Quantum Error Correction

Joseph M. Renes

May 21, 2024

version 1.7
git d892691, branch *QEC2023*, 2024-05-19
Lecture Notes ETH Zürich FS2023

Contents

Essential features of quantum error correction 1

1.1 But quantum error correction is impossible!

In the early days of quantum information theory, it was thought that quantum error correction is impossible. The goal of the first chapter is to sketch Shor's[1] original construction of a quantum error correcting code and show how it avoids this conclusion.

There were two main arguments to the claim that quantum error correction is impossible.

1. *No cloning.* The simplest method of classical error correction is to simply repeat the message. However, in the quantum setting this method would require the ability to copy an arbitrary quantum state, in violation of the no-cloning theorem.

2. *Finite precision control.* Quantum information appears to be analog information, not digital or discrete, in that specifying a qubit state $|\psi\rangle$ requires specifying two real numbers in the representation $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ (where $\alpha, \beta \in \mathbb{C}$ subject to $|\alpha|^2 + |\beta|^2 = 1$ and the overall phase of the state is irrelevant).

   If the analog properties of the information carriers are truly vital for the computation to work, then it would appear to be necessary to perform error correction to arbitrary precision. But this is not going to be possible with only finite precision control. For instance, how can we possibly correct arbitrarily tiny rotations with relatively crude qubit control? If, on the other hand, we only need to keep the information from being disturbed by errors above a fixed finite precision, then we have effectively discretized the problem. In this case, error correction with finite precision control might be possible, were it not for the non-cloning objection above.

Shor was confronted with these arguments after developing his famous factoring algorithm, causing him to reconsider them quite carefully. We shall see that quantum information defies easy categorization, and while the information itself is sort of analog, the noise that affects the information is in some sense discrete. No-cloning turns out not to be an obstacle, and indeed quantum error correction is possible.

1.2 Repetition coding

Let us step back and consider the simplest classical error-correcting code, the repetition code. Suppose a sender would like to transmit a single bit message, 0 or 1, to a receiver. However, the communication channel connecting them is noisy and occasionally flips the bit value. To transmit 0 using the repetition code, the sender transmits three zeros: 000; to transmit a 1, three 1s: 111.

A noisy version of the original transmission is delivered to the receiver, in which some (or even all) of the bits have been flipped to the opposite value. The task of the receiver is to determine what message the sender transmitted. Assuming bit flips only happen occasionally, a reasonable course of action is for the receiver to assume that the sender's intended message is whatever bit value appears most often in the noisy received version. This is called majority vote decoding.

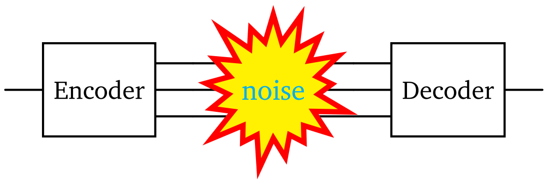
The entire procedure ensures that the intended message is properly received even if there is a single error in the transmission. Supposing that errors occur independently on the transmitted bits

[1] Peter Shor

with identical probability p, the probability of error using the three-bit repetition code is $3p^2(1-p)+p^3$, the probability of two or three errors. When using no code at all, the error rate would be simply p, so the code has succeeded in reducing the probability of error from p to $O(p^2)$ (provided $p<\frac{1}{2}$).

1.2.1 Structure of the correction procedure

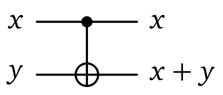
This simple example illustrates the important features of general error correction procedures. First, the message is *encoded* into the *codewords*, here 000 and 111, by the transmitter. The set of codewords is the *code*. Here the code $\mathcal{C}=\{000,111\}$. Next, the noise acts on the codewords in some manner, here by flipping single or multiple bits. Finally, the receiver *decodes* the noisy codeword by analyzing it in some manner, here by computing the majority vote, and outputs an estimate of the original message. The procedure is successful when the output equals the input. The following block diagram, reading left to right, depicts the setup.



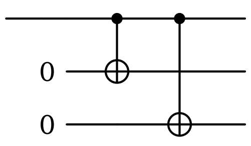
Note that the noise is assumed to only act on the encoded information on its way from the sender to the receiver. This is the case of interest in communication engineering, as reflected in the language of "transmitter" and "receiver". Here the operations of the encoder and decoder are assumed to be free of noise. If this is not the case, more intricate methods of error correction are required. We will revisit this issue in ...

1.2.2 Encoding

The encoding operation simply needs to copy the value of the single-bit message, call it $x\in\mathbb{F}_2$, to two additional *ancilla* bits which are initialized to 0. This can be accomplished by CNOT gates in a Boolean circuit. The CNOT gate has the action $(x,y)\mapsto(x,x+y)$ for $(x,y)\in\mathbb{F}_2^2$, so that addition is modulo 2, and it is represented diagramamtically as follows.



We may then specify the action of the encoding operation by the following circuit diagram.



1.2.3 Syndrome decoding

It is not actually necessary to employ majority vote in order to decode the repetition code. Instead, we can use the fact that it is a *linear code* and employ *syndrome decoding*. Trivially in this case, the sum of two codewords is also a codeword. Observe that an arbitrary codeword $x\in\mathcal{C}$, whose bit values we call $x_1x_2x_3$, satisfies the two *parity checks* $s_1=x_2+x_3=0$ and $s_2=x_1+x_3=0$. A pattern of bit flip errors $z \in \mathbb{F}_2^3$ acts on a codeword by addition, producing from x the noisy codeword $y = x + z$. Computing the parity checks on y now produces the *syndromes* $s_1 := y_2 + y_3 = z_2 + z_3$ and $s_2 := y_1 + y_3 = z_1 + z_3$. These syndromes allow the decoder to diagnose and correct the error. For example, if $z = 100$, then $s_1 = 0$ and $s_2 = 1$. No other single bit flip will result in this syndrome, so upon seeing this syndrome, the decoder can simply flip the first bit. All the syndrome possibilities are given in the following table:

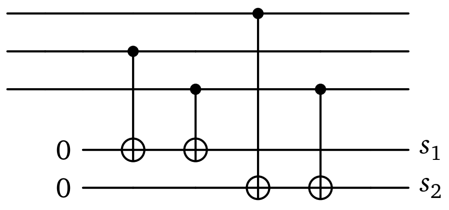
Syndrome s_1s_2	Error position
00	$\emptyset$
01	1
10	2
11	3

Note that, as claimed, the syndromes uniquely specify an error location (under the assumption that only one error occurred).

Syndrome decoding therefore consists of three steps.

1. The syndrome of the received noisy codeword is computed.
2. The location of the error is determined from the syndrome value, here by simply looking it up in the table.
3. The corresponding bit is flipped in order to (hopefully) restore the original codeword.

The following circuit diagram depicts the computation of the syndrome, which is stored in two ancilla bits.



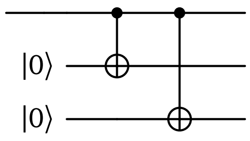
Syndrome decoding is a kind of *codespace decoding* (note: this is not standard terminology), since the codeword is restored, not the message directly. The original codeword can be further decoded back to an estimate of the message by using the inverse of the encoding operation.

1.3 Quantum repetition code

As we have specified the encoding and decoding operations of the repetition code via Boolean circuits, we may instead regard these circuits as quantum circuits and apply them to superposition input states. That is, instead of trying to create a quantum repetition code by repeating arbitrary states to be protected, which is forbidden by no cloning, let us just quantize the classical repetition code. The resulting code will protect a single qubit from flips of the standard basis states $\{|0\rangle, |1\rangle\}$, errors which correspond to the action of the Pauli operator σ_x, which we shall denote by X.

1.3.1 Encoding

In particular, we replace the encoding circuit with

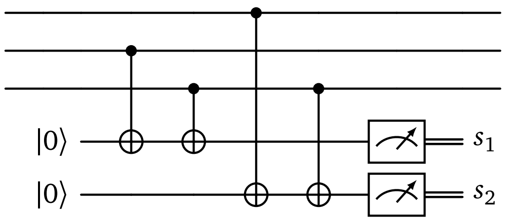


and the noise with Pauli X operators.

Therefore an input state $|\psi\rangle$ of the form $|\psi\rangle = a|0\rangle + b|1\rangle$ with $a, b \in \mathbb{C}$ will be transformed by the encoder into $|\widehat{\psi}\rangle = a|0\rangle \otimes |0\rangle \otimes |0\rangle + b|1\rangle \otimes |1\rangle \otimes |1\rangle$. We will usually drop the tensor product symbols and simply write $|000\rangle$ and $|111\rangle$, or sometimes $|000\rangle_{123}$ to emphasize that we are dealing with qubits 1, 2, and 3. The code $\mathcal{C}$ is now the span of such states, i.e. $\mathcal{C} = \text{span}(|000\rangle, |111\rangle)$. Observe that the operator $X_1X_2X_3 = X_1 \otimes X_2 \otimes X_3$ interchanges the two codewords $|000\rangle$ and $|111\rangle$, while $Z_1 = Z_1 \otimes \mathbb{1}_2 \otimes \mathbb{1}_3$ for $Z = \sigma_z$ gives a -1 phase to $|111\rangle$. The pair $\widehat{X} = X_1X_2X_3$ and $\widehat{Z} = Z_1$ are *logical operators* for the code; they act on the codespace the way X and Z act on a single qubit.

1.3.2 Decoding

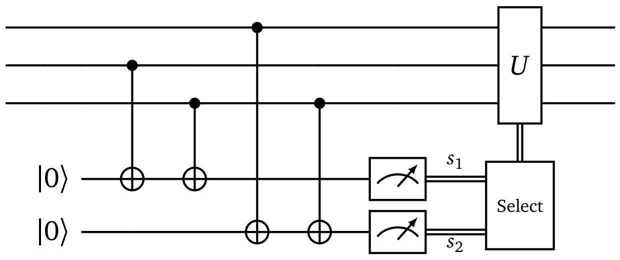
On the decoding side, we must now measure in order to obtain the syndromes, like so:



Just as in the classical case, the CNOT gates add the value of the control qubit to that of the ancilla, and so each ancilla accumulates one of the syndrome parities. Then the value of the syndrome parity is read out by measurement.

Another way to understand this operation is that measurement of the first ancilla has the same effect as measuring the observable $Z_2Z_3 = \mathbb{1}_1 \otimes Z_2 \otimes Z_3$, whose corresponding eigenprojectors are $P_0 = \mathbb{1}_1 \otimes (|00\rangle\langle 00| + |11\rangle\langle 11|)_{23}$ and $P_1 = \mathbb{1}_1 \otimes (|01\rangle\langle 01| + |10\rangle\langle 10|)_{23}$. Measurement of the second is equivalent to measuring $Z_1Z_3 = Z_1 \otimes \mathbb{1}_2 \otimes Z_3$, which has similar parity-based eigenprojectors. Observe that for both measurements, it is not the case that Z_1, Z_2, and Z_3 are individually measured and then the parities computed. Instead, the parity calculation is done quantum-mechanically by the CNOT gates, and then the resulting parity values are measured from the ancilla qubits.

The remainder correction procedure continues very much as before. Indeed, the second step of determining the error location is precisely the same as before; this is a classical computation. The third step differs in that now the correction operation is an X at the appropriate qubit location. The following circuit diagram illustrates all three steps of the correction procedure. After the syndrome measurement, a classical computation (denoted 'Select') determines which correction operation U to apply. Observe that the fact that the computation of the correction operation is purely classical is reflected in the diagram by the 'Select' operation only having classical inputs (the horizontal double wires) and outputs (the vertical double wire).



1.3.3 Correction maintains superposition of encoded information

It can be seen by direct calculation that any single qubit bit flip error will be corrected, no matter the encoded state $|\widehat{\psi}\rangle$. For example, if the second qubit is flipped, then $|\widehat{\psi}\rangle$ becomes $|\widehat{\psi'}\rangle = a|010\rangle + b|101\rangle$. Measurement of s_1 will yield $s_1 = 1$ since the parity of the second and third bits differs. Moreover, importantly, the state $|\widehat{\psi'}\rangle$ will be unchanged by the measurement. Measurement of s_2 will yield $s_2 = 0$, again leaving the state unchanged. Finally, the correction procedure will indicate that qubit two should be flipped, which restores the original state $|\widehat{\psi}\rangle$. Thus, the syndrome decoding procedure is able to correct the error without destroying the superposition!

1.3.4 Discretization of errors

Instead of a bit flip of a single qubit, consider rotation of a single qubit around the $\widehat{x}$ axis by some angle $\theta \in \mathbb{R}$. The unitary operator which implements this rotation is simply $U = e^{-i\frac{\theta}{2}\sigma_x} = \cos\frac{\theta}{2}\mathbb{1} + \sin\frac{\theta}{2}X$, a superposition of correctable errors. In this case the error correction procedure, specifically the syndrome measurement, breaks the superposition and discretizes the error to either of these two cases!

Suppose the rotation affects the first qubit. Then the noisy codeword is given by

$$|\widehat{\psi'}\rangle = U|\widehat{\psi}\rangle = \cos\tfrac{\theta}{2}|\widehat{\psi}\rangle + \sin\tfrac{\theta}{2}X_1|\widehat{\psi}\rangle\,. \tag{1.1}$$

For s_1, both terms in the superposition lead to a measurement result of $s_1 = 0$. That is, the measurement outcome is deterministic, and will necessarily be $s_1 = 0$. However, for s_2 the first term leads deterministically to $s_2 = 0$, while the second leads to $s_2 = 1$. The probabilities of the two cases are $\cos^2\frac{\theta}{2}$ and $\sin^2\frac{\theta}{2}$, respectively. Therefore, for $s_1s_2 = 00$ the post-measurement state is proportional to $|\widehat{\psi}\rangle$, while for $s_1s_2 = 01$ the post-measurement state is proportional to $X_1|\widehat{\psi}\rangle$.

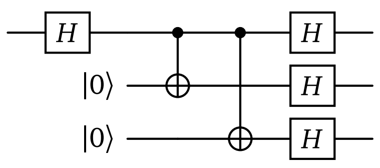
In the former case the syndrome correctly indicates that there is no error, so the decoder will take no action. In the latter case the syndrome correctly indicates that the error is on the first qubit, so the decoder will apply X_1 to $X_1|\widehat{\psi}\rangle$, which restores $|\widehat{\psi}\rangle$. Hence, in both cases the encoded state is correctly recovered. The angle of the rotation only determines the relative probability of the two different cases $s_2 = 0$ and $s_2 = 1$. For a small angle, the syndrome measurement will most often result in $s_1s_2 = 00$, while for a large angle it will often result in $s_1s_2 = 10$. In this way the analog rotation error is discretized to a probabilistic mixture of either no rotation ($\mathbb{1}$) or a π rotation (X).

The coefficients of the two terms in (1.1) come from the unitary operation, but actually their particular form played no role in the argument. The syndrome decoding scheme can correct an arbitrary single-qubit error of the form $E = c_1\mathbb{1} + c_2X$, where $c_1, c_2 \in \mathbb{C}$. As before, the coefficients only determine the probabilities of the various syndrome measurement results. The trivial syndrome indicating no error will occur with probability $|c_1|^2/(|c_1|^2 + |c_2|^2)$, while the nontrivial syndrome will occur with probability $|c_2|^2/(|c_1|^2 + |c_2|^2)$.

Syndrome decoding of the quantum repetition code illustrates how the original objections to the possibility of quantum error correction are overcome. First, it is not necessary to clone the qubit state $|\psi\rangle$. Second, the decoding procedure does not damage the analog nature (superposition) of the quantum information. Third, the decoding procedure does discretize all errors which are superpositions of $\mathbb{1}$ and X to a probabilistic mixture of either $\mathbb{1}$ or X. The drawback is that this repetition code only protects against single-qubit errors of the form $E = c_1\mathbb{1} + c_2X$.

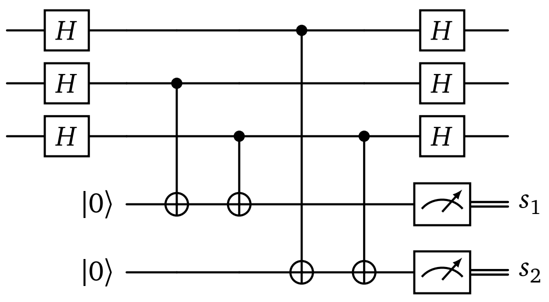
1.4 Repetition coding to correct phase errors

Of course, there is nothing special about X error operators. We could just rotate the entire error correction scheme so that it protects against Z errors, or any errors along any axis. Single qubit Z errors are called *phase flips*. To protect against single-qubit phase flips it is sensible to change the encoding so that $|\psi\rangle = a|+\rangle + b|-\rangle$ is transformed to $|\widehat{\psi}\rangle = a|{+}{+}{+}\rangle + b|{-}{-}{-}\rangle$, where $|\pm\rangle = \frac{1}{\sqrt{2}}(|+\rangle \pm |-\rangle)$. This can be accomplished by applying the *Hadamard gate*, denoted H, to the inputs and outputs of the repetition code encoder. The Hadamard gate has the action $|x\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle + (-1)^x|1\rangle)$ for $x \in \mathbb{F}_2$. The resulting encoder for the *phase flip repetition code* takes the following form.



Note that the logical operators are now X_1 and $Z_1Z_2Z_3$.

Similarly, we may utilize the Hadamard gate in order to recycle the decoding circuit from the bit flip repetition code. To check for $\pm$ parity in the second two qubits, for instance, simply apply H to all qubits in order to rotate $|\pm\rangle$ to the standard basis, compute the parities as usual, and finally apply H to rotate back. The end result is to measure the observables X_2X_3 and X_1X_3. The resulting circuit is just the following.

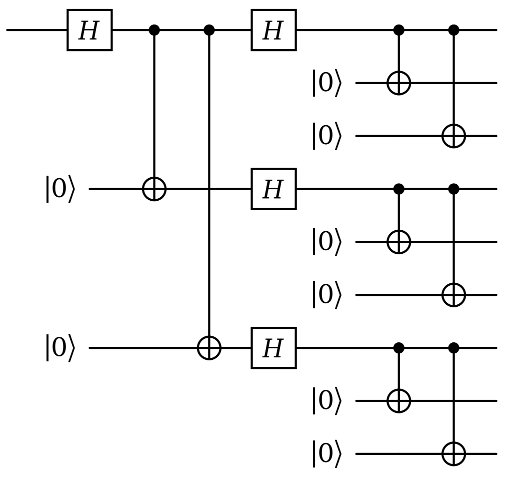


The syndrome is used to determine the location of the error exactly as before. Now, however, the correction operation is Z at the appropriate location. The phase-flip repetition code corrects all single-qubit phase-flip errors, as well as single-qubit errors of the form $E = c_1\mathbb{1} + c_2Z$.

1.5 The Shor code

1.5.1 Concatenation

Shor's insight was to realize that *concatenating* the bit flip and phase flip repetition codes produces a code which can correct *any* single-qubit error. Code concatenation refers to concatenating the encoders, first applying the encoder of a given code and then applying to each of its outputs the encoder of another code. In the Shor code, the qubit to be protected is first encoded into the phase flip repetition code, and then each physical qubit is separately encoded in the bit flip repetition code. The encoding circuit is a concatenation of the phase and bit flip repetition encoders and takes the form



The second encoder, in this case the bit flip repetition code, is the "inner code" (in the sense that it is closer to the noisy channel), while the first encoder is the "outer code".

We can think of concatenation in a different but also very useful way: The codewords of the outer encoder are not built from the physical qubits directly, but from the codewords of the inner code. That is, instead of taking the phase flip repetition codewords to be products of the single qubit $|\pm\rangle$ states, we instead use the encoded $|\hat{\pm}\rangle$ states from the bit flip repetition code. The encoded $|\pm\rangle$ states of the Shor code, denoted by $|\tilde{\pm}\rangle$, are the following nine-qubit states:

$$|\tilde{+}\rangle = |\hat{+}\rangle|\hat{+}\rangle|\hat{+}\rangle = (|000\rangle + |111\rangle)(|000\rangle + |111\rangle)(|000\rangle + |111\rangle)\,, \tag{1.2}$$

$$|\tilde{-}\rangle = |\hat{-}\rangle|\hat{-}\rangle|\hat{-}\rangle = (|000\rangle - |111\rangle)(|000\rangle - |111\rangle)(|000\rangle - |111\rangle)\,. \tag{1.3}$$

The encoded $|\tilde{0}\rangle$ and $|\tilde{1}\rangle$ states are simply $|\tilde{0}\rangle = |\tilde{+}\rangle + |\tilde{-}\rangle$ and $|\tilde{1}\rangle = |\tilde{+}\rangle - |\tilde{-}\rangle$. Expanding out the expressions yields

$$\begin{aligned}
|\tilde{0}\rangle &= |\widehat{\tilde{0}\tilde{0}\tilde{0}}\rangle + |\widehat{\tilde{0}\tilde{1}\tilde{1}}\rangle + |\widehat{\tilde{1}\tilde{0}\tilde{1}}\rangle + |\widehat{\tilde{1}\tilde{1}\tilde{0}}\rangle && (1.4)\\
&= |000000000\rangle + |000111111\rangle + |111000111\rangle + |111111000\rangle && (1.5)\\
|\tilde{1}\rangle &= |\widehat{\tilde{0}\tilde{0}\tilde{1}}\rangle + |\widehat{\tilde{0}\tilde{1}\tilde{0}}\rangle + |\widehat{\tilde{1}\tilde{0}\tilde{0}}\rangle + |\widehat{\tilde{1}\tilde{1}\tilde{1}}\rangle && (1.6)\\
&= |000000111\rangle + |000111000\rangle + |111000000\rangle + |111111111\rangle && (1.7)
\end{aligned}$$

This way of viewing a concatenated code implies that the logical operators should essentially be the logical operators of the phase-flip repetition code, but the physical qubit operators therein should be replaced with the logical operators of the bit-flip repetition code. That is, the logical operators ought to be $\tilde{X} = X_1X_2X_3$ and $\tilde{Z} = Z_1Z_4Z_7$. It is straightforward to verify that these operators indeed have the action on $|\tilde{0}\rangle$ and $|\tilde{1}\rangle$ that X and Z have on $|0\rangle$ and $|1\rangle$, respectively.[2]

1.5.2 Correction

The decoding operation is also based on concatenation, proceding in reverse order to the encoding. First bit flip errors are corrected using the inner code, and subsequently phase flip errors are corrected by the outer code. Let us examine this procedure more carefully to see how single-qubit errors are corrected. The first step to establishing that the Shor code can correct any single-qubit error is to establish that it can correct any single-qubit Pauli error. To this end we begin by considering single X and Z errors.

Bit flips are corrected just as before by the inner code, in each of the three blocks of three qubits. Syndrome measurements as defined above are made in each block of three qubits, and the damaged qubit is identified and corrected by possibly applying an X operator. When the error is a single X, this procedure will restore the original Shor-encoded state. On the other hand, when the error is a single Z, then the procedure does nothing. The single Z commutes with the syndrome measurement, and therefore the syndrome is trivial. No correction operation will be applied, but the single Z remains. And finally, when the error is a single Y, then the syndrome measurment will again infer its location and apply a correction X. Therefore the overall action on the code is $XY = iZ$, meaning the Y error is converted into a Z error.

After the correction operations of the inner code, only a possible Z error remains. Observe that the codewords are such that a Z error acting on any position in a block is equivalent to it acting on the first position in the block. That is, for example, Z_2 and Z_1 have the same effect on $|\widehat{\pm}\rangle_{123}$. Namely, they both produce $|\widehat{\mp}\rangle_{123}$. Two errors that have the same effect on the codespace are said to be *degenerate*. Due to the degeneracy, we can assume that only the first qubit in any block could be affected by a phase flip.

To correct phase flips, syndrome measurements for the phase repetition code need to be performed, in order to indicate which block is in error. The outer layer essentially sees the encoded states of the inner layers, not the bare qubit states themselves; after all, the inner layer decoder ensures that the qubits are in the bit-flip repetition codespace. Following the procedure for constructing the logical operators, the observables to be measured are therefore $\widehat{X}_{ii}\widehat{X}_{iii}$ and $\widehat{X}_i\widehat{X}_{iii}$ where the hat indicates the bit flip logical operator and the index i, ii, or iii indicates the block. In terms of Pauli operators on the physical qubits, we have $\widehat{X}_{ii}\widehat{X}_{iii} = X_4X_5X_6X_7X_8X_9$ and $\widehat{X}_i\widehat{X}_{iii} = X_1X_2X_3X_7X_8X_9$.

Thus, $s_1 = 0$ corresponds to the projector $\mathbb{1}_{123} \otimes (|\widehat{+}\widehat{+}\rangle\langle\widehat{+}\widehat{+}| + |\widehat{-}\widehat{-}\rangle\langle\widehat{-}\widehat{-}|)_{456789}$ and $s_1 = 1$ to the projector $\mathbb{1}_{123} \otimes (|\widehat{+}\widehat{-}\rangle\langle\widehat{+}\widehat{-}| + |\widehat{-}\widehat{+}\rangle\langle\widehat{-}\widehat{+}|)_{456789}$. The syndrome indicates which block of three qubits, be it 123, 456, or 789, has the error. The error is then removed by applying Z to qubit 1, 4, or 7 as appropriate. This procedure restores the original codeword from any single-qubit Pauli error. If we wish to obtain the original encoded qubit at this point, we may simply run the encoding circuit in reverse.

Just as the syndrome decoder measurements of the bit flip repetition code discretizes superpositions of $\mathbb{1}$ and X, the syndrome decoder measurements of the Shor code will discretize any superposition of Pauli errors, any error of the form $E = c_0\mathbb{1} + c_1X + c_2Y + c_3Z$ for $c_i \in \mathbb{C}$. This holds because each term in this expression leads to a different value of the measured syndromes. Since every linear operator E on qubits can be written in this way (the Pauli operators span the set

[2]Note that our convention for the codewords and logical operators differs from that of Nielsen and Chuang. We have the logical X operator composed of Pauli X operators and similarly for logical Z.

of 2x2 matrices when complex linear combinations are allowed), every single-qubit error can be corrected.

1.6 The stabilizer formalism

It is a lot more convenient to describe the Shor code and many other quantum error-correcting codes in terms of their *stabilizers*. Let us start back with the repetition code.

1.6.1 Stabilizers

Recall the codewords are $|000\rangle$ and $|111\rangle$, so the code subspace itself is the span of these two vectors. Observe that this subspace is the simultaneous $+1$ eigenspace of the operators Z_2Z_3 and Z_1Z_3: For $x \in \{0,1\}$ we have

$$I_1 \otimes Z_2 \otimes Z_3|xxx\rangle = |xxx\rangle \qquad \text{and} \qquad Z_1 \otimes I_2 \otimes Z_3|xxx\rangle = |xxx\rangle\,, \tag{1.8}$$

simply because there are an even number of Z factors in each operator and $Z|1\rangle = -|1\rangle$.

These two operators are *stabilizers* of the code, since they leave the codewords invariant. A *stabilizer code* is a quantum code (a subspace of the full quantum state space) which is defined as the simultaneous $+1$ eigenspace of some collection of stabilizer operators. We typically only consider stabilizer operators which are products of Pauli operators, as here.

Note that the product of stabilizers is also a stabilizer, e.g. Z_1Z_2 is also a stabilizer. In general, the stabilizers will form a group, and it is enough to specify *generators* of the group. For the three-qubit repetition code, generators are Z_2Z_3 and Z_1Z_3, while the group itself is $\{I, Z_1Z_2, Z_2Z_3, Z_1Z_3\}$. Any two of the nontrivial group elements could serve as generators.

1.6.2 Syndromes

Not coincidentally, the syndromes of the repetition code are precisely the result of measuring the stabilizer operators. For Pauli stabilizers, which is the case of interest to us, the stabilizers are not only unitary but also Hermitian. Thus they are valid quantum observables, and we may consider measurement corresponding to their eigensubspaces.

