higher than the error rate for one-qubit gates.

Basic models for fault tolerance capture the main effects that are important in developing the theory of fault tolerance. While a particular physical realization for quantum computation may not fulfill precisely the assumptions of a basic model, provided it does not violate the assumptions too much, we expect fault-tolerant protocols to work more or less according to the analysis done for the basic model. We'll discuss in detail in chapter 15 what happens when we change the assumptions behind the basic model.

Conversely, if we attempt to simplify the basic model by leaving out parts of it, the analysis can fail badly for realistic systems. For instance, if we assume the storage error rate is 0, so wait locations are perfect, we might come up with a circuit that is very sensitive to storage errors and thus fails badly when the storage error rate becomes non-zero. Nevertheless, it is sometimes interesting to consider models that do include such simplifications, either because they represent the qualities of a particular physical system of interest or because they may give us insight into particular aspects of the theory of fault tolerance.

10.1.2 Error Propagation

Building fault-tolerant circuits that still work when subjected to independent stochastic errors is challenging because of two problems beyond those we solved to make QECCs. First, we can no longer count on any circuit or gate to do exactly what we intended it to do, since any gate can fail with probability p_G. Second, once errors appear, they don't stay put. Instead, they spread through the qubits of our computer like a disease being transmitted by sick air travellers. Even a portion of a circuit that has no faults in it can still cause problems by contributing to error propagation.

To see how this works, consider the CNOT gate. Assume it acts correctly, but that the initial state is wrong. Suppose instead of $|\psi\rangle$, the CNOT acts on $E|\psi\rangle$. Then

$$\text{CNOT}(E|\psi\rangle) = (\text{CNOT } E \text{ CNOT})\text{CNOT}|\psi\rangle. \tag{10.1}$$

The correct final state is $\text{CNOT}|\psi\rangle$, so the true final state has an error CNOT E CNOT on it, much as we saw for the analysis of the Clifford group in chapter 6. For example, suppose $E = X \otimes I$. Then CNOT E CNOT $= X \otimes X$, so a bit flip error spreads from the control qubit to the target qubit of the CNOT. Whereas only one qubit had an error before the CNOT, now two qubits have errors, even though the gate itself functioned perfectly. This is clearly a problem for a QECC which is designed to correct only a small number of errors — an error in one qubit, initially something the code could correct, might spread quickly to affect more qubits than the code is designed to handle.

Now consider $E = I \otimes Z$. We have CNOT E CNOT $= Z \otimes Z$, meaning the phase flip error has spread from the target qubit to the control qubit. While a classical CNOT can spread errors from control to target, the quantum CNOT can also spread errors the other way. This means that error propagation, already a challenge for classical fault-tolerant computers, is even more pervasive in a quantum computer. In general, any two-qubit entangling gate will cause errors to propagate in both directions through the gate. (The SWAP gate is an interesting exception in that it causes errors to switch between the two qubits without directly causing them to spread. But then, the SWAP gate doesn't really entangle the two qubits involved in the gate either.)

10.1.3 Example Transversal Gate

Single-qubit gates don't cause error propagation, so circuits composed of single-qubit gates will behave much better than general circuits involving multiple-qubit gates. For instance, consider the seven-qubit code. It turns out that the Hadamard transform applied to all 7 qubits of the code will perform the logical Hadamard transform:

$$H^{\otimes 7}|\overline{0}\rangle = \frac{1}{\sqrt{2}}(|\overline{0}\rangle + |\overline{1}\rangle) \tag{10.2}$$

$$H^{\otimes 7}|\overline{1}\rangle = \frac{1}{\sqrt{2}}(|\overline{0}\rangle - |\overline{1}\rangle) \tag{10.3}$$

(You can check this yourself, if you like.) Imagine we have a codeword $|\psi\rangle = \alpha|\overline{0}\rangle + \beta|\overline{1}\rangle$, and consider what happens if there is a single-qubit error on the codeword, say a bit flip on qubit 3. Then, as above, we find that the state post-Hadamard is

$$(H^{\otimes 7} X_3 H^{\otimes 7}) H^{\otimes 7}|\psi\rangle = Z_3 H^{\otimes 7}|\psi\rangle. \tag{10.4}$$

The single-qubit Hadamard transforms have changed the *type* of error from a bit flip to a phase flip, but crucially, it is still a single-qubit error. Before the Hadamard transforms, the code could correct the error, and the code can still correct the error after the Hadamard transforms.

This is the key idea that takes us a long way towards fault tolerance. The tensor product of 7 Hadamard transforms is an example of a *transversal gate*, a concept that we'll explore in much more detail in chapter 11. Of course, single-qubit gates alone won't get us very far, so we'll also see how to design multiple-qubit transversal gates. These gates will cause some error propagation, but the error propagation will be carefully controlled to be of a form we can handle. It turns out that even multiple-qubit transversal gates are still not enough to provide a universal set of gates on the encoded qubits, so we'll need to add some additional techniques discussed in chapter 13 to have a full range of control.

10.1.4 Starting and Ending Encoded

There is an additional pitfall to starting or ending a fault-tolerant computation. One natural approach is to start with unencoded qubits given to you by someone else, then to encode them in a QECC. Once it is encoded, perform the fault-tolerant quantum computation, then decode, leaving unencoded qubits once more. However, this approach leaves the qubits vulnerable to errors at the beginning and end of the computation. Even if we discount errors occurring before we're given the qubits or after we finish with them, there is always the possibility of errors early in the encoding circuit, before we've had time to develop any error correction capability, or late in the decoding circuit. Techniques exist to minimize the period of vulnerability, but it cannot be completely eliminated. Therefore, any quantum computation performed this way has a minimum error probability (derived from the error rates p_G, etc.) which cannot be reduced by using more powerful quantum error-correcting codes. The logical error rate during the main part of the computation can be reduced arbitrarily far via the threshold theorem, meaning the encoding and decoding error will dominate the error rate, regardless of how big a computation we are performing.

This is unsatisfactory. Luckily, there is a simple solution. Most of the time, we are interested in solving a problem with a classical description, and getting a classical answer out at the end of the computation. Then we can start our quantum computation in a standard encoded state, such as $|\overline{00\ldots0}\rangle$, which can be reliably created via fault-tolerant state preparation. At the end of the computation, we take the state, still encoded, and measure it using fault-tolerant measurement, getting a classical answer. Indeed, the classical answer is also encoded in some classical error-correcting code at this point, perhaps just the repetition code, or perhaps something more complicated. Then classical processing is used to extract the output of the computation. Since the basic model for fault-tolerance assumes classical computation is noiseless, it is safe to decode the classical data. In this way, the logical error rate can be made arbitrarily low, using the techniques we'll discuss in the following chapters.

If we have a quantum computation which takes qubits as input and also outputs qubits, this won't work. If possible, we should demand to be given qubits which are already encoded, and insist on outputting encoded qubits as well. Possibly we'll want to switch codes to do our fault-tolerant computation using a different QECC than the one provided to us, but fault-tolerant techniques exist for switching between codes. If the code used for input and output qubits is not a very good code (e.g., small distance), there may still be some unavoidable error in the process of switching into and out of the better code used for fault tolerance, but the error will likely be less than if the input and output qubits were completely unencoded.

Some years ago, I was at a conference where a skeptic about quantum error correction tried to explain what he thought was a problem with QECCs by making an analogy. He pointed out the unavoidable error due to encoding and decoding, as described above, and said it was like getting out of a car in the rain — there is not enough space to open your umbrella in the car, so you must get a least a little bit wet while simultaneously getting out and opening the umbrella. The solution of starting and ending encoded was already known (the skeptic was behind the times), so I pointed out that there is an analogous solution to avoid getting wet: If you always make sure to park in a covered lot, you can get out of the car and open the umbrella where there is more space while still being protected from the rain. Once the umbrella is open, you can safely leave the covered area and walk around under the protection of the umbrella. Fault-tolerant protocols work the same way. Of course, you might have to walk a bit further if the covered lot is not right where you want to park, but there is indeed a cost to fault tolerance.

10.1.5 Fault Tolerant Simulations and the Components of a Fault Tolerant Circuit

So, putting this together, what do we want out of a fault-tolerant protocol? The basic idea is this: We are given a circuit, broken down into a universal set of gates. We'd like to replace the qubits of the circuit with the logical qubits of a QECC. Then we'd like to perform a sequence of gates on the qubits, without ever leaving the protection of the code, that transforms the logical qubits with the same unitary transformation as the original ideal circuit we were given. And we'd like the sequence of gates we use to be fault tolerant, in the sense that errors on a small but constant fraction of the gates do not interfere with the getting the correct outcome for the logical qubits.

We can describe what we want as a *fault-tolerant simulation* of the circuit. We replace each location in the original circuit with a fault-tolerant *gadget* that does the same thing to logical qubits as the location is supposed to do in the ideal circuit.

Definition 10.4. Given a particular type L of location, specified precisely (e.g., not just that it is a gate location, but also the type of gate), a *gadget for* L is a QECC Q coupled with a circuit G_L, such that if the qubits involved in L are encoded into Q, then subjected to the circuit G_L, then decoded, all without errors, the effect is the same as if L were performed directly. The circuit G_L may involve adding and/or discarding physical qubits, which may or may not also be encoded in Q.

In other words, a gadget for a specific function is supposed to perform that function on the encoded state. If the encoder for Q is $\mathcal{E}$ and the decoder is $\mathcal{D}$, then

$$\mathcal{D} \circ G_L \circ \mathcal{E} = L. \tag{10.5}$$

Definition 10.4 does not guarantee that the gadget is in any sense fault-tolerant; I'll discuss in section 10.2 what is needed for a gadget to be fault-tolerant. In the simplest examples, the QECC used in a fault-tolerant protocol has only one encoded qubit, and in the definitions below, I will assume that. If the location involves multiple qubits but each code block only encodes one logical qubit, each logical qubit should be encoded in a separate block of the QECC, and the gadget circuitry will interact the different blocks as required. Another option for dealing with multiple qubits per block is to distinguish between locations which interact qubits encoded in the same block and locations which interact qubits encoded in different blocks. Then we'll need different kinds of gadgets to deal with these two different kinds of locations, even when the gate involved is otherwise the same.

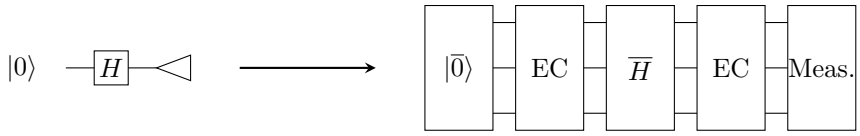


Figure 10.2: The fault-tolerant simulation of a circuit replaces each location with the corresponding gadget and puts error correction gadgets between each pair of other gadgets.

Gadgets can also be defined for other tasks, not directly corresponding to a location. In that case, the gadget is simply a QECC with a circuit. The proper functioning of the gadget has to be defined separately. The most useful sort of gadget that doesn't correspond to a location is an error-correction gadget. Obviously, an error-correction gadget is supposed to use the QECC to correct any errors pre-existing in the code, and a fault-tolerant error-correction gadget is supposed to do so even while new errors are occurring. The formal definition of a fault tolerant error-correction gadget is given in section 10.2.

Definition 10.5. A *fault-tolerant protocol* consists of a QECC Q along with the following types of gadgets:

- State preparation gadget for at least one type of state
- Measurement gadget for at least one type of measurement
- Gate gadgets for a universal set of gates
- Storage gadget
- Error correction gadget

The protocol may contain more than one gadget for a given type of location, and may contain some additional gadgets as well. All gadgets should satisfy some fault-tolerance criteria such as those given in section 10.2. Normally all gadgets associated with a given protocol use the same QECC; in cases where they do not, the protocol should also contain a code switching gadget that changes the QECC in which one or more logical qubits are encoded.

We don't usually bother explicitly describing a storage gadget, since a storage gadget can always be implemented by just putting a wait location for all physical qubits in the code. In principle, though, a fault-tolerant protocol could contain a different storage gadget involving non-trivial gates.

How do we put together gadgets in order to make a fault-tolerant circuit? Since each gadget for a location is supposed to perform the function of the location on the encoded state, obviously what we want to do is to replace each location in the original circuit with a corresponding gadget. However, that's not sufficient. In a noisy circuit, errors will occur, and in a large circuit, those errors will build up over time unless we are constantly performing error correction to eliminate them. The most common approach is to do a round of error correction as often as possible, namely between every adjacent pair of locations.