Indeed, the parity projection operators $P_0 = \mathbb{1}_1 \otimes (|00\rangle\langle 00| + |11\rangle\langle 11|)_{23}$ and $P_1 = \mathbb{1}_1 \otimes (|01\rangle\langle 01| + |10\rangle\langle 10|)_{23}$ introduced in §1.3.2 are precisely the eigensubspace projectors of Z_2Z_3. Since this operator squares to the identity, we have

$$P_0 = \tfrac{1}{2}(I + Z_2Z_3) \qquad \text{and} \qquad P_1 = \tfrac{1}{2}(I - Z_2Z_3)\,. \tag{1.9}$$

The former is associated with syndrome value 0 and the latter with syndrome value 1. This fits the standard formalism of quantum mechanics by defining the syndrome value s in terms of the eigenvalue λ of Z_2Z_3 as $\lambda = (-1)^s$. This framework also applies to the stabilizer Z_1Z_3.

1.6.3 Errors

It is important to see that the stabilizers have relatively simple measurment circuits, but it is not necessary to use these circuits or the projector description in order to understand what happens in the decoding process. Instead, note that an error X_k on qubit k will transform the encoded state into a state which is again the simultaneous eigenstate of the two stabilizer operators. The reason is that the operator X_k either commutes or anticommutes with each stabilizer. If it commutes with the stabilizer, say we have X_1 and Z_2Z_3, then clearly $X_1|\widehat{\psi}\rangle$ is still a $+1$ eigenstate of Z_2Z_3: $Z_2Z_3X_1|\widehat{\psi}\rangle = X_1Z_2Z_3|\widehat{\psi}\rangle = X_1|\widehat{\psi}\rangle$. On the other hand, if the two anticommute, as X_1 does with Z_1Z_3, then the eigenvalue is simply flipped to -1: $Z_1Z_3X_1|\widehat{\psi}\rangle = -X_1Z_1Z_3|\widehat{\psi}\rangle = -X_1|\widehat{\psi}\rangle$.

Since the syndrome is the eigenvalue of the associated stabilizer for the noisy codeword, *the effect of an error is simply to change the syndrome of all stabilizers with which it anticommutes*. The stabilizer measurement will not cause further changes to the state. It is then easy to check that all single-qubit X errors lead to distinct syndromes, and therefore can be corrected.

1.6.4 Correctable errors

Which errors are correctable depends on the properties of the code; this is not a property which defines a code. That is, we define a code, perhaps by giving stabilizer operators, and then determine which errors are correctable.

Moreover, there can be different sets of correctable errors. For instance, the repetition code can correct errors X_2X_3, X_1X_3, and X_1X_2, along with no error (I). The syndrome 00 still indicates no error, 01 indicates X_2X_3, 10 indicates X_1X_3, and 11 indicates X_1X_2. Each syndrome is associated with one of the correctable errors, so the appropriate correction can be performed.

Note that this does not imply that the repetition code can correct all single and two-qubit errors simultaneously! It can correct all single-qubit errors or it can correct all two-qubit errors, but not both sets at the same time.

Additionally, in the quantum case it turns out that the above property that each syndrome is associated to a single correctable error is sufficient for correction, but not necessary. As we have already seen with the Shor code, different errors can have the same effect on the code space. Since multiple errors can have the same effect, it is enough for the syndrome to identify which of these effects occurred on the codespace and correct it.

1.6.5 Shor code stabilizer structure

We have already implicitly seen the stabilizers of the Shor code; they are the observables measured to obtain the syndrome. There are six of Z-type: Z_1Z_2, Z_2Z_3, Z_4Z_5, Z_5Z_6, Z_7Z_8, and Z_8Z_9, and two of X-type: $X_4X_5X_6X_7X_8X_9$ and $X_1X_2X_3X_7X_8X_9$. Laid out the nine qubits as a square lattice and using $\cdot$ to indicate an identity operator, the six Z-type stabilizers are

$$\begin{matrix} Z & Z & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{matrix}, \quad \begin{matrix} \cdot & Z & Z \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{matrix}, \quad \begin{matrix} \cdot & \cdot & \cdot \\ Z & Z & \cdot \\ \cdot & \cdot & \cdot \end{matrix}, \quad \begin{matrix} \cdot & \cdot & \cdot \\ \cdot & Z & Z \\ \cdot & \cdot & \cdot \end{matrix}, \quad \begin{matrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ Z & Z & \cdot \end{matrix}, \quad \begin{matrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & Z & Z \end{matrix} \tag{1.10}$$

while the two X-type stabilizers are

$$\begin{matrix} X & X & X \\ X & X & X \\ \cdot & \cdot & \cdot \end{matrix}, \qquad \begin{matrix} \cdot & \cdot & \cdot \\ X & X & X \\ X & X & X \end{matrix} \tag{1.11}$$

Finally, the logical operators are just X operators in the first row or Z operators in the first column:

$$\widetilde{X} = \begin{matrix} X & X & X \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{matrix}, \qquad \widetilde{Z} = \begin{matrix} Z & \cdot & \cdot \\ Z & \cdot & \cdot \\ Z & \cdot & \cdot \end{matrix} \tag{1.12}$$

It is straightforward to check that all single-qubit Pauli errors are correctable. They either lead to distinct syndromes, or have equivalent action on the codespace for the same syndrome, as shown in the following table. The syndromes are partitioned into pairs, the first three corresponding to the three pairs of Z-type checks in each row and the final pair the two X-type checks.

Error	Syndrome
X_1	10 00 00 00
X_2	11 00 00 00
X_3	01 00 00 00
X_4	00 10 00 00
X_5	00 11 00 00
X_6	00 01 00 00
X_7	00 00 10 00
X_8	00 00 11 00
X_9	00 00 01 00

Error	Syndrome
Y_1	10 00 00 10
Y_2	11 00 00 10
Y_3	01 00 00 10
Y_4	00 10 00 11
Y_5	00 11 00 11
Y_6	00 01 00 11
Y_7	00 00 10 01
Y_8	00 00 11 01
Y_9	00 00 01 01

Error	Syndrome
$\emptyset$	00 00 00 00
$Z_7 \simeq Z_8 \simeq Z_9$	00 00 00 01
$Z_1 \simeq Z_2 \simeq Z_3$	00 00 00 10
$Z_4 \simeq Z_5 \simeq Z_6$	00 00 00 11

2 General quantum codes

In this chapter we investigate quantum error correction at a more general level. We start with how noise is described, then give a definition of quantum error correction, and finally establish the Knill-Laflamme conditions on exact correction.

2.1 Quantum noise

2.1.1 Unitary description

Noise affecting quantum information comes from interaction with its surroundings, which is usually just called the *environment*. In the standard formalism of quantum mechanics, any interaction is described by a unitary transformation of the information of interest, stored in quantum system labelled A, and the environmental degrees of freedom, which we will label E. The environment is essentially everything outside of A, and is regarded as initially independent of A and in a pure state, say $|0\rangle_E$. Then any noise process is of the form

$$|\psi\rangle_A \otimes |0\rangle_E \to |\psi'\rangle_{AE} = U_{AE}|\psi\rangle_A|0\rangle_E\,, \tag{2.1}$$

for some unitary U_{AE}.

2.1.2 Kraus operator description

We can describe the effect of the noise solely in terms of A as a *quantum channel* $\mathcal{N}$, by tracing out the environment E in the expression above:

$$\mathcal{N} : |\psi\rangle\langle\psi|_A \mapsto \mathrm{Tr}_E[U_{AE}(|\psi\rangle\langle\psi|_A \otimes |0\rangle\langle 0|_E)U_{AE}^\dagger]\,. \tag{2.2}$$

It is convenient to expand in an orthonormal basis of E and define the *Kraus*[1] *operators*:

$$|\psi'\rangle_{AE} = \sum_k (M_k|\psi\rangle)_A \otimes |k\rangle_E\,, \tag{2.3}$$

which holds for $M_k = {}_E\langle k|U_{AE}|0\rangle_E$, an operator on A. Unitarity of U_{AE}, i.e. $\langle\psi'|\psi'\rangle = 1$ for all input $|\psi\rangle$, translates into the condition $\sum_k M_k^\dagger M_k = \mathbb{1}$. Then the action of the channel on A is given more simply in terms of the Kraus operators:

$$\mathcal{N} : |\psi\rangle\langle\psi| \mapsto \sum_k M_k|\psi\rangle\langle\psi|M_k^\dagger\,. \tag{2.4}$$

In §1.3.4 the error operator $E = c_0\mathbb{1} + c_1\sigma_x + c_2\sigma_y + c_3\sigma_z$ was just an arbitrary Kraus operator acting on a qubit.

2.1.3 Examples: qubit channels

Examples of qubit channels include

[1]Karl Kraus

Pauli channels. We have already implicitly considered the case of a *Pauli channel* in the previous lecture, when considering Pauli operators as errors, i.e. as Kraus operators. A general Pauli channel takes the form $\rho \mapsto \sum_{k=0}^{3} p_k \sigma_k \rho \sigma_k$, where σ_k are the Pauli operators with $\sigma_0 = \mathbb{1}$ and $p_k \geq 0$ are probabilities, meaning $\sum_k p_k = 1$.

Depolarization. Several special cases of Pauli channels are of interest. In the *depolarizing channel* $p_1 = p_2 = p_3$, so that the Bloch vector of the input qubit simply shrinks towards the origin.

Dephasing. The *dephasing channel* describes loss of coherence in superpositions of the standard basis states. Here there are no bit flip errors, i.e. $p_1 = p_2 = 0$, and the Bloch vector shrinks toward the $\hat{z}$ axis.

Independent bit and phase errors. The case of independent bit and phase noise is a convenient channel to study. For bit errors occuring at probability p and phase errors occuring at probability q, this channel is the Pauli channel with $p_1 = p(1-q)$, $p_2 = pq$, $p_3 = (1-p)q$.

Leakage. The *quantum erasure channel* describes leakage of the qubit information into some other levels of the physical system (e.g. in a trapped ion qubit in which two specified electronic levels are used as the qubit, leakage occurs when the population moves to some other level). Formally, the input qubit remains as it was with some probability $1-q$ and moves to another level, call it $|2\rangle$, with probability q: $\rho \mapsto (1-q)\rho + q|2\rangle\langle 2|$. It can be shown that this action has Kraus operators $M_0 = \sqrt{1-q}\mathbb{1}$, $M_1 = \sqrt{q}|2\rangle\langle 0|$ and $M_2 = \sqrt{q}|2\rangle\langle 1|$.

Note that the output is no longer a qubit. It can be restored to the qubit space by simply randomly mapping the state $|2\rangle$ to either $|0\rangle$ or $|1\rangle$. Physically this would correspond to measuring the system to see if the population is still in the qubit subspace, and mapping it back if not. In this case the measurement result indicates whether the qubit is ok or leaked, which is an additional piece of information (a classical bit). The channel can then be described by $\rho \mapsto (1-q)\rho \otimes |\text{ok}\rangle\langle\text{ok}| + q\tau \otimes |\text{leak}\rangle\langle\text{leak}|$, where $\tau = \frac{1}{2}\mathbb{1}$ is the maximally-mixed state.

Amplitude damping. The process of spontaneous decay of the $|1\rangle$ to the $|0\rangle$ state is described by the *amplitude damping* channel. It is defined by the damping parameter $\gamma \in [0,1]$ and has two Kraus operators

$$M_0 = |0\rangle\langle 0| + \sqrt{1-\gamma}|1\rangle\langle 1| \qquad \text{and} \qquad M_1 = \sqrt{\gamma}|0\rangle\langle 1|\,. \tag{2.5}$$

The latter corresponds to decay, as $|1\rangle$ jumps to $|0\rangle$, which occurs with probability γ (when the initial state is $|1\rangle$). The former Kraus operator corresponds to the reweighting of the state in favor of $|0\rangle$ which occurs when no decay occurs. Essentially, the decay event should be accompanied by some observable change in the environment, e.g. a photon is emitted by an atom jumping from an excited state $|1\rangle$ to a lower-energy state $|0\rangle$, and when no photon is observed the conclusion is that the state was more likely to already have been in the state $|0\rangle$.

2.1.4 I.I.D. channels

Noise acting on a collection of qubits can take an arbitrary form consistent with (2.1). In these notes we will focus on the case of noise in which each qubit experiences the same noise channel, independently of all other qubits. For example, consider noise $\mathcal{N}$ acting on the nine qubits of the Shor code of the form $\mathcal{N} = \mathcal{N}_1' \otimes \mathcal{N}_2' \otimes \cdots \otimes \mathcal{N}_9'$, where the qubit channel $\mathcal{N}_j'$ is a Pauli channel that acts on qubit j. When all the $\mathcal{N}_j'$ are identical, the Pauli errors on the qubits are *independently and identically distributed*, so this kind of channel is called i.i.d. The Kraus operators of an i.i.d. channel (or even just independent but not identical) are tensor products of the Kraus operators of the individual qubit channels.

From the starting point of i.i.d. noise we can branch out into independent but not identical noise, for instance, or consider weak correlations between noise on different qubits. Other noise models, such as collective rotation of all qubits, are also of interest, but are less immediately relevant to contemporary experiments. Note that a collective rotation of some collection of qubits, say three, is implemented by an operator of the form $e^{i\theta((\sigma_x)_1+(\sigma_x)_2+(\sigma_x)_3)}$, where $(\sigma_x)_2 = \mathbb{1}\otimes\sigma_x\otimes\mathbb{1}$ and so forth. The exponential of a sum of product operators is not itself a product operator, and hence collective rotation is not an instance of i.i.d. noise.

2.1.5 Errors versus channels

Often we specify noise directly in terms of error operators, some $\{M_k\}$ such that the noise action is given by (2.4). The distinction to starting with a bone fide quantum channel is that the error operators might not obey the normalization condition $\sum_k M_k^\dagger M_k = \mathbb{1}$. At the mathematical level, the resulting mapping $\mathcal{N}$ is completely-positive, meaning it maps positive operators to positive operators, even when acting only on parts of entangled states, but it is not necessarily trace-preserving. It turns out that the trace-preserving condition is not terribly crucial to the mathematical formulation of error correction, as we will see below.

2.2 Definition of quantum error correction

2.2.1 Error correction

Setup Suppose that $\mathcal{N}$ is a noisy channel that transforms a collection of n qubits in some way, maybe even to some other kinds of quantum systems. Let's lump all the qubits together and call them system A and the output of the channel system B. Usually we will consider the case that $B \simeq A$, as in the Shor code, where the output of the noise is also a nine-qubit system. However it is useful to keep the input and output systems distinct in the following arguments, and no simplification is gained by having $B \simeq A$.

We would like to store some quantum information in A and protect it from $\mathcal{N}_{B|A}$. At least formally, we can imagine that initially the quantum information is stored in system Q and then it is encoded into A by an encoding operation $\mathcal{E}_{A|Q}$. After the action of the noise $\mathcal{N}$, the quantum information is decoded from B back to Q by a decoding operation $\mathcal{D}_{Q|B}$. Any quantum operation, noise or not, is described by some quantum channel, and therefore the encoding operation $\mathcal{E}$ and the decoding operation $\mathcal{D}$ are each described by a quantum channel.

Exact correction If the quantum information is indeed protected from $\mathcal{N}_{B|A}$, then there is a decoding quantum operation $\mathcal{D}_{Q|B}$ which transforms B back to Q such that the composition $\mathcal{D}_{Q|B} \circ \mathcal{N}_{B|A} \circ \mathcal{E}_{A|Q}$ is equal to the identity channel $\mathcal{I}_Q$ on Q:

$$\mathcal{D}\circ\mathcal{N}\circ\mathcal{E} = \mathcal{I}\,. \tag{2.6}$$

This is the basic requirement for exact quantum error correction. Error correction is specified by the encoder and decoder. Depending on their properties, they either can or cannot correct the action of $\mathcal{N}$.

An equivalent condition is that $\mathcal{D}\circ\mathcal{N}\circ\mathcal{E}$ preserves the maximally-entangled state. Let $|\Phi\rangle_{QQ'}$ be the maximally-entangled state on the bipartite system QQ', where $|Q'| = |Q|$. Then (2.6) is equivalent to

$$\mathcal{D}_{Q|B}\circ\mathcal{N}_{B|A}\circ\mathcal{E}_{A|Q}[\Phi_{QQ'}] = \Phi_{QQ'}\,. \tag{2.7}$$

Note that $\mathcal{D}\circ\mathcal{N}\circ\mathcal{E}$ acts only on Q, while Q' is an ancilla system. The equivalence follows from the Choi isomorphism of quantum channels and certain bipartite quantum states, which says that the action of the channel $\mathcal{D}_{Q|B}\circ\mathcal{N}_{B|A}\circ\mathcal{E}_{A|Q}$ on an arbitrary input can be computed using the state $\mathcal{D}_{Q|B}\circ\mathcal{N}_{B|A}\circ\mathcal{E}_{A|Q}[\Phi_{QQ'}]$. Therefore (2.7) implies (2.6); the other direction is immediate.

Approximate correction If some pair $(\mathcal{E},\mathcal{D})$ cannot exactly correct $\mathcal{N}$, it is natural to ask how well (or poorly) they can correct $\mathcal{N}$. This is the notion of approximate correction. Nominally, we would like to have $\mathcal{D}\circ\mathcal{N}\circ\mathcal{E}\approx_\varepsilon\mathcal{I}$ for some small $\varepsilon>0$. But we must define the notion of approximation, what $\approx_\varepsilon$ actually means. There are several ways to do this. For simplicity, in these notes we will say that $\mathcal{D}\circ\mathcal{N}\circ\mathcal{E}\approx_\varepsilon\mathcal{I}$ holds when the fidelity (squared) of the output of $\mathcal{D}\circ\mathcal{N}\circ\mathcal{E}$ applied to a maximally-entangled state $\Phi_{QQ'}$ with $\Phi_{QQ'}$ itself is larger than $1-\varepsilon$:

$$\mathrm{Tr}[\Phi_{QQ'}\mathcal{D}\circ\mathcal{N}\circ\mathcal{E}[\Phi_{QQ'}]]\geq 1-\varepsilon\,. \tag{2.8}$$

Setting the approximation parameter $\varepsilon=0$ returns us to the case of exact correction above. Another common approximation measure again uses the fidelity (squared), but considers the worst-case input pure state, e.g. the lefthand side of (2.8) is replaced by $\min_{\Psi_{QQ'}}\mathrm{Tr}[\Psi_{QQ'}\mathcal{D}\circ\mathcal{N}\circ\mathcal{E}[\Psi_{QQ'}]]$.

A common example of approximate error correction is when a subset of the Kraus operators of a given channel are exactly correctable, but the remainder are not. This is the case with the Shor code faced with an i.i.d. Pauli channel $\mathcal{N}=\mathcal{N}_1'\otimes\mathcal{N}_2'\otimes\cdots\otimes\mathcal{N}_9'$, where the qubit channel $\mathcal{N}_j'$ is a Pauli channel that acts on qubit j. Since the Kraus operators of i.i.d. channels (or even just independent but not identical) are tensor products of the Kraus operators of the individual qubit channels, the Kraus operators of $\mathcal{N}$ are products of Pauli operators. One of the Kraus operators is proportional to the identity, and there are a further 27 which have only a single nontrivial Pauli factor. As we have seen, this collection of 28 Kraus operators is correctable. These are the Pauli products with *weight* 0 and 1, respectively. Not all of the higher-weight Pauli products are exactly correctable along with the lowest weight set (though some are). Taking $\mathcal{N}_j'$ to be the depolarizing channel with small depolarization parameter $p\ll 1$, the higher-weight errors occur with probability $O(p^2)$ and smaller. Therefore the Shor code corrects depolarization noise up to and including order p. It is easy to show that the fidelity criterion above is $1-O(p^2)$, and so $\varepsilon=O(p^2)$.

Errors versus channels The example above is not the only kind of approximate correction; Section 2.4 will discuss an example in which no Kraus operator is corrected exactly. However, this simple example explains the focus on "errors", i.e. sets of not-necessarily-normalized Kraus operators (which are hopefully correctable) instead of "channels", i.e. full sets of proper Kraus operators. For i.i.d. Pauli noise, the name of the game in error correction is to be able to correct the likely errors, i.e. those with low weight.

We can define exact correctability of a set of errors M_j by regarding the errors as Kraus operators of an improperly-normalized channel $\mathcal{N}$, as discussed in §2.1.5, and requiring there to exist an encoder $\mathcal{E}$ and decoder $\mathcal{D}$ (both proper quantum channels) such that

$$\mathcal{D}\circ\mathcal{N}\circ\mathcal{E}\propto\mathcal{I}\,. \tag{2.9}$$

As with properly-normalized channels, this is equivalent to

$$\mathcal{D}\circ\mathcal{N}\circ\mathcal{E}[\Phi_{QQ'}]\propto\Phi_{QQ'}\,. \tag{2.10}$$

Moreover, (2.9) is equivalent to demanding that each error is exactly corrected, which is perhaps a definition more often encountered in the literature. Specifically, with $\mathcal{M}_j : \rho \mapsto M_j \rho M_j^\dagger$,

$$\mathcal{D} \circ \mathcal{M}_j \circ \mathcal{E} \propto \mathcal{I} \qquad \forall j \tag{2.11}$$

To see the equivalence, write (2.9) as $\sum_j \mathcal{D} \circ \mathcal{M}_j \circ \mathcal{E} \propto \mathcal{I}$. Applied to the pure state $|\psi\rangle_Q$, the statement becomes $\sum_j \rho_j = a|\psi\rangle\langle\psi|$ for some $a > 0$, where $\rho_j = \mathcal{D} \circ \mathcal{M}_j \circ \mathcal{E}[|\psi\rangle\langle\psi|]$. Picking out some particular ρ_j, we have $a|\psi\rangle\langle\psi| - \rho_j = \sum_{j' \neq j} \rho_{j'}$ and hence $a|\psi\rangle\langle\psi| - \rho_j \geq 0$. Now we can easily show that ρ_j must be proportional to $|\psi\rangle\langle\psi|$. Consider a vector $|\psi'\rangle$ which is orthogonal to $|\psi\rangle$ but contained in the support of ρ_j (if no such $|\psi'\rangle$ exists, then certainly $\rho_j \propto |\psi\rangle\langle\psi|$). We should have $\langle\psi'|(a|\psi\rangle\langle\psi| - \rho_j)|\psi'\rangle \geq 0$ by the positivity statement. However, the expression on the lefthand side is manifestly negative, and so it must indeed be that $\rho_j \propto |\psi\rangle\langle\psi|$.

Developing a notion of approximate correctability for general sets of errors is complicated by their improper normalization. Here we will only consider approximate correction at the level of channels and not errors.

2.2.2 Error-correcting codes

Isometric encoding In the Shor code example the encoding quantum operation was *isometric*, meaning it simply isometrically embeds the two-dimensional qubit space from Q into the nine qubits A. This is guaranteed by the fact that the codewords $|\widetilde{\pm}\rangle$ are orthogonal, so inner products of general states are preserved: $\langle\psi|\phi\rangle_Q = \langle\widetilde{\psi}|\widetilde{\phi}\rangle_A$. Isometries can always be implemented by unitary operators acting on the input as well as additional ancilla systems, as we will see more concretely later. Taking $\mathcal{E}$ to be an isometry in the above definitions of exact and approximate error correction, the image of Q under $\mathcal{E}$ defines the (quantum) *code*, which is just a subspace $\mathcal{C}$ in A. A collection of orthogonal states in the code are called *codewords*. While non-isometric encoding is possible, we will not consider it further here.

Codespace decoding The decoder we considered for the Shor code was actually not of the kind described above in Section 2.2.1. Instead, we called it a *codespace decoder*. This kind of decoder is sensible when $\mathcal{E}$ is an isometry, for then we can define the codespace decoder $\widehat{\mathcal{D}}$ to map B not to Q, but back to the code subspace $\mathcal{C}$ in A. Then the decoder $\mathcal{D}$ of (2.6) and (2.8) can be constructed as $\mathcal{D} = \mathcal{E}^{-1} \circ \widehat{\mathcal{D}}$, where the inverse is defined only on the range of $\mathcal{E}$, i.e. the code. Codespace decoding is useful in the setting of quantum computation, where the information needs to remain encoded until the very last step of the computation. In contrast, the original notion of decoding is more appropriate in a communication setting.

Isometric decoding The decoder in the Shor code is also of a particular nature. The stabilizer measurement projects onto orthogonal two-dimensional subspaces, each of which is then isometrically mapped back to the codespace by the corresponding correction operation. An *isometric decoder*[2] has Kraus operators $V_j^\dagger$ which are adjoints of isometries V_j from Q to B, such that the images of V_j and V_k for $j \neq k$ are disjoint. The action of such a decoder can be thought of just as in the Shor decoder, first performing a measurement described by the projection operators $\Pi_j = V_j V_j^\dagger$ and subsequently isometrically mapping the identified subspace to Q using $V_j^\dagger$.

[2]Called a *homomorphic* decoder by Kretschmann and Werner, for reasons we won't get into here.

Degenerate errors As we saw in the Shor code, it can happen that distinct errors have the same effect on the code subspace. There, a single-qubit phase error in any of the three blocks of three qubits has the same effect as a single-qubit phase error on any of the other qubits in the same block.

For an arbitrary code $\mathcal{C}$, two errors M_1 and M_2 are said to be degenerate precisely when, for all $|\widehat{\psi}\rangle \in \mathcal{C}$,

$$M_1|\widehat{\psi}\rangle = M_2|\widehat{\psi}\rangle\,. \tag{2.12}$$

Note that the errors M_1 and M_2 are not necessarily correctable in this definition. But error degeneracy does mean that it is not necessary to determine precisely which error occurred in order to recover from the noise. It is only necessary to determine the error up to degeneracy.

Examples: Four- and five-qubit codes Two particularly simple but interesting codes with fewer qubits than the Shor code are the four- and five-qubit codes. The four-qubit code has the following (unnormalized) codewords

$$|\bar{0}\rangle = |0000\rangle + |1111\rangle \qquad \text{and} \qquad |\bar{1}\rangle = |0011\rangle + |1100\rangle\,. \tag{2.13}$$

There are several versions of the five-qubit code. A particular choice has the following codewords

$$|\bar{0}\rangle = |00000\rangle - |00110\rangle - |01001\rangle - |01111\rangle + |10011\rangle - |10101\rangle - |11010\rangle - |11100\rangle\,, \tag{2.14}$$
$$|\bar{1}\rangle = |11111\rangle - |11001\rangle - |10110\rangle - |10000\rangle - |01100\rangle + |01010\rangle + |00101\rangle + |00011\rangle\,. \tag{2.15}$$

2.3 Knill-Laflamme conditions for exact correction

It would seem that determining whether a given set of errors acting on a given code subspace is exactly correctable would require either constructing a decoder, as we did with the Shor code, or proving that no decoding operation exists. For instance, we might like to determine if the four- or five-qubit codes can correct single-qubit Pauli errors (and therefore all single-qubit errors). However, there is a simple set of conditions on the code subspace and set of errors which is equivalent to exact correctability, known as the Knill[3]-Laflamme[4] conditions.

2.3.1 Statement of the conditions

Suppose $\{M_{B|A}(j)\}_{j=1}^r$ is a set of errors and $\mathcal{C} \subset \mathcal{H}_A$ is a code of dimension d. Consider the action of the errors on a maximally-entangled state on the codespace and a reference system Q' of dimension d, described using the unitary interaction with the environment. This interaction results in the state

$$|\Psi\rangle_{Q'BE} = \frac{1}{\sqrt{d}} \sum_{j=1}^{r} \sum_{k=1}^{d} |k\rangle_{Q'} \otimes M_{B|A}(j)|\widehat{k}\rangle_A \otimes |j\rangle_E \tag{2.16}$$

for orthonormal bases $\{|k\rangle_{Q'}\}$ in Q', $\{|\widehat{k}\rangle_A\}$ the encoded version of the basis in Q', i.e. an orthonormal set of codewords, and $\{|j\rangle_E\}$ in system E of dimension r. When the errors are Kraus operators of a channel, this state is properly normalized, but normalization will not be crucial in what follows, so we may consider an arbitrary set of errors. As formalized by the following statement, error correction is possible precisely when the marginal state on $Q'E$ is completely uncorrelated.