Definition 10.6. Let C be a circuit, broken down into locations as in definition 10.1. Then $FT(C)$, the *ideal fault-tolerant simulation of* C associated with a given fault-tolerant protocol, is the circuit created as follows: Take each location in C and replace it with the corresponding gadget of the fault-tolerant protocol, in the process replacing each qubit of C with a block of the QECC Q. After each state preparation gadget, gate gadget, or storage gadget, place an error correction gadget on each of the blocks of Q involved in the gadget. The *fault-tolerant simulation of* C is then $\widetilde{FT(C)}$. The *circuit size overhead* of $FT(C)$ is the number of locations of $FT(C)$ divided by the number of locations in C. The *space overhead* or *qubit overhead* is the ratio of the number of physical qubits involved in $FT(C)$ to the number of qubits involved in C, and the *depth overhead* or *time overhead* is the ratio of the depth of $FT(C)$ to the depth of C.

Note that we place error correction gadgets after the storage gadgets as well as after gate gadgets, since storage errors can occur even in wait locations. We don't need error correction gadgets after measurement gadgets, because the output of a measurement gadget is just classical information, and we are assuming that classical circuits don't suffer errors.

If the QECC used in the fault-tolerant protocol encodes $k > 1$ qubits per block, the fault-tolerant simulation must divide up the qubits of C into groups of k and encode them that way. The fault-tolerant protocol will probably then contain multiple different gadgets for a single two-qubit gate U, depending on whether U acts between qubits in the same block or between qubits in different blocks.

10.2 Formal Definition of Fault Tolerance

Now let us consider what we require for a fault-tolerant simulation to successfully simulate a low-noise logical computation using noisy physical gates. The definition of a gadget, definition 10.4, only requires that the gadget act correctly in the absence of errors. For a fault-tolerant protocol to truly tolerate faults, we'll need the gadgets to degrade gracefully in the presence of noise. If there are too many errors, we don't expect the gadget to work right, but if there aren't, there are two conditions we require: it should still behave correctly on the logical state, and it shouldn't propagate errors too badly.

The definitions in this section represent the most common notion of fault tolerance, but weaker or stronger definitions are possible as well, and have their place. Some protocols may need additional properties, requiring stronger definitions. On the other hand, when put together properly, gadgets satisfying weaker definitions can still achieve fault tolerance on the protocol as a whole.

10.2.1 Ideal Decoder and r-Filter

In order to define precisely what it means for a gadget to be fault tolerant, we'll need to talk about the logical state encoded in a QECC, and about the number of errors present in a noisy codeword. However, by just looking locally at a circuit, there is no way to tell what the "correct" encoded state is supposed to be. Luckily, we don't need to discuss that to define fault tolerance. A gadget is given some encoded state, which may be right or may be wrong, and it does something to that; we must check if the "something" is what we intended to do or not.

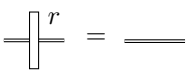
To talk about the number of errors in a state, we therefore only consider the number of errors relative to *some* codeword, not necessarily the "correct" one, since we don't know what the correct codeword is supposed to be. We can codify this by thinking of a projector:

Definition 10.7. The *r-filter* for the QECC Q is the projector on the subspace spanned by states of the form $E|\psi\rangle$, where $|\psi\rangle \in Q$ and E is an error of weight at most r. Pictorially, I will draw an r-filter as:

(10.6)

The doubled horizontal lines in this picture represent a block of the QECC Q rather than single physical qubits.

When we apply an r-filter to some state, we get a state which contains only errors of weight less than or equal to r. Thus, any codeword passes through the r-filter unchanged. Components of the state with more errors on them are dropped. The r-filter is thus a test we can apply to see if there are many errors on the state (relative to any codeword). If



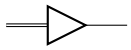
(10.7)

then the input state has r or fewer errors on it.

To talk about the logical state of a code, we should decode it. To do so, we would need to correct errors, and using a real circuit, the attempt at decoding would have a chance of introducing additional errors, and

the state we end up with might not be the same logical state as we had before we tried to decode. However, we're only *talking* about the logical state; we don't actually have to produce it. Therefore, we can imagine decoding using an ideal decoder that does not have any faults.

Definition 10.8. The *ideal decoder* for the QECC Q is the quantum channel that takes a state (possibly with errors) encoded in Q, corrects the errors, and then decodes the logical state, discarding all ancilla qubits. I will draw an ideal decoder as follows:



(10.8)

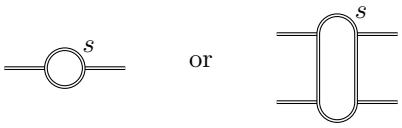
The doubled horizontal line on the left of the picture represents a block of Q. The single horizontal line on the right of the picture is the logical qubit after decoding.

For now, it is sufficient for the decoder to keep only the logical qubit. Later on, we'll also want to consider decoders that keep the syndrome information they extracted during error correction.

10.2.2 Fault-Tolerant Gate Properties

We're now ready to define what it means for a gate gadget to be fault tolerant. Using the pictorial language we started to develop above, we'll be able to draw two pictures that define properties that a fault-tolerant gate gadget must satisfy. For the purposes of these definitions, we can include storage gadgets as gate gadgets. Just think of them as performing the identity gate.

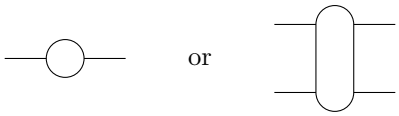
Definition 10.9. The picture to represent a noisy gate gadget with exactly s faults is



(10.9)

-for one- and two-qubit gates, respectively. Each doubled horizontal line represents a single block of the QECC.

The picture to represent an ideal gate performed on unencoded qubits without noise is



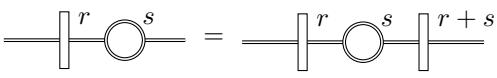
(10.10)

for one- and two-qubit gates, respectively. Each single horizontal line represents a single unencoded qubit.

In an independent stochastic noise model, these pictures can be interpreted as completely positive maps. For some particular realization of the noisy gate gadget circuit, there are some number s of faults specified by a fault path S. Replacing the locations in S with the CP maps corresponding to faults in the noise model (usually we will want to rescale to make the maps trace-preserving, if possible) gives a linear map corresponding to the picture. The picture for a gadget with s faults thus represents many different maps, depending on the exact locations which are faulty and on the precise error model.

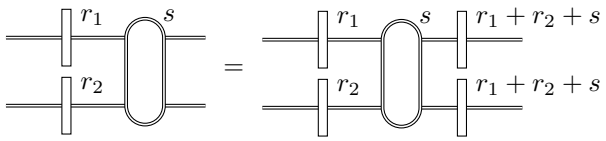
The ideal decoder and r-filter are also CP maps; the decoder is trace preserving, but the r-filter is not. We can compose them with the CP map corresponding to a realization of a gate gadget to get more CP maps, and the properties for fault tolerance will be defined as equations relating CP maps derived this way. We can thus represent such equations simply as pictures.

Definition 10.10 (FT Gate Error Propagation Property)**.** Suppose we have a gate gadget associated with a QECC with distance $2t + 1$ (i.e., it corrects t errors). If it is a single-qubit gate gadget, it satisfies the *FT Gate Error Propagation Property*, abbreviated *GPP*, if, whenever $r + s \leq t$,



(10.11)

A two-qubit gate gadget satisfies the GPP if, whenever $r_1 + r_2 + s \leq t$,



(10.12)

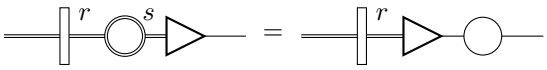
For both equations, the faulty locations on both sides of the equation have the same fault path and error maps substituted in.

These equations use filters on the left to ensure that the input state to the CP maps can be considered to be at most r errors away from a codeword. The equations then demand that, given such an input state, the state exiting the gadget be at most $r + s$ or $r_1 + r_2 + s$ errors away from a codeword. In the case of the two-qubit gate gadget, we allow errors to propagate between the two blocks of the code. Thus, even if $s = 0$, so the gadget itself is faulty, the number of errors in a block may increase as a result of error propagation. However, we insist that the number of errors ending up in a single block is limited by the total number of errors $r_1 + r_2$ input in the two blocks plus the number s of new faults. This is the sense in which this property limits the error propagation.

Note that for the two-qubit gate gadget, the condition uses *separate* filters on the two blocks of the code involved in the gadget. Thus, the final state of the two blocks taken together might have a total of $2(r_1 + r_2 + s)$ errors on it.

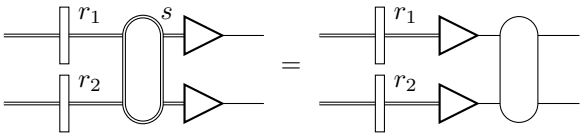
For the Gate Error Propagation Property, we only impose the equations when the total number of errors on input states plus faults in the gadget is at most t. When there are more than t total errors floating around, we don't in general insist that our gadgets have to work correctly. In many cases, fault-tolerant gate constructions do continue to satisfy these equations even when there are more errors, but that is an additional property not required by the basic definition.

Definition 10.11 (FT Gate Correctness Property)**.** Suppose we have a gate gadget associated with a QECC with distance $2t+1$. If it is a single-qubit gate gadget, it satisfies the *FT Gate Correctness Property*, abbreviated *GCP*, if, whenever $r + s \leq t$,



(10.13)

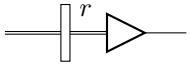
A two-qubit gate gadget satisfies the Gate Correctness Property if, whenever $r_1 + r_2 + s \leq t$,



(10.14)

These equations must hold for any choice of CP maps substituted in for the faulty locations on the left-hand side.

What do these conditions mean? They say that if we perform the noisy gate gadget and then decode, we get the same thing as if we decode and then perform the gate gadget. The combination



(10.15)

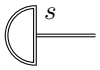
extracts the logical state of the input after limiting the number of errors. The RHS of each equation thus outputs the state we should get by doing an ideal gate on the logical state. The LHS is the state we actually get. The Gate Correctness Property thus ensures that a noisy gate gadget performs the correct operation on the encoded state, generalizing the definition of a gadget to the case in which there are errors.

Again, we only insist that the condition hold when the total number of errors floating around is at most t. In this case, that is usually all we can hope for. That is because if there is any arrangement of errors that allows all $r_1 + r_2 + s$ errors to concentrate in a single block, on the LHS, the concentration will happen, possibly giving a logical error which is not corrected by the ideal decoder, whereas on the RHS, all the errors are corrected immediately before there is any opportunity for propagation.

10.2.3 Fault-Tolerant Preparation and Measurement Properties

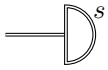
The properties for fault-tolerant state preparation and measurement gadgets are similar:

Definition 10.12. The picture for a single-qubit state preparation gadget with exactly s faults is



(10.16)

The picture for a single-qubit measurement gadget with exactly s faults is



(10.17)

In both cases, the doubled horizontal lines represent encoded blocks of the QECC. The measurement gadget outputs classical information, generally a single bit.

The pictures for ideal preparation of a single-qubit state and ideal single-qubit measurement are

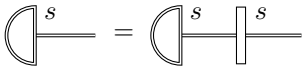
(10.18)

The horizontal lines represent single unencoded qubits, and the ideal measurement outputs a classical bit.

Again, these pictures represent CP maps. A noisy preparation gadget is a CP map that takes no input and outputs a state in the Hilbert space of the QECC. A noisy measurement gadget is a CP map that takes a noisy codeword as input and outputs a classical bit.

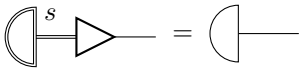
For the fault-tolerant preparation gadgets, there is no input, so we don't need to precede them by filters.

Definition 10.13 (FT Preparation Error Propagation Property)**.** Suppose we have a preparation gadget associated with a QECC with distance $2t + 1$. The gadget satisfies the *FT Preparation Error Propagation Property*, abbreviated *PPP*, if, whenever $s \leq t$,



(10.19)

Definition 10.14 (Preparation Correctness Property)**.** Suppose we have a preparation gadget associated with a QECC with distance $2t+1$. The gadget satisfies the *FT Preparation Correctness Property*, abbreviated *PCP*, if, whenever $s \leq t$,

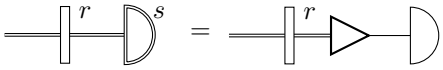


(10.20)

In other words, when $s \leq t$, a preparation gadget with s faults in it should output a state with no more than s errors in it, and the state should be an encoding of the correct state.

For the measurement gadget, we only have one property. We don't need to worry about error propagation through a measurement gadget because the output only involves classical information, which we are assuming is error-free.

Definition 10.15 (FT Measurement Correctness Property)**.** Suppose we have a measurement gadget associated with a QECC with distance $2t + 1$. The gadget satisfies the *FT Measurement Correctness Property*, abbreviated *MCP*, if, whenever $r + s \leq t$,



(10.21)

That is, if the initial state has no more than r errors, and there are not too many errors in the measurement gadget, the classical outcome of the measurement should be the same as if we did an ideal measurement on the decoded state.