[3]Emanuel Knill
[4]Raymond Laflamme

Proposition 2.1: Knill-Laflamme conditions

The set of errors $\{M_{B|A}(j)\}$ is exactly correctable using the code $\mathcal{C}$ if and only if there exists a normalized density operator σ_E such that, for $\pi_{Q'} = \frac{1}{d}\mathbb{1}_{Q'}$ and some $a > 0$,

$$\mathrm{Tr}_B[|\Psi\rangle\langle\Psi|_{Q'BE}] = a\pi_{Q'} \otimes \sigma_E\,. \tag{2.17}$$

Equivalently, in matrix components this condition reads as follows, with $s_{j,j'} = \frac{1}{d}\langle j|\sigma|j'\rangle$,

$$\langle\widehat{k}'|M(j')^\dagger M(j)|\widehat{k}\rangle = a s_{j,j'}\delta_{k,k'} \qquad \forall\, j,j' \in \{1,\dots,r\}\,, k,k' \in \{1,\dots,d\}\,. \tag{2.18}$$

The above expressions hold with $a = 1$ when the errors are Kraus operators of a normalized quantum channel, i.e. when they satisfy $\sum_j M_{B|A}(j)^\dagger M_{B|A}(j) = \mathbb{1}_A$.

2.3.2 Examples

Applied to the four- and five-qubit codes, the Knill-Laflamme conditions imply that the former cannot correct single-qubit Pauli errors, while the latter can. Thus, we expect that the five-qubit code can correct an arbitrary single-qubit error by the discretization argument described in Section 1.5.2. However, that argument was made using a syndrome decoder, which we have not (yet) established works for the five-qubit code.

For the four-qubit code, consider phase flips on the first and third qubits, i.e. errors $Z_1 = ZIII$ and $Z_3 = IIZI$. Then we find $\langle\overline{0}|Z_1Z_3|\overline{0}\rangle = 1$ while $\langle\overline{1}|Z_1Z_3|\overline{1}\rangle = -1$. Therefore (2.18) is not satisfied by any $s_{j,j'}$ independent of the encoded state. Put differently, Z_1Z_3 is a logical Z operator for the information encoded in the four-qubit code. In contrast, for the five-qubit code, a tedious calculation reveals that the Knill-Laflamme conditions are satisfied with $s_{j,j'} \propto \delta_{j,j'}$, where the j index all 16 possible single-qubit or no-qubit Pauli errors.

Not all errors are Pauli errors, though, and indeed the four-qubit code can correct single-qubit erasure errors. By the discussion in Section 2.1.3, the Kraus operators of erasure have the form $|\mathrm{leak}\rangle \otimes \sigma_k$ for some Pauli operator σ_k. Therefore the quantity $M_{j'} \dagger M_j$ will be zero whenever the error M_j concerns erasure of some qubit and $M_{j'}$ does not, i.e. when $M_{j'}$ describes no error, or erasure of a different qubit. Thus it is sufficient to check the Knill-Laflamme conditions for the case that $M_{j'}^\dagger M_j$ is at most a Pauli operator of weight *one* (as opposed to weight up to two for the case of single-qubit Pauli errors which have no side information indicating their location). Then it is not difficult to see that the conditions are satisfied in this case.

2.3.3 Properties of the KL conditions

Before examining in more detail why the Knill-Laflamme conditions are necessary and sufficient for the existence of an exact decoder, a few observations are in order.

Correctability is a property of the span of error operators Given a set of correctable errors $\{M_j\}_{j=1}^r$, every other set formed by arbitrary linear combinations of the M_j is also correctable. That is, the operators $K_\ell = \sum_{j=1}^r b_{\ell j}M_j$ for arbitrary coefficients $b_{\ell j} \in \mathbb{C}$ with $\ell \in \{1,\dots,r'\}$ are correctable if the M_j are. This follows directly from (2.18):

$$\langle\widehat{k}'|K_{\ell'}^\dagger K_\ell|\widehat{k}\rangle = \sum_{j,j'=1}^r b_{\ell'j'}^* b_{\ell j}\langle\widehat{k}'|M_{j'}^\dagger M_j|\widehat{k}\rangle \propto \sum_{j,j'=1}^r b_{\ell'j'}^* s_{j'j} b_{\ell j}\delta_{kk'} = s'_{\ell'\ell}\delta_{kk'}\,, \tag{2.19}$$

where we have implicitly defined $s'_{\ell'\ell}$ in the last equality. The matrix s' is positive if s is, as $s' = b^\dagger s b$. Therefore (2.18) holds for the K_ℓ as well.

Further equivalent forms There are several further equivalent forms of the KL conditions which are often more immediately useful than (2.17) or (2.18). To state them, let P_A be the projector onto the codespace. Then the further conditions below are equivalent to (2.17) and (2.18):

- There exists a positive semidefinite matrix with components $s_{j,j'}$ such that

$$P_A M_{B|A}(j)^\dagger M_{B|A}(j) P_A = s_{j,j'} P\,, \tag{2.20}$$

- There exists a positive semidefinite matrix with components $s_{j,j'}$ such that for all $|\widehat{\psi}\rangle, |\widehat{\psi}'\rangle \in \mathcal{C}$

$${}_A\langle\widehat{\psi}|M_{B|A}(j)^\dagger M_{B|A}(j)|\widehat{\psi}'\rangle_A = s_{j,j'}\langle\widehat{\psi}|\widehat{\psi}'\rangle\,, \tag{2.21}$$

- There exists a positive semidefinite matrix with components $s_{j,j'}$ such that for all $|\widehat{\psi}\rangle \in \mathcal{C}$

$${}_A\langle\widehat{\psi}|M_{B|A}(j)^\dagger M_{B|A}(j)|\widehat{\psi}\rangle_A = s_{j,j'}\langle\widehat{\psi}|\widehat{\psi}\rangle\,. \tag{2.22}$$

The condition (2.20) follows from (2.18) since

$$PM_{j'}^\dagger M_j P = \sum_{k,k'=1}^{d} |\widehat{k}\rangle\langle\widehat{k}|M_{j'}^\dagger M_j|\widehat{k}'\rangle\langle\widehat{k}'| = a s_{j,j'} \sum_k |\widehat{k}\rangle\langle\widehat{k}| = (a s_{j,j'})P\,. \tag{2.23}$$

Then (2.20) implies (2.21) by multiplying (2.20) on the left by $\langle\widehat{\psi}|$ and the right by $|\widehat{\psi}'\rangle$. This immediately implies (2.22). Finally, (2.22) implies (2.18). The $k = k'$ case is immediate; the $k \neq k'$ case can be obtained by considering $|\widehat{\psi}\rangle = |\widehat{k}\rangle + |\widehat{k}'\rangle$ and $|\widehat{\psi}\rangle = |\widehat{k}\rangle + i|\widehat{k}'\rangle$ for all orthonormal $|\widehat{k}\rangle$ and $|\widehat{k}'\rangle$ in $\mathcal{C}$.

Relaxed positivity requirement Often the KL conditions are stated without the positivity requirement on $s_{j,j'}$; instead any $s_{j,j'} \in \mathbb{C}$ are allowed. This is possible since the lefthand side of the KL conditions (in whatever form) imply that that $s_{j,j'}$ has to be Hermitian, and can therefore be diagonalized as in the linear span argument. Then, setting $|\widehat{\psi}'\rangle = |\widehat{\psi}\rangle$ in (2.21), it is apparent that the eigenvalues of $s_{j,j'}$ have to be positive. Hence it is no loss of generality to simply require positivity up front.

2.3.4 Necessity of the KL conditions

Now we turn to the proof of Proposition 2.1. We begin with the necessity of the KL condition.

Noiseless quantum channels do not leak information A crucially important fact about quantum channels is that noiseless channels cannot leak any information about the input to the environment, i.e. the output in the environment must be completely uncorrelated with the input. This is completely different from the case of noiseless classical channels, which could just as well copy the input to the environment. For instance, a copy machine which keeps the copy is a noiseless classical channel, since the input is returned unchanged to the user.

More formally, we have the following statement.

Proposition 2.2: Noiseless quantum channels do not leak information

Suppose $\mathcal{N}_A$ is a quantum channel stemming from a unitary interaction U_{AE} with the environment system E. If $\mathcal{N}_A[|\psi\rangle\langle\psi|_A] = |\psi\rangle\langle\psi|_A$ for all $|\psi\rangle_A$, then there exists a state $|\xi\rangle_E$ such that for all $|\psi\rangle_A$, $U_{AE}(|\psi\rangle_A \otimes |0\rangle_E) = |\psi\rangle_A \otimes |\xi\rangle_E$.

This can be regarded as a statement that, in quantum mechanics, there is "no information (gain) without disturbance". If the interaction U should deposit some information about the input state $|\psi\rangle_A$ in the environment system E, then it must change the state of A in some way.

The proof is simple. By assumption, the marginal state $\mathrm{Tr}_E[|\Psi\rangle\langle\Psi|_{AE}]$ of $|\Psi\rangle_{AE} = U_{AE}(|\psi\rangle_A \otimes |0\rangle_E)$ satisfies $\mathrm{Tr}_E[|\Psi\rangle\langle\Psi|_{AE}] = |\psi\rangle\langle\psi|_A$. Consider purifications of these two states. In particular, $|\Psi\rangle_{AE}$ is a purification of the former, while $|\psi\rangle_A \otimes |0\rangle_E$ is a purification of the latter. Since all purifications of a given state having the same purifying system, here E, are related by unitary operators on that system, it follows that there exists some unitary U_E such that $|\Psi\rangle_{AE} = |\psi\rangle_A \otimes U_E|0\rangle_E$. Set $|\xi\rangle_E = U_E|0\rangle_E$. Since $|\Psi\rangle_{AE} = |\psi\rangle_A \otimes |\xi\rangle_E$ holds for all $|\psi\rangle_A$, it must be that $|\xi\rangle$ is independent of $|\psi\rangle$. Otherwise the lefthand side would depend linearly on $|\psi\rangle$ while the righthand side would depend quadratically on $|\psi\rangle$. More concretely, suppose $|\psi_0\rangle$ leads to $|\xi_0\rangle$ while $|\psi_1\rangle$ leads to $|\xi_1\rangle$. Then the action of U_{AE} on $|\psi_0\rangle + |\psi_1\rangle$ produces $|\psi_0\rangle|\xi_0\rangle + |\psi_1\rangle|\xi_1\rangle$ by superposition of these two cases. But by assumption U_{AE} produces $(|\psi_0\rangle + |\psi_1\rangle) \otimes |\xi'\rangle$ for some $|\xi'\rangle$. This can only hold if $|\xi'\rangle = |\xi_0\rangle = |\xi_1\rangle$.

For application to error correction, we need a slight generalization of the statement, basically dropping the unitary requirement. Given a set $\{M_A(j)\}$ of error operators on system A, define the mapping $V_{AE|A} : |\psi\rangle_A \mapsto \sum_j M_A(j)|\psi\rangle_A \otimes |j\rangle_E$. A similar argument to that above implies the following. If $\sum_j M_A(j)|\psi\rangle\langle\psi|_A M_A(j)^\dagger \propto |\psi\rangle\langle\psi|_A$ for all $|\psi\rangle_A$, then there exists a state $|\xi\rangle_E$ such that $V_{AE|A}|\psi\rangle_A \propto |\psi\rangle_A \otimes |\xi\rangle_E$ for all $|\psi\rangle_A$.

Necessity argument We can apply the above considerations to exact correction, and for this purpose (2.9) is particularly convenient. Assume that exact correction is possible and so $\mathcal{D}\circ\mathcal{N}\circ\mathcal{E} \propto \mathcal{I}$. Consider the state resulting from applying $\mathcal{D}\circ\mathcal{N}\circ\mathcal{E}$ to some state $|\psi\rangle$, where we keep track of the environment E and the ancillary degrees of freedom involved in the decoding operation, which has Kraus operators $D_{Q|B}(k)$:

$$|\Psi\rangle_{QFE} = \sum_{jk} D_{Q|B}(k) M_{B|A}(j) V_{Q|A} |\psi\rangle_Q |k\rangle_F |j\rangle_E \,. \tag{2.24}$$

By the non-normalized version of Proposition 2.2, since the Q part of the state is returned to $|\psi\rangle_Q$, it must be that there exists a state $|\sigma\rangle_{EF}$ such that $|\Psi\rangle_{QFE} \propto |\psi\rangle_Q \otimes |\sigma\rangle_{EF}$. Now compute the marginal state of E:

$$\sigma_E \propto \mathrm{Tr}_{QF}[\Psi_{QFE}] = \sum_{jj'k} \langle\widehat{\psi}|M_{B|A}(j')^\dagger D_{Q|B}(k)^\dagger D_{Q|B}(k) M_{B|A}(j)|\widehat{\psi}\rangle_A |j\rangle\langle j'|_E \tag{2.25a}$$

$$= \sum_{jj'} \langle\widehat{\psi}|M_{B|A}(j')^\dagger M_{B|A}(j)|\widehat{\psi}\rangle_A |j\rangle\langle j'|_E \,. \tag{2.25b}$$

This holds for all $|\widehat{\psi}\rangle$, and taking matrix elements gives (2.22) with $s_{jj'} = \langle j|\sigma|j'\rangle$. We have established the 'only if' part of Proposition 2.1.

2.3.5 Sufficiency of the KL conditions

Sufficiency of the KL conditions can be established immediately from (2.17). Consider purifications of the states $\Psi_{Q'E}$ and $a\pi_{Q'}\otimes\sigma_E$. For the former we can simply use $|\Psi\rangle_{Q'BE}$ from (2.16). For the latter may define $|\sigma\rangle_{EE'}$ to be a purification of σ_E, with $|E'|$ equal to the rank of σ_E, and take $\sqrt{a}|\Phi\rangle_{QQ'}\otimes|\sigma\rangle_{E'E}$. Any two purifications of the same state, here $\Psi_{Q'E}$, are related by a partial isometry from the purifying systems of one purification to the purifying systems of the other. (A partial isometry is an operator which is an isometry on its support; it may annihilate some part of the input space.) That is, there exists a partial isometry $W_{B|QE'}$ such that $|\Psi\rangle_{Q'BE} = \sqrt{a}\,W_{B|QE'}|\Phi\rangle_{QQ'}|\sigma\rangle_{E'E}$. If the dimension of B is larger than that of QE', then $W_{B|QE'}$ can be taken to be an isometry, not just a partial isometry. This will always be the case here. Since the overall state is pure, the rank of Ψ_B is the same as that of $\Psi_{Q'E} = a\pi_{Q'}\otimes\sigma_E$. This implies $|B|\geq \mathrm{rank}(\Psi_B) = |Q'||E'| = |Q||E'|$.

Then the decoder $\mathcal{D}_{Q|B}$ is implemented by first applying $W^\dagger$, which is a partial isometry from B to QE', followed by tracing out of E'. The first step of the decoder gives

$$W^\dagger|\Psi\rangle_{Q'BE} = \sqrt{a}\,W^\dagger W|\Phi\rangle_{QQ'}|\sigma\rangle_{E'E} = \sqrt{a}\,|\Phi\rangle_{QQ'}|\sigma\rangle_{E'E}\,. \tag{2.26}$$

Therefore $\mathcal{D}_{Q|B}\circ\mathcal{N}_{B|A}\circ\mathcal{E}_{A|Q}[\Phi_{QQ'}] = a\Phi_{QQ'}$, and the condition for exact correction in (2.10) is satisfied. This establishes the 'if' statement of Proposition 2.1.

In fact, the decoder just constructed is isometric. The Kraus operators of the decoder are $D_{Q|B}(j) = {}_{E'}\langle j|W^\dagger_{B|QE'}$, whose adjoints are the isometries $V_{B|Q}(j) = W_{B|QE'}|j\rangle_{E'}$. Their images are distinct, since $V_{B|Q}(k)^\dagger V_{B|Q}(j) = {}_{E'}\langle k|W^\dagger_{B|QE'}W_{B|QE'}|j\rangle_{E'} = \langle k|j\rangle = \delta_{jk}$. Note that there is in fact an entire family of isometric decoders corresponding to the choice of basis in system E'.

The isometric property of the decoder also implies that the decoder so constructed for some set of errors $\{M_j\}$ also corrects any other set of errors K_ℓ formed by linear combination of the M_j. Previously we established that if one set is correctable, then the other is as well. However, that left open the question of whether different decoders would be needed for the different sets of errors. Note that this is not an issue when thinking about correcting channels: If two sets of Kraus operators represent the same channel, then a decoder for one set necessarily works for the other. For errors, though, the K_ℓ will generally give rise to a different improperly-normalized channel than the M_j. Return to (2.16) and notice that the analogous state for the errors K_ℓ is just $|\Psi'\rangle_{Q'BE} = L_E|\Psi\rangle_{Q'BE}$, where $L = \sum_{\ell,j} b_{\ell j}|\ell\rangle\langle j|$. But then we have $|\Psi'\rangle_{Q'BE} = \sqrt{a}\,W_{B|QE'}|\Phi\rangle_{QQ'}|\sigma'\rangle_{EE'}$ for $|\sigma'\rangle_{EE'} = L_E|\sigma\rangle_{EE'}$, and so the decoder based on $W_{B|QE'}$ will correct any set of errors formed from the span of the M_j.

2.3.6 Degeneracy for correctable errors

Error degeneracy is present in a set of correctable errors whenever σ_E in (2.17) does not have full rank. That is, σ_E has a zero eigenvalue for some set $\{M_j\}$ of errors if and only if it is possible to find two errors K_1' and K_2' which are linear combinations of the M_j and which act degenerately on the code space. Here is the proof.

- $\Rightarrow$ Suppose $\{M_\ell\}$ is a set of errors such that $s_{jj'}$ has a zero eigenvalue. Moving to the errors K_j which diagonlize $s_{jj'}$, there is some j such that $K_j = \sum_\ell b_{j,\ell}M_\ell$ satisfies $PK_j^\dagger K_jP = 0$. This implies that K_j annihilates all codewords $|\widehat{\psi}\rangle\in\mathcal{C}$. Now define $K_1' = b_{j,\ell'}M_{\ell'}$ for the first ℓ' such that $b_{j,\ell'}\neq 0$ and $K_2' = -\sum_{\ell>\ell'} b_{j,\ell}M_\ell$. Then $K_j = K_1' - K_2'$. Hence $K_1'|\widehat{\psi}\rangle = K_2'|\widehat{\psi}\rangle$ for all $|\widehat{\psi}\rangle\in\mathcal{C}$.

$\Leftarrow$ Suppose K_1' and K_2' are linear combinations of the M_ℓ which act degenerately on $\mathcal{C}$. Then $K_1' - K_2'$ is a linear combination which annihilates $\mathcal{C}$. Say it has coefficients b_ℓ'. Then the vector corresponding to b_ℓ' is an eigenvector of $s_{\ell,\ell'}$ with eigenvalue 0.

Evidently, the single-qubit and no-qubit errors are not degenerate on the five-qubit code described above. Note that five qubits is precisely the minimal number of qubits needed for a code to correct such errors when they act nondegenerately. There are 16 possible error actions, and since their action in this case maps the code subspace to disjoint subspaces, each of which has dimention 2, there need to be at least 32 dimensions in total to contain all of the subspaces. This is precisely the dimension of the state space of five qubits.

2.4 Approximate error correction

2.4.1 Example: Amplitude damping and the four-qubit code

An example of approximate error correction is furnished by the four-qubit code subject to amplitude damping noise. Unlike the simpler example mentioned in Section 2.2.1, here no Kraus operator is exactly corrected at all. Instead, each Kraus operator is corrected up to order γ, the damping parameter.

The amplitude damping channel has but two Kraus operators, given in (2.5), meaning the i.i.d. channel on four qubits has 16 Kraus operators, each of the form $K_{x_1x_2x_3x_4} = M_{x_1} \otimes M_{x_2} \otimes M_{x_3} \otimes M_{x_4}$ for $x_j \in \{0,1\}$. Note that the Kraus operators are all of weight four, since no factor is trivial. However, most of the operators only contribute at high orders in γ, and the code does not attempt to correct these at all. These high-order Kraus operators are the equivalent of the uncorrected Pauli errors in a Pauli channel. To see which Kraus operators one should attempt to at least approximately correct, consider M_1, which describes the damping event. It occurs with probability γ, and, clearly, M_1 is linear in $\sqrt{\gamma}$. On the other hand, $M_0 = \mathbb{1} - \frac{1}{2}\gamma|1\rangle\langle 1| + O(\gamma^2) = \mathbb{1} - O(\sqrt{\gamma})$.

Therefore, it is sensible to focus on Kraus operators with only one damping event and furthermore to consider an approximation of these errors in which M_0 is replaced by $\mathbb{1}$. Then, to first order in $\sqrt{\gamma}$, the action of the channel is described by the five Kraus operators $K_0' = \mathbb{1}\mathbb{1}\mathbb{1}\mathbb{1}$, $K_1' = \mathbb{1}\mathbb{1}\mathbb{1}M_1$, $K_2' = \mathbb{1}\mathbb{1}M_1\mathbb{1}$, $K_4' = \mathbb{1}M_1\mathbb{1}\mathbb{1}$, and $K_8' = M_1\mathbb{1}\mathbb{1}\mathbb{1}$. The labelling corresponds to the binary strings $x_1x_2x_3x_4$. Direct calculation shows that these five error operators satisfy the Knill-Laflamme conditions for the four-qubit code. In particular, the $s_{j,j'}$ is diagonal, with $s_{0,0} = 1$ and $s_{j,j} = \gamma/2$ for $j \in \{1,2,3,4\}$. Therefore, there exists a decoder which exactly corrects these errors. Note that the decoder constructed in this way is only defined on the subspace of the four-qubit state space which can be reached by action of the K_j'. To extend the decoder to the entire space, we can first measure if the qubits are in the subspace, apply the exact decoder if so, and simply output a fixed state, say $|0000\rangle_B$, if not.

The correctability of the K' error operators implies that the corresponding decoder will approximately correct the actual output of the amplitude damping channel to first order in γ. To see why, define the joint state

$$|\Psi'\rangle_{Q'BE} = \tfrac{2-\gamma}{2\sqrt{1-\gamma}}\beta V_{B|Q}|\Phi\rangle_{QQ'}|0\rangle_E + \beta\sum_{j=0}^{3} K_B'(2^j)V_{B|Q}|\Phi\rangle_{QQ'}|2^j\rangle_E\,, \tag{2.27}$$

for $\beta = \frac{2\sqrt{\gamma^2-3\gamma+2}}{\sqrt{7\gamma^3-18\gamma^2+4\gamma+8}}$. Here the label j in the orthonormal states $|j\rangle_E$ ranges from 0 to 15, and corresponds to the 4-bit binary strings x used in the discussion of K_x above. This state is close to the actual state $|\Psi\rangle_{Q'BE}$ from (2.16) encountered in the error-correction procedure. By direct calculation it can be verified that $|\langle\Psi|\Psi'\rangle|^2 = 1 - \frac{5}{2}\gamma^2 + O(\gamma^3)$. (Indeed, the state was constructed to maximize this fidelity.) The decoder for the K'_x, applied to $|\Psi'\rangle$ will result in exact recovery of $|\Phi\rangle_{QQ'}$ since each term will be exactly corrected. Because the fidelity will not decrease under the decoding operation, the action of the amplitude damping channel will be corrected to first order in γ, and therefore (2.8) is satisfied with $\varepsilon = O(\gamma^2)$.

2.4.2 Approximate Knill-Laflamme conditions

The Knill-Laflamme conditions enable us to decide if exact decoding is possible without having to explicitly construct a decoder. One may then wonder if something similar is possible for approximate correction. Fortunately, the answer is yes.

Proposition 2.3

For a code $\mathcal{C}$ and noisy channel $\mathcal{N}_{B|A}$ the state $\Psi_{Q'E}$ from (2.16) satisfies $\Psi_{Q'E} \approx_\varepsilon \pi_{Q'} \otimes \sigma_E$ for some σ_E in the sense that the fidelity squared satisfies $F(\Psi_{Q'E}, \pi_{Q'} \otimes \sigma_E)^2 \geq 1 - \varepsilon$ if and only if there exists a decoder such that (2.8) is satisfied.

Proof. The proof is very similar to the case of exact correction and additionally involves using Uhlmann's theorem. Specifically, suppose that $F(\Psi_{Q'E}, \pi_{Q'} \otimes \sigma_E)^2 \geq 1 - \varepsilon$. Then, by Uhlmann's theorem there exists a partial isometry $W_{B|QE'}$ such that $|_{Q'BE}\langle\Psi'|W_{B|QE'}|\Phi\rangle_{QQ'}|\sigma\rangle_{EE'}|^2 \geq 1 - \varepsilon$, where $|\sigma\rangle_{EE'}$ is a purification of σ_E. Tracing out E gives the decoder as before, and preserves the fidelity. This establishes the 'only if' part of Proposition 2.3.

For the other direction, suppose a decoder $\mathcal{D}_{Q'|B}$ exists such that $F(\Phi_{QQ'}, \mathcal{D}_{Q|B}[\Psi_{Q'B}])^2 \geq 1 - \varepsilon$. Consider the purifications $|\Phi\rangle_{QQ'}|\sigma\rangle_{EE'}$ and $V'_{QE'|B}|\Psi\rangle_{Q'BE}$, for V' the isometry which implements the decoding operation and involves the ancillary system E' and $|\sigma\rangle_{EE'}$ an arbitrary bipartite pure state. Proceeding via Uhlmann's theorem just as above implies that there exists a unitary on EE' such that $|_{Q'BE}\langle\Psi|V^\dagger_{QE'|B}U_{EE'}|\Phi\rangle_{QQ'}|\sigma\rangle_{EE'}|^2 \geq 1 - \varepsilon$. Absorbing U_{EE} into $|\sigma\rangle_{EE'}$ and tracing out QE' implies $F(\Phi_{Q'E}, \pi_{Q'} \otimes \sigma_E)^2 \geq 1 - \varepsilon$. This establishes the 'if' part of Proposition 2.3. $\square$

Unlike the case of exact correction, for approximate correction the decoder might not be isometric. As we saw there, isometric decoding relies on $|B| \geq |Q||E'|$. However, this inequality is violated for the optimal decoder in the example of the four qubit code and amplitude damping noise. The optimal σ_E in the squared fidelity can be computed by means of semidefinite programming, and appears to have rank 11. Since $|Q| = 2$ and $|B| = 16$, the inequality is violated. Indeed, the optimal decoder can also be directly constructed by semidefinite programming, and does not appear to have the isometric structure. It is still an open question whether for any decoder of an approximate code there always exists an isometric decoder with similar performance. For the four-qubit code example, the optimal squared fidelity appears to be $1 - \frac{5}{4}\gamma^2 + O(\gamma^3)$ from numerical calculation. In this case the performance of the optimal decoder with approximation parameter ε is nearly replicated by the isometric decoder, which has parameter $\approx 2\epsilon$.

3 Stabilizer codes and the Pauli group

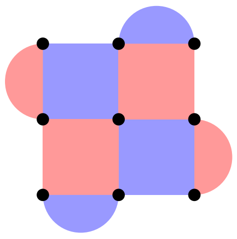
We are mostly interested in noise which is described by Pauli operators and stabilizer codes whose generators are Pauli operators. In particular, in this chapter we define the Pauli group and stabilizer codes, see how to determine the size and logical operators of a stabilizer code given its stabilizers, and finally how to decompose a general Pauli error into a stabilizer contribution, a logical operator contribution, and a contribution from the “destabilizers”.