10.2.4 Fault-Tolerant Error Correction Properties

Fault-tolerant error correction gadgets are a bit different from gadgets associated with locations. For other gadgets, we are concerned that error propagation is not too severe, but error correction gadgets are supposed to actually *reduce* the number of errors in the state. Of course, if there are faults in the error correction gadget itself, they may introduce more errors that must be corrected in a later error correction gadget.

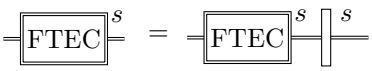
Definition 10.16. A fault-tolerant error correction gadget (which I will usually abbreviate FTEC in the future) is represented by the following picture:

(10.22)

The horizontal lines represent blocks of the QECC.

As usual, we can interpret this picture as a CP map for any particular realization of the noisy circuit. Error correction gadgets invariably involve adding extra ancilla qubits and produce syndrome information. The syndrome information may either be in the form of classical bits or as extra qubits, and is usually discarded at the end of the gadget.

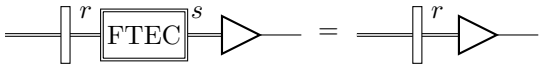
Definition 10.17 (FT Error Correction Recovery Property)**.** Suppose we have an error correction gadget associated with a QECC with distance $2t + 1$. The gadget satisfies the *FT Error Correction Recovery Property*, abbreviated *ECRP*, if, whenever $s \leq t$,



(10.23)

Note that the ECRP does not involve a filter on the input state. This is important: We want our FTEC gadgets to return us to a codeword (or a codeword with s errors when there are s faults in the FTEC gadget) no matter how many errors there are to start with. Otherwise, if we ever accumulate too many errors in a block, the rest of the circuit is a loss. When there are more than t errors in a block, we may not be able to decode correctly, but at least we'd like the remaining gadgets after the FTEC to do the right thing, even if it's on the wrong logical operator. That localizes the logical error to a small number of gadgets. In particular, when we prove the threshold theorem, we'll use concatenated codes to get more and more accuracy, but that won't work if sub-blocks of the code aren't brought back to *some* codeword, even if it is not the correct one.

Definition 10.18 (FT Error Correction Correctness Property)**.** Suppose we have an error correction gadget associated with a QECC with distance $2t + 1$. The gadget satisfies the *FT Error Correction Correctness Property*, abbreviated *ECCP*, if, whenever $r + s \leq t$,



(10.24)

This property says that when the total number of errors (input errors plus faults in the gadget) is at most t, the encoded state does not change during an FTEC gadget.

10.2.5 Fault Tolerance Properties for Different Error Models

In the basic model for fault tolerance, we allow the faults in a gadget to be general CP maps, but ones which are independent between different locations. However, if you look at the definitions of the fault tolerance properties given in this section, you'll see that all of them are linear equations. That means that a version of theorem 2.4 applies in this case as well:

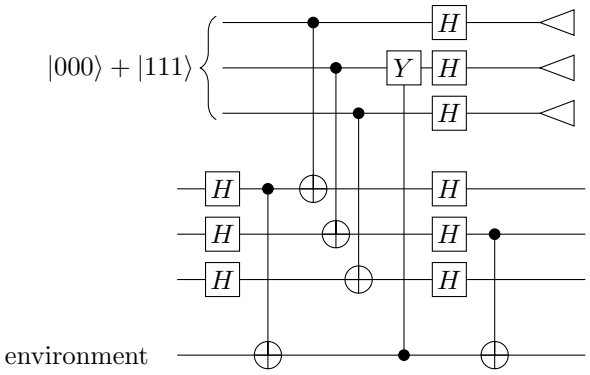


Figure 10.3: An example of a gadget interacting with a persistent environment on a fault path of size 3.

Theorem 10.1. *Consider a realization C_1 of a noisy gadget circuit with a fault path of size s, and suppose that the location of each fault is replaced by a linear Hilbert space operator (e.g., specific Kraus operators of the CP maps). Let C_2 be C_1 with one particular faulty location L replaced by a different linear Hilbert space operator. Suppose the operator for L in C_1 is E and the operator for L in C_2 is F, and let C_3 be the circuit realization C_1 with L replaced by $\alpha E + \beta F$, for any complex numbers α and β. If the gadget satisfies the relevant fault tolerance properties for realizations C_1 and C_2, then it also satisfies them for C_3.*

As a consequence of this, we need not check the fault tolerance properties for arbitrary CP maps inserted for faults. It is sufficient to check them for a basis of errors. In particular, Pauli errors suffice.

Corollary 10.2. *If a gadget satisfies the relevant fault tolerance properties when arbitrary Pauli errors are inserted for all faults, then it satisfies the fault tolerance properties when arbitrary CP maps are inserted for faults.*

Recall that a Pauli error model means that the location acts correctly and then afterwards there is a Pauli error, so when I say a "Pauli error is inserted" at a location, I mean that it is inserted after the correct action of the location (unless it is a measurement location, in which case the error flips the outcome of the measurement). This corollary is very convenient, since it is much easier to check the properties for just Pauli errors than for general CP maps.

The logic behind theorem 10.1 has a more far-reaching consequence than just convenience. We could also take the superposition when we change the faults in multiple locations, and the result would still work. This is not necessary in a basic model for fault tolerance, but it allows us to talk about certain correlations between errors, and will actually be needed for proving the threshold theorem. Indeed, once we have checked the fault tolerance properties for Pauli errors, we know they hold under the most general error that can happen to a circuit with faults in those s specific locations.

Theorem 10.3. *Consider a realization of a gadget with a particular fault path S, $|S| = s$. Suppose a gadget satisfies the relevant fault tolerance properties when arbitrary Pauli errors are inserted at the locations of S. Then it also satisfies the fault tolerance properties when subjected to a persistent environment which starts in an arbitrary state (possibly even one entangled with the qubits involved in the gadget), and interacts with the locations of S through arbitrary unitaries, as illustrated in figure 10.3.*

This picture allows for arbitrary dynamics within the environment, since the unitaries implementing the self-interaction can be absorbed into the unitaries interacting the system and environment.

We haven't even defined the fault tolerance properties in this case, but it is straightforward to do so by extending the pictures to include the environment and then tracing over it at the end of each picture. We don't put a restriction on the initial state of the environment, but consider it as extra input qubits for the circuit. For instance, see figure 10.4. The resulting equations are again equalities of CP maps; indeed, since

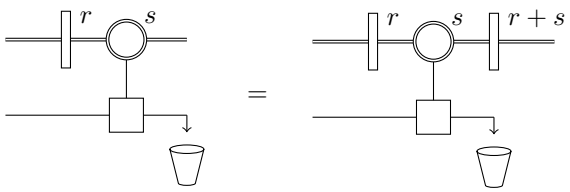


Figure 10.4: The one-block GPP with an environment. The s locations of the fault path within the gate gadget are arbitrary interactions with the environment register (bottom line), and the left-hand side and right-hand side have the same interactions.

the interactions with the environment are unitary, we can consider the picture to represent a linear operation on the Hilbert space. The gadget may involve measurement, so to make a linear Hilbert space operator, we purify each measurement, replacing the classical computations afterwards with quantum gates (with no additional faults). The filters that appear in the pictures are projections, so the operators we get will not be unitary.

Because we don't care what happens to the environment, we need not compare the environment on the LHS of each equation with the environment on the RHS of each equation. Indeed, in the various correctness properties, the environment on the LHS interacts with the faults in the gadget, whereas there *are* no faults on the RHS. The final state of the environment therefore differs for the LHS and RHS of the correctness properties.

Proof. Let us label the interactions between the environment and the faulty locations in the circuit as $U_1, \ldots, U_s$, and say that the ith faulty location L_i involves g_i qubits. We can write

$$U_i = \sum_{P \in \hat{\mathsf{P}}_{g_i}} P \otimes V_{i,P}, \tag{10.25}$$

where $V_{i,P}$ acts on the environment qubits. Since U_i is unitary,

$$\sum_{P,Q \in \hat{\mathsf{P}}_{g_i}} PQ \otimes V_{i,P}^\dagger V_{i,Q} = I \otimes I, \tag{10.26}$$

which means

$$\sum_{P \in \hat{\mathsf{P}}_{g_i}} V_{i,P}^\dagger V_{i,P} = I \tag{10.27}$$

$$\sum_{P \in \hat{\mathsf{P}}_{g_i}} V_{i,P}^\dagger V_{i,PQ} = 0 \quad \forall Q \in \hat{\mathsf{P}}_{g_i} \setminus \{I\} \tag{10.28}$$

Let M be the operator corresponding to a picture involving the environment, with interactions U_i in place. Let $M_{\{P_i\}}$ be the operator corresponding to the same picture, but where U_i is replaced by $P_i \otimes V_{i,P_i}$. Then

$$M = \sum M_{\{P_i\}}. \tag{10.29}$$

That is, the operator corresponding to the picture with unitary interactions with an environment can be written as the sum of operators corresponding to different Pauli fault paths.

In the FT error propagation properties and the FT error correction recovery property, both sides of the equation involve the same set of faults in the system. By the hypothesis, the LHS and RHS match when we substitute Pauli errors, resulting in pictures $M_{\{P_i\}}$. Therefore, the sums M match term by term, so the totals match as well.

For the various FT correctness properties, let $M = \sum M_{\{P_i\}}$ be the LHS of the equation. The LHS with Pauli errors in the faulty locations and V_{i,P_i} in the corresponding environment locations matches the RHS side picture, which is a tensor product between the system qubits and the environment (still with V_{i,P_i} acting in appropriate places). The total LHS M is therefore equal to the RHS picture tensored with a sum of the V_{i,P_i}s acting on the environment.

The only question is whether the trace over the environment gives a different value for the LHS and RHS. We can calculate this by looking at what happens to the environment's trace for each faulty location L_i. The environment undergoes

$$\sum_{P \in \hat{\mathsf{P}}_{g_i}} V_{i,P} \tag{10.30}$$

at interaction location i, and its trace becomes

$$\operatorname{tr} \rho \left(\sum_{P,Q \in \hat{\mathsf{P}}_{g_i}} V_{i,P}^\dagger V_{i,Q} \right) = \operatorname{tr} \rho \tag{10.31}$$

by equations (10.27) and (10.28). □

As a consequence of theorem 10.3, once we fix the locations of the faults in a circuit, anything, including horrible correlated errors, can happen at the faulty locations, and the fault tolerance properties will still pull us through, allowing things to work correctly. However, the theorem does *not* immediately allow us to generalize the independent stochastic error model to arbitrary correlated models. We'll want to calculate the probability of having a logical error, and that involves summing over multiple fault paths. The probability of having a particular fault path is not even well-defined in more general error models, so we'll need to be careful.

10.3 Statement of the Threshold Theorem for the Basic Model

The main goal of part II is to prove the threshold theorem. This is one of the key results in the theory of quantum information. If it were not true, it would probably be impossible to build big quantum computers. The threshold theorem sets a definite goal for experimentalists hoping to build a scalable quantum computer: It says that if you can make the basic components of the computer good enough, and can make enough qubits, then that suffices, and error-correcting codes and fault-tolerant protocols exist that can take you the rest of the way. The point here is that "good enough" is a value independent of the algorithm you want to run, so you don't need to continue to improve your experimental error rates to solve a new, bigger problem. (Of course, you may need more qubits to solve a bigger problem.)

Theorem 10.4 (Threshold Theorem for Basic Model of Fault Tolerance)**.** *Suppose a system satisfies a basic model for fault tolerance, with $p_P = p_G = p_S = p_M = p$. Then there exists a family of fault-tolerant protocols $\mathcal{F}_l$ and a threshold error rate p_T such that, if $p < p_T$, then for any ideal quantum circuit C which starts with preparation locations and ends with measurement locations, and any $\epsilon > 0$, there exists an l such that the output distribution of $\widetilde{FT_l(C)}$, the noisy fault-tolerant simulation of C by $\mathcal{F}_l$, has statistical distance at most ϵ from the output of C. When C has T locations, then $FT_l(C)$ has $O(T \operatorname{polylog}(T/\epsilon))$ locations (i.e., the circuit size overhead is $O(\operatorname{polylog}(T/\epsilon))$.*

In other words, if the physical error rates for all locations are below the magical threshold error value p_T, there exists a fault-tolerant protocol that makes the logical error rate on the complete circuit as small as we like. Note that we actually need a *family* of FT protocols to make this work — as the circuit gets bigger, or the logical error rate we'd like to achieve gets smaller, than we need to work harder at getting rid of errors, which means more overhead and a more involved FT protocol. The nice thing about the threshold theorem is that the extra work needed scales reasonably, only as a polynomial in the log of T/ϵ. Scaling up to a really really big quantum computer is a lot of work even for an ideal circuit, but adding fault-tolerance

to the circuit does not require much more effort than it does to add fault-tolerance to a merely big quantum computer.