3.1 Motivation: Surface code

Besides the Shor code, another stabilizer code is the *surface code*, one of the leading codes for experimental implementation of quantum error correction. The surface code is a family of codes, one for each size (m_1, m_2). The stabilizers of the 3×3 surface code are, in the same format as (1.10) and (1.11) for the Shor code,

$$\begin{array}{ccc} Z & Z & \cdot \\ Z & Z & \cdot \\ \cdot & \cdot & \cdot \end{array}, \quad \begin{array}{ccc} \cdot & Z & Z \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{array}, \quad \begin{array}{ccc} \cdot & \cdot & \cdot \\ Z & Z & \cdot \\ \cdot & \cdot & \cdot \end{array}, \quad \begin{array}{ccc} \cdot & \cdot & \cdot \\ \cdot & Z & Z \\ \cdot & Z & Z \end{array},$$
$$\begin{array}{ccc} X & \cdot & \cdot \\ X & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{array}, \quad \begin{array}{ccc} \cdot & X & X \\ \cdot & X & X \\ \cdot & \cdot & \cdot \end{array}, \quad \begin{array}{ccc} \cdot & \cdot & \cdot \\ X & X & \cdot \\ X & X & \cdot \end{array}, \quad \begin{array}{ccc} \cdot & \cdot & \cdot \\ \cdot & \cdot & X \\ \cdot & \cdot & X \end{array}. \tag{3.1}$$

It is simpler to represent the stabilizers by “plaquettes” in a 3×3 lattice, as shown below.



Here the black dots represent qubits, the blue “plaquettes” represent Z-type stabilizers which involve the qubits on the corners of the plaquette, and the red plaquettes represent X-type stabilizers. Again there are a total of eight stabilizers, just like the Shor code, though this time there are an equal number of X-type and Z-type stabilizers. Observe that the left and right boundaries are red, in the sense that looking from left and right one sees red plaquettes. Similarly, the top and bottom boundaries are blue. By alternating plaquettes in this checkerboard pattern and satisfying the red/blue boundary constraint, one can define the surface code for an arbitrary $m_1 \times m_2$ lattice of qubits.

How many qubits does the 3×3 surface code encode? What are its logical operators? How would we encode information into the code? How do we decode it? To answer these questions for the surface code and codes in general, we look now at the structure of the Pauli group in more detail.

3.2 Definition and basic properties

3.2.1 Single-qubit Pauli operators

Just for concreteness, we start with the definitions of the single-qubit Pauli operators in terms of a fixed basis with elements $|0\rangle$ and $|1\rangle$: For $k \in \mathbb{F}_2$ we have

$$\begin{array}{ll} \sigma_0 = \mathbb{1}|k\rangle = |k\rangle & \sigma_1 = \sigma_x|k\rangle = |k+1\rangle \\ \sigma_3 = \sigma_z|k\rangle = (-1)^k|k\rangle & \sigma_2 = \sigma_y|k\rangle = i\sigma_x\sigma_z|k\rangle \end{array} . \tag{3.2}$$

The matrix representatives of these operators, when acting to the right on the vectors $|0\rangle \simeq \begin{pmatrix}1\\0\end{pmatrix}$ and $|1\rangle \simeq \begin{pmatrix}0\\1\end{pmatrix}$ are therefore

$$\mathbb{1} = \begin{pmatrix}1 & 0\\0 & 1\end{pmatrix}, \qquad \sigma_x = \begin{pmatrix}0 & 1\\1 & 0\end{pmatrix}, \qquad \sigma_y = \begin{pmatrix}0 & -i\\i & 0\end{pmatrix}, \qquad \sigma_z = \begin{pmatrix}1 & 0\\0 & -1\end{pmatrix}. \tag{3.3}$$

We will usually denote these operators by I, X, Y, Z, respectively.

The single qubit Pauli operators have the property that the product of any two is just another, up to some phase factor ± 1 or $\pm i$. That is, for every $j, k \in \mathbb{Z}_4$, $\sigma_j\sigma_k = i^c\sigma_\ell$ for some $c, \ell \in \mathbb{Z}_4$. The following multiplication table for the Pauli operators gives all the particular phases; each entry is the product of the corresponding Pauli operator listed in the row with the corresponding Pauli operator listed in the column.

	I	X	Y	Z
I	I	X	Y	Z
X	X	I	iZ	$-iY$
Y	Y	$-IZ$	I	iX
Z	Z	iY	$-iX$	I

. (3.4)

3.2.2 Definition of the Pauli group

A Pauli operator on n qubits is an operator of the form

$$i^b \sigma_{j_1} \otimes \sigma_{j_2} \otimes \cdots \otimes \sigma_{j_n} , \tag{3.5}$$

where b and all the j_k for $k = 1, \dots, n$ are elements of $\mathbb{Z}_4$. Call this set of operators P_n.

3.2.3 Properties of the Pauli group

Anticommutation Since the single-qubit Pauli operators either commute or anticommute, the elements of P_n either commute or anticommute depending on the parity of the number of anticommuting factors. That is, if two Pauli operators g and g' are such that ℓ of their factors anticommute and the remaining $n-\ell$ commute, then g and g' commute when ℓ is even and anticommute when ℓ is odd.

Group structure By (3.4), the product of any two elements of P_n is also in P_n. The identity operator is clearly an element of P_n: Choose $b = 0$ and $j_k = 0$ for all $k \in 1, \dots, n$ in (3.5). Moreover, for any $g \in P_n$ with particular b and j_k, the operator $g' \in P_n$ with identical $j'_k = j_k$ but $b' = 3b$ is the inverse of g. Hence P_n forms a finite group.

Center The center of P_n (the subgroup of elements of P_n which commute with everything in P_n) is just the four operators $i^b\mathbb{1}$ for $b \in \mathbb{Z}_4$. Denote this $Z(P_n)$.

Hermitian subset Evidently, a Pauli operator is Hermitian when $b \in \{0,2\}$ in (3.5), so that the prefactor to the Pauli tensor product is real-valued. Call $P_n^{\pm}$ the subset of Hermitian operators in P_n; note that it is not a group since $XZ \notin P_n^{\pm}$. The elements of $P_n^{\pm}$ are also unitary, since their eigenvalues are ± 1. Hence $g^2 = \mathbb{1}$ for all $g \in P_n^{\pm}$.

It will also be convenient to define P_n^{+} as the set of all Pauli operators with $b = 0$ in (3.5).

The non-Hermitian subset of P_n is just $iP_n^{\pm}$, that is, the elements of $P_n^{\pm}$ multiplied by i. So $P_n = P_n^{\pm} \cup iP_n^{\pm}$.

Real Pauli group The collection of $g \in P_n$ such that the matrix representation using (3.3) has only real-valued entries forms the real Pauli group. Denote it by P_n^r. It is a group since the product of two real-valued matrices is necessarily itself real-valued. We will not make that much use of it, though, because the Hermitian properties of $P_n^{\pm}$ are more useful than the group properties of P_n^r.

3.2.4 Representation of the Pauli group over $\mathbb{F}_2^{2n+2}$

Since $Y = iXZ$, all elements of P_n can be uniquely written as

$$i^c(-1)^t X^u Z^v, \tag{3.6}$$

for $c, t \in \mathbb{F}_2$ and $u, v \in \mathbb{F}_2^n$. Here X^u denotes $X^{u_1} \otimes X^{u_2} \otimes \cdots \otimes X^{u_n}$ with $X^0 = I$ and $X^1 = X$, and similarly for Z^v. Each $g \in P_n$ is specified by a vector $w = (u, v) \in \mathbb{F}_2^{2n}$ and a phase $(t, c) \in \mathbb{F}_2^2$.

For instance, consider the example $X_1 Y_2$. This satisfies $X_1 Y_2 = (X \otimes I)(I \otimes Y) = (X \otimes I)(I \otimes iXZ) = i(X \otimes X)(I \otimes Z)$. Therefore $(t, c) = (0, 1)$, $u = (1, 1)$ and $v = (0, 1)$; $w = (1, 1, 0, 1)$.

The real Pauli group P_n^r corresponds to $c = 0$ and t unconstrained. Elements of the $P_n^{\pm}$ satisfy $c = u \cdot v$, where the dot product is computed modulo 2. For elements of P_n^{+} things are slightly more complicated. Let $|uv|$ be the number of 1s in the pointwise product of u and v (i.e. $u \cdot v$ computed modulo 2 as above is just $|uv| \mod 2$). Then $t = 0$ if $|uv| \mod 4$ is 0 or 1, and $t = 1$ otherwise.

The product of two Pauli operators has a nice representation in these terms as well. Since X anticommutes with Z,

$$X^u Z^v = (-1)^{u \cdot v} Z^v X^u, \tag{3.7}$$

which is to say that when reversing the order of X^u and Z^v there is one factor of (-1) for every qubit $j \in 1, \ldots n$ such that $u_j = v_j = 1$. Therefore we have

$$(i^c(-1)^t X^u Z^v)(i^{c'}(-1)^{t'} X^{u'} Z^{v'}) = i^{c+c'}(-1)^{t+t'}(-1)^{u' \cdot v} X^{u+u'} Z^{v+v'}. \tag{3.8}$$

Thus multiplication in P_n can be expressed as addition in $\mathbb{F}_2^{2n}$ along with some adjustment to the phase. In particular, for $g \in P_n$ represented by (t, c, w) and $g' \in P_n$ represented by (t', c', w'), the product gg' is represented by

$$(t, c, w) * (t, c', w') := (t + t' + u' \cdot v + cc', c + c', w + w'), \tag{3.9}$$

where addition in each entry is modulo 2. If we ignore the phase, meaning we work with the Pauli group modulo its center, $P_n/Z(P_n)$, then multiplication is just addition in $\mathbb{F}_2^{2n}$.

3.3 (Pauli) Stabilizer groups and codes

Now we turn to stabilizer codes based on the Pauli group. We give the definition, then determine the size of the stabilizer code for a given stabilizer group, and finally define the logical operators of the code.

3.3.1 Definition and basic properties

Stabilizers and their associated codes Consider a subset $G \subseteq P_n$ of Pauli operators and define $\mathcal{C}$ to be the common $+1$ eigenspace of all the g_j. Thus the g_j stabilize $\mathcal{C}$ in the sense that $g_j|\psi\rangle = |\psi\rangle$ for all $g_j \in G$ and $|\psi\rangle \in \mathcal{C}$. $\mathcal{C}$ is the stabilizer code defined by G.

Properties Note that for the trivial case $\mathcal{C} = 0$, G can be composed of arbitrary operators. However, assuming that $\mathcal{C} \neq 0$, the following properties hold:

1. By construction, the g_j have a $+1$ eigenvalue, and hence $G \subseteq P_n^{\pm}$.

2. Each element of G is its own inverse (immediately implied by $G \subseteq P_n^{\pm}$).

3. All elements of G commute. For suppose there exist $g, g' \in G$ such that $[g, g'] \neq 0$. Since gg' is also a stabilizer of $\mathcal{C}$, $gg' \in P_n^{\pm}$ and hence $(gg')^2 = \mathbb{1}$. On the other hand, $(gg')^2 = gg'gg' = -g^2(g')^2 = -\mathbb{1}$, a contradiction.

Stabilizer group As we just saw, gg' stabilizes $\mathcal{C}$ if g and g' do. This implies that G generates a group, since every element will be its own inverse. We denote the resulting *stabilizer group* by S; its elements are *stabilizers*, and the elements of G are the *stabilizer generators*. By the above properties, the interesting groups S (with nontrivial $\mathcal{C}$) are those that are consist of commuting Pauli operators drawn from $P_n^{\pm}$. There is one particular element of $P_n^{\pm}$ which cannot be part of any nontrivial stabilizer group S, namely $-I$.

Independent generators Suppose that S is a stabilizer group with generators G. It is most useful to consider the case of independent generators, i.e. when no element of G is the product of any collection of the remainder. Independence of a given set G of elements of $P_n^{\pm}$ is equivalent to the statement that the product of any subset of Gs elements is unequal to $\mathbb{1}$; if the product is $\mathbb{1}$, then any one of them is the product of the remainder.

In terms of the vector representation, independence of a set $G = \{g_j \in P_n^{\pm}\}_{j=1}^m$ is equivalent to linear independence of the associated vectors $\{w_j \in \mathbb{F}_2^{2n}\}$. Suppose the elements of G are dependent, meaning $\prod_{j=1}^m g_j^{a_j} = \mathbb{1}$ for some $a_j \in \mathbb{F}_2$. Then the associated vector representatives satisfy $\sum_{j=1}^m a_j w_j = 0$, meaning the w_j are dependent. Conversely, suppose that $\sum_{j=1}^m a_j w_j = 0$ holds for some $a_j \in \mathbb{F}_2$. Then the product of the g_j satisfies $\prod_{j=1}^m g_j = i^b \mathbb{1}$ for some $b \in \mathbb{Z}_4$. But b must equal zero since the product of the g_j is a stabilizer and therefore $b \neq 0$ is ruled out.

Syndrome decoding For a stabilizer code on n qubits with m independent stabilizer generators, the syndrome $s \in \mathbb{F}_2^m$ is the classical data which results from quantum measurement of each of the stabilizer generators. Measurement of an element of $P_n^{\pm}$ has two possible outcomes, and our convention is that the jth bit s_j of s corresponds to the eigenvalue $(-1)^{s_j}$ of the jth stabilizer generator.

The task of a syndrome decoder, which is a type of codespace decoder, is to determine an appropriate correction operator from the syndrome and then apply it to the qubits. For Pauli noise models, it turns out that taking the correction operator to be Pauli operators loses nothing in error-correction performance. Indeed, we may as well choose the correction to be an element of P_n^+ as the phase of the correction operator will not be important. Nonetheless, for more general noise models more general correction operations are possible. Moreover, it may be that syndrome decoding is also no longer appropriate.

3.3.2 Examples

The five qubit code It turns out that five physical qubits are necessary in order to protect a single qubit of information from arbitrary single-qubit errors. Here are the stabilizers of such a code:

$$XZZXI\,, \qquad IXZZX\,, \qquad XIXZZ\,, \qquad ZXIXZ\,. \tag{3.10}$$

Note that the stabilizers are cyclic shifts of each other. The final shifted stabilizer, $ZZXIX$, is not independent; it is the product of all of the listed generators.

The Steane code Independently of Shor, Steane also discovered that quantum error correction is possible. He constructed a seven-qubit code encoding one qubit based on the classical Hamming code of seven bits. Like the Shor code, the stabilizers come in X and Z types:

$$\begin{matrix} X & X & I & X & X & I & I \\ X & I & X & X & I & X & I \\ I & X & X & X & I & I & X \end{matrix} \quad \text{and} \quad \begin{matrix} Z & Z & I & Z & Z & I & I \\ Z & I & Z & Z & I & Z & I \\ I & Z & Z & Z & I & I & Z \end{matrix}\,. \tag{3.11}$$

In this case the two sets of stabilizers are identical, except for $X \longleftrightarrow Z$. Since each stabilizer has weight four and overlaps with any of the others in only two qubit positions, all the stabilizers commute.

The four-qubit code and general Shor codes Instead of concatenating two three-bit repetition codes, we could concatenate an m_1-bit repetition code with an m_2-bit repetition code. For example, concatenating two two-bit repetition codes results in a four-qubit code that encodes one qubit. Its stabilizers are

$$ZZII\,, \qquad IIZZ\,, \qquad XXXX\,. \tag{3.12}$$

The four qubit code cannot recover from arbitrary errors, but can recover from a single erasure.

Types of stabilizer codes There are four major types of stabilizer codes:

1. General stabilizer codes; the stabilizers come from $P_n^{\pm}$.
2. Codes whose stabilizers come from $P_n^{\pm} \cap P_n^r$, i.e. the stabilizers are Hermitian and come from the real Pauli group. For instance, the five-qubit code.
3. Calderbank-Shor-Steane (CSS) codes, which have a generating set in which each stabilizer generator is either of X type or of Z type, e.g. the Shor and Steane codes.
4. Classical linear codes, a special case of CSS codes in which all the stabilizers are of Z type.

3.3.3 Size of $\mathcal{C}$

For a given stabilizer group S, how large is its stabilizer code $\mathcal{C}$, i.e. what is the dimension of $\mathcal{C}$?

Proposition 3.1

For a stabilizer group having m independent generators (and which does not contain $-\mathbb{1}$), the dimension of $\mathcal{C}$ is equal to 2^{n-m}. That is, a stabilizer code with m independent stabilizer generators can encode $k = n - m$ qubits.

Proof. The caveat that S not contain $-\mathbb{1}$ rules out the case $\mathcal{C} = 0$.

For each $g_j \in G$, consider the operator $\pi_j = \frac{1}{2}(\mathbb{1} + g_j)$. Since $g_j^2 = \mathbb{1}$, each π_j is a projector: $\pi_j^2 = \frac{1}{4}(\mathbb{1} + 2g_j + g_j^2) = \pi_j$. And because the g_j all commute, the π_j are a collection of commuting projectors. Their common $+1$ eigensubspace is V, and indeed this common eigensubspace is the $+1$ eigensubspace of their product $\pi = \prod_{j=1}^{m} \pi_j$.

The size of the $+1$ eigenspace of π is simply equal to its trace. Expanding this out, we have

$$\mathrm{Tr}[\pi] = \mathrm{Tr}[\prod_{j=1}^{m} \tfrac{1}{2}(\mathbb{1} + g_j)] = \tfrac{1}{2^m}\mathrm{Tr}[\mathbb{1} + \sum_j g_j + \sum_{j \neq j'} g_j g_j + \sum_{j \neq j' \neq j''} g_j g_{j'} g_{j''} + \cdots + g_1 g_2 \cdots g_m], \tag{3.13}$$

where the $\cdots$ indicate the missing terms, which are sums over products of distinct quadruples of the g_js, then products of five, then six, and so on. The last term, as given, is the product of all the generators. By assumption the generators are independent, and thus all terms but the first are not equal to $\mathbb{1}$. Hence the trace of all terms but the first is zero. This gives $\mathrm{Tr}[\pi] = \frac{1}{2^m}\mathrm{Tr}[\mathbb{1}] = 2^{n-k}$ as claimed. □

A simple way to remember this is that each independent generator halves the size of the common $+1$ eigenspace.

3.3.4 Logical operators

Idea of the logical operators Consider a quantum error-correcting code $\mathcal{C}$ which encodes k qubits. The idea of the logical operators on $\mathcal{C}$ is that they are to the code space what the single-qubit Pauli operators are to the space of k qubits. Their action on $\mathcal{C}$ should be a representation of P_k.

Normalizer of S in P_n One thing the logical operators should do is map $\mathcal{C}$ to itself. In the setting of stabilizer codes, a logical operator $g \in P_n$ should satisfy

$$sg|\psi\rangle = g|\psi\rangle \qquad \forall s \in S, |\psi\rangle \in \mathcal{C}\,. \tag{3.14}$$

Multiplying both sides by g^{-1} yields $g^{-1}sg|\psi\rangle = |\psi\rangle$ for all $s \in S$ and $|\psi\rangle \in \mathcal{C}$. Therefore $g^{-1}sg$ has to be an element of S. The set of such $g \in P_n$ is called the normalizer of S in P_n, denoted by $N(S, P_n)$. Formally,

$$N(S, P_n) := \{g \in P_n : g^{-1}sg \in S \quad \forall s \in S\}\,. \tag{3.15}$$

Since all g and s in the definition commute or anticommute for the Pauli group, $gsg^{-1} = \pm s$. And indeed it must be that $gsg^{-1} = s$, as otherwise gsg^{-1} would not be in S. Hence the normalizer is actually equal to the *centralizer* $C(S, P_n) := \{g \in P_n : gs = sg \quad \forall s \in S\}$.

Defintion of logical operators The logical operators of a stabilizer code with stabilizer group S are the operators in $N(S, P_n)/S$, i.e. all Pauli operators in the normalizer of S modulo elements of S itself. In the following section, we will see that the logical operators as so defined do indeed represent P_k.

Examples Looking at (3.1), we can easily verify that the logical operators of the Shor code given in (1.12) are also logical operators of the 3×3 surface code. That is, the logical Z operator is a product of Z operators on a vertical line of qubits stretching from the bottom to the top, while the logical X operator is a product of X operators on a horizontal line of qubits between the left and right. This holds for surface codes of arbitrary size. The logical operators are indeed independent of the stabilizers: no product of Z-type stabilizers will result in the product of Z along a vertical line.

Similarly, one can confirm that $XXXXX$ and $ZZZZZ$ are logical operators of the five-qubit code, $XXXXXXX$ and $ZZZZZZZ$ are logical operators of the Steane code, and $XXII$ and $ZIZI$ are logical operators of the four-qubit code. Since the logical operators are really only defined up to stabilizers, these choices are not unique; an alternate set of logical operators for the Steane code is for instance $XIIIXXI$ and $ZIIIZZI$.

3.4 Symplectic structure of the Pauli group

The commutation structure of the Pauli group endows the vector representation on $\mathbb{F}_2^{2n}$ with a symplectic structure, making it a symplectic vector space. This structure allows us to easily construct logical operators for a stabilizer code.

3.4.1 Commutation

Symplectic form To understand the nature of the normalizer $N(S, P_n)$, it is useful to first start with the commutation structure of the Pauli group. Consider two elements g, g' of P_n with associated vectors $w = (u, v)$ and $w' = (u', v')$ from §3.2.4. By the (3.8) it follows that

$$[g, g'] = 0 \qquad \Longleftrightarrow \qquad u \cdot v' + u' \cdot v \mod 2 = 0\,. \tag{3.16}$$

This condition defines a symplectic structure on $\mathbb{F}_2^{2n}$. First define

$$J = \begin{pmatrix} 0 & \mathbb{1} \\ \mathbb{1} & 0 \end{pmatrix}. \tag{3.17}$$

Then using J we define the symplectic bilinear form $\langle w, w' \rangle \in \mathbb{F}_2$ of $w, w' \in \mathbb{F}_2^{2n}$ by regarding w and w' as column vectors and setting

$$\langle w, w' \rangle := w^T J w'\,. \tag{3.18}$$

Then (3.16) becomes

$$[g, g'] = 0 \qquad \Longleftrightarrow \qquad \langle w, w' \rangle = 0\,. \tag{3.19}$$

"Symplectic" just means that the bilinear form is alternating ($\langle w, w \rangle = 0$) and non-degenerate ($\langle w, w' \rangle = 0$ for all $w' \in \mathbb{F}_2^{2n}$ implies $w = 0$). (The symplectic form is not an inner product in the usual sense, due to the alternating property: every vector would have "length" zero.)

Parity-check matrices The symplectic form and vector representation gives a simple means to compute the syndrome of a given error for a given code. Suppose that the stabilizer code $\mathcal{C}$ has independent stabilizers $\{g_j\}_{j=1}^m$, with associated vectors $w_j \in \mathbb{F}_2^{2n}$. For a Pauli error g with associated vector w, by (3.19) the syndrome bit $s_j = w_j^T J w$. Collecting the w_j^T into an $m \times 2n$ matrix H, whose jth row is w_j^T, the entire syndrome of w is given by

$$s = HJw\,. \tag{3.20}$$

Hence the syndrome s is a convenient, linear function of w. Note that the commutation requirement of the stabilizer generators can be expressed as $HJH^T = 0$.

For CSS codes with m_x X-type checks and m_z Z-type checks, the matrix H has the form

$$H = \begin{pmatrix} H_X & 0 \\ 0 & H_Z \end{pmatrix} \tag{3.21}$$

for an $m_x \times n$ matrix H_X and an $m_z \times n$ matrix H_Z. The commutation condition above reduces to $H_X H_Z^T = 0$ in this case.

The formalism also encompasses the case of classical linear codes. In this setting the only errors we care about are of X-type, while the stabilizers are of Z-type (or vice versa). Thus $w = (u, 0)$ for $u, 0 \in \mathbb{F}_2^n$, while H has block form $H = \begin{pmatrix} 0 & H_c \end{pmatrix}$, where the 0 is now the zero matrix. Then $s = HJw = H_c u$, which is the standard syndrome calculation in terms of the classical parity check matrix H_c and error vector u.

3.4.2 Properties of the normalizer

Normalizer as symplectic complement Overloading notation, let S also denote the subspace of $\mathbb{F}_2^{2n}$ spanned by the vector representatives w_j of a set of independent stabilizer generators. By the discussion in §3.3.1, the dimension of S is m. And by the discussion in the previous section, the elements of the normalizer have vector representatives w such that $\langle w, w' \rangle = 0$ for all $w \in S$. Denote this set, the *symplectic complement*, by $S^\perp := \{w \in \mathbb{F}_2^{2n} : \langle w, w' \rangle\ \forall w' \in S\}$; we also overload $S^\perp$ to refer to $N(S, P_n)$.

Size from rank-nullity The dimension of $S^\perp$ is the number of independent generators of $N(S, P_n)$. To determine its size, consider the map $f : \mathbb{F}_2^{2n} \to \mathbb{F}_2^m$ defined by $f(w) = HJw$. It has rank equal to m, since the m rows of H are independent and J is invertible. The kernel of the map is $S^\perp$, and therefore by the rank-nullity theorem it follows that $m + \dim(S^\perp) = 2n$. Thus the dimension of $S^\perp$ is $2n - m$. As a subspace of $S^\perp$, S contributes m of these dimensions, leaving $2(n - m) = 2k$ dimensions for $S^\perp/S$. This subspace corresponds to $2k$ independent generators for $N(S, P_n)/S$, an appropriate number of generators needed to represent P_k, as an X and a Z are needed for each qubit. But we still need to confirm that the algebraic structure of $N(S, P_n)/S$ is indeed P_k.

Isotropic and Lagrangian subspaces This argument also implies that at most n linearly independent elements in $\mathbb{F}_2^{2n}$ can be pairwise symplectic orthogonal. More than n would imply $S^\perp$ has dimension less than n, but this would contradict the assumption that $S \subseteq S^\perp$. In the study of symplectic vector spaces, a subspace like S which is contained in its symplectic complement ($S \subseteq S^\perp$) is called *isotropic*. If S has maximal dimension n, such that $S = S^\perp$, it is called *Lagrangian*. A subspace S is symplectic when $S \cap S^\perp = 0$. Observe that the rank-nullity argument above applies for arbitrary H, not just those corresponding to isotropic subspaces. When $m = 2n$ and the rows of H are independent, it follows that $S^\perp = 0$ and S is symplectic. That is, no Pauli operator can commute with every one of a set of $2n$ independent generators.

3.4.3 Symplectic bases for the Pauli group

Standard generators for P_n The Pauli group P_n has a standard set of generators, namely the X and Z operators acting on the individual qubits. These generators have the property that they come in pairs, one pair for each qubit, such that the operators in the pair anticommute, but they each commute with all the elements of all other pairs.

Symplectic basis The structure of the standard generators is related to the symplectic structure given by J on $\mathbb{F}_2^{2n}$: The vectors associated to the standard generators of P_n form a *symplectic basis* for $\mathbb{F}_2^{2n}$. A symplectic basis for $\mathbb{F}_2^{2n}$ is a basis with elements $e_1, e_2, \ldots, e_n$ and $f_1, f_2, \ldots, f_n$ such that $\langle e_j, e_k \rangle = \langle f_j, f_k \rangle = 0$ while $\langle e_j, f_k \rangle = \delta_{j,k}$. It is easy to see that a collection of vectors w_j, arranged as rows of a $2n \times 2n$ matrix M is a symplectic basis when, for an appropriate ordering of the w_j, $MJM^T = J$. Such a matrix is called a *symplectic matrix*.

Associated to a symplectic basis of $\mathbb{F}_2^{2n}$ is a set of *symplectic generators* for the Pauli group. This is a set of $2n$ Pauli operators, which we may group into pairs $(g_j, \tilde{g}_j)$ for $j = 1, \ldots, n$, such that g_j commutes with all g_k and $\tilde{g}_k$ for $k \neq j$ and anticommutes with $\tilde{g}_j$, and similarly for $\tilde{g}_j$. A set of symplectic generators is also sometimes called a *full tableau*.