Unfortunately, while the asymptotic scaling for large T/ϵ is good, the actual amount of overhead needed is still pretty large. The problem is that the constants hidden by big-O notation are significant. Even relatively efficient fault-tolerant protocols have a space overhead of hundreds, thousands, or even tens of thousands of physical qubits per logical qubit, and many are much worse than that. While it's not mentioned explicitly in the theorem, the overhead is also dependent on p/p_T, so it helps to get the physical error rates down as much as possible, but even then the costs are high. One approach is to abandon the notion of a family of protocols with a threshold, and just pick a specific FT protocol that works for the size of the circuit you are interested in. By carefully choosing a protocol, you may be able to get the overhead down, although with existing protocols, it is still likely be at least a few dozen physical qubits per logical qubit. And of course, there may exist fault-tolerant protocols with much lower overhead than any we know of today.

The threshold theorem is proven using concatenated codes, introduced in section 9.1. The basic idea is straightforward: Given a fault-tolerant protocol $\mathcal{F}$, we can take any circuit and produce the fault-tolerant simulation $FT(C)$ of that circuit. Provided the error rate p per location is small enough, the logical error rate for the FT simulation should be less than the physical error rate, and we have improved matters by using the fault-tolerant simulation. Then we do it again, and take the FT simulation of the simulation, i.e., $FT(FT(C))$. That improves the error rate some more. With multiple iterations, we can drive the error rate down very rapidly. The family of fault-tolerant protocols $\mathcal{F}_l$ that we need for the threshold theorem is then the fault-tolerant protocol $\mathcal{F}$ concatenated l times. The result is really an FT protocol that uses an l-level concatenated code. I will explain how to make this intuitive argument precise in chapter 14. The main challenge is to properly define and determine the "logical error rate" for a fault-tolerant simulation.

Topological codes, discussed in chapter 19, can also be used to get a threshold, as can families of low-density parity check codes (LDPC codes) with a constant rate. It may be other families of codes exist that can give a threshold. We sometimes talk about the threshold for a specific code family, and sometimes just about "the threshold" for fault tolerance. In the latter case, we are talking about the best (i.e., highest) threshold p_T optimized over all possible families of codes. In addition, different fault-tolerant protocols for the same code family can lead to substantially different "threshold" values, but the real threshold is defined using the best possible FT protocol.

The threshold theorem can be extended in various ways. One obvious way, still sticking to a basic model for fault tolerance, is to let the error rates for different types of locations be different. In that case, the threshold is no longer a single number, but a surface in the 4-dimensional space of error rates (p_P, p_G, p_S, p_M). The threshold surface separates the noiseless point $(0, 0, 0, 0)$ from the completely noisy point $(1, 1, 1, 1)$, and the extended theorem then says that we can achieve error rate ϵ for the outcome distribution for any point on the noiseless side of the surface with polylogarithmic overhead. It is also possible to go beyond a basic model by loosening or removing some of the assumptions. Not all of the assumptions can be relaxed; which ones can and which ones can't is discussed in chapter 15. Even when it is possible to derive a threshold with relaxed conditions, the actual numerical value of the threshold may decrease significantly, or the overhead may increase, or both. Building a fault-tolerant quantum computer will therefore likely involve making a number of trade-offs in order come up with an implementation for which the error rate is below the relevant threshold value.

10.4 Different Ways of Computing the Threshold and Overhead

Since the threshold error rate determines whether a given experimental implementation will or will not be useable for fault-tolerance, a great deal of effort has gone into determining the actual value of the threshold for various code families. Likewise, the overhead (of every kind) determines how many resources will be needed for a given computation, and has also been a subject of intense study. However, comparing threshold and overhead values you might find in the literature is not completely straightforward because of different techniques used to derive them. First all, remember that for a specific code family, it might be possible to come up with better FT gadgets that give a better threshold value or lower overhead. Secondly, to be

fair, you should only compare estimates derived using comparable techniques, since different techniques have different sources of inaccuracy.

An estimate of the threshold and overhead will generally involve some combination of rigorous proof, simplifying assumptions, and statistical sampling. Pretty much every approach uses some simplifying assumptions — indeed, by working with a basic model of fault tolerance, you are already making a number of simplifying assumptions — but some approaches go further. It is perhaps worth distinguishing between assumptions about the quantum computer (e.g., assuming an independent stochastic error model) and assumptions about the calculation (e.g., assuming that a particular type of state preparation gadget can be performed reliably under conditions of interest). The former type of assumption gives you a complete threshold estimate, albeit one that only applies under the given conditions.

Assumptions about the calculation are a bit more tricky to evaluate: Sometimes the assumption is a conservative one, in which an effect which may lower the threshold value is simplified to have the worst possible effect it can have. An approach which only uses conservative assumptions will give a threshold value that is definitely lower than the true value and an overhead that is higher than the true value by an unknown amount. The advantage of this is that we can be sure we have a lower bound on the threshold and an upper bound on the overhead. Sometimes the assumption is a reasonable approximation, in which case we may expect that the calculated threshold and overhead are roughly the same as it would be without the assumption, maybe slightly higher or lower. However, as long as the approximation hasn't been fully checked, there is a chance it is wrong, so we cannot have full confidence in any value derived with the assumption. Occasionally, the assumption is obviously an over-optimistic one, giving a significant *overestimate* of the threshold value. Optimistic assumptions are sometimes used to tease out how much of the threshold error rate comes from different effects, or to estimate upper bounds on thresholds for a code family. It is important not to think of thresholds calculated using optimistic assumptions as actual threshold values, since they don't provide any kind of guarantee for an experimentalist achieving that error rate.

One particularly widespread example of an optimistic assumption is a *phenomenological error model*. In a phenomenological error model, there is an error rate per physical qubit per time step (regardless of gates) and an error rate on each bit of the error syndrome that is measured. The phenomenological error model is a greatly simplfied model which still includes constantly occurring errors and imperfect error syndrome measurements. It is used to get a quick handle on the general performance of a QECC, syndrome extraction, and syndrome decoding method without having to design and analyze detailed fault-tolerant circuits. Usually this is done primarily to learn about the threshold and not for overhead. A threshold in a phenomenological model should *not* be confused with the threshold derived from a full circuit model; they are not at all comparable. It makes some sense to compare phenomenological thresholds for different FT protocols with each other, but even there, caution is needed, as the relationship between the phenomenological threshold and the circuit threshold can vary between codes and even between different FT protocols for the same code. For instance, the number of gates needed to extract a bit of the syndrome in a full circuit analysis can be greatly different between different protocols, but a phenomenological noise model completely ignores this important contribution to the threshold.

The next big distinction is between rigorous proofs, simulations of FT protocols using statistical sampling, and analytical estimates. Rigorous proofs use only conservative assumptions about the calculations and carefully justify any assumptions needed to be sure they actually are conservative. Consequently, a rigorous proof provides a lower bound on the threshold and upper bound on the overhead given the stated assumptions about the computer.

Simulations, in contrast, simulate a big chunk of a FT circuit on a classical computer and run it many times, randomly generating errors according to the appropriate error model and calculating the logical error rate for the run. Classical simulations of general quantum circuits are unlikely to be possible, but as you'll see, the main parts of many FT protocols are composed just of Clifford group gates, and simulating Clifford circuits with Pauli errors can be done efficiently using procedure 6.4. In some cases, additional properties of the FT circuit and QECCs used can lead to additional simplifications of the simulation. Simulations invariable require some assumptions which may lead to good approximations, but are not necessarily conservative ones. Consequently, a well-done simulation can give more accurate threshold and overhead estimates than a rigorous proof, but we can never be quite certain if the true values are higher or lower than the simulated

values.

An analytical estimate focuses on formulas for the performance of a fault-tolerant protocol. They may be derived in part from simplifying assumptions and may use empirical results from simulations to determine constants in the formulas. The purpose of an analytical estimate is clarify the role of certain parameters and to develop some insight into what changes when they are varied. For instance, you might approximate the logical error rate or overhead of a protocol as a function of p/p_T in order to understand what can be gained by improving the physical error rate p in the quantum computer.

Note that some proofs also use statistical methods as part of the threshold calculation. In this case, there is some statistical error, but the size of the error is controlled and known, so the technique comes with a guarantee that we cannot be much above the true threshold value, in contrast with simulations, which cannot have such a guarantee.

Another distinction between proofs and simulations comes in the error models used. Often, rigorous proofs use an adversarial error model within the basic model for fault tolerance. That is, they work with the worst possible error model consistent with a given value of the error rate p. Indeed, as we'll see in chapter 14, the threshold proof actually uses an even broader error model which involves some potential correlation between qubits. Simulations usually have to work with some particular Pauli error model. Frequently X, Y, and Z errors are given the same probability, so the error model is related to the depolarizing channel. Thus, proofs generally apply to a broader range of quantum computers than simulations.

There is often a difference of an order of magnitude or more between the rigorously proven value of the threshold for a given family of codes and that derived using simulations. For instance, the best known threshold value comes from Knill's technique of using concatenated error-detecting codes to prepare ancilla states (see section **??**). Simulations of the technique suggest a threshold error rate of a few percent, say 3%, depending on the error model, whereas the best existing proof for this approach shows that the threshold is at least 10^{-3}, thirty times smaller. Most likely, the simulations are closer to the correct value, but as noted above, we cannot have full confidence that they are underestimates and not overestimates.

10.4.1 Tips For Comparatively Evaluating Fault-Tolerant Protocols

FT protocols are complicated things, with many different components. Moreover, the various different aspects of an FT protocol can interact in complicated and unexpected ways, so, for instance, a gate gadget or FTEC technique that works well for one protocol can be poorly suited for another protocol even if the QECC is the same. That makes it challenging to determine which of the many existing FT techniques is a good choice for a protocol you are designing.

Today, most comparative information about the relative strengths and weakness of different FT protocols comes from simulations. In the early days of fault-tolerant quantum computation, there were a number of unreliable simulations of the threshold. The methodology has improved, but if you are not careful, it is easy to find yourself making a misleading comparison. Here is a checklist to think about as you go about deciding between FT protocols.

- *There is no "best":* It's rare to have an FT technique which, in isolation, is uncategorically *better* than another. As I have already noted, the different components of an FT protocol interact, and so they need to be chosen to be well-suited to each other. If you go through a list and try to pick the "best" QECC, the "best" state preparation method, and so on, without seeing how they work together, you will not even end up with a functional protocol, and certainly not the "best" protocol that could be devised. Some techniques might be *usually* worse than others, but perhaps in the right context, they can be the optimal solution to some problem. I tend to view the array of FT techniques in the literature as a toolbox, and you want to pick a tool that is right for the job you are actually trying to do.

- *Check the assumptions:* Make sure to read carefully what assumptions and simplifications are made for the simulations (or proofs or analytic estimates) you are relying on for the comparison. Certainly, you should be careful about comparing a threshold estimated via simulation to one bounded below by a proof. Also, pay close attention to things like use of a phenomenological error model. In general,

you should be aware of what error model is being used in a simulation; a comparison between two simulations with different error models may be misleading.

- *Check which aspects of the protocol are included:* It has become common to cite threshold values based on simulations of a pure storage scenario, that is, one for which only a FTEC gadget is run repeatedly, and not any gate gadgets or even preparation or measurement gadgets. While this makes comparison between two such threshold values relatively straightforward, it can sometimes overestimate the threshold by an amount that may differ between protocols. Generally, simulating only storage is reasonable because FTEC consumes a large fraction of the error budget of an FT protocol, but sometimes there are additional reductions to the threshold that come from other gadgets. Overhead computations are sometimes for storage only and sometimes for full protocols. Be sure of which you are looking at, and only compare two overhead calculations if they are calculating the same thing. Also, be aware that different protocols for FTEC might constrain the rest of the protocol in ways that produce a very different overhead for the full protocol even when overhead for pure storage is similar.

- *Consider the tradeoffs:* Sometimes the tradeoffs between FT protocols are reasonably straightforward, such as one protocol with a higher threshold vs. a second with a lower overhead. But there are other properties which can also be important, including connectivity of the physical qubits (for instance, 2-dimensional nearest-neighbor gates vs. long-range gates), speed of the syndrome decoding algorithms, behavior under different error models, and more. In any given implementation, some of these properties may be more salient than others. A good understanding of the full range of properties of the FT protocols under consideration will help you choose one that is most appropriate for the specific implementation you have in mind (if any).

- *Compare the degree of optimization:* Finally, be aware that some protocols have been intensely studied and have undergone a high degree of optimization to improve performance in all aspects. By comparison, a newly proposed protocol may look inefficient or seem to have too many requirements. But that doesn't mean it is useless — it may be that with further study, or in conjunction with other new fault-tolerant tricks, the new protocol can become competitive with the old one. Again, keeping a large toolbox of FT techniques allows us to study different combinations which may allow apparently inferior approaches to shine.

Chapter 11

If Only It Were So Easy: Transversal Gates

This chapter and the next two chapters (chapters 12 and 13) will explain how to create gadgets that satisfy the fault-tolerance properties we defined in chapter 10. We'll start with the simplest case, that of transversal gates. You already saw an example of a transversal gate in section 10.1.3, so you know how straightforward they are. There's not much to be said about performing transversal gates, so in this chapter, I'll mostly focus on the question of figuring out which gates can be performed transversally for a given QECC.

Since transversal gates are so nice, we generally like to build FT protocols using codes for which many gadgets can be performed transversally. The 7-qubit code is particularly nice, as measurement and all Clifford group gates can be performed transversally. Sadly, to get a full universal set of gates, we need to add something non-transversal. This is not just a property of the 7-qubit code, but is true for any QECC. If it were not true, the theory of fault tolerance would be a lot simpler.

11.1 What is a Transversal Gate?

11.1.1 Definition of Transversal

Transversal gates are constructed so that they automatically satisfy the FT Gate Error Propagation Property. In particular, a transversal implementation of a gate gadget should never change the weight of a pre-existing error within a single block of the QECC. One way to achieve this is by just using single-qubit gates, but we can also talk about multiple-qubit transversal gates, using the following definition:

Definition 11.1. Let $\mathcal{F}$ be a quantum operation acting on a Hilbert space $\mathcal{H}_{2^n}^{\otimes m}$, interpreted as m blocks each consisting of n qubits. Then $\mathcal{F}$ is *transversal* if $\mathcal{F} = \bigotimes_{i=1}^{n} \mathcal{G}_i$, where $\mathcal{G}_i$ acts on the 2^m-dimensional Hilbert space composed of the ith qubit from each of the m blocks.

Of course, this definition can easily be extended to work for qudits as well as for qubits. Since we're interested in fault-tolerance, we invariably consider situations where each block of n qubits forms a QECC. Almost always, all m blocks are separate blocks of the same QECC. The operation $\mathcal{F}$ is frequently a unitary, but we don't require that, and occasionally we want to talk, for instance, about transversal measurements.

Often, all the $\mathcal{G}_i$ are the same gate, but I will not require that for the definition of transversal. If all $\mathcal{G}_i$ are equal to $\mathcal{G}$, I may describe the gate $\mathcal{F}$ as a *transversal* $\mathcal{G}$ to indicate its construction. (The phrase "bitwise $\mathcal{G}$" is also sometimes used.) Because it is the most common case, sometimes in the literature, you will find the notion of transversal gates restricted to having all $\mathcal{G}_i$ identical, but there is no real reason to do so.

A gate consisting of a tensor product of single-qubit gates certainly is transversal, with $m = 1$, but it is also possible to have transversal gates with $m > 1$. For instance, consider figure 11.1, illustrating the transversal CNOT on two blocks of 7 qubits each.

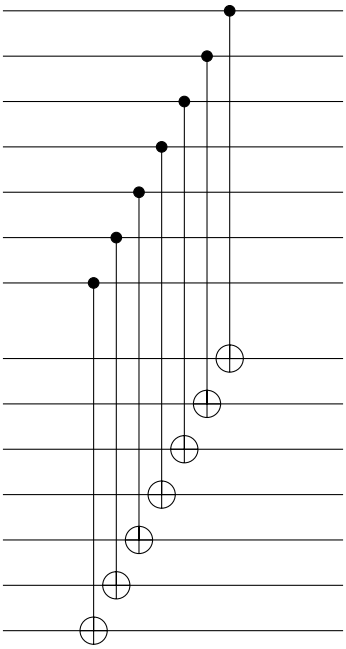


Figure 11.1: The transversal CNOT gate acting on two blocks of the 7-qubit code.

11.1.2 Transversal Gates Satisfy the FT Gate Properties

The point of this definition is that a transversal gate will always satisfy the GPP. The GCP requires a little more work, but actually, all the work just goes into checking that the transversal gate is in fact a gadget for the desired type of location — in other words, that it maps codewords correctly to codewords.

Proposition 11.1. *Suppose the circuit $\mathcal{F}$ is a gate gadget for the QECC Q and $\mathcal{F}$ is transversal. Then $\mathcal{F}$ satisfies the GPP and the GCP.*

Note that the definition of gadget actually requires that we specify a circuit, not just a quantum operation. This is an important distinction, since different circuits for the same quantum operation can produce different results in the presence of noise. In the case of the proposition, assume that the circuit implementing $\mathcal{F}$ is one that directly implements the decomposition required by the definition of transversal.

Proof. We will consider the case where $\mathcal{F} = U$ is unitary; non-unitary $\mathcal{F}$ can be handled by purification. Since U is transversal, $U = \bigotimes_{i=1}^{n} V_i$. Suppose $|\psi\rangle$ is the input to the gadget, and is a state on mn qubits which is produced by putting an r_a filter in front of the ath block of n qubits. Then $|\psi\rangle$ is a superposition of codewords with errors of weight at most r_a on block a,

$$|\psi\rangle = \sum_j E_j|\psi_j\rangle \quad (\mathrm{wt}\, E_j|_{\mathrm{block}\ a} \leq r_a), \tag{11.1}$$

with $|\psi_j\rangle$ a possibly entangled state of valid codewords on all m blocks.

Suppose the faulty circuit for U has errors at s locations given by the fault path S. By corollary 10.2, it suffices to check the case where the errors are Pauli errors. Let $\mathcal{H}_i$ be the Hilbert space $\mathcal{H}_{2^m}$ composed of the ith qubits of all m blocks. Suppose the fault path performs U and then a Pauli error $\bigotimes_i F_i$, where F_i acts on $\mathcal{H}_i$. We can also assume E_j is a tensor product (by increasing the number of terms in the sum over j, if needed), and further write $E_j = \bigotimes_{i=1}^{n} E_{ij}$, where E_{ij} acts on $\mathcal{H}_i$. Then

$$\widetilde{U}|\psi\rangle = \sum_j \left(\bigotimes_i F_i\right)\left(\bigotimes_i V_i\right)\left(\bigotimes_i E_{ij}\right)|\psi_j\rangle = \sum_j \bigotimes_i \left[F_i\left(V_i E_{ij} V_i^\dagger\right)\right] U|\psi_j\rangle \tag{11.2}$$

Note that $V_i E_{ij} V_i^\dagger$ acts only on $\mathcal{H}_i$ always. If $E_{ij} = I$, then $V_i E_{ij} V_i^\dagger = I$, so the term $F_i \left(V_i E_{ij} V_i^\dagger \right)$ is not the identity only if either E_{ij} and F_i is not the identity.

Now, for any j, at most $\sum_{a=1}^m r_a$ of the E_{ij}s are not the identity, and the number of non-identity F_i terms is at most s, the size of the fault path. Thus, the maximal possible number of non-identity tensor factors in $F_i \left(V_i E_{ij} V_i^\dagger \right)$ is $s + \sum_{a=1}^m r_a$. That is,

$$\widetilde{U}|\psi\rangle = \sum_j G_j U |\psi_j\rangle, \tag{11.3}$$

where G_j is not the identity on at most $s + \sum_a r_a$ tensor factors $\mathcal{H}_i$. Since U is a gate gadget for the code and $|\psi_j\rangle$ is a valid codeword for each block, $U|\psi_j\rangle$ also consists of valid codewords for each block. Therefore, this state will pass a $\sum_a r_a + s$ filter on each n-qubit block, so the gadget satisfies the GPP.

For the GCP, we take equation (11.3) and apply an ideal decoder for the QECC to each n-qubit code block. G_j has weight $s + \sum_a r_a \leq t$ for each n-qubit code block, with t the number of errors the code corrects, so if $\mathcal{D}$ is the ideal decoder,

$$\mathcal{D}(G_j U |\psi_j\rangle) = U|\psi_j\rangle. \tag{11.4}$$

Now, E_j also has weight $\leq t$ for each n-qubit code block, so

$$\mathcal{D}(E_j|\psi_j\rangle) = |\psi_j\rangle, \tag{11.5}$$

and

$$\mathcal{D}(\widetilde{U} E_j |\psi_j\rangle) = U|\psi_j\rangle = U\mathcal{D}(E_j|\psi_j\rangle). \tag{11.6}$$

Thus, by linearity,

$$\mathcal{D}(\widetilde{U}|\psi\rangle) = U\mathcal{D}(|\psi\rangle), \tag{11.7}$$

which is the GCP. □

Of course, this proof is a long-winded way of saying something that is fairly obvious, which is that transversal gates can spread errors between blocks, but they cannot cause errors to propagate within a block of the code. Thus, the total number of errors in any single block of the output of a transversal gate is at most the total number of errors summed over all input blocks plus the total number of faulty locations in the circuit.

Note that the proof of the GPP does not make use of the constraint $\sum_a r_a + s \leq t$, so transversal gates actually satisfy a stronger version of the GPP.

One nice property of transversal gates is that a series of them can be strung together, and the result is another transversal gate.

Proposition 11.2. *The product of transversal gate gadgets is a transversal gate gadget.*

Proof. From the definition of transversal, it immediately follows that the product of two transversal gates is a transversal gate. The product of gadgets for locations L_1 and L_2 is also a gadget, for the circuit consisting of L_1 followed by L_2. □

In the standard fault-tolerant simulation, we put an error correction gadget after each gate gadget. However, the notion of what a fundamental gadget is can be relative. As a consequence of proposition 11.2, the composite gadget consisting of a product of some number of consecutive transversal gate gadgets also satisfies the GPP and GCP, so is also fault-tolerant. Therefore, we still get a valid fault-tolerant simulation if we only put a single error correction gadget after the whole composite gadget rather than after each of the gate gadgets making it up.

Of course, if we combine two m-block transversal gadgets that act on some different blocks, the result can be a gadget that acts on as many as $2m-1$ blocks (assuming the gadgets share at least one block). Even though transversal gates satisfy the GPP, each output block of the combined gadget may contain the errors combined from all the input blocks. This sets a real limit to how many transversal gate gadgets one can safely do in sequence before doing quantum error correction. Single-block transversal gadgets (composed of single-qubit gates) don't have this problem, although it's still true that faults in the physical gates making up a composite single-block transversal gadget will eventually add up.

11.2 Transversal Gates for Stabilizer Codes

So, transversal gate gadgets are automatically fault-tolerant. How do we determine, for a given code, what transversal gates are available? In general, this problem doesn't have a satisfactory known solution. A good first step is to ask, for a given QECC and a given transversal circuit, is it a valid gadget for some logical gate, or does it take some codewords to non-codewords? If it is a gate gadget, what is the logical gate? For a general circuit and QECC, we can answer the latter questions by applying the circuit to a set of basis states for the code and determining if we always get codewords. If we do, we can then compute the overall logical operation. This works, but takes exponential time in the number of qubits involved. For the special case of stabilizer codes and Clifford group circuits, we can do much better.

11.2.1 Transversal Paulis

Let's start with the easiest case, transversal gates made up of Pauli gates. In fact, we've already answered the question for this case: a Pauli P is a gate gadget for stabilizer code S iff $P \in \mathsf{N}(\mathsf{S})$. Furthermore, by theorem 3.11, P is a gate gadget for a logical Pauli gate location, and we can determine which one by looking at which coset in $\mathsf{N}(\mathsf{S})/\mathsf{S}$ contains P.

Note that all the Paulis in the same coset in $\mathsf{N}(\mathsf{S})/\mathsf{S}$ implement the same logical Pauli. All are transversal and therefore fault-tolerant. This makes it clear that FT gate gadgets are not unique. In this case, there are a total of 2^{n-k} perfectly good transversal implementations of the same gate gadget. However, from the point of view of fault tolerance, the different implementations are not completely equivalent. In particular, they involve different sets of physical locations, which can lead to different error rates and different kinds of errors. For instance, for the five-qubit code, $X \otimes X \otimes X \otimes X \otimes X$ and $I \otimes Y \otimes Y \otimes I \otimes X$ are both representatives of the $\overline{X}$ equivalence class. However, the first involves 5 X-gate locations, whereas the second involves only 3 gate locations (two Ys and one X) and 2 wait locations. If p_S is less than p_G, the second implementation is probably preferred. In other circumstances, the first implementation might be better, for instance if Y locations are noticeably more error-prone than X locations, or just because of its greater symmetry.