Every Pauli operator $g \in P_n$ can be written as a product of the g_j and $\tilde{g}_j$, times a prefactor ± 1 or $\pm i$. Which operators appear is determined by the commutation structure. Specifically, g_j is a factor in the decomposition of g precisely when g anticommutes with $\tilde{g}_j$, $\tilde{g}_j$ when g anticommutes with g_j. This follows from the biorthogonality property of the symplectic basis.

Stabilizer tableau Importantly, the standard generators are not the only set of generators of P_n which have the symplectic structure. Put differently, there are many symplectic bases of $\mathbb{F}_2^{2n}$. Given a stabilizer S, it is possible to construct a symplectic basis for P_n which has as elements generators of both S and of $N(S, P_n)/S$. Such a basis for P_n is called a (full) stabilizer tableau.

For example, Table 3.1 gives a stabilizer tableau of the five-qubit code.

1	$X\ Z\ Z\ X\ I$	$X\ I\ X\ X\ X$
2	$I\ X\ Z\ Z\ X$	$I\ X\ X\ I\ I$
3	$X\ I\ X\ Z\ Z$	$X\ I\ I\ I\ X$
4	$Z\ X\ I\ X\ Z$	$I\ X\ X\ X\ X$
5	$X\ X\ X\ X\ X$	$Z\ Z\ Z\ Z\ Z$

Table 3.1: Stabilizers (left), destabilizers (right), and logical operators (bottom) associated with the five-qubit code.

The stabilizer tableau is not unique; for example, Table 3.2 gives another tableau for the five-qubit code which has the same stabilizer generators but different logical operators and destabilizers.

1	$X\ Z\ Z\ X\ I$	$X\ Z\ X\ I\ I$
2	$I\ X\ Z\ Z\ X$	$X\ Z\ I\ I\ I$
3	$X\ I\ X\ Z\ Z$	$X\ Z\ I\ X\ I$
4	$Z\ X\ I\ X\ Z$	$X\ I\ I\ I\ I$
5	$X\ Z\ I\ I\ Z$	$I\ Z\ I\ X\ X$

Table 3.2: Another stabilizer tableau for the five-qubit code.

3.4.4 Symplectic Gram-Schmidt procedure

Idea of SGS The standard Gram-Schmidt procedure for constructing an orthonormal basis for an arbitrary collection of vectors can be easily modified to yield a means of constructing a basis adapted to the symplectic structure from an arbitrary collection of linearly independent vectors. Let us call it SGS, and describe how it works directly on elements of the Pauli group P_n. Due to the vector correspondence, SGS has essentially the same action on $\mathbb{F}_2^{2n}$.

Input and output of SGS The input to SGS is an arbitrary sequence of independent Pauli operators, call it $G \subseteq P_n$. The outputs of SGS are two-fold: a sequence C of independent commuting Pauli operators and a sequence Q of pairs of Pauli operators, such that all operators are independent and every operator commutes with all other operators in C and Q, except their partner in their pair. For m the number of independent elements of G, the sizes $|C|$ and $|Q|$ will satisfy $|C| + 2|Q| = m$.

Procedure Initially C and Q are empty sets and G is given. Then we repeat the following SGS iterations until G is empty. First pick the first element from G (remove it from G), call it g_a. Next, check if it anticommutes with any other element of G. If not, append g to C, and start the next SGS iteration. If it does, pick/remove the first anticommuting element from G, call it g_b. Then, for all elements $g \in G$, compute $j = \langle w(g), w(g_a)\rangle$ and $k = \langle w(g), w(g_b)\rangle$ and replace g by $g_a^k g_b^j g$. Append the pair (g_a, g_b) to Q and start the next SGS iteration.

Analysis

- The SGS procedure is designed to adjust the commutation of Pauli operators, or equivalently the symplectic property of vectors in $\mathbb{F}_2^{2n}$. It does not deal with independence, which is why the input sequence G is assumed to be independent. This is unlike the familiar case of Gram-Schmidt over $\mathbb{R}^n$, which targets orthogonality under the inner product. Since orthogonality and independence are related in that setting, Gram-Schmidt over $\mathbb{R}^n$ simultaneously handles independence. (Note that there are some independence constraints from commutativity in the symplectic case, as we saw for Lagrangian subspaces.)

- The output C will contain operators which commute with themselves and everything in Q. Thus, it will essentially be the center of G (it will generate the center of the group generated by G).

- The symplectic structure is enforced by the adjustment to G when an anticommuting pair (g_a, g_b) is found. This step ensures that all the updated elements of G commute with both g_a and g_b. Any $g \in P_n$ which anticommutes with g_a can be made to commute by multiplying it with g_b, since then the product $g_b g$ has two factors which anticommute with g_a.

3.4.5 Constructing stabilizer tableau

Logical operators Now it is straightforward in principle to construct the logical operators of a stabilizer code given an independent set of stabilizer generators. The generators are given by the rows of the matrix H. First construct generators of $S^\perp$ by finding a basis for the (right) null space of the matrix HJ. In particular, the basis should be chosen to include the generators of S as the first m elements. Then apply SGS to the basis of $S^\perp$. This will return a set of generators of S in C and a set of generators of $S^\perp/S$ in Q. Hence the logical operators do indeed represent P_k for k encoded qubits.

Full stabilizer tableau Given an independent set of generators of S is it also nice to construct a symplectic basis for P_n which is adapted to S. Such a symplectic basis is called a (full) stabilizer tableau, and it also contains generators of $S^\perp/S$. More specifically, a stabilizer tableau is a symplectic basis consisting of pairs $(g_j, \tilde{g}_j)$ for $j = 1, \ldots, n$ such that $\{g_j\}_{j=1}^m$ generates S and the pairs $\{(g_j, \tilde{g}_j)\}_{j=m+1}^n$ generate $S^\perp/S$. The partners $\{\tilde{g}_j\}_{j=1}^m$ of the stabilizers are often called the "destabilizers". Another good name would be "antistabilizers", but "destabilizers" is funnier.

Note that the stabilizer S does not uniquely determine the tableau. We will say two stabilizer tableau are equivalent when one can be obtained by the following operations on the other. Stabilizers and logical operators can be multiplied by arbitrary stabilizers and destabilizers by any operators, such that the symplectic structure is maintained.

One way to construct a stabilizer tableau from $\{g_j\}_{j=1}^m$ is as follows. First append to $\{g_j\}$ the standard generators to obtain a sequence of $m+2n$ Pauli operators. These are not all independent, so remove elements from the end of the sequence until an independent set of n generators is reached. Applying SGS to the result will generate a symplectic basis adapted to the stabilizer S. In general, for $2n$ independent generators, SGS will return $C = \{\}$ and Q a symplectic basis for P_n. This is because at most n independent Pauli operators can commute. Due to the ordering of the input to SGS, in this case the first elements of the first m pairs in Q will generate S. The remaining $n-m$ pairs generate $S^\perp/S$, just as above.

3.5 Correction of Pauli errors

3.5.1 Syndrome decoding

Consider the effect of a Pauli error $g \in P_n$ on the codeword $|\widehat{\psi}\rangle$ of a stabilizer code $\mathcal{C}$. For stabilizer generators $\widehat{g}$ which commute with g, the state $g|\widehat{\psi}\rangle$ will again be stabilized by $\widehat{g}$. However, when $\widehat{g}$ anticommutes with g, the effect is that the stabilizer is changed to $-\widehat{g}$:

$$\widehat{g}g|\widehat{\psi}\rangle = -g\widehat{g}|\widehat{\psi}\rangle = -g|\widehat{\psi}\rangle\,. \tag{3.22}$$

Thus, the effect of a Pauli error on a stabilizer code is simply to change the sign of some stabilizer generators. Measuring the stabilizers to generate the syndrome reveals precisely which ones are changed, and moreover, this measurement does not cause any change to the state $g|\widehat{\psi}\rangle$.

Given the observed syndrome, the decoder decides on a correction operation g', and therefore upon correction the state becomes $g'g|\widehat{\psi}\rangle$. When $g'g \in \mathcal{S}$, the state is properly restored. Denoting by synd : $P_n \to \mathbb{F}_2^m$ the function which maps an error g to its syndrome $s \in \mathbb{F}_2^m$ and by corr : $\mathbb{F}_2^m \to P_n^\pm$ the function which specifies the correction operation g' given the syndrome s, we can write the condition that an error g is correctly decoded as

$$\mathrm{corr}(\mathrm{synd}(g))g \in \mathcal{S}\,. \tag{3.23}$$

It is illuminating to consider the operation of the syndrome decoder using the decomposition of the error g into stabilizers, destabilizers, and logical operators.

- *Only the destabilizer part can be detected.* This is the only part of g which anticommutes with the stabilizer and can thus change the stabilizer measurement outcome.
- *The stabilizer contribution is responsible for error degeneracy*. Errors which only differ by a stabilizer will certainly have the same syndrome and therefore the same correction operation will be applied. There is also no point in including a stabilizer contribution to the correction Pauli operator, as it will have no effect on the qubits.

- *Logical operators are uncorrectable errors*. This is by design, since they should manipulate the encoded information in the codespace. But therefore they will have a trivial syndrome and be completely undetectable to syndrome decoding.

Therefore, what the syndrome decoder needs to do for each correctable error is determine whether (and what kind) of logical operator it contains, given its destabilizer contribution. The correction operation is then the destabilizer times the logical operator. The former ensures that the system is returned to the codespace, while the latter (hopefully) ensures that the vectors in the code subspace are each properly restored. Both of these aspects are contained in the statement (3.23).

There is a subtlety in this picture of error correction in that the logical contribution to a given error is not independent of the destabilizer contribution. The decomposition is relative to a given stabilizer tableau, and different choices of logical generators will lead to different statements about whether a given error does or does not contain a logical operator in its decomposition. Put differently, the correction operation is independent of the particular stabilizer tableau, but its destabilizer and logical contributions can each vary. This is essentially just the fact that a given vector has different components relative to different bases. Nonetheless, in the context of error correction, it can be tempting to think that that the logical contribution to a given Pauli operator is unique.

3.5.2 Correctable errors

The syndrome decoder can only choose one correction operation for each value of the syndrome, but this does not mean that the decoder can only correctly decode a single error per syndrome value. If $g \in P_n$ is correctly decoded, then so too is g', provided $gg' \in \mathcal{S}$. The destabilizer contributions to g and g' must therefore be identical, and hence the correction will be identical. The correction operation will be $g''g$ for some $g'' \in S$, so that g is corrected. Applied to g', the combined action of error and correction is simply $g''gg'$, which acts trivially on the codespace.

The syndrome decoder will therefore succeed on any set of Pauli errors such that when the errors are partitioned into subsets according to syndrome (destabilizer contribution), the errors in each subset only differ by stabilizers. The stabilizer measurement will determine the subset, and within each subset the logical contribution of all the errors is identical. So the correction operation can be chosen to be the destabilizer indicated by the syndrome times the logical operator indicated by the error subset. Put differently, a sufficient condition for correctability of a set of Pauli errors by syndrome decoding is that the product gg' of any two errors g and g' is either a stabilizer (meaning g and g' have the same syndrome) or anticommutes with some stabilizer (meaning they have different syndromes).

By the same logic, this condition is also necessary. That is, syndrome decoding cannot correctly decode both g and g' if they have the same syndrome (same destabilizer contribution), but different logical contributions. The combined effect of error and correction would in this case be a logical operator, meaning the input state would not be correctly restored.

In fact, these are the same necessary and sufficient conditions that arise directly from the Knill-Laflamme conditions. Hence if a set of Pauli errors are exactly correctable by *some* decoder, then they are correctable by the syndrome decoder. Applied to stabilizer codes and Pauli errors, the KL conditions have the simpler form

Proposition 3.2

A set $\{g_j \in P_n\}_{j=1}^t$ of Pauli errors is correctable using a Pauli stabilizer code $\mathcal{C}$ defined by a stabilizer group S if and only if for all j,k either $g_j^\dagger g_k \in S$ or $g_j^\dagger g_k$ anticommutes with some element of S.

That the condition is sufficient for correctability is established by the syndrome decoder, but let us give a completely self-contained proof.

Proof. To see that correctability follows from the stated conditions, we give a set of $c_{j,k}$ such that the Knill-Laflamme conditions hold. When $g_j^\dagger g_k \in S$, it follows that $Pg_j^\dagger g_k P = P$, and therefore $c_{j,k} = 1$. On the other hand, when $g_j^\dagger g_k$ anticommutes with some $\widehat{g} \in S$, then $c_{j,k} = 0$ since

$$Pg_j^\dagger g_k P = Pg_j^\dagger g_k g P = -Pg g_j^\dagger g_k P = -Pg_j^\dagger g_k P\,. \tag{3.24}$$

To show necessity, we establish the contrapositive, that the negation of the condition implies uncorrectability. So suppose that for some j,k $g_j^\dagger g_k$ commutes with S but is not contained in S. Therefore it is a logical operator, and thus the value of $\langle\widehat{\psi}|g_j^\dagger g_k|\widehat{\psi}\rangle$ will vary with the state $|\widehat{\psi}\rangle$. For instance, $|\widehat{\psi}\rangle$ an eigenstate of $g_j^\dagger g_k$ will lead to a different value than an eigenstate of a logical operator which anticommutes with $g_j^\dagger g_k$. In this case the Knill-Laflamme conditions cannot possibly be satisfied, and the set of errors is necessarily uncorrectable. □

Note that the number of possible syndromes may be much larger than the number of subspaces which are ostensibly correctable, or even the number of errors which are ostensibly correctable. For instance, the Shor code can correct all single-qubit errors as well as no error, which is a total of 28 possibilities. But the eight-bit syndrome can identify $2^8 = 256$ different subspaces. These correspond to higher-weight errors.

Distance The distance d of a stabilizer code $\mathcal{C}$ with stabilizer S is the smallest weight of the logical operators $N(S,P_n)/S$. By the discussion above, it is the smallest number of qubits that can be in error such that the error is undetectable by the code, assuming that the identity operator is a correctable "error". In terms of the Knill-Laflamme conditions, $PMP \neq cP$ for some $c \in \mathbb{C}$ for an operator $M \in P_n$ of weight d. A code of distance d can correct all Pauli errors of weight no larger than $t = \lfloor \frac{d-1}{2} \rfloor$: Since the project of any two such errors M_j and M_k has weight $2t$, $PM_j^\dagger M_k P = cP$ is satisfied for some $c \in \mathbb{C}$.

Encoding, decoding, and the Clifford group 4

In this chapter we study the Clifford group of unitary operators, which give a means to construct encoding and measurement circuits of stabilizer codes. We will also see how to simulate Clifford unitary operations and measurements of Pauli operators just using the formalism of stabilizer tableau, a result is usually known as the Gottesman-Knill theorem.

4.1 The Clifford group

4.1.1 Unitaries which preserve the Pauli group

An important class of n-qubit unitary operations are those U which map Pauli operators to Pauli operators under conjugation: $UgU^\dagger \in P_n$ for $g \in P_n$, i.e. the normalizer $N(G_n, U_n)$ of the Pauli group in the group of unitaries U_n. Since the overall phase of an operator U drops out when U acts by conjugation, we are ultimately interested in $N(G_n, U_n)/U(1)$, the quotient group in which Us with different phases are identified. This group is known as the Clifford group C_n. Since unitaries in the normalizer cannot change eigenvalues, the elements of the Clifford group map $P_n^\pm$ to itself. This allows the action of C_n to be described in terms of the Pauli group itself.

4.1.2 Single- and two-qubit Clifford operators

We have already met two examples of Clifford unitaries, namely the Hadamard and CNOT gates. Another single-qubit Clifford unitary is the $S = \sqrt{Z}$ gate, whose action is $|x\rangle \mapsto i^x|x\rangle$ for $x \in \mathbb{F}_2$. In the matrix representation on the standard basis states of one and two qubits, these operators take the form

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \qquad S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}, \qquad \text{CNOT}_{12} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{4.1}$$

In the CNOT gate, qubit 1 is the control and qubit 2 is the target.

Action on X and Z

- We have already seen that $HXH = Z$ and $HZH = X$.
- For the S gate, clearly $SZS^\dagger = Z$. And by direct calculation we find $SXS^\dagger = Y = iXZ$.
- The CNOT gate is more interesting. When qubit 1 is the control and qubit 2 the target, we have $U_{\text{CNOT}} = |0\rangle\langle 0|_1 \otimes \mathbb{1}_2 + |1\rangle\langle 1|_1 \otimes (\sigma_x)_2$. Therefore I_1X_2 is invariant, i.e. $U_{\text{CNOT}} I_1 X_2 U_{\text{CNOT}}^\dagger = I_1X_2$, since X is the operator in the target. Similarly, Z_1I_2 is also invariant, since the control is done in the Z basis. We will just write these as X_2 and Z_1 from now on, omitting the identity I factors. For X_1 on the other hand, it is clear that flipping the control and then applying CNOT is equivalent to first applying CNOT and then flipping both the control and target. Therefore, X_1 is transformed into X_1X_2. Rewriting the target in the eigenbasis of σ_x gives the equivalent expression $U_{\text{CNOT}} = \mathbb{1}_1 \otimes |+\rangle\langle +|_2 + (\sigma_z)_1 \otimes |-\rangle\langle -|_2$, i.e. while CNOT_{12} is nominally a controlled bit flip on qubit 2, controlled on the Z eigenstate of qubit 1, it is equally well a controlled phase flip on qubit 1, controlled on the X eigenstate of qubit 2. Hence, by the same logic as above, Z_2 is transformed into Z_1Z_2.

Altogether we have the following tables:

	H
X	Z
Z	X

	S
X	Y
Z	Z

	CNOT_{12}
X_1	X_1X_2
Z_1	Z_1
X_2	X_2
Z_2	Z_1Z_2

Generator representation on $\mathbb{F}_2^{2n+2}$ These transformation rules completely determine how H, S, and CNOT gates will act on any given Pauli operator, since single-qubit X and Z operators are the standard basis of P_n. In terms of the $\mathbb{F}_2^{2n}$ representation, we can write $g = (-1)^t i^c X^u Z^v$ for $t, c \in \mathbb{F}_2$ and $u, v \in \mathbb{F}_2^n$, and then compute

$$UgU^\dagger = (-1)^t i^c UX^uU^\dagger UZ^vU^\dagger = (-1)^t i^c UX_1^{u_1}U^\dagger UX_2^{u_2}U^\dagger \cdots UX_n^{u_n}U^\dagger UZ_1^{v_1}U^\dagger \cdots UZ_n^{v_n}U^\dagger\,. \quad (4.2)$$

The H and S gates only act on a single qubit, and their action on, say, qubit j in the $\mathbb{F}_2$ representation is simply

$$H_j : (t, c, (u_j, v_j)) \mapsto (t + u_jv_j, c, (v_j, u_j)) \quad (4.3)$$
$$S_j : (t, c, (u_j, v_j)) \mapsto (t + cu_j, c + u_j, (u_j, u_j + v_j)) \quad (4.4)$$
$$S_j^\dagger : (t, c, (u_j, v_j)) \mapsto (t + u_j + cu_j, c + u_j, (u_j, u_j + v_j))\,. \quad (4.5)$$

Here we include only the u_j and v_j components of w, since all others will be unaffected.

The CNOT gate is more complicated, as it involves two qubits. Suppose that qubit j is the control and qubit k is the target. Then the effect of $\text{CNOT}_{j,k}$ is

$$\text{CNOT}_{j,k} : (t, c, (u_j, u_k, v_j, v_k)) \to (t, c, (u_j, u_j + u_k, v_j + v_k, v_k))\,. \quad (4.6)$$

In this case the phase is completely unaffected, since the CNOT gate neither changes the ordering of X and Z operators when conjugating the expression $(X^{u_1} \otimes X^{u_2})(Z^{v_1} \otimes Z^{v_2})$, nor introduces Y operators.

Other useful single- and two-qubit Clifford operations

- Conjugation by a Pauli operator is a Clifford operation, too, but one which only affects the sign. In particular,

$$X_j : (t, c, (u_j, v_j)) \mapsto (t + v_j, c, (u_j, v_j)) \quad (4.7)$$
$$Z_j : (t, c, (u_j, v_j)) \mapsto (t + u_j, c, (u_j, v_j))\,. \quad (4.8)$$

- The $\text{SWAP}_{j,k}$ operator, which interchanges qubits j and k, can be decomposed into three CNOT gates: $\text{SWAP}_{j,k} = \text{CNOT}_{j,k}\text{CNOT}_{k,j}\text{CNOT}_{j,k}$. Clearly $X_j \longleftrightarrow X_k$ and $Z_j \longleftrightarrow Z_k$, meaning

$$\text{SWAP}_{j,k} : (t, c, (u_j, u_k, v_j, v_k)) \to (t, c, (u_k, u_j, v_k, v_j))\,. \quad (4.9)$$

- Related to CNOT is the controlled-phase gate CPHASE, $U = |0\rangle\langle 0|_1 \otimes \mathbb{1}_2 + |1\rangle\langle 1| \otimes (\sigma_z)_2$. Unlike CNOT, the CPHASE gate is even more symmetric in how it operates on X and Z. Clearly it commutes with both Z_1 and Z_2. Since $\text{CPHASE}_{12} = H_2\text{CNOT}_{12}H_2$, it maps X_1 to X_1Z_2 and likewise X_2 to Z_1X_2. Hence we have

$$\text{CPHASE}_{j,k} : (t, c, (u_j, u_k, v_j, v_k)) \to (t + u_1u_2, c, (u_j, u_k, u_k + v_j, u_j + v_k))\,. \quad (4.10)$$

4.1.3 Representation of C_n over $\mathbb{F}_2$

As we have seen above, since $Ug_1g_2U^\dagger = UgU^\dagger Ug_2U^\dagger$ for $g_1, g_2 \in P_n$, the action of Clifford operators on the standard generators of P_n completely specifies the action on all Pauli operators. Therefore, a Clifford unitary $U \in C_n$ can be specified by a set of symplectic generators $\{(g_j, \tilde{g}_j)\}_{j=1}^n$ taken from $P_n^\pm$, the image under U of the standard generators. We may then recycle the $\mathbb{F}_2^{2n+2}$ representation of P_n to represent elements of C_n. The only restriction is that the phase contribution c for each g_j and $\tilde{g}_j$, with $\mathbb{F}_2^{2n}$ vector (u, v), must satisfy $c = u \cdot v$. Hence an element of C_n is specified by a vector of phases $(t_1, \ldots, t_n) \in \mathbb{F}_2^n$ and a symplectic matrix M whose rows are the $w \in \mathbb{F}_n^{2n}$.

Ignoring phases, the action of $U \in C_n$ on some $g \in P_n$ is given by $w^T M$. This corresponds to expanding g as a product of standard generators, replacing these with their images under U, and then performing the multiplication by addition in $\mathbb{F}_2^{2n}$. To determine the correct phase of the output, we do precisely the same thing, but use (3.9) to perform the multiplication. Denoting the representation of g by $(t, c, (u, v))$, the product we want to compute is $(-1)^t i^c \prod_{j=1}^n g_j^{u_j} \prod_{k=1}^n \tilde{g}_k^{v_k}$ using (3.9).

4.2 Decomposing Clifford operations

It turns out that the three operations CNOT, Hadamard, and phase are enough to generate the Clifford group.

4.2.1 Setup

Consider a Clifford unitary $U \in C_n$, specified by a full tableau T consisting of a $2n \times 2n$ symplectic matrix M and a $2n \times 2$ matrix of pairs (t_j, c_j) which specify the phase. Initially the c_j are such that the Pauli operator corresponding to (t_j, c_j, w_j), for w_j the jth row of M, is Hermitian. The full tableau corresponds to a sequence of $g_j \in P_n^\pm$, $j = 1, \ldots, 2n$ such that g_j and g_{n+j} are anticommuting pairs, for $j = 1, \ldots, n$. Here we switch convention of the symplectic basis to a length-$2n$ sequence of Pauli operators instead of a length-n sequence of pairs of Pauli operators.

The following procedure will generate a sequence $K = (K_1, K_2, \ldots)$ of Clifford generators in which each element is either CNOT, H, $S^\dagger$, X, or Z, such that $\prod_{j=\ell} K_\ell = U^\dagger$. This is done by making use of a block decomposition of M as $M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$, where the blocks are all $n \times n$ matrices.

The procedure sequentially transforms the Pauli operators g_j and g_{n+j} into $\pm X_j$ and $\pm Z_j$, respectively, and then fixes the signs. A sequential procedure is possible because the commutation relations will require that the constructed g_k and g_{n+k} for $k > j$ have only identity factors on qubits 1 to j. Thus, the operations required to bring the original g_j to and g_{n+j} to the desired form do not inadvertently undo the work done in previous steps.

The procedure has four specific parts. In the first step the qubits are reordered so that the jth qubit of the $(n+j)$th generator g_{n+j} is either a Y or a Z. In the second step the $(n+j)$th generator is transformed to $\pm Z_j$. In the third step the jth generator g_j is transformed to $\pm X_j$. Finally, in the last step, the signs are fixed to $+1$.

4.2.2 Procedure

Start with $K = \{\}$. For j in $1, \ldots, n$, do the following:

1. Pick the first $k \geq j$ such that one of $C_{j,k}$ and $D_{j,k}$ is nonzero. Append ($\textsc{cnot}_{j,k}, \textsc{cnot}_{k,j}, \textsc{cnot}_{j,k}$) to K and update the tableau by interchanging columns (j,k) and $(n+j, n+k)$ of M.

2.1. For $k \geq j$ with $C_{j,k} = 1$, if $D_{j,k} = 0$ append H_k to K, else append $(S_k^\dagger, H_k)$ to K. Update the tableau by using (4.3) and/or (4.5) as necessary.

2.2. For $k > j$ with $D_{j,k} = 1$, append $\textsc{cnot}_{k,j}$ to K and update M using (4.6).

3.1. For $k \geq j$ with $B_{j,k} = 1$, if $A_{jk} = 0$ append H_k to K, else append $S_k^\dagger$ to K. Update the tableau by using (4.3) or (4.5) accordingly.

3.2. For $k > j$ with $A_{j,k} = 1$, append $\textsc{cnot}_{j,k}$ to K and update M using (4.6).

4. If $t_j = 1$ append Z_j to K and if $t_{n+j} = 1$ append X_j to K. Update the tableau by setting $t_j = t_{n+j} = 0$.

4.2.3 Analysis

This procedure is essentially equivalent to a method suggested by Gidney.[1]

The second step first removes all the X factors of the $(n+j)$th generator, converting Xs and Ys to Zs. Then it removes all the Z factors except Z_j. The third step fixes the jth generator to be the anticommuting partner X_j of Z_j. Since all the operations preserve the symplectic structure of M, it must be that $A_{j,j} = 1$ at the beginning of the third step. If the jth qubit of the jth generator is X, then no futher action is required on this qubit. If it is Y, then we need to transform Y to X while preserving Z, which is precisely the action of $S^\dagger$. Then step 3.1 continues, and removes all Z factors in the jth generator; these steps leave Z_j unchanged. Finally, all X factors except X_j are removed by $\textsc{cnot}$ gates controlled on the jth qubit, so that again Z_j is unchanged. The result is that $B_{j,k} = C_{j,k} = 0$ and $A_{j,k} = D_{j,k} = \delta_{j,k}$. The symplectic constraint then implies that $A_{k,j} = B_{k,j} = C_{k,j} = D_{k,j} = 0$ for $k > j$. Hence the jth and $(n+j)$th generators are completely decoupled from the remaining generators. They must be $\pm X_j$ and $\pm Z_j$, respectively, as the Clifford operations will ensure the output is Hermitian. The phases can be easily fixed by conjugation with appropriate Pauli operators, and the procedure can proceed with the next pair of generators.

There should be at most $O(n^2)$ gates.

4.2.4 The real Clifford group

Observe that if M is such that none of the Pauli operators contains a Y factor, then the $S^\dagger$ gate will never be necessary. The resulting decomposition will be in terms of H and $\textsc{cnot}$ only. These two operations plus the X and Z Pauli operators generate the *real Clifford group*, the subgroup of Cliffords whose matrix representatives on $\mathbb{C}_2^n$ have real entries. It turns out that the real Clifford group is actually the normalizer of the real Pauli group in the orthogonal group of size n, i.e. those matrices O on $\mathbb{C}_2^n$ for which $OO^T = O^T O = \mathbb{1}$.