11.2.2 Clifford Gadgets Preserve the Stabilizer

Moving to more general Clifford group gates takes a little more work, but we've seen most of what to do in chapter 6. Under the action of physical Clifford group gate U, $M \in \mathsf{S}$ gets mapped to $UMU^\dagger$, and the stabilizer of the system afterwards is $U\mathsf{S}U^\dagger$. So when is a Clifford group circuit a valid gadget? Certainly, if $UMU^\dagger = M$ for all $M \in \mathsf{S}$, that means that the stabilizer post-operation is the same as the stabilizer before the circuit, which in turn means that the final state is still a codeword of the same code. However, this logic reveals that the condition $UMU^\dagger = M$ is too stringent. It is sufficient if the stabilizer group as a whole remains the same, and in particular, it is acceptable if conjugation by U permutes the elements of the stabilizer.

Conversely, if $U\mathsf{S}U^\dagger = \mathsf{T} \neq \mathsf{S}$, then U maps some codewords to non-codewords. Why is that? Well, the conjugation action by U is a group isomorphism. That means that not only do distinct Paulis remain distinct, but independent Paulis remain independent. In particular, that means that T has the same number of generators as S, and thus has a code space of the same dimension. Since U is unitary, the map from $\mathcal{T}(\mathsf{S})$ to $\mathcal{T}(\mathsf{T})$ is onto. But $\mathcal{T}(\mathsf{T}) \neq \mathcal{T}(\mathsf{S})$, so there is some $|\phi\rangle \in \mathcal{T}(\mathsf{T})$ which is not a codeword of S, and there is some $|\psi\rangle \in \mathcal{T}(\mathsf{S})$ such that $U|\psi\rangle = |\phi\rangle$.

Summing up, we find a straightforward condition for U to give a valid gate gadget:

Theorem 11.3. *Clifford group circuit U maps codewords of S to codewords of S iff $UMU^\dagger \in \mathsf{S}$ for all $M \in \mathsf{S}$.*

In other words, U is in the normalizer *in the Clifford group* of S. This generalizes the fact that Paulis give logical operations of the code space if they are in the Pauli group normalizer of S.

Checking all 2^{n-k} elements of the stabilizer could take a long time, but as usual, it is sufficient to check only generators, which is much quicker.

Corollary 11.4. *Clifford group circuit U maps codewords of S to codewords of S iff $UMU^\dagger \in \mathsf{S}$ for all generators M of S.*

This is true because conjugation is a group homomorphism, so the image of a product of generators is the product of the images, which is again in S.

Theorem 11.3 and corollary 11.4 hold for arbitrary Clifford group circuits, but for the purposes of fault-tolerance, we are most interested in transversal gates. Theorem 11.3 combined with proposition 11.1 tells us that if we have a transversal Clifford gate in the normalizer of S than we have a fault-tolerant gadget. For single-block transversal gates, that is straightforward enough, but you might wonder how exactly to apply these ideas to multiple-block transversal gates. In this case, we should apply theorem 11.3 to the stabilizer for multiple blocks of the code. For instance, a two-block transversal Clifford gate should preserve the stabilizer $\mathsf{S}^{\otimes 2} = \{M \otimes N | M, N \in \mathsf{S}\}$.

Let us illustrate by considering the 4-qubit code, as given in table 3.5. We will start with single-block transversal gates, implemented by a tensor product of single-qubit operations. The stabilizer of the 4-qubit code contains four elements, $\{I \otimes I \otimes I \otimes I, X \otimes X \otimes X \otimes X, Y \otimes Y \otimes Y \otimes Y, Z \otimes Z \otimes Z \otimes Z\}$. Conjugation can never mix I up with the other three Paulis, but we should consider permutations of the other elements of S.

Indeed, we can implement any permutation of the three non-identity Paulis in S. The trivial permutation can be achieved by the logical Paulis, and indeed, these are the only gates that will leave the stabilizer elements unchanged. Otherwise, we want to permute X, Y, and Z the same way on all four qubits. This can be achieved by performing the same single-qubit Clifford gate on all four qubits. For instance, transversal H (i.e., $H \otimes H \otimes H \otimes H$) will swap $X \otimes X \otimes X \otimes X$ with $Z \otimes Z \otimes Z \otimes Z$, and map $Y \otimes Y \otimes Y \otimes Y$ to itself. Note that H maps Y to $-Y$ on a single qubit, but when applied to all four qubits, the minus signs cancel out.

Since single-qubit Clifford group elements can perform any permutation of the single-qubit Paulis, we can achieve any possible permutation of the stabilizer by picking an appropriate single-qubit Clifford U and then performing the transversal U. We need to do the same permutation of X, Y, and Z on each qubit, but what happens to the phases of X, Y, and Z can be different on different qubits. The single-qubit Cliffords which change the phase of Paulis without otherwise permuting them are exactly the Pauli subgroup of C_1. Thus, the most general single-block transversal Clifford gadget for the four-qubit code can be written uniquely as PU, where $P \in \mathsf{N}(\mathsf{S})/\mathsf{S}$ and $U = V^{\otimes 4}$, with $V \in \check{\mathsf{C}}_1$ an arbitrary single-qubit Clifford group operation.

Now let us consider the two-block transversal Clifford gadgets. The stabilizer of two blocks of the code consists of 16 elements. It will perhaps be clearest if we group them in pairs consisting of the ith qubit of both blocks. Thus, for instance, two elements of $\mathsf{S}^{\otimes 2}$ would be $XI \otimes XI \otimes XI \otimes XI$ and $ZY \otimes ZY \otimes ZY \otimes ZY$. We can take the generators of $\mathsf{S}^{\otimes 2}$ to be

$$\{(XI)^{\otimes 4}, (ZI)^{\otimes 4}, (IX)^{\otimes 4}, (IZ)^{\otimes 4}\}. \tag{11.8}$$

Imagine we take a two-qubit Clifford gate and perform it transversally. For instance, what happens when we perform $\mathrm{CNOT}^{\otimes 4}$? The four generators become

$$\{(XX)^{\otimes 4}, (ZI)^{\otimes 4}, (IX)^{\otimes 4}, (ZZ)^{\otimes 4}\}. \tag{11.9}$$

But $(XX)^{\otimes 4} = (XI)^{\otimes 4}(IX)^{\otimes 4}$ and $(ZZ)^{\otimes 4} = (ZI)^{\otimes 4}(IZ)^{\otimes 4}$, so $\mathrm{CNOT}^{\otimes 4}$ is a valid transversal gadget.

Indeed, *any* transversal two-qubit Clifford group gate will give a valid gadget. If $U \in \mathsf{C}_2$, then $U^{\otimes 4}$ maps any Pauli of the form $P^{\otimes 4}$ to $Q^{\otimes 4}$, with $P, Q \in \mathsf{P}_2$. But $Q^{\otimes 4} \in \mathsf{S}^{\otimes 2}$, since Q can be written as a product of XI, ZI, IX, and IZ, and any phase will disappear when taken four times. The same argument applies to m-block Cliffords for arbitrary m: If $U \in \mathsf{C}_m$, then $U^{\otimes 4}$ is a valid transversal Clifford gate gadget. Indeed, by the same argument as in the single-block case, up to logical Paulis, these are the only valid transversal Clifford gadgets for the 4-qubit code.

11.2.3 Determining the Encoded Gate for a Clifford Gadget

So, now we know how to determine if a Clifford circuit is a gadget for the stabilizer code. But a gadget for what?

In chapter 6, you learned to characterize Clifford group gates by their action on Paulis. We can use the same concept to characterize logical Clifford gates. Since $\mathsf{N}(\mathsf{S})/\mathsf{S}$ is the logical Pauli group, the transformation performed on it by a Clifford gate will tell us what Clifford operation has been performed on the encoded state. (Of course, this only applies to Clifford circuits which are gadgets, since a circuit which is not will change S and therefore also not preserve $\mathsf{N}(\mathsf{S})/\mathsf{S}$.)

Let us illustrate the procedure by again considering the 4-qubit code. We know that any gate of the form $U^{\otimes 4}$ for $U \in \mathsf{C}_m$ is a transversal gadget. Suppose we do $H^{\otimes 4}$. It changes the logical Paulis as follows:

$$\overline{X}_1 = X \otimes X \otimes I \otimes I \rightarrow Z \otimes Z \otimes I \otimes I \tag{11.10}$$

$$\overline{X}_2 = X \otimes I \otimes X \otimes I \rightarrow Z \otimes I \otimes Z \otimes I \tag{11.11}$$

$$\overline{Z}_1 = I \otimes Z \otimes I \otimes Z \rightarrow I \otimes X \otimes I \otimes X \tag{11.12}$$

$$\overline{Z}_2 = I \otimes I \otimes Z \otimes Z \rightarrow I \otimes I \otimes X \otimes X \tag{11.13}$$

What we'd like to do is to write everything on the right-hand side as a product of logical Paulis in order to figure out what transformation is being performed. But, hold on a second: $Z \otimes Z \otimes I \otimes I$ can't possibly be written as a product of $I \otimes Z \otimes I \otimes Z$ and $I \otimes I \otimes Z \otimes Z$! What's wrong?

Don't panic. Remember that the logical Paulis are not unique. We've chosen particular representatives of cosets in $\mathsf{N}(\mathsf{S})/\mathsf{S}$, but other representatives are equally valid. What's happened here is that the transversal gate has changed, by itself, to a different representative of the coset. Yes, it's rude to make the change without telling us, but what can you do? Fault-tolerant gates do what they want and don't listen to complaints.

In this case, it's not too difficult to recover and figure out what is going on. We can immediately recognize $Z \otimes Z \otimes I \otimes I = (Z \otimes Z \otimes Z \otimes Z)(I \otimes I \otimes Z \otimes Z) = \overline{Z}_2$, and can similarly identify the other logical Paulis by multiplying them by stabilizer elements. Overall, we find the following transformation of the logical Paulis:

$$\overline{X}_1 \rightarrow \overline{Z}_2 \tag{11.14}$$

$$\overline{X}_2 \rightarrow \overline{Z}_1 \tag{11.15}$$

$$\overline{Z}_1 \rightarrow \overline{X}_2 \tag{11.16}$$

$$\overline{Z}_2 \rightarrow \overline{X}_1 \tag{11.17}$$

We can recognize this as a gate which swaps the two logical qubits and applies the Hadamard to both.

Notice that when the QECC encodes multiple qubits, as in the case of the 4-qubit code, even single-block transversal gates can do multiple-qubit *logical* gates. However, in the case of the 4-qubit code, not every logical two-qubit Clifford gate can be performed by transversal operations — there simply aren't enough of them.

11.3 Transversal Gates for the 7-Qubit Code

In section 11.2, we saw that the set of transversal gates for a stabilizer code is based on the set of symmetries of the stabilizer. The more symmetric the stabilizer, the more transversal gates are available. The 4-qubit code is pretty symmetric, but the 7-qubit code is more so, extending the symmetry to the logical Paulis as well as the stabilizer generators. Consequently, the 7-qubit code is particularly attractive based on the number of transversal gates it has available.

Looking at the stabilizer of the 7-qubit code and going through the same logic as for the 4-qubit code, we immediately see that for the 7-qubit code, it is also true that $U^{\otimes 7}$ is a valid m-block transversal gate gadget for any m-qubit Clifford U. All elements of the stabilizer for the 7-qubit code also have weight 4, so the phases will take care of themselves. Let's go through some examples to see what logical operations they perform.

First, Hadamard:

$$\overline{X} = X \otimes X \otimes X \otimes X \otimes X \otimes X \otimes X \rightarrow Z \otimes Z \otimes Z \otimes Z \otimes Z \otimes Z \otimes Z = \overline{Z} \tag{11.18}$$

$$\overline{Z} = Z \otimes Z \otimes Z \otimes Z \otimes Z \otimes Z \otimes Z \rightarrow X \otimes X \otimes X \otimes X \otimes X \otimes X \otimes X = \overline{X} \tag{11.19}$$

Thus, transversal Hadamard implements the logical Hadamard.

Next comes $R_{\pi/4}$:

$$\overline{X} = X \otimes X \otimes X \otimes X \otimes X \otimes X \otimes X \to Y \otimes Y \otimes Y \otimes Y \otimes Y \otimes Y \otimes Y \tag{11.20}$$

$$\overline{Z} = Z \otimes Z \otimes Z \otimes Z \otimes Z \otimes Z \otimes Z \to Z \otimes Z \otimes Z \otimes Z \otimes Z \otimes Z \otimes Z = \overline{Z} \tag{11.21}$$

We need to be careful with $Y^{\otimes 7}$. $Y = iXZ$, so

$$Y^{\otimes 7} = i^7 X^{\otimes 7} Z^{\otimes 7} = -i\overline{XZ} = -\overline{Y}. \tag{11.22}$$

Thus, the transversal $R_{\pi/4}$ does not do the logical $R_{\pi/4}$. Instead, we can identify the gate as $\overline{R_{\pi/4}}^\dagger = \overline{R_{-\pi/4}}$. This is an important example of a place where you need to be careful with phases or you get the wrong answer. If we want to do $\overline{R_{\pi/4}}$, we can instead do it using the transversal $R_{-\pi/4}$.