4.3 Encoding & measurement circuits

The decomposition procedure in the previous section allows us to construct encoding and measurement circuits for a stabilizer code given a list of independent stabilizer generators.

[1] See here.

4.3.1 Encoding circuits

An encoding circuit for a quantum stabilizer code should map the single-qubit Z stabilizers of the ancilla qubits to the stabilizer generators. Since this will take a commuting part of a symplectic basis to a commuting part of a different symplectic basis, the transformation can be accomplished by a Clifford operation.

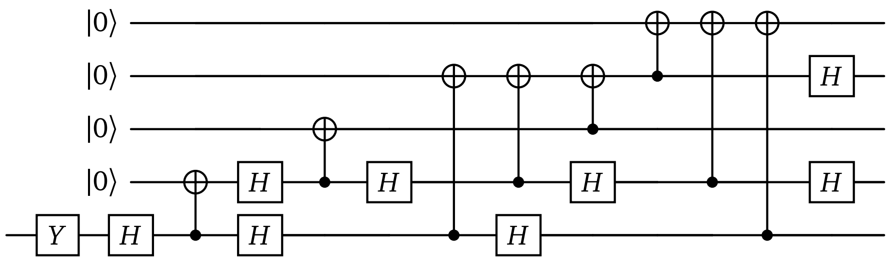
To construct an encoding circuit, simply regard the stabilizer generators as defining $m \times n$ matrices C and D, where m is the number of stabilizer generators, set $M = \begin{pmatrix} C & D \end{pmatrix}$, and apply steps 1 and 2 of the procedure for $j = 1, \dots, m$. Applying the inverses of the gates in K in reverse order gives a circuit which encodes an arbitrary state of $m+1, \dots, n$ and m ancilla qubits in the $|0\rangle$ state into n qubits.

I believe the encoding circuit will require $O(nm)$ gates.

Returning to the types of stabilizer codes, we find

1. General stabilizer codes will have encoding circuits with all three Clifford generators.
2. Codes with stabilizers in the real Pauli group will have encoding circuits generated by H and CNOT.
3. With a slight modification to the decomposition procedure, it is easily seen that encoding circuits of CSS codes are entirely composed of CNOT gates, plus a round of Hadamard gates in the first step. The modification is that only steps 2.2 and 3.2 need be performed, on the Z-type generators and X-type generators, respectively, and finally the resulting individual X_j generators should subsequently be transformed to Z_j by H_j.
4. Classical linear codes have encoding circuits composed solely of CNOT gates.

Example: The five-qubit code Here is an encoding circuit for the five-qubit code. The first input corresponds to the stabilizer $ZXIXZ$, the second to $IXZZX$, the third to $XZZXI$, the fourth to $XIXZZ$, and the final qubit is the input qubit. The initial Y and H gates ensure that the X and Z operators of the input bit are mapped to logical $XXXXX$ and $ZZZZZ$. More directly, X and Z are mapped to $-XZIIZ$ and $-IZIXX$, which are equivalent to $XXXXX$ and $ZZZZZ$ modulo the stabilizers (by multiplication with the last three in the former case and the first two in the latter case).



4.3.2 Stabilizer measurement circuits

Measurement of the stabilizer generators can be accomplished by employing m additional ancilla qubits initialized to $|0\rangle$, applying a suitable Clifford operation on the $m+n$ qubits, and finally individually measuring the ancilla qubits in the Z basis. Here are two possibilities for the Clifford operation.

First, we could measure the individual stabilizers separately. Each separate measurement requires only a single ancilla, which should be initialized to the $|+\rangle$ state by applying H to the $|0\rangle$ state. Then, for each non-trivial factor X, Y, or Z in the stabilizer generator, apply a control operation from the ancilla to the jth qubit, where the operation on the target is the corresponding X, Y, or Z. Note that controlled-Y from qubit j to k is simply $S_k \text{CNOT}_{jk} S_k^\dagger$. Finally, a Hadamard is needed on the ancilla state to convert X basis measurement to Z basis measurement.

Second, we could reuse the decomposition procedure above. Modify stabilizer generators $g_1, \dots, g_m$, to be $g_1' = g_1 \otimes Z_{n+1}$, $g_2' = g_2 \otimes Z_{n+2}$, and so forth, enlarging the Pauli group to $n+m$ elements. Now consider a Clifford operator which transforms Z_{n+j} to g_j', which is specified by a $m \times 2(n+m)$ matrix M. Utilizing the Clifford decomposition procedure used in the encoding circuit construction gives a quantum circuit such that applying the circuit and measuring Z_{n+j} gives the same result as measuring g_j directly.

4.4 Simulation of Clifford unitaries and Pauli measurements

We can simulate the action of Clifford unitaries and Pauli measurements just using the $\mathbb{F}_2$ representation of the Pauli group. That is, we can avoid using the quantum state space representation, and therefore the simulation can be much more efficient.

4.4.1 Simulating unitaries

Section 4.1.3 describes the procedure for simulating the action of Clifford unitaries on individual Pauli operators, collections of them, all the way up to full tableau.

Another useful simulation is of Pauli errors. Here we are less concerned about keeping track of the phase, since the effect of a Pauli error on an error-correcting code state for the purposes of error correction does not depend on the phase. Suppose U describes a quantum circuit and E is a Pauli error. The effect of E followed by U can be expressed as a modified error E' following the circuit U, where $E' = UEU^\dagger$ so that $UE = E'U$. In this sense E is propagated through the circuit U. The vector w' corresponding to E' can be easily computed from the vector w corresponding to E and the matrix M corresponding to U: $w'^T = w^T M$.

4.4.2 Simulating Pauli measurement

Setup Luckily, we can also describe the effect of Pauli measurement on a stabilizer code in terms of the Pauli operators. The reason is that measurement of a Pauli operator $g \in P_n^\pm$ will definitely result in a state which is an eigenstate of g, either stabilized by g or $-g$. So measurement can potentially modify the stabilizer. It can also modify the logical operators, and we will see that *Pauli measurements can induce Clifford operations on stabilizer code states*.

Measurement of Pauli operator g on a state $|\psi\rangle$ which is stabilized by a Pauli operator g' that anticommutes with g will lead to a completely random outcome. This is simply because the probability of a 0 outcome, $\langle\psi|\frac{1}{2}(\mathbb{1}+g)|\psi\rangle$, is equal to the probability of a 1 outcome, $\langle\psi|\frac{1}{2}(\mathbb{1}-g)|\psi\rangle$:

$$\langle\psi|\tfrac{1}{2}(\mathbb{1}+g)|\psi\rangle = \langle\psi|\tfrac{1}{2}(\mathbb{1}+g)g'|\psi\rangle = \langle\psi|\tfrac{1}{2}(\mathbb{1}-g)|\psi\rangle\,. \tag{4.11}$$

Thus, there are no difficulties in determining complicated measurement outcome probability distributions.

Nominally we should aim to simulate the measurement of a given Pauli operator on a particular state: Unlike the case of describing unitaries, surely a state is needed to decide whether the measurement outcome is deterministic (and what value it will take) or random. We could therefore hope to simulate Pauli measurements on *stabilizer states*, states $|\psi\rangle$ on n qubits which have a stabilizer group with n independent generators. However, we do not need to be so restrictive. Instead, we can use the full tableau to simulate the action of measurement for a wide class of input states.

4.4.3 Example: Teleportation

Let us see how teleportation can be modelled in this formalism. Recall that the teleportation protocol is specified as follows. The initial state is comprised of three qubits: an arbitrary input qubit, number 1, and two qubits 2 and 3 in the maximally-entangled state $|\Phi\rangle = |00\rangle + |11\rangle$. Then qubits 1 and 2 are measured in the basis of the Bell states, which amounts to measuring X_1X_2 and Z_1Z_2. Finally, qubit three is subject to the Pauli operator X^bZ^a, where a is the outcome of the X_1X_2 measurement and b is the outcome of the Z_1Z_2 measurement.

The full tableau initially is just

$$\begin{array}{c|c} X_2X_3 & Z_2 \\ Z_2Z_3 & X_3 \\ \hline \multicolumn{1}{c}{X_1} & Z_1 \end{array} \tag{4.12}$$

Here the input qubit is listed last and the left column of the first two rows is the stabilizer.

Now we are meant to measure X_1X_2. It anticommutes with the stabilizer Z_2Z_3, so the measurement result will be uniformly distributed. After the measurement the system will certainly be in the eigenspace of X_1X_2 corresponding to the measurement result $a \in \mathbb{F}_2$. That is, $(-1)^aX_1X_2$ will be a stabilizer. But what will be the effect on the remainder of the tableau? The simplest case would be if X_1X_2, anticommuted with just one of the stabilizer operators, i.e. if X_1X_2 were the destabilizer associated to one of the stabilizers. Then the two operators would switch roles, and there would be no effect on the rest of the tableau.

In fact we can find a tableau equivalent to the initial tableau for which this is the case. The initial destabilizer of interest is the one which anticommutes with the stabilizer with which X_1X_2 anticommutes, i.e. X_3. Evidently, multiplying it by the other X-type stabilizer X_2X_3 and the logical operator X_1 results in X_1X_2. Consider these steps one at a time. Multiplication with X_2X_3 does not change anything, since X_2X_3 is a stabilizer. But we will need to adjust the Z-type operators in the tableau to maintain the commutation relations. In particular, when replacing X_3 by $X_2 = X_2X_3X_3$ we should also replace Z_2 by $Z_3 = Z_2Z_2Z_3$, i.e. multiply the associated partner operators together. Doing so produces the tableau

$$\begin{array}{c|c} X_2X_3 & Z_3 \\ Z_2Z_3 & X_2 \\ \hline \multicolumn{1}{c}{X_1} & Z_1 \end{array} \tag{4.13}$$

(Note that we could have started with this tableau in the first place.) We can do the same to multiply by X_1. Replacing a destabilizer with a destabilizer times a logical operator does not change anything, since the destabilizer does not determine the state anyway. Meanwhile, the logical Z_1 operator will be multiplied by Z_2Z_3 in order to preserve commutation. But this is a stabilizer, so this does not meaningfully change the logical operators. The tableau is now

$$\begin{array}{c|c} X_2X_3 & Z_3 \\ Z_2Z_3 & X_1X_2 \\ \hline \multicolumn{1}{c}{X_1} & Z_1Z_2Z_3 \end{array} \tag{4.14}$$

We are ready to update the tableau to include the measurement operation. Since X_1X_2 is a destabilizer, we simply swap it with Z_1Z_2 and include the phase factor from the measurement. The tableau becomes

$$\begin{array}{c|c} X_2X_3 & Z_3 \\ (-1)^a X_1X_2 & Z_2Z_3 \\ \hline X_1 & Z_1Z_2Z_3 \end{array} \tag{4.15}$$

Next we measure Z_1Z_2, which anticommutes with X_2X_3 and X_1. Again the measurment result is uniformly distributed. Adjusting the tableau and inserting the measurement result $(-1)^b$, we have

$$\begin{array}{c|c} (-1)^b Z_1Z_2 & X_2X_3 \\ (-1)^a X_1X_2 & Z_2Z_3 \\ \hline X_1X_2X_3 & Z_1Z_2Z_3 \end{array} \tag{4.16}$$

Now observe that multiplying the $Z_1Z_2Z_3$ logical operator by the $(-1)^b Z_1Z_2$ stabilizer, gives $(-1)^b Z_3$ as an equivalent logical operator. To maintain the symplectic basis, we should also multiply the X_2X_3 destabilizer by the $X_1X_2X_3$ logical oeprator to obtain

$$\begin{array}{c|c} (-1)^b Z_1Z_2 & X_1 \\ (-1)^a X_1X_2 & Z_2Z_3 \\ \hline X_1X_2X_3 & (-1)^b Z_3 \end{array} \tag{4.17}$$

Doing the same for the X-type logical operator gives

$$\begin{array}{c|c} (-1)^b Z_1Z_2 & X_1 \\ (-1)^a X_1X_2 & (-1)^b Z_2 \\ \hline (-1)^a X_3 & (-1)^b Z_3 \end{array} \tag{4.18}$$

Finally, we apply the Pauli operator X^bZ^a to qubit 3, resulting in the tableau

$$\begin{array}{c|c} (-1)^b Z_1Z_2 & X_1 \\ (-1)^a X_1X_2 & (-1)^b Z_2 \\ \hline X_3 & Z_3 \end{array} \tag{4.19}$$

Therefore, the input Pauli operators on qubit 1 have been transferred to qubit 3. The first two qubits are, unsurprisingly, left in a maximally entangled state determined by a and b. It can be shown that if the Pauli operators are transferred without any change, then all operators are as well, meaning the overall action on the input qubit 1, no matter its state, is to transfer it to qubit 3. We have proven that teleportation works as intended by using the stabilizer formalism.

4.4.4 Gottesman-Knill

Symplectic Gaussian elimination Before launching into how the simulation works in general, we first formalize the tool we used in the above example, a form of Gaussian elimination adapted to the symplectic setting at hand. Suppose $\{(g_j, \tilde{g}_j)\}_{j=1}^n$ is a symplectic basis for P_n and $g \in P_n$ is given. We can adapt the symplectic basis so that g only anticommutes with one particular generator, as follows.

Suppose g anticommutes with $\{g_{j_\ell}\}_{\ell=1}^s$ and $\{\tilde{g}_{j_{\ell'}}\}_{\ell'=1}^t$. This means that $g \sim \prod_{\ell,\ell'} \tilde{g}_{j_\ell} g_{j'_\ell}$. Let us change the basis so that g only anticommutes with g_{j_1}. That is, we can construct a new basis consisting of pairs $(g'_j, \tilde{g}'_j)$ in which $g'_{j_1} = g_{j_1}$ and $\tilde{g}'_{j_1} = g$. Simply set $g'_k = g_{j_1}^{\langle g, g_k \rangle} g_k$ and $\tilde{g}'_k = g_{j_1}^{\langle g, \tilde{g}_k \rangle} \tilde{g}_k$ for $k \neq j_1$.

This doesn't change the commutation structure of the non-j_1 pairs, since g_1 commutes with all of those operators. For the same reason, g'_{j_1} commutes with all the operators in the non-j_1 pairs. By construction, g'_{j_1} definitely anticommutes with $\tilde{g}'_{j_1}$. It remains to check that $\tilde{g}'_{j_1}$ commutes with all the operators in non-j_1 pairs. This holds because for every g_k which does anticommute with $\tilde{g}'_{j_1}$, a factor of g'_{j_1} is multiplied into the operator to cancel the anticommutation.

Procedure Suppose $\{(g_j, \tilde{g}_j)\}_{j=1}^n$ is a symplectic basis for P_n and we would like to simulate the measurement of some $g \in P_n$ under the condition that the input state is stabilized by the g_j for $j = 1$ to $m \leq n$. Thus, the corresponding $\tilde{g}_j$ are the associated destabilizers, while g_k and $\tilde{g}_k$ for $k > m$ are logical operators associated to the stabilizer. The simulation output should give the measurement result and it should describe the possible post-measurement states, by giving the new logical operators.

There are three cases we need to consider:

1. g commutes with both the stabilizer S and the logical operators $S^\perp/S$,

2. g commutes with the stabilizer, but anticommutes with some of the logical operators,

3. g anticommutes with some of the stabilizer generators.

In case 1, g can only have stabilizer generators in its decomposition. In case 2 it can have some stabilizer contribution, plus some logical contribution. In case 3 it has some destabilizer contribution, and possibly some logical operator contribution. Here is the procedure in each case:

1. In this case the measurement result is determined, and the symplectic basis describing the system need not change under the measurement. (Applying the symplectic Gaussian elimination procedure above will at most change the set of stabilizer generators and their associated destabilizers. But since the system is definitely in an eigenstate of the stabilizer group, the particular generators do not matter.)

   The remaining task is to determine the measurement result. One of g and $-g$ is in the stabilizer, which is to say that either $g = \prod_\ell g_{j_\ell}$ for some collection of stabilizer generators g_{j_ℓ} or $g = -\prod_\ell g_{j_\ell}$. By checking anticommutation with the various destabilizers, we can determine the set of g_{j_ℓ} which contribute to g. Finally, we simply check if $g = \prod_\ell g_{j_\ell}$ and report the measurement result 0 if this equality holds, and report measurement result 1 if not.

2. In this case the measurement result is indeterminate, one additional operator will be added to the stabilizer generators, and the logical operators will change. We cannot determine the probability distribution of the measurement results, since we do not have a full specification of the input state.

   Pick a logical operator pair $(g_k, \tilde{g}_k)$ such that one anticommutes with g and relabel them if necessary so that $\tilde{g}_k$ anticommutes with g. Apply the symplectic Gaussian elimination procedure for $\tilde{g}_k$, then replace $(g_k, \tilde{g}_k)$ by $((-1)^a g, \tilde{g}_k)$ with $a \in \mathbb{F}_2$, and move this pair to the list of stabilizer/destabilizer pairs. The parameter a is the measurement result, but we do not know the probability of outcome 0 or 1.

3. In this case the measurement result is random, the stabilizer group will change but keep the same size, and the logical operators may change.

   Pick a stabilizer operator pair $(g_k, \tilde{g}_k)$ such that one anticommutes with g and relabel them if necessary so that $\tilde{g}_k$ anticommutes with g. Apply the symplectic Gaussian elimination procedure for $\tilde{g}_k$. At most this will multiply operators in the tableau by a stabilizer. Finally, replace $(g_k, \tilde{g}_k)$ by $((-1)^a g, \tilde{g}_k)$ with a a random choice of 0 or 1.

5 Optimal decoding of Pauli channels

Here we define two "optimal" decoders for a stabilizer code, for two notions of optimal. Both are based on the probabilistic model of the noisy channel. One returns the most likely error pattern given the syndrome and the other the most likely logical operator caused by the noise.

Recall from §3.3 that the task of a decoder is to determine an appropriate correction operator $g \in P_n^+$ given the output of the stabilizer measurement, the syndrome s. The correction operator is then applied to the qubits. Decoding is successful if the actual error $g' \in P_n^+$ and correction g are such that their product is in the stabilizer group: $gg' \in S$. From the Pauli decomposition, this condition implies that the state of the qubits is returned to the codespace and there is no logical error. For a Pauli noise model, specified by a probability function P_E so that $P_E[g' \in P_n^\pm]$ is in $[0,1]$, syndrome function $f(g) = HJw(g)$, and decoding function $r(s) \in P_n^\pm$, the probability of a logical error after error correction is then $\sum_{g' \in P_n^\pm} P_E[g']\mathbb{1}[r(f(s))g' \notin S]$, where $\mathbb{1}[]$ denotes the indicator function.

5.1 Most likely error

The most straightforward decoder simply looks for the most likely error given the syndrome. Call this function MLE for "most likely error". Nominially it is just

$$\mathrm{MLE}(s) := \underset{g \in P_n^+ : f(g) = s}{\arg\max}\; P_E[g]. \tag{5.1}$$

However, it can easily happen that there are multiple errors of largest probability, and then some method is needed to break ties. So it is perhaps better to speak of *an* MLE decoder rather than *the* MLE decoder. Conceptually the simplest tie breaking method is to just choose randomly, which we can denote by MLE_r.

5.1.1 Minimum-weight decoder

For error models P_E which have an i.i.d. structure, the probability of an error is larger the smaller the weight of the error. Hence in this case MLE looks for errors of minimum weight. Unfortunately, the general task of minimum-weight decoding is known to be NP-complete, meaning an efficient general-purpose algorithm (not taking into account the details of the specific code) is exceedingly unlikely to exist. For small enough codes we can simply brute force search through all possible error patterns consistent with each syndrome, but this rapidly becomes intractable as the code size increases. From the Pauli decomposition, for a code on n qubits with m stabilizer generators, the possible error patterns consistent with a given syndrome are 2^{2n-m} in number.

5.1.2 Bounded-distance decoder

Related to the minimum-weight decoder is the *bounded-distance decoder* (BDD), specified by a distance d, which looks for a minimum weight error which has weight no larger than t, for $t = \lfloor \frac{d-1}{2} \rfloor$, and gives up if such an error is not found. For a code of distance d, the bounded-distance decoder correctly recovers from all of the correctable errors up to the distance, but not more. The BDD's performance is precisely characterized by the distance of the code. Note that by the perfect error-correcting conditions, if there are two errors of minimum weight, their product will necessarily be an element of the stabilizer group. Hence all errors of minimum weight for a given syndrome are degenerate errors, and so any scheme of breaking ties is as good as any other. We define this decoder mainly to point out that a useful code need not have a very high distance, and that the BDD often has poor performance except at very low noise rates. Figure 5.1 shows the decoding error probability for the Shor code under BDD, MLE, and the concatenated repetition decoder (CRD) defined in §1.5.2.

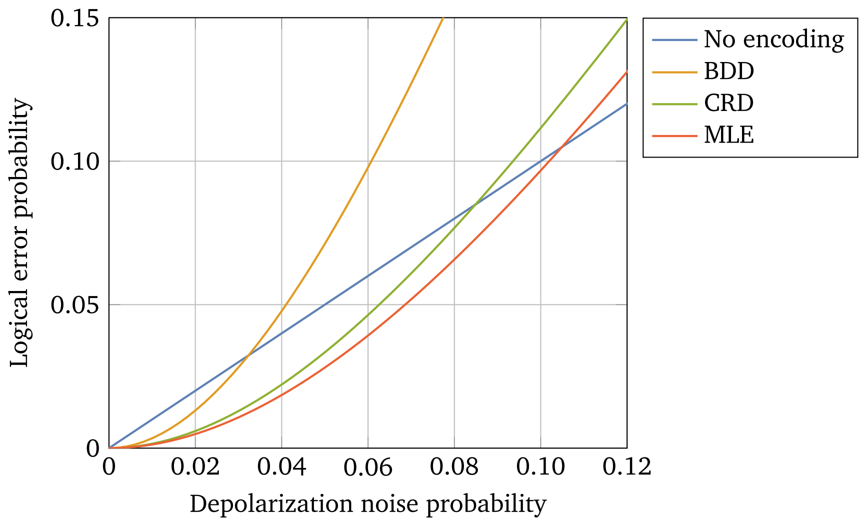


Figure 5.1: Logical error rate of various decoders for the Shor code subject to depolarizing noise. The three decoders all have quadratic logical error rate for very small rates of physical noise, indicating they each successfully recover from all single-qubit errors. Nevertheless, the break-even points of the decoders differ substantially.

5.2 Most likely logical

Using the decomposition of the Pauli group into stabilizer, destabilizer, and logical generators, the MLE function in (5.1) can be expressed as

$$\mathrm{MLE}(s) := \xi^\star \eta(s)\tau^\star \qquad \text{for} \qquad (\xi^\star,\tau^\star) = \operatorname*{arg\,max}_{\xi\in S,\tau\in L} P_E[\xi\eta(s)\tau]\,, \tag{5.2}$$

where S is the stabilizer group of the code, L is the group of logical operators, and $\eta(s)$ is the destabilizer operator determined by the syndrome. The expression $\xi^\star\eta(s)\tau^\star$ is therefore an element of $P_n^\pm$.

Error degeneracy implies that all errors $\xi\eta(s)\tau^\star$ for $\xi \in S$ have the same effect on the code. Thus, it is better to look for the τ leading to the most likely class of errors, not the pair (ξ,τ) leading to the most likely individual error. The probability of the logical error class is just the sum over the probabilities for each of the possible stabilizer contributions, which leads to the *most likely logical* decoder

$$\mathrm{MLL}(s) := \eta(s)\tau^\star \qquad \text{for} \qquad \tau^\star = \operatorname*{arg\,max}_{\tau\in L} \sum_{\xi\in S} P_E[\xi\eta(s)\tau]\,. \tag{5.3}$$

Again, it is necessary to specify how ties are broken to obtain a concrete decoding function.

Since the logical contribution in a given Pauli error is the only thing relevant to correct recovery, MLL is the optimal decoder for the given noise model. Note, though, that the noise model used in the MLL and MLE decoders need not be the same as the noise model of the channel. For instance, we could make use of a simpler noise model in the decoder than the one we think accurately describes the channel, in order to reduce the computational complexity of the decoder.

5.3 Role of degeneracy in decoding

5.3.1 Sometimes there is no advantage

The MLL and MLE decoders are usually distinct, but it can happen that they are identical. This happens for the Shor code under depolarizing noise, for instance; the red MLE curve in Figure 5.1 is in fact the lowest logical error rate possible for any decoder. The equivalence between MLL and MLE holds in this case even though some syndromes have non-unique minimum weight errors. By simply finding the minimum weight errors and computing the logical class probabilities for each syndrome (with the assistance of a computer), it can be verified that whenever there are multiple minimum-weight errors, the corresponding logical classes all have the same probability. So no matter how ties are broken for each decoder, their performance is identical.

For example, suppose the syndromes corresponding to Z_1Z_2 and $X_4X_5X_6X_7X_8X_9$ are nontrivial. There are six possible errors of weight two which are consistent with the syndrome: X_1Z_7, X_1Z_8, and X_1Z_9 on the one hand, and Y_1Z_4, Y_1Z_5, and Y_1Z_6 on the other. All the errors in either of these triples are equivalent by degeneracy, namely shifting the location of the Z contribution with the weight-two Z-type stabilizers. The errors in the former triple anticommute with logical Z (i.e. $Z_1Z_4Z_7$) and so act as logical X (i.e. $X_1X_2X_3$) on the codespace. Meanwhile, the errors in the latter triple act as logical Y on the codespace, since they each anticommute with both logical X and logical Z. The equivalence classes of errors corresponding to logical X and to logical Y have identical probability, and so the MLL decoder's performance is the same as that of the MLE decoder.

5.3.2 Advantage by tie-breaking

We need not look too far to find an example in which MLL and MLE behave differently: Already for the 3×3 surface code the two decoders are distinct. This is depicted in Figure 5.2.

In this case one can verify (again by computer) that again there are sometimes multiple possible errors of minimum weight consistent with the syndrome. It turns out that the possible correction operations suggested by the MLL probability calculation are always a subset of these, sometimes a strict subset. Thus, not all minimum-weight errors consistent with the syndrome are equivalent in terms of decoding performance. Any MLE decoder with a simple method of breaking ties will not know which one to pick, however. Ties are broken in the MLL and MLE decoders of Figure 5.2 by choosing the first element in the list of possibilities generated by the particular algorithm.

For example, consider the case that both X and Z plaquettes in the first row are nontrivial. Take the logical operators to be XXX along the bottom row and ZZZ along the right column. There are again six errors of minimum weight which are compatible with the syndrome: Y_2X_3 and X_2Y_3, along with Z_2X_4, Z_3X_4, X_1Z_2, and X_1Z_3. The latter four commute with both logical operators, while the first two anticommute with the logical Z operator and therefore act as logical X. Just looking at the minimum weight errors, the probability for the effect of the error to be logical identity is twice as large as for it to be logical X, so the decoder decides for the former. (The same conclusion

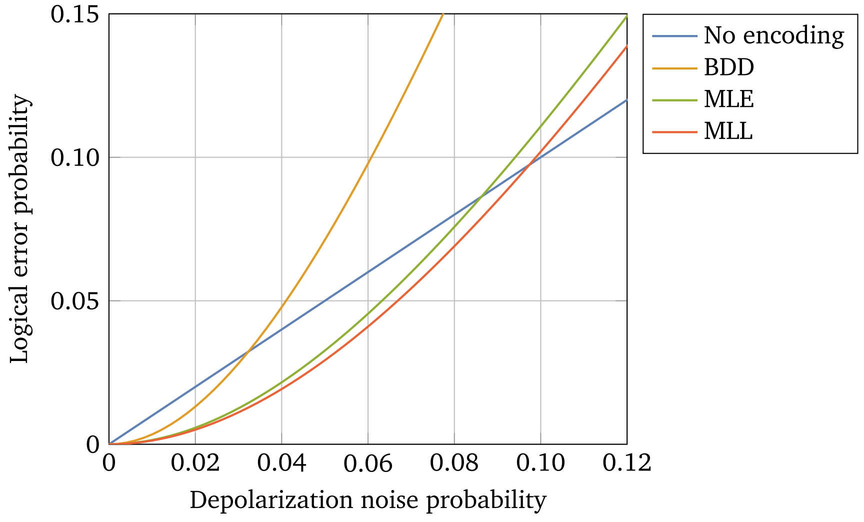


Figure 5.2: Logical error rate of various decoders for the 3 × 3 surface code subject to depolarizing noise. The BDD curve is the same as in Figure 5.1. For this code the MLL decoder outperforms the MLE decoder; both recover from all single-qubit errors of course.

is reached when performing the full calculation, including all possible errors consistent with the syndrome, not just those which have minimum weight.)