Next, let's try a two-block gate, the transversal CNOT:

$$\overline{X \otimes I} = (XI)^{\otimes 7} \to (XX)^{\otimes 7} = \overline{X \otimes X} \tag{11.23}$$

$$\overline{I \otimes X} = (IX)^{\otimes 7} \to (IX)^{\otimes 7} = \overline{I \otimes X} \tag{11.24}$$

$$\overline{Z \otimes I} = (ZI)^{\otimes 7} \to (ZI)^{\otimes 7} = \overline{Z \otimes I} \tag{11.25}$$

$$\overline{I \otimes Z} = (IZ)^{\otimes 7} \to (ZZ)^{\otimes 7} = \overline{Z \otimes Z}. \tag{11.26}$$

We can identify this as the logical CNOT between the qubits encoded in the two blocks of the code.

Now, H, $R_{\pi/4}$, and CNOT are generators of the Clifford group, and we have transversal implementations of all of them. That means we can implement the whole logical Clifford group on m blocks of the 7-qubit code just by doing transversal operations. The transversal U does not always give us the logical U — instead it gives us the complex conjugate U^* — but nevertheless, it is straightforward to perform any logical Clifford group operation for this code. This gives us high hopes for finding a fault-tolerant protocol for the 7-qubit code, and indeed this is a good start, but unfortunately, it gets harder from here. This is the full list of transversal gate gadgets for the 7-qubit code. To get a gate outside the Clifford group, we will need another construction, discussed in chapter 13.

11.4 Transversal Gates and Measurement for CSS Codes

The 7-qubit code is not bad, but there's a whole world of codes out there, so it's worth looking at a broader class of codes to see what we can say about transversal gates. One gate that plays a particularly big role in quantum computation is the CNOT gate, so it's convenient that the 7-qubit code allows a transversal implementation of it. What other codes have this property? It's not straightforward to make a completely general statement about transversal implementations of CNOT, but for the 7-qubit code, the gadget for CNOT is just transversal CNOT. Let's try to generalize that.

Take an arbitrary stabilizer code S, and consider two blocks of the code. Let's see what happens when we apply transversal CNOT. If $M \in \mathsf{S}$, then $M \otimes I \in \mathsf{S}^{\otimes 2}$. Suppose M has binary symplectic representation $(x_M|z_M)$; then transversal CNOT applied to $M \otimes I$ gives $(x_M x_M|z_M 0)$. For this to be in $\mathsf{S}^{\otimes 2}$, we need $(x_M|0) \in \mathsf{S}$. By the fact that S is a group, $(0|z_M)$ is also in S. By choosing generators appropriately, we can thus always have all generators of the form $(x_M|0)$ or $(0|z_M)$. That means it is a CSS code.

For a CSS code, we can always choose the logical X operators to be tensor products of X and I and the logical Z operators to be tensor products of Z and I. The choice of basis codewords given in section 5.1.3 has this property. In that case, we can see that transversal CNOT will perform logical CNOT between all corresponding pairs of encoded qubits. In other words, we find the following:

Theorem 11.5. *A stabilizer code* S *has a gadget consisting of transversal* CNOT *iff* S *is a CSS code. If so,* S *has a choice of logical operators for which the gadget simulates a location consisting of the tensor product of* CNOT*s from logical qubit* i *in the first block to logical qubit* i *in the second block, for all* i*.*

Theorem 11.5 makes CSS codes very attractive for fault tolerance. Just by choosing to work with a CSS code, we know for sure we have logical CNOT gates available, albeit performed together on all logical qubits in the block. It turns out that CSS codes always have another transversal gadget that is just as useful: measurement.

Recall from section 5.1.3 that we can write the basis codewords of a CSS code as

$$|u + {C_2}^\perp\rangle = \sum_{v \in {C_2}^\perp} |u + v\rangle. \tag{11.27}$$

The logical state encoded by this basis state is given by the coset of $C_1/{C_2}^\perp$ which u lies in. Thus, to measure the encoded data in the standard basis, it suffices to measure u up to an element of ${C_2}^\perp$. But we can achieve this by simply measuring transversally in the standard basis: When we do so, we get $u+v$, with v a random element of ${C_2}^\perp$. Thus, transversal standard basis measurement followed by classical decoding is a gadget for measurement of logical qubits in the standard basis. The classical decoding part to determine the coset of u is not implemented transversally, but in a basic error model, we assume that classical processing is completely reliable.

Proposition 11.1 only tells us transversal gate gadgets are fault tolerant, so we should also check that transversal measurement satisfies the MCP. In fact, for a general stabilizer code, transversal measurement is *not* fault-tolerant, although it does serve as a gadget for some codes. For instance, for the five-qubit code, transversal measurement of a perfect codeword will tell you the encoded state — consulting equations (3.11) and (3.12), we see that $|\overline{0}\rangle$ always gives an even parity string and $|\overline{1}\rangle$ always gives an odd parity string — but even a single bit flip error due to a faulty measurement location will cause us to get the wrong answer.

CSS codes, however, have the special property that transversal measurement is fault tolerant.

Theorem 11.6. *For any CSS code, transversal measurement followed by classical decoding is a fault-tolerant gadget implementing logical measurement of all encoded qubits.*

Proof. The theorem is true because the codewords are superpositions of states from a classical error-correcting code. That means a small number of errors in the classical output cannot confuse us between different logical codewords.

To prove the MCP formally, let us consider the output state of an r-filter

$$|\psi\rangle = \sum_j E_j |\psi_j\rangle \quad (\mathrm{wt}\, E_j \leq r). \tag{11.28}$$

Invoking corollary 10.2 again, we can assume the errors E_j are Pauli errors. Furthermore, we can assume that all E_j in the sum have distinct error syndromes: If the code is non-degenerate, this is automatically true. If the code is degenerate, since $\mathrm{wt}\, E_j \leq r \leq t$, degenerate errors will act the same on the codewords $|\psi_j\rangle$, and we can re-write the decomposition over j to have only one error of each syndrome.

Let us write $|\psi_j\rangle = \sum_u c_{ju} |\overline{u}\rangle$. Then the ideal decoder applied to equation (11.28) gives

$$\sum_{j,u} c_{ju} |u\rangle |\sigma(E_j)\rangle, \tag{11.29}$$

where the second tensor factor contains the syndrome bits. Since all $\sigma(E_j)$ are distinct, measurement of this state simply gives codeword u with probability $\sum_j |c_{ju}|^2$. This is the RHS of the MCP.

Now let us figure out what a faulty measurement gadget does to equation (11.28). We have

$$|\psi\rangle = \sum_u \left(\sum_j c_{ju} E_j \right) |\overline{u}\rangle. \tag{11.30}$$

The s faults in the transversal measurement have the effect of up to s additional bit-flip errors, which we can encapsulate as F, with $\mathrm{wt}\, F \leq s$. That gives us the state

$$F|\psi\rangle = \sum_u \left(\sum_j c_{ju} F E_j \right) |\overline{u}\rangle, \tag{11.31}$$

on which we perform perfect single-qubit measurement. Notice that $\mathrm{wt}(FE_j) \leq r + s \leq t$. Let's go even further and expand $|\overline{u}\rangle = \sum_{v \in {C_2}^\perp} |u + v\rangle$. Then

$$F|\psi\rangle = \sum_{u \in C_1/{C_2}^\perp} \sum_{v \in {C_2}^\perp} \left(\sum_j c_{ju} F E_j \right) |u + v\rangle. \tag{11.32}$$

Now, C_1 has distance d_1. If the CSS code is non-degenerate, $d_1 \geq 2t + 1$, and $FE_j|w\rangle$ is orthogonal to $FE_{j'}|w'\rangle$ for any $w \neq w' \in C_1$. When the code is degenerate, it is possible that $FE_j|w\rangle = \pm FE_{j'}|w'\rangle$ for some $w \neq w' \in C_1$. However, since the quantum code does have distance $2t + 1$, the only way this can happen is if $w + w' \in {C_2}^\perp$. Thus, if we want to calculate the probability of some output string x, at most one coset u can contribute. This means the classical decoding of x will be unambiguous, giving a unique coset u, which we can interpret as the measurement outcome.

It just remains for us to check that the probability of getting the outcome u is the same as in the ideal case. The probability of the measured string being x for that particular measurement error F is

$$|\langle x|F|\psi\rangle|^2 = \left| \langle x| \left(\sum_j c_{ju} F E_j \right) |\overline{u}\rangle \right|^2, \tag{11.33}$$

where u is the unique u consistent with x. The probability of outcome u is (11.33) summed over x:

$$\Pr(u) = \sum_x \sum_{j,j'} c_{ju} c^*_{j'u} \langle \overline{u}|E^\dagger_{j'} F^\dagger|x\rangle \langle x|FE_j|\overline{u}\rangle. \tag{11.34}$$

The sum over x is summed over all xs that have non-zero probability for u, so we can replace $\sum_x |x\rangle \langle x|$ with the identity. Then

$$\Pr(u) = \sum_{j,j'} c_{ju} c^*_{j'u} \langle \overline{u}|E^\dagger_{j'} E_j|\overline{u}\rangle. \tag{11.35}$$

But we have restricted the sum over j to have just a single E_j with each error syndrome. Therefore, the inner product gives 0 unless $j = j'$. Thus, $\Pr(u) = \sum_j |c_{ju}|^2$, as desired. □

It's also worth discussing briefly when a CSS code has additional transversal operations beyond CNOT (or product of CNOTs) and measurements. Thinking briefly about the structure of a CSS code, it's clear that transversal Hadamard is a gadget iff $C_1 = C_2$. The 7-qubit code has this property, as does the 4-qubit code. In general, we can form a CSS code with $C_1 = C_2$ whenever ${C_1}^\perp \subseteq C_1$. However, as in the case of the 4-qubit code, the action of the transversal Hadamard can be more than just Hadamard applied to all encoded qubits. Since transversal Hadamard changes Xs into Zs and vice-versa, it does perform the logical Hadamard on all qubits, but there may be some additional classical operation (such as SWAP or CNOT) interacting the encoded qubits within a single block.

To have transversal $R_{\pi/4}$ as a valid gadget for the code, we need further additional structure. $R_{\pi/4}$ maps X to Y, so the X generators of the code will get mapped to tensor products of Y and I, which must also be in the stabilizer. Furthermore, the product of an X generator M and $N = R^{\otimes n}_{\pi/4} M R^{\dagger \otimes n}_{\pi/4}$ must also be in the stabilizer. The product will be a tensor product of Z and I, in fact what would be obtained from M via a transversal Hadamard, but with an additional phase which depends on the weight of M. The phase is $i^{\mathrm{wt}\, M}$. Since the stabilizer must be Abelian, $\mathrm{wt}\, M$ must be even, or else M and N would not commute, but we can still get a phase of -1. In other words, to have transversal $R_{\pi/4}$ as a valid gadget, we must have $C_1 = C_2$ *and* the weight of every generator must be a multiple of 4 (the classical code ${C_1}^\perp$ is *doubly even*). However, even if ${C_1}^\perp$ is not doubly even, there is still a corresponding transversal gate, consisting of transversal $R_{\pi/4}$ followed by some phase flips to fix up the phase. The logical operation performed by the transversal $R_{\pi/4}$ (with or without bit flips) does not necessarily involve logical $R_{\pi/4}$s, however.

11.5 Other Topics Relating to Transversal Gates

11.5.1 Codes With Transversal $\pi/8$ Gates

All the examples we've seen so far of transversal gates are Clifford group gates, but naturally it's also possible to have codes with non-Clifford group transversal gates. Either the physical operations involved in the gate or the logical effect of the gadget — or most often, both — can be a non-Clifford unitary. However, stabilizer codes have a particular affinity for Clifford group gates because both are so closely connected with the Pauli group. For many stabilizer codes, all the transversal gates are Clifford group gates. Nevertheless, a few examples are known of stabilizer codes with non-Clifford transversal gates. We don't currently have a systematic understanding of when this is possible and which gates can be done with which codes, but in this section I'll present one particular family of codes with a non-Clifford transversal gate. This will illustrate the principle, and the smallest code from this family will be of use in section 13.5.

The gate we'll be using is the transversal $\pi/8$ gate $R_{\pi/8}^{\otimes n}$. The $R_{\pi/8}$ gate on a single qubit produces a relative phase $e^{i\pi/4}$ between $|0\rangle$ and $|1\rangle$. Therefore, n of them applied to a basis state $|x\rangle$ with $\operatorname{wt} x = w$ produce a phase $e^{i\pi w/4}$ relative to the $|00\ldots0\rangle$ state. In particular, basis states whose weight is a multiple of 8 acquire no overall phase relative to the all-0's state.

Recall that the codewords of CSS codes are superpositions over cosets of some classical code. Suppose, therefore, that we consider a classical code ${C_2}^\perp$ whose codewords all have weights 0 mod 8. (Such a code is *quadruply even*.) The logical 0 of the corresponding CSS code is thus

$$|\overline{0}\rangle = \sum_{v \in {C_2}^\perp} |v\rangle, \tag{11.36}$$

which is a superposition of basis states with weight 0 mod 8. Therefore, $R_{\pi/8}^{\otimes n}|\overline{0}\rangle = e^{i\phi}|\overline{0}\rangle$. There is a global phase $e^{i\phi}$ with $\phi = -n\pi/8$ because of our definition of $R_{\pi/8}$, but the point is that $|\overline{0}\rangle$ is an eigenstate of the transversal $\pi/8$ gate. Furthermore, any other logical basis codeword

$$|u + {C_2}^\perp\rangle = \sum_{v \in {C_2}^\perp} |u + v\rangle \tag{11.37}$$

is a superposition of basis states of weight w mod 8, where $w = \operatorname{wt} u$. Therefore,

$$R_{\pi/8}^{\otimes n}|u + {C_2}^\perp\rangle = e^{i(\phi + \pi w/4)}|u + {C_2}^\perp\rangle. \tag{11.38}$$

This is also an eigenstate of the transversal $\pi/8$ gate.

The transversal $\pi/8$ is thus a valid gadget for any CSS code with quadruply even ${C_2}^\perp$. The resulting logical gate is a diagonal gate. Its action on the logical basis state corresponding to $u \in C_1$ is to produce a phase $e^{i(\phi+\pi w/4)}$, where $w = \operatorname{wt} u$.

The smallest interesting distance 3 code for which this happens is the 15-qubit code with stabilizer given in table 11.1. C_1 is the $[15, 5, 8]$ punctured Reed-Muller code $\mathcal{R}(1, 4)$ (i.e., $\mathcal{R}(1, 4)$ with the last register deleted). ${C_2}^\perp$ is the even subcode, namely the set of codewords of C_1 with even weight, which is the punctured $\mathcal{R}(1, 4)$ with the all 1's vector removed. The 15-qubit code is a $[[15, 1, 3]]$ code. Since $\mathcal{R}(1, 4)$ is quadruply even, the codewords in C_1 have weight either 0 mod 8 or 7 mod 8. ${C_2}^\perp$ therefore contains only the 0 mod 8 codewords, so this CSS code has the right form.

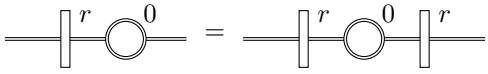
Indeed, from the above discussion, we can see immediately that the $|\overline{0}\rangle$ state is a superposition of those codewords of C_1 with weight 0 mod 8 and the $|\overline{1}\rangle$ state is a superposition of the codewords of C_1 with weight 7 mod 8. Therefore, the transversal $\pi/8$ gate applied to $|\overline{a}\rangle$ gives a phase $e^{i\pi(1/8-a/4)}$, which is the logical $-\pi/8$ gate $\overline{R_{-\pi/8}}$.

Z	Z	Z	Z	I	I	I	I	I	I	I	I	I	I	I
Z	Z	I	I	Z	Z	I	I	I	I	I	I	I	I	I
Z	I	Z	I	Z	I	Z	I	I	I	I	I	I	I	I
Z	Z	I	I	I	I	I	I	Z	Z	I	I	I	I	I
Z	I	Z	I	I	I	I	I	Z	I	Z	I	I	I	I
Z	I	I	I	Z	I	I	I	Z	I	I	I	Z	I	I
Z	Z	Z	Z	Z	Z	Z	Z	I	I	I	I	I	I	I
Z	Z	Z	Z	I	I	I	I	Z	Z	Z	Z	I	I	I
Z	Z	I	I	Z	Z	I	I	Z	Z	I	I	Z	Z	I
Z	I	Z	I	Z	I	Z	I	Z	I	Z	I	Z	I	Z
X	X	X	X	X	X	X	X	I	I	I	I	I	I	I
X	X	X	X	I	I	I	I	X	X	X	X	I	I	I
X	X	I	I	X	X	I	I	X	X	I	I	X	X	I
X	I	X	I	X	I	X	I	X	I	X	I	X	I	X

Table 11.1: The stabilizer for the 15-qubit code.

11.5.2 Transversal Gates Plus Permutations

Transversal gates are useful for fault-tolerance in part because they conjugate pre-existing errors into errors of the same weight. Phrased in terms of our pictorial language, they have the property



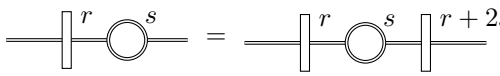
(11.39)

regardless of whether $r \leq t$ or $r > t$.

Transversal gates are not the only set of gates that behave this way. Another option is the set of gates which permute the qubits in the code but don't otherwise change their values. Under a permutation gate of this sort, an error on a single qubit just gets moved to a different qubit. You could think of this as propagation, since an existing error causes an error to appear on a new qubit, but in the process, the error is eliminated from the old qubit. Thus, property (11.39) is satisfied.

For some codes, it is possible to make interesting gates out of permutations. For instance, in the 4-qubit code, suppose we swap the second and third qubits. The stabilizer generators are left unchanged, but $\overline{X}_1 \leftrightarrow \overline{X}_2$ and $\overline{Z}_1 \leftrightarrow \overline{Z}_2$. $\mathrm{SWAP}_{2,3}$ is thus a gadget for the logical SWAP gate between the code's two encoded qubits.

However, permutation gates are not fault tolerant unless we are careful about the implementation. The permutation group can be generated by transpositions of pairs of elements, so it is sufficient to consider circuits where every gate is the SWAP gate. The problem is that a fault on a SWAP gate can cause both of the qubits involved in the gate to have errors. Therefore, instead of the GPP, we have



(11.40)

In order to get around this, we need to be careful how to implement the SWAP. The circuit given in figure 11.2 is one solution. To swap two data qubits, add a third ancilla qubit to act as an intermediary. By performing three SWAP gates as in the figure, the overall effect is to do a SWAP on the data qubits, but none of the three gates directly interacts the two data qubits. Thus, a single faulty gate can only cause an error on one of the data qubits. After the circuit is finished, the ancilla qubit can be discarded. Also note that the initial state of the ancilla qubit is irrelevant — the circuit works just as well no matter what it is. It is also OK to reuse the ancilla qubit many times in SWAPs like this.

The upshot is that a circuit of SWAP gates each implemented via the method of figure 11.2 is fault-tolerant, satisfying both the GPP and GCP. For some codes, this can significantly expand the set of available fault-tolerant gadgets. Another possibility is to consider gadgets built of a transversal gate followed by a

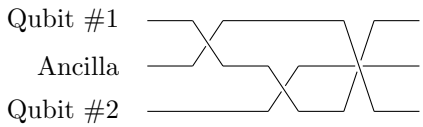


Figure 11.2: A circuit for swapping two physical qubits for which one fault only causes one error in the final state. The ancilla can be in any state.

fault-tolerant implementation of a permutation gate. The combination will be fault-tolerant as well. However, whereas any product of transversal gates remains fault tolerant, as does any product of permutation gates, the combination (transversal gate – permutation gate – transversal gate) is *not* in general fault tolerant. In particular, consider a case for which the transversal gates interact two different blocks of the code. A fault in the first transversal gate can create an error in qubit i of both blocks of the code. If the permutation gate treats the two blocks differently, it could move one of those errors to qubit $j \neq i$ on one block, and then the second transversal gate could propagate the error to qubit j on the other block, which would then have errors on the two qubits i and j even though there was only a single fault in the circuit. Therefore, if we want to combine transversal gates and permutations, we will need to periodically do error correction.

11.5.3 Transversal Gates Cannot Be Universal

For the 7-qubit code, transversal gate gadgets implement the full Clifford group. This is tantalizing — if we can just do one more transversal gate, by theorem 6.9, we get a universal set of gates. Unfortunately, for the 7-qubit code, there are no more transversal gates. The Clifford group by itself can be classically simulated, so we definitely need something more to achieve the full power of quantum computation. Perhaps there is another code which lets us do a universal set of gates transversally? Unfortunately, there is not:

Theorem 11.7 (Eastin-Knill)**.** *If Q is an $((n, K, d))_q$ QECC with $d \geq 2$, and G is the set of logical unitary gates that have transversal gadgets for Q, then G is a discrete group. In particular, G is not dense in $\mathsf{SU}(K)$.*

Theorem 11.7 tells us that transversal gate gadgets can never be enough to get a universal set of gates. We will always need to do something else; the most common source of something else is to add in magic states, which will be discussed in chapter 13.

Note that there does exist a classical code with a classically universal set of transversal gadgets: The repetition code. Any classical gate that we wish to perform on the encoded state of a repetition code can just be done by doing it separately on each copy.

The intuition behind this theorem is that if G is not discrete, it must have some non-trivial small unitaries in it. But there is no way the QECC can distinguish between a small transversal gate that is supposed to be a gadget and an error: As in theorem 1.1, the tensor product of unitaries very close to the identity is also very close to a one-qubit error channel, and with distance 2, the code should at least detect if not correct any one-qubit error channels. Presented with a gate gadget which is a small transversal gate, the code will treat it as an error and correct it, meaning the logical action of the gadget is the identity. Thus, any sufficiently small elements of G are trivial and G is discrete. To make this intuition precise, we should look at infinitesimal unitaries, which means dealing with the Lie algebra.

Proof. It is sufficient to consider single-block transversal gadgets. An m-block transversal gadget can be thought of as a single-block transversal gadget for the code $Q^{\otimes m}$, which can be interpreted as an $((n, K, d))_{q^m}$ code, with the ith register consisting of the ith registers of all m blocks.

Let $\tilde{G}$ be the set of transversal gates which are gadgets for the code. G is equal to $\tilde{G}/K$, with K the set of transversal implementations of the identity.

The proof will use a few facts about Lie groups and Lie algebras, which I will bring up as needed. The first observation is that $\tilde{G}$ is a topologically closed subgroup of the compact finite-dimensional Lie group $\mathsf{U}(q)^{\otimes n}$, and is therefore a Lie group itself. $\tilde{G}$ is a subgroup by proposition 11.2, and it is closed because it

is the intersection of the set of transversal gates $\mathsf{U}(q)^{\otimes n}$ and the set $\mathsf{U}(K) \otimes \mathsf{U}(q^n - K)$ of unitaries which preserve the code space, both of which are closed sets.

Since $\tilde{G}$ is a Lie group, it has a Lie algebra $\mathfrak{g}$ which is a subalgebra of the Lie algebra $\mathfrak{t}$ of $\mathsf{U}(q)^{\otimes n}$. $\mathfrak{t}$ is spanned by elements of the form iH, where H is a weight-1 Hermitian operator. Therefore, an arbitrary element iD of $\mathfrak{g}$ can be written as a sum of weight-1 operators. However, the code has distance at least 2, so it detects any single-register error. By theorem 2.4, D must therefore be a detectable error. That is, for any codeword $|\psi\rangle$, $D|\psi\rangle = \alpha|\psi\rangle + |\phi\rangle$, where $|\phi\rangle$ is orthogonal to the code space Q.

Now, a neighborhood of the identity in $\tilde{G}$ is generated from the Lie algebra by the exponential function, $U = e^{itD}$, $D \in \mathfrak{g}$.

$$U|\psi\rangle = e^{itD}|\psi\rangle = \sum_r \frac{(itD)^r}{r!}|\psi\rangle = |\psi\rangle + (itD)|\psi\rangle + O(t^2). \tag{11.41}$$

For any t, U is a logical operation on the code, so this sum is a codeword for all t and all $|\psi\rangle \in Q$. The only way this can be is if $D|\psi\rangle \in Q$. Combining with the error detection property, we find

$$D|\psi\rangle = \alpha_D|\psi\rangle \tag{11.42}$$

when $|\psi\rangle \in Q$ and $D \in \mathfrak{g}$. α_D can depend on D, but by the error correction conditions, it cannot depend on $|\psi\rangle$.

Therefore, given any element $U = e^{iD}$ in a neighborhood of the identity in $\tilde{G}$, we have

$$U|\psi\rangle = e^{iD}|\psi\rangle = e^{i\alpha_D}|\psi\rangle. \tag{11.43}$$

U performs the trivial logical gate, and the neighborhood of the identity lies within K.

The finite-dimensional Lie group $\tilde{G}$ is a union of a discrete set of connected components. The connected component containing the identity is generated by a neighborhood of the identity, and therefore the whole connected component of the identity lies within K. The other connected components are cosets of the identity component, so $G = \tilde{G}/K$ is a discrete group. □