Figure 5.3 shows a different example involving just X-type errors.

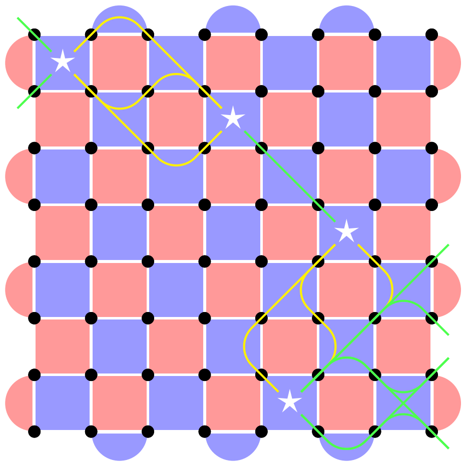


Figure 5.3: Importance of degeneracy in breaking ties between multiple minimum-weight errors. The four nontrivial Z-type syndromes (white stars) can be explained by bit flip error patterns of weight 6. Each possible error pattern is a "path" through the qubits from one nontrivial syndrome to another, or to the boundary. In yellow are 9 different error patterns and in green 12 possible error patterns. The product of any "yellow" error with a "green" error is a logical X operator, spanning from the left boundary to the right boundary. Hence yellow and green correspond to different logical classes. Green is the more likely logical class, at least when considering only the minimum-weight errors. (This conclusion also holds when considering higher weight errors.)

5.3.3 Advantage by probability

The most striking example of the difference between MLL and MLE is the case in which the most likely logical class does not contain an error of minimum weight. To give an example we need only modify the previous example slightly, using a larger surface code, as depicted in Figure 5.4.

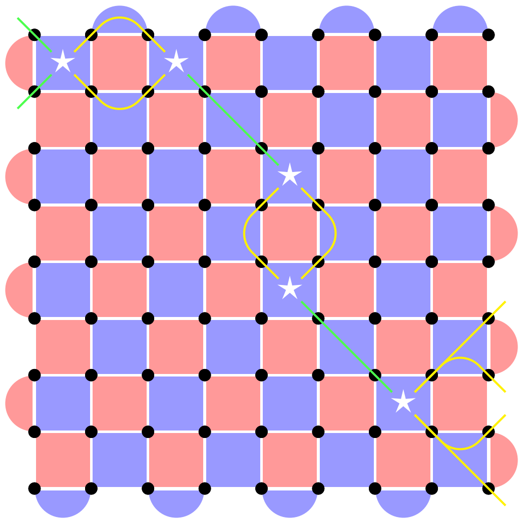


Figure 5.4: Importance of degeneracy over minimum-weight decoding. For the given syndrome there are two minimum-weight errors, of weight 5, depicted in green. Meanwhile there are 16 error patterns of weight 6, shown in yellow. The yellow and green errors are in different logical class, since the product of a yellow error with a green error is a logical X operator. It turns out that the MLL decoder will decide for yellow when the bit flip error noise rate is above roughly 5%.

5.4 Decoding CSS codes

For CSS codes, an appealing option is to correct the X- and Z-type errors separately and independently, using the syndrome from the Z-type checks for the former and the syndrome from the X-type checks for the latter. The advantage of doing so is that each correction task is simpler, dealing with only one kind of error. Both MLE and MLL methods are applicable; the MLE task is precisely the same as for a classical binary linear code and can potentially be handled with classical decoding methods.

Regardless of which decoding method is used, if the two separate decoders are accurate, then the overall decoder is accurate. Concretely, suppose p_Z is the probability of logical Z-type error under the given decoder and p_X the probability of X-type error. Then by the union bound, the probability of any logical error is no larger than $p_X + p_Z$. If the actual error is entirely of Y-type, it will appear to be both an X error and a Z error. In this case this bound for the logical error probability under separate decoding is tight.

For noise with correlated bit and phase flip errors, e.g. depolarization, the performance of separate decoding can be enhanced somewhat by decoding separately but sequentially. After decoding the X-type errors, the resulting correction operation can be used to update the probability model for phase errors (treating the correction as a stand-in for the actual error). For instance, if the noise is described by the depolarizing channel with probability p, then given that a bit flip has occurred, the probability of a phase flip is $1/2$, irrespective of p. On the other hand, given that a bit flip has not occurred, the probability of a phase flip drops by roughly a factor of 2, from $2p/3$ to $p/(3-2p)$. The output of the bit flip decoder thus informs the phase flip decoder where the phase flips are more or less likely to be.

Code construction 6

6.1 Concatenation

We encountered the idea of concatenated codes already in the construction of the Shor code. The broad idea is to take the encoders from two codes C_{outer} and C_{inner} and run the inner encoder on the output of the outer encoder. The names are chosen so that the inner code is closer to the channel. The simplest case is that C_{inner} encodes a single qubit, and its encoder is applied to each of the individual qubits in the output of the C_{outer} encoder. This is depicted in Figure 6.1.

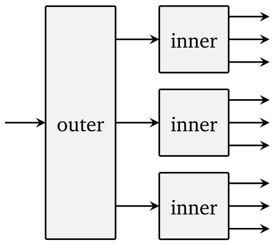


Figure 6.1: Schematic depiction of encoding into a concatenated code.

For instance, encoding each of the qubits themselves in the five-qubit code into the five-qubit code results in a code of size 25 which encodes a single qubit. The figure makes clear that one could in principle use different codes for each different qubit of the outer code. Even more generally, one can choose an inner code which encodes k qubits and an outer code whose blocklength is some multiple of k, so that k-sized blocks of the outer code are each encoded by the inner code.

Figure 6.1 also suggests using concatenation for decoding as well. In the first stage, the inner codes are decoded to single qubits and then in the second stage these are passed to the outer decoder. This is precisely the original decoding scheme we developed for the Shor code. Figure 5.1 shows that concatenated decoding is not necessarily even as accurate as MLE decoding. Nonetheless, decoding complexity scales favorably with the code size.

6.2 CSS codes from classical codes

CSS codes are combinations of two classical codes, so it is natural to construct (hopefully) interesting CSS codes from interesting classical codes. The difficulty is the CSS constraint between the two classical codes. Recall that two classical codes with parity check matrices H_X and H_Z can be combined to a CSS code (with H_X specifying the X-type stabilizers and H_Z the Z-type stabilizers) if and only if $H_X H_Z^T = 0$.

6.2.1 General Shor construction

For any two binary linear codes C_1 and C_2, concatenating C_1 as the inner code to correct bit flips with C_2 as the outer code to correct phase flips results in a CSS code. If H_1 is an $m_1 \times n_1$ parity check matrix of C_1 and H_2 an $m_2 \times n_2$ parity check matrix of C_2, then the Z-type parity check matrix of the resulting CSS code is simply

$$H_Z = \mathbb{1}_{n_2} \otimes H_1 \,. \tag{6.1}$$

For the X-type parity check matrix, first define the generator matrix for C_1, i.e. an $(n_1 - m_1) \times n_1$ matrix G_1 such that $G_1 H_1^T = 0$. The rows of the generator matrix, by definition, are codewords of

C_1. Then the X-type parity check matrix is

$$H_X = H_2 \otimes G_1 \,. \tag{6.2}$$

Observe that the commutation relation $H_X H_Z^T = 0$ is satisfied since $H_X H_Z^T = H_2 \mathbb{1}_{n_2} \otimes G_1 H_1^T$. The resulting CSS code has $n_1 n_2$ qubits, $n_2 m_1$ Z-type stabilizers, and $m_2(n_1 - m_1)$ X-type stabilizers. Thus there are $k_1 k_2 = (n_1 - m_1)(n_2 - m_2)$ encoded qubits.

6.2.2 LDPC codes

Among the most interesting classical codes are *low-density parity check* (LDPC) codes. As the name implies, they have parity check matrices with a small number of nonzero entries per row and per column. That is to say, each bit is involved in only a small number of parity checks (the row weight) and each check is only a function of a small number of bits (the column weight). LDPC codes are appealing classically as they encode a large number of bits and still manage to achieve very low error rates with reasonable decoding complexity. Often one considers a family of LDPC codes of increasing blocklength and usually column and row weight which is *constant* in the blocklength. It is not difficult to construct classical LDPC codes of constant rate and distance scaling in proportion to the blocklength.

For encoding quantum information, having codes whose stabilizers are all of constant weight has advantages in the setting of noisy error correction, as we will see later. However, the CSS constraint makes it somewhat difficult to construct good quantum LDPC codes from good classical LDPC codes. Suppose that both H_X and H_Z are to be low density parity check matrices of a CSS code. But then the fact that $H_X H_Z^T = 0$ implies that the rows of H_X are codewords of the code defined by H_Z. Since H_X has low-weight rows, this implies that the code defined by H_Z has low-weight codewords. Hence, it is not a very good classical code. Put differently, if a quantum LDPC code has good decoding performance, it is not because the underlying classical LDPC codes have good performance. Instead, quantum LDPC codes can only have good performance because it is not necessary to find the exact error, i.e. degeneracy.

We have already met one example of a quantum LDPC code: the surface code. Since it has parity checks of weight two or four, no matter the size of the code, and each qubit is only involved in at most four stabilizers, it is an LDPC code.

6.2.3 Hypergraph product codes

Despite the above difficulties, there is one simple construction that yields quantum LDPC codes with decent performance from classical LDPC codes. For reasons to do with how it was originally discovered, it is called the *hypergraph product construction*. In terms of parity check matrices it is extremely simple. Suppose, as above, H_1 and H_2 are two classical codes. Then define X-type and Z-type parity check matrices as the following block matrices:

$$H_X = \begin{pmatrix} H_1 \otimes \mathbb{1}_{n_2} & \mathbb{1}_{m_1} \otimes H_2^T \end{pmatrix} , \tag{6.3}$$
$$H_Z = \begin{pmatrix} \mathbb{1}_{n_1} \otimes H_2 & H_1^T \otimes \mathbb{1}_{m_2} \end{pmatrix} . \tag{6.4}$$

It can be readily verified that the CSS constraint is satisfied. The number of qubits is $n_1 n_2 + m_1 m_2$ and the number of encoded bits is $k_1 k_2 = (n_1 - m_1)(n_2 - m_2)$ as in the Shor construction. It turns out that the distance of the quantum code is at least the smaller of the distances of two classical codes. For instance, if the two codes are identical and have constant rate $r = k/n$ and distance d, then the resulting quantum code has size $n^2(1+(1-r)^2)$, rate $r^2/(1+(1-r)^2)$, and distance at least d.

The simplest example of the hypergraph product is the product of two repetition codes, which gives the original version of the surface code. (The version we have considered already came later and is often called the "rotated" surface code.) Following the construction for the length-3 repetition code gives a code of 13 physical qubits encoding a single logical qubit, depicted in Figure 6.2.

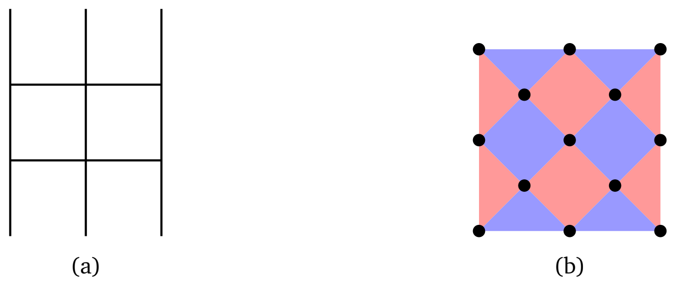


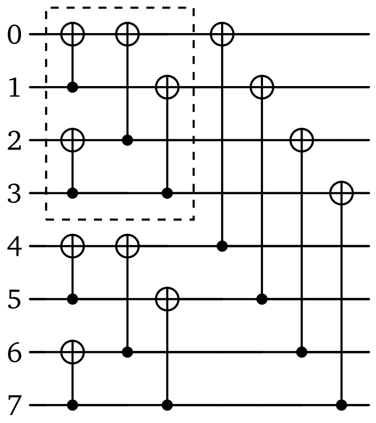
Figure 6.2: Graphical depiction of the original surface code. In (a), qubits are located on edges. Z-type stabilizers involve all qubits around a plaquette, including the open plaquettes at the top and bottom. Called "rough boundaries", these are weight-three stabilizers. X-type stabilizers involve all qubits around a vertex. The weight three X-type stabilizers on the left and right form the "smooth" boundaries. The logical X operator can be taken to be the X operator acting on all three qubits on the top, while the logical Z operator can be taken to be the Z operator acting on all three qubits on the left. Panel (b) shows the same stabilizers in our previous format. The rotation of the two versions of the surface code relative to each other is immediately apparent.

6.2.4 Reed-Muller and polar codes

A different way to construct a CSS code is to take a reversible encoding circuit composed of CNOT gates and specify certain inputs as the data qubits, and the remaining inputs as fixed in the $|0\rangle$ or $|+\rangle$ state. This construction relies on the fact that CSS codes have encoding circuits of this form. This fact was shown in §4.3.1, but can also be understood directly by propagating the stabilizer at the input (namely single-qubit Z and X operators at the locations of $|0\rangle$ and $|+\rangle$, respectively) through the encoding circuit. The CNOT gates will leave Z-type operators as Z-type operators, and the same for X, meaning the stabilizers will be of purely X- or Z-type.

The inputs $|0\rangle$ thus specify the Z-type stabilizers of a code C_Z and the $|+\rangle$ inputs the X-type stabilizers of a code C_X, and these satisfy the CSS condition by construction. The trick is to find an encoding circuit and a specification of $|0\rangle$ and $|+\rangle$ inputs which leads to a good quantum code.

One option is to try to ensure that C_X and C_Z are good classical codes. Then the decoders of these codes can be put to use correcting bit flips and phase flips. *Quantum Reed-Muller codes* and *quantum polar codes* are examples of this approach. In fact, the encoding circuit in both cases is the same; the only difference between the two kinds of codes is which inputs are set to $|0\rangle$ and which to $|+\rangle$. Either code has a size $n = 2^m$ for integer m. The circuit for $n = 8$, i.e. $m = 3$ has the form

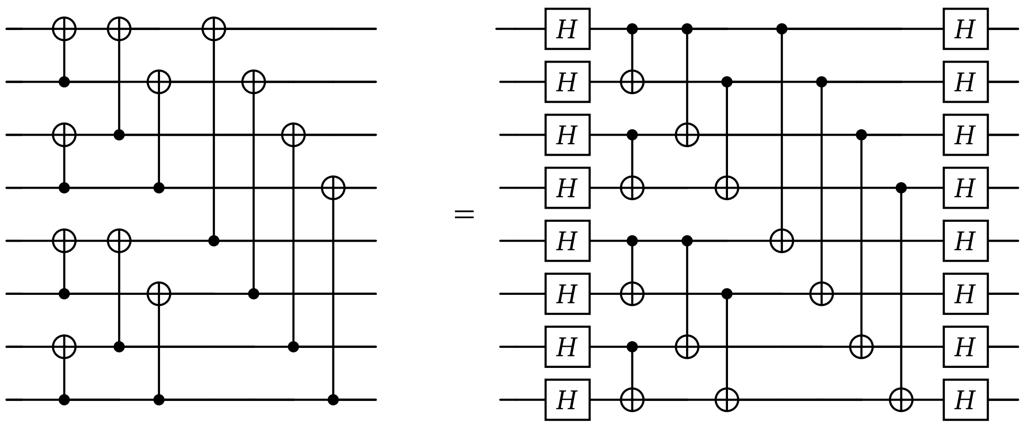


The inputs are labelled starting at zero on the left. The encoding circuit for the $n = 4$ code is shown in the dashed-line box. Observe that the $n = 8$ encoding circuit is simply two copies of the $n = 4$ circuit connected *transversally* by CNOT gates. Indeed the $n = 4$ circuit has this structure as well, starting from the base case $n = 2$ where the circuit is a single CNOT gate. Using this recursive structure defines the encoding circuit for arbitrary m.

The classical Reed-Muller codes choose data inputs and $|0\rangle$ inputs according to the Hamming weight of the binary representation of the input index. In particular, in the (r, m) Reed-Muller code, all inputs whose binary expansion has weight greater than or equal to $m - r$ are used as data inputs and the rest set to $|0\rangle$. Thus, the $(1, 3)$ code uses inputs 3, 5, 6, and 7 (i.e. 011_2, 101_2, 110_2, and 111_2) for data, and hence encodes four bits. The $(2, 3)$ is larger and fixes only the first bit, producing a code with a single parity check which includes all n bits. The $(0, m)$ codes are repetition codes in general, since only the last input is used for data and a bit flip in this input translates into a joint bit flip of all encoded bits, which the encoded X operator of the repetition code.

Polar codes make a different choice of data inputs, for reasons that we do not go into here. There are several variants of how to pick the data inputs precisely, and one that will be useful to us called the *polarization weight* method. Consider the function $f_\beta(i) = \sum_{j=1}^{m} b_j(i)\beta^j$, where $i \in \{0, \ldots, 2^m - 1\}$, $\beta \in \mathbb{R}$, and $b_1(i), b_2(i), \ldots, b_m(i)$ is the m-bit binary expansion of i. For a desired code size k and choice of β, the indices with the k largest values of $f_\beta(i)$ are used as data inputs. The typical choice of β, for reasons we won't go into here, is $2^{1/4}$.

To construct CSS codes from Reed-Muller or polar codes, notice an interesting property of the encoding circuit:



The Hadamard gates on the right just swap the order of control and target for each gate. Owing to the form of the original circuit, the circuit between the two sets of Hadamard gates has the same action as the original, just upside down. Note that the circuit itself is not precisely the upside-down version of the other, as the ordering of the CNOT gates in the second and third layers is not quite the same in both cases. However, since these gates (in the individual layers) act on different qubits, their ordering in the circuit does not affect the action of the circuit. The original circuit specifies rather directly what happens to Z-basis inputs. But due to the Hadamard gates, we can infer that the same action is applied to X-basis inputs, just with the qubits in the reverse order.

Now consider fixing some collection of inputs $i_1, i_2, \ldots, i_{m_z}$ to each start in the state $|0\rangle$ and some other collection of distinct inputs $i'_1, i'_2, \ldots, i'_{m_x}$ to each start in the state $|+\rangle$. For instance, suppose we pick the former set to be the $m - r_z$ indices with lowest binary weight and the latter to be the $m - r_x$ indices with the highest binary weight. As long as $m - r_z + m - r_x \leq m$, i.e. $r_x + r_z \geq m$ then the two sets will not overlap. The former set specifies the (r_z, m) Reed-Muller code for bit flip errors. Meanwhile the above property of the encoding circuit means the latter specifies the (r_x, m) Reed-Muller code for phase flip errors, since upon reversing the order of the inputs the highest weight indices become lowest weight indices. Together the two Reed-Muller codes form a CSS code. (Note that if $r_x + r_z = m$, then the codespace has dimension 1, so only encodes a single state.) This construction can be understood more directly from the properties of parity check matrices of Reed-Muller codes, but this approach also reveals the highly structured form of the encoding circuit.

The same construction also works for PW polar codes. The reason is that $f_\beta(i) + f_\beta(2^m - 1 - i)$ is constant for all i (namely $\sum_j \beta^j$); if $b_0, \ldots, b_m$ is the binary expansion of i, then the bitwise complement $\bar{b}_0, \ldots, \bar{b}_m$ is the binary expansion of $2^m - 1 - i$ (since then their sum is $2^m - 1$, whose binary expansion is m 1s). Indeed, the Reed-Muller example is just $\beta = 1$, where clearly the weight of an index i and that of its binary complement $2^m - 1 - i$ add up to m. Therefore, reversing the order of the inputs takes those with highest weight to those with lowest weight. This gives a quantum PW polar code.

6.3 Subsystem codes

A subsystem code is essentially a stabilizer code which nominally encodes multiple qubits, say k, but we only use $k' < k$ of the logical operators for data. Not encoding into the remainder can have some uses in simplifying syndrome measurement. We will not delve deeply into subsystem codes and instead focus on the example of the Bacon-Shor code.

6.3.1 Bacon-Shor codes

The Shor code is actually somewhat related to the surface code. Returning to the Shor code in Table 6.1, notice the symmetry between the stabilizers / destabilizers and the two logical operators; evidently we could have weight-two X-type checks and weight-six Z-type checks and essentially have the same code. That is, it's not the same code subspace, since the stabilizers are different, but it's clear that its logical operators and ability to correct single-qubit errors are both the same.

We can make things even more symmetrical between X and Z as follows. First, think of the qubits as living on a 3×3 lattice. We can arrange things so that the X stabilizers are pairs of rows. The Z checks are weight-two operators in a single row. But multiplying them appropriately, we can construct two weight-six stabilizers that are all Zs in adjacent columns. Thus, the Shor code is a

1	$Z\,Z\,I\,I\,I\,I\,I\,I\,I$	$I\,X\,X\,I\,I\,I\,I\,I\,I$
2	$I\,Z\,Z\,I\,I\,I\,I\,I\,I$	$X\,X\,I\,I\,I\,I\,I\,I\,I$
3	$I\,I\,I\,Z\,Z\,I\,I\,I\,I$	$I\,I\,I\,I\,X\,X\,I\,I\,I$
4	$I\,I\,I\,I\,Z\,Z\,I\,I\,I$	$I\,I\,I\,X\,X\,I\,I\,I\,I$
5	$I\,I\,I\,I\,I\,I\,Z\,Z\,I$	$I\,I\,I\,I\,I\,I\,I\,X\,X$
6	$I\,I\,I\,I\,I\,I\,I\,Z\,Z$	$I\,I\,I\,I\,I\,I\,X\,X\,I$
7	$I\,I\,I\,X\,X\,X\,X\,X\,X$	$Z\,Z\,Z\,Z\,Z\,Z\,I\,I\,I$
8	$X\,X\,X\,X\,X\,X\,I\,I\,I$	$I\,I\,I\,Z\,Z\,Z\,Z\,Z\,Z$
9	$Z\,Z\,Z\,Z\,Z\,Z\,Z\,Z\,Z$	$X\,X\,X\,X\,X\,X\,X\,X\,X$

Table 6.1: Stabilizers (left), destabilizers (right), and logical operators (bottom) associated with the Shor code.

kind of combination of two classical repetition codes. Furthermore, since the two X checks in the rows are sufficient to recover from phase errors, it stands to reason that the two Z checks in the columns are sufficient to recover from bit errors; no other stabilizers are really needed. Except that that analysis isn't quite right, since the weight-two Z checks ensure enough degeneracy of phase errors so that the two weight-six X checks are sufficient for recovery.

The interesting thing noticed by Bacon is that we don't need to fix the Z stabilizers for the degeneracy argument to go through. The weight-six stabilizers are consistent with (i.e. commute with) both weight-two Z-type checks running horizontally and weight-two X-type checks running vertically, though these are not consistent with each other. So if we had started with the coding having weight-two X checks, then clearly the two weight-six Z stabilizers would be sufficient.

Recall that phase error correction is just done by block of three qubits in a row: From the syndrome we compute which row has a phase error and correct it by applying Z to the first qubit in the row. Let's just try the same thing for bit flip errors, but on the columns instead of rows. Suppose we start with a state encoded in the usual Shor code, and a bit flip afflicts the fifth qubit. This is noticed by the both of the Z-type stabilizers, so we conclude the error is in the second column and we apply the correction X_2. Overall, the error and correction procedure applies X_2X_5 to the state. This commutes with the logical operators, and the weight-six stabilizers are all $+1$, so in this sense error correction succeeded.

However, we're outside the original codespace, since this operator does not commute with all the weight-two stabilizers; four of them have flipped. We could fix these incorrect stabilizers by applying the corresponding destabilizers (in this case, that would imply multiplying by $X_1X_3X_4X_6$, the product of the first four destabilizers). But there's really no reason to. For one thing, flipping the signs of the stabilizers results in an equally-good error-correcting code. However, we might need to know which codespace we're in to perform the correct decoding procedure. But in this case the modified decoding procedure is the same in both cases.

In terms of the stabilizer tableau in Table 6.2, what happens is that the error and recovery cause changes to qubits 5 through 8, i.e. the new encoded qubits that are created by reducing the size of the stabilizer. But as long as the action is confined to these qubits, it doesn't affect subsequent correctability since they are not needed for the decoder. In this case X_2X_5 is the product of the X-type operators of qubits five through eight, as well as the second X-type weight-six stabilizer.

These qubits are called "gauge qubits" since we can "fix" the weight-two Z operators to $+1$ to get the standard Shor code, or we can fix the weight-two X operators to get the X version of the Shor code, or we can just let them "float". One aspect that is quite convenient is that we can determine the important weight-six stabilizers by measuring all of the weight-two stabilizers of both types. They do not all commute with each other, but they do allow reconstruction of the weight-six stabilizers. Measuring the Z type first and then the X type converts the code to X gauge, while the reverse order converts it to Z gauge. Other intermediate gauges are also possible.

1	$ZZIZZIZZI$	$IIIIIIIXX$
2	$IZZIZZIZZ$	$IIIIIIXXI$
3	$IIIXXXXXX$	$ZZZZZZIII$
4	$XXXXXXIII$	$IIIZZZZZZ$
5	$ZZIIIIIII$	$IXXIIIIXX$
6	$IZZIIIIII$	$XXIIIIXXI$
7	$IIIZZIIII$	$IIIIXXIXX$
8	$IIIIZZIII$	$IIIXXIXXI$
9	$ZZZZZZZZZ$	$XXXXXXXXX$

Table 6.2: Stabilizer tableau associated with the nine-qubit Bacon-Shor code.

This is a *subsystem code*, where the quantum information is not stored in a subspace *per se*, but rather a subsystem within this subspace. In terms of the overall 9-qubit state space, we have a decomposition into $C^c \oplus (L \otimes E)$, where L stands for the logical qubit, E for the extra four qubits, C the codespace, and C^c its complement. For stabilizer codes as here, L is composed of the original logical operators, and E are the additional operators which are promoted from the original stabilizer. C is the subspace picked out by the new (smaller) stabilizer. Together, the operators in S, the new stabilizer, and E generate the gauge group. As long as the error correcting condition is fulfilled on L, then correctability is still assured; that is, the environment can learn as much as it wants about E, but nothing about L. And correctability on just L means we only need to use S for correction, so E is really irrelevant. It should be mentioned that the error-correcting properties of stabilizer subsystem codes are a bit more immediate than general subsystem codes, but we won't go into those here.

It is not difficult to see that we can extend the Bacon-Shor construction to an arbitrary $n \times m$ lattice, as in the surface code. The stabilizers are products of all Z operators in adjacent columns and all X operators in adjacent rows. The standard decoder is just repetition coding for each case, with correction operations applied at a single position in the given column or row, respectively.

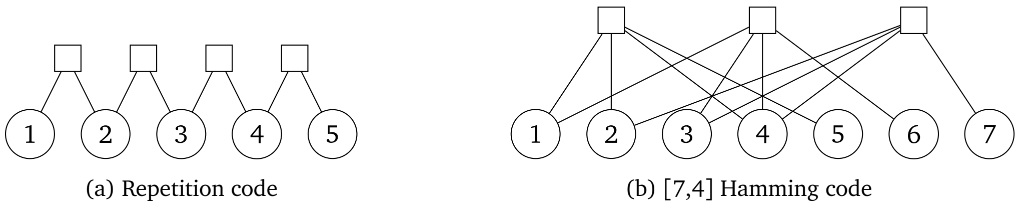
7 Decoding by perfect matching

Thus far we have no specific decoder designed for the surface code. Since it is a CSS code, we may correct X- and Z-type errors separately, using the syndrome from the Z-type checks for the former and the syndrome from the X-type checks for the latter. The simple structure of these syndromes enables the problem of finding the minimum-weight X or Z error to be cast as the problem of finding a *minimum-weight perfect matching* on a particular graph. Fortunately there is a polynomial-time algorithm to solve this problem, and therefore we can construct a reasonably-efficient decoder.

The intuition behind the decoder can already be appreciated from Figures 5.3 and 5.4: The possible error patterns form paths which join the nontrivial syndromes in pairs. Therefore the decoder simply needs to decide how to pair up the syndromes, and the most immediate approach is to find the pairing which has the error of minimal weight.

7.1 Tanner graphs

The matching decoder works on a graphical representation of the code, so let us begin by examining the *Tanner graph* of a code. The Tanner graph is a bipartite graph representing the parity check constraints of the code. There are two sets of vertices or nodes in the graph: variable nodes and check nodes. Each variable node is associated with a position in which a single error can occur, i.e. a bit or qubit position. Each check node is associated with one of the parity checks of the code. A given variable node is connected to a given check node if that variable participates in that check. Figure 7.1 depicts some examples. Note that we can also define Tanner graphs for stabilizer codes, e.g. the five-qubit code as well, and in that case are invited to consider three different kinds of edges corresponding to whether X, Y, or Z of a particular qubit participates in a particular check. However, we will not make much use of this kind of Tanner graph here.



(a) Repetition code

(b) [7,4] Hamming code

Figure 7.1: Examples of Tanner graphs

7.2 Cycle codes and almost cycle codes

7.2.1 Cycle codes and their correction by matching

Now consider a *cycle code*, a binary linear code that has a parity check matrix such that each column has precisely two 1s. In terms of the Tanner graph, every variable node has degree two. Columns correspond to individual bits (or qubits) and rows to individual stabilizers, so an error on a single bit flips precisely two syndrome values. Further errors will only create new pairs of errors or move parts of existing pairs around. Hence a valid correction which returns the system to the codespace must be a collection of individual errors which join the nontrivial syndrome positions in pairs. Two nontrivial syndrome positions are joined by a “path” of errors, where the ends of the path are responsible for the nontrivial syndrome values themselves, and the errors in the interior join the two ends with trivial syndromes along the way.

7.2.2 Almost cycle codes

The X and Z parity check matrices associated with the surface code, as well as that of the repetition code, nearly describe cycle codes. In both cases there are weight-one columns in the check matrices, i.e. single errors at certain positions which affect only one stabilizer. For both codes these positions are on the boundaries: Variable nodes of degree one. Weight-one checks complicate the pairing or matching strategy just described because now there can be an odd number of nontrivial syndrome values.

However, for any $m \times n$ parity check matrix H having weight-one and weight-two columns, we can construct an $(m+1) \times n$ check matrix H' which describes the same code and has only weight-two columns. Simply append an additional row to H which is the sum modulo 2 of its rows. The summation in the weight-two columns will result in zero, leaving only 1s in the weight-one columns. The resulting H' describes the same code as H since the last parity check is not linearly independent of the rest. To generate the syndrome s' for H' given the syndrome s for H, simply append the parity of s.

If s has even parity, then the additional entry in s' is 0. In this case the nontrivial syndromes in H can be paired without involving the additional check, but pairings are also possible which do involve the additional check. This can be seen in the green paths of Figure 5.3. The additional check is connected to every qubit on both the left and right boundaries, and so the green path can be understood as a pairing of two nontrivial syndromes via the boundary. On the other hand, if s has odd parity, the additional entry in s' is 1. Now one of the nontrivial syndromes of H can be and indeed must be paired with the new syndrome. This is the situation in Figure 5.4, where one of the nontrivial syndromes must be matched to the boundary.

7.2.3 Matching graphs

Since all variable nodes in a Tanner graph of a cycle code have degree two, we may as well just represent them as edges between the corresponding check nodes. Call the resulting graph the *matching graph* (though *syndrome graph* might be a better name). It can happen that the additional check can be connected to a boundary check by multiple edges, i.e. the matrix H' can very well have several identical columns. For the purposes of finding the minimum-weight error, we can simply drop any linearly dependent columns from H'. Figure 7.2 depicts the matching graphs for the repetition code and the Z-type checks of the 5×5 surface code.

7.3 Matching

7.3.1 Repetition code

We can get some further intution on the matching problem by examining the n-bit repetition code with parity checks Z_iZ_{i+1} for $n = 1, \dots, n-1$. The additional check is essentially the parity check Z_nZ_1, whose value is then set to be the parity of the syndrome s. A nontrivial syndrome for Z_iZ_{i+1} means the bits in positions i and $i+1$ take different values. Put differently, if Z_iZ_{i+1} and Z_jZ_{j+1} are nontrivial, but all Z_kZ_{k+1} for $i < k < j$ are trivial, then all the bits from $i+1$ to j have to have the same value. The question is only whether these two nontrivial syndromes should be paired

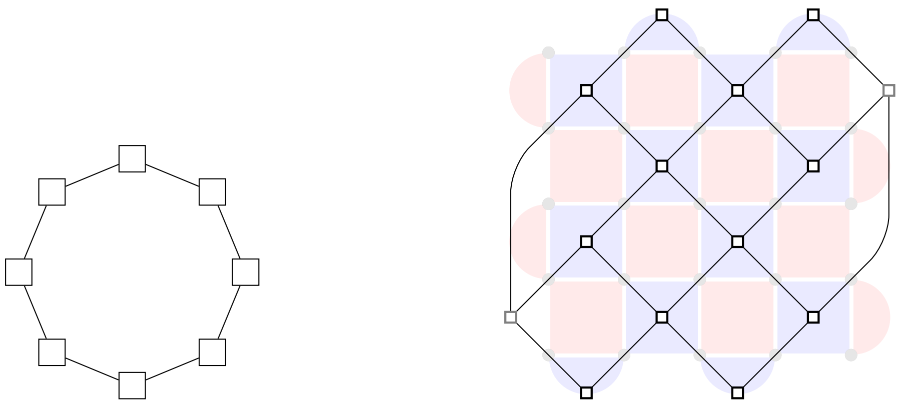

(a) Repetition code of length 8 (b) Z-type checks of the 5×5 surface code

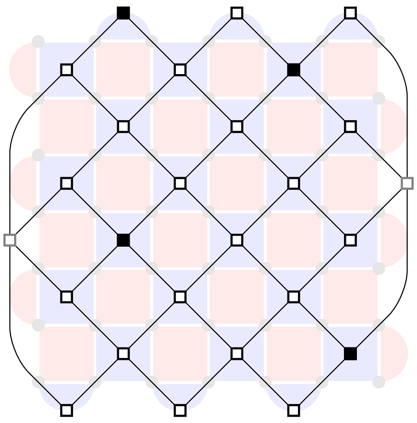
Figure 7.2: Matching graphs. In the surface code the two gray checks in the lower left and upper right are to be identified: There is only one extra check of degree 6 due to the boundary qubits. Extraneous edges have been removed.

together, which corresponds to deciding that the bits from $i+1$ to j all take the value 1, or they should each be matched to other nontrivial syndromes, meaning the bits in this range all take the value 0. We can think of the nontrivial syndromes as "domain walls" between adjacent bits.

Finding the lowest-weight error pattern is greatly simplified in this case, as there are only two possible pairings of domain walls. The problem is inherently one-dimensional: either a given domain wall pairs with the domain wall to its left or to its right (possibly cycling around the code using the additional check in either case). The two possible pairings are binary complements of each other. To find the lowest-weight error pattern, we may therefore just check the weight of one of the solutions. If it is $n/2$ or less, we take it as the correction, otherwise we take its complement. Hence the decoding can be performed in $O(n)$ steps.

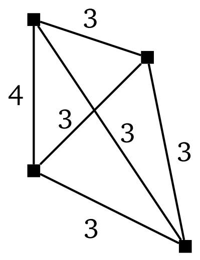
7.3.2 General cycle codes

Pairing nontrivial syndromes such that their joining error paths have minimal weight is much less straightforward in the general case. Consider a particular syndrome pattern, say for the 7×7 surface code shown below. The black squares represent nontrivial syndrome values.



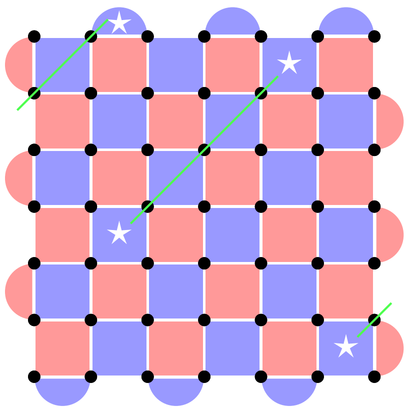
We now create the *path graph* to perform the matching. The path graph is the complete graph whose vertices correspond to nontrivial syndromes in the matching graph. Moreover, the edges of the path graph are labelled by the smallest distance between the associated syndromes in the matching graph, as given by the number of edges that are traversed. This equates to the number of qubits which need to be flipped to generate those two nontrivial syndromes. The value of each edge in the path graph needs to be computed from the matching graph. For general matching graphs this can be accomplished with the Floyd-Warshall algorithm. The lattice structure of the surface code enables more direct calculation. There are two cases of paths to consider: paths joining a pair of nontrivial syndromes which go directly through the "bulk" of the latticework, and those that go via the boundary. Both can be computed directly from the position information of the pair of syndromes.

The path graph for the example is then (scaled down, but otherwise in the same orientation)



Now we have transformed the problem such that the solution is given by finding a minimum-weight perfect matching. A perfect matching is a pairing of vertices in the path graph. It has minimum weight if the sum of the values of the edges in the matching is minimal among all matchings. The "blossom" algorithm can be employed for this purpose. In the example a minimum weight perfect matching is to pair diagonally, leading to a value of 6. (Note that it is not unique; matching the top and bottom edges also gives a weight of 6.)

Given the minimum-weight perfect matching in the path graph, the only thing left to do is translate this back into a path of errors on the matching graph. In the running example we thus obtain



7.3.3 Performance of matching on the surface code

For the 3×3 surface code subjected to depolarizing noise, decoding X and Z errors independently via matching is almost as accurate as MLE, as depicted in Figure 7.3. For large surface codes, we can estimate the logical error rate under the matching decoder by randomly sampling errors according to the error model. Doing so reveals a *threshold* in the noise rate. For noise rates below the threshold, larger and larger codes will have smaller and smaller logical error rates. Above the threshold the logical error rates get larger and larger. The value of matching decoder threshold is about 10% for the (rotated) surface code subject to independent bit and phase errors.

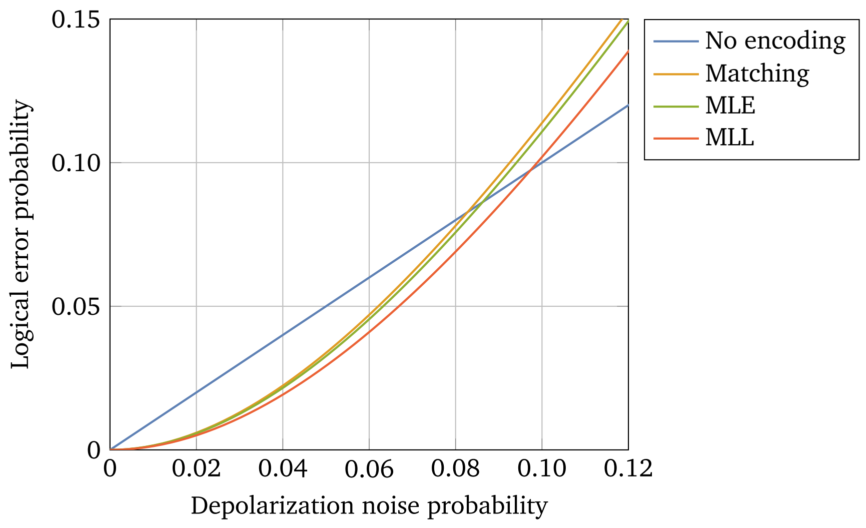


Figure 7.3: Logical error rate of individual X- and Z-type matching decoders versus MLE and MLL for the 3×3 surface code subject to depolarizing noise.

Note that the bounded distance decoder has a threshold of zero since it refuses to correct errors of weight beyond half the distance of the code. For the $d \times d$ surface code the distance is d, meaning only errors of weight roughly $\sqrt{n}$ for $n = d^2$ will be corrected. But having a threshold value of p means being able to decode at least a sizeable fraction of the errors to be expected at this noise rate, which have weight $\approx np$. In this sense, distance is not a very useful indicator of the performance of a code.

8 Tensor network decoding methods

Tensor networks are an elegant way to represent functions which decompose into sums of products of simpler functions. Such functions arise in various decoding algorithms, where the tensor network representation suggests approximation methods of calculating these quantities so as to have efficient but also accurate means of decoding.

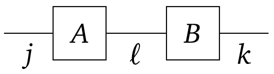
8.1 Tensor networks

Consider a function $f(x_1,\dots,x_n)$ which can be written as a sum of products of functions involving additional variables $y_1,\dots,y_m$, e.g. something of the form

$$f(x_1,x_2,x_3) = \sum_{y_1,y_2} h_1(x_1,x_2,y_1)h_2(y_1,y_2,x_3)h_3(x_2,y_2)\,. \tag{8.1}$$

For instance, the components of the matrix M which is a product of some matrices A and B is simply $M(j,k) = \sum_\ell A(j,\ell)B(\ell,k)$.

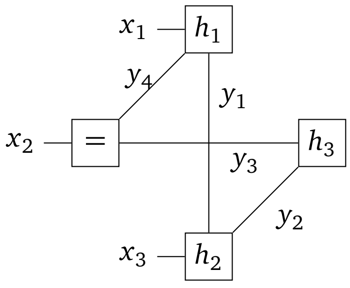
A tensor network, for our purposes, is a graph which represents this structure. Each vertex corresponds to a factor or tensor h_i in the decomposition. Here "tensor" has no geometrical meaning, as it does in general relativity. For us it is just a function or equivalently a collection of function values for each possible input. Edges and half-edges correspond to variables which are arguments to the tensors. Edges between two nodes respresent the internal or bound variables $y_1,\dots,y_m$ of the summation, while half-edges connected to single nodes correspond to the free variables $x_1,\dots,x_n$ which are arguments of f. The tensor network associated with matrix multiplication is therefore just



Nominally, no free variable can be connected to more than one node and no bound variable can be connected to more than two nodes. For example, in the choice of f above, the free variable x_2 appears in two factors. This issue can be resolved by including additional internal variables and δ tensors:

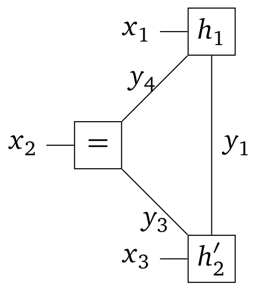
$$f(x_1,x_2,x_3) = \sum_{y_1,y_2,y_3,y_4} h_1(x_1,y_4,y_1)h_2(y_1,y_2,x_3)h_3(y_3,y_2)\delta(x_2,y_3,y_4)\,. \tag{8.2}$$

The tensor network corresponding to the righthand side of this equation is then



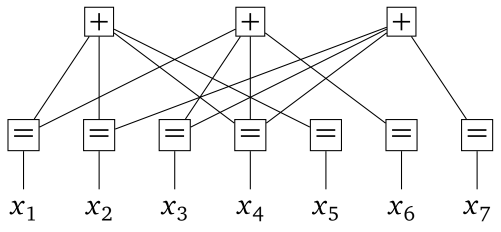
Here we denote the δ function by the tensor labelled '=', as this makes clear that all attached edges must take the same value.

Suppose now we define the function $h_2'(y_1, x_3, y_3) = \sum_{y_2} h_2(y_1, y_2, x_3) h_3(y_3, y_2)$. The tensor network representation now looks like



We have *contracted* h_2 and h_3 to form a new tensor, and as a result the bound variable y_2 no longer appears. We can think of drawing a box around h_2 and h_3 in the original network; h_2' is essentially this box.

We will be mostly interested in the case of binary variables. A useful tensor in this context is the parity tensor, labelled by '+', which is just the indicator function for even parity of its arguments. For instance, we can represent the indicator function of a binary linear code by interpreting the Tanner graph as a tensor network. Recall the Tanner graph of the [7,4] Hamming code in Figure 7.1(b). Regarding the check nodes as parity tensors, the variable nodes as equality tensors, and adding additional open edges for the variables gives



Algebraically, this tensor network represents the function $\delta(x_1 + x_2 + x_4 + x_5 = 0)\delta(x_1 + x_3 + x_4 + x_6 = 0)\delta(x_2 + x_3 + x_4 + x_7 = 0)$, which can be written in sum of product form as

$$\begin{aligned} f(x_1, \dots, x_7) = \sum_{w,y,z} & p(y_1, y_2, y_4, y_5) p(z_1, y_3, z_4, y_6) p(z_2, z_3, w_4, y_7) \\ & \times \delta(x_1, y_1, z_1)\delta(x_2, y_2, z_2)\delta(x_3, y_3, z_3)\delta(x_4, y_4, z_4, w_4) \\ & \times \delta(x_5, y_5)\delta(x_6, y_6)\delta(x_7, y_7). \end{aligned} \tag{8.3}$$

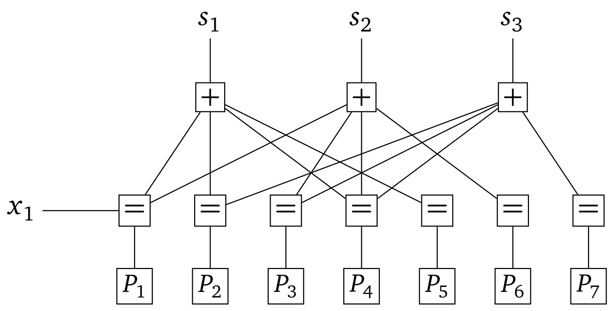
Here $p()$ denotes the parity function. The last three δ functions are trivial, but they maintain the structure of the Tanner graph.

8.2 Qubitwise decoding

8.2.1 Tensor network representation

The most common approach to decoding classical LDPC codes is to decode bitwise. Instead of looking for the most likely error pattern given the syndrome, the bitwise decoder looks for the most likely value of each individual bit of the error pattern. Suppose P_E is the joint distribution of the entire error pattern $x_1, \dots, x_n$ and $P_E^{(i)}$ is the marginal probability $P_E^{(i)}[x_i] = \sum_{\sim i} P_E[x_1, \dots x_n]$. The bitwise MLE decoder then computes $\hat{x}_i = \underset{x\in\mathbb{F}_2^n:f(x)=s}{\arg\max}\; P_E^{(i)}[x_i]$ for each i and outputs the correction operation $\hat{x} = \hat{x}_1,\dots\hat{x}_n$. We simply have to hope that $\hat{x}$ satisfies the syndrome, i.e. $f(\hat{x}) = s$, otherwise we will get a decoding error.

The conditional distribution of x_i given the syndrome s is simply $\sum_{\sim i} P_E[x_1,\dots,x_n]\delta(f(x_1,\dots,x_n),s)$, where f is the function $f(x) = Hx$ for H the parity check matrix of the code. For an independent error model in which $P_E = P_1 \times P_2 \times \cdots \times P_n$, this expression factorizes even further. We have seen above how to respresent $\delta(f(x),0)$ as a tensor network, and so all that is needed is to include P_E and shift the value of the syndrome from 0 to an arbirary value s. For instance, the conditional distribution of x_1 for the [7,4] Hamming code example is represented by



An i.i.d. error model would have $P_1 = P_2$ and so on, but in the tensor network depiction it is clear that indpendence is the important thing, not that the marginals are all identical.

8.2.2 Belief propagation

Now we need a means of actually computing the two values of the conditional distribution, not just representing the function itself. Suppose that the Tanner graph were a *tree*, meaning a graph with no loops. For a fixed value of the syndrome, we can include the s_i into their corresponding parity check tensors, and the resulting tensor network will also be a tree. The equality node connected to x_1 in the example can be considered the *root* node of the tree. Since there are no loops, there is only one path from the root to any other node. In particular, the P_i nodes are *leaves* of the tree, since they have only one edge. (For a nontrivial code there should be no other leaves; single edge check nodes are trivial checks which just fix the value of a particular bit.)

The tree structure gives an efficient means of computing the values of the conditional distribution by recursively contracting the tensor network from the leaves towards the root. That is, for each neighboring node of the leaf tensors we contract it with its neighboring leaves. Due to the tree structure this contraction creates a new leaf node, and then we repeat the procedure. Each such leaf tensor created has a single binary-valued edge going towards the root, and since we start with P_i tensors which represent probability distributions of individual bits, the leaf tensors also represent probability distributions of individual bits. These are the “beliefs” about the value of the edges, and the contraction of the tensor network amounts to “propagation” of these beliefs towards the root. This is the belief propagation (BP) algorithm. We can regard the probability functions as “messages” which are “passed” along edges from the leaves to the root, and are combined with other messages at the nodes. Hence BP is a “message passing” algorithm.

Because the Tanner graph has only P_i tensors, parity or check tensors, and equality tensors, there are really only two contraction rules one needs to consider. Either leaf tensors (P_i type tensors) are contracted at an equality node or at a check node. The base case is two leaf tensors, as more than two can be contracted successively onto the same node, and check or variable nodes of degree two can simply be removed from the graph. We are not overly concerned with the precise form of the rules, but they are

$$P_+(x) = \sum_{y,z} P_1(y)P_2(z)\delta(x+y+z,0)\,, \tag{8.4}$$

$$P_=(x) = \sum_{y,z} P_1(y)P_2(z)\delta(x,y)\delta(x,z)\,. \tag{8.5}$$

Note that $P_=$ is not a normalized probability distribution, which is as it should be, since we were not careful with the normalization of the conditional marginal distribution. Nevertheless, we can think of the edges as carrying probability messages, flowing from the leaves to the root and combining and check and variable nodes.

Now, as depicted in the example, there are certainly codes for which we cannot find a Tanner graph which is a tree (the Tanner graph depends on the particular parity checks chosen, so it is not unique for a given code). Nevertheless, we may still apply the contraction rules, or message combining rules, and hope for the best. We only need to shift perspective on how BP operates slightly. First, to simplify matters, we can contract the channel nodes P_j with their neighboring equality nodes. Second, consider the direction of "flow" of the messages in BP. When estimating a particular bit in a tree graph, the edges are traversed in only one direction. But the direction will change depending on the root node. So in general messages are flowing in both directions at different points in the algorithm. We can set a simple schedule for generating the messages in which the messages are sent from equality nodes (now combined with their associated channel nodes) to the check nodes and then the messages are sent back from check nodes to equality nodes. This is known as the parallel or flooding schedule. In each odd-numbered (even-numbered) round, for each edge we look at the connected equality (check) node, and then use (8.4) and (8.5) to generate the message from the messages on other connected edges of the equality (check) node. To get things started, we set the messages to just come from the channel nodes directly.

In classical LDPC coding, it is possible to construct high-rate codes such that the loops in the Tanner graph are fairly large. Hence it is locally tree-like, and BP is a reasonable approximation. In practice BP is a widely-used decoder, and LPDC codes can currently be found in the 5G specification for mobile communication as well as the Wi-Fi 6 specification for wireless networking. However, quantum LDPC codes are likely to have short loops, as in the surface code.

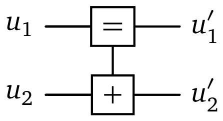
BP can in principle be used to decode quantum codes. We have discussed the binary version of BP above, which is suitable for binary classical codes, but one can also formulate a 4-valued version to directly handle the case of Pauli errors. Alternately, one may use the binary BP to decode X and Z errors separately in a CSS code. The difficulty is that typically the BP output does not satisfy the observed syndrome, so it does not give a useful correction operation.

However, since BP computes the marginal probabilities, at least approximately, these probabilities can be used as inputs to another decoder. An example is matching. In our discussion of matching we focused on finding the minimum-weight error, and thus implicitly set every edge weight in the matching graph to 1. We could instead set it so that if an qubit is thought to be more likely in error, the weight of the edge is smaller. A typical choice is to set the edge weight to $\log \frac{p}{1-p}$ when the probability of error for the associated qubit is p.

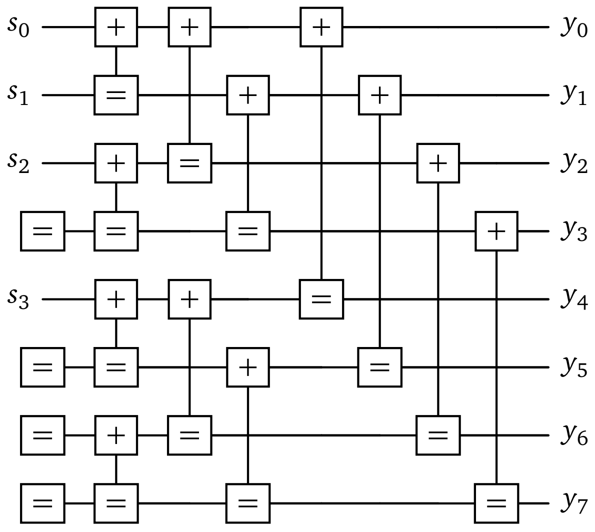
8.2.3 Successive cancellation

Another way to generate a tensor network representation of the projection onto the codespace of a classical code as in (8.3) is to use a tensor network formed from a reversible encoding circuit. Consider a circuit on n bits composed entirely of CNOT gates. Certain of the inputs to the circuit carry data to be encoded, and the remaining inputs are to be initialized as 0. If the $n \times n$ matrix U describes the reversible encoding circuit and the last m inputs are the ancilla inputs, then the projection onto the codespace is simply the function $y^n = (y_0, \dots, y_{n-1}) \mapsto \sum_{x_0,\dots,x_{k-1}} \delta(y^n, U(x^k, 0^m))$. The projection onto vectors which have syndrome s can be obtained by replacing 0^m with s in this expression.

Now observe that we can reinterpret the circuit diagram as a tensor network by replacing each CNOT gate with a parity node at the not operation and an equality node at the check:

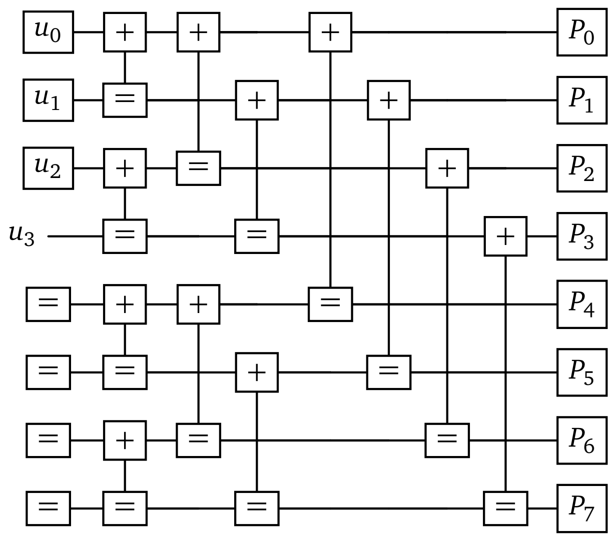


This tensor network captures the action of the CNOT gate, since $(u_1', u_2') = (u_1, u_1 + u_2)$, where the u_i and u_i' are all in $\mathbb{F}_2$. To obtain the tensor network representation of the projection to a given syndrome, we simply remove the x_1 inputs in the network so that they are summed over. Suppose that $k = 4$ and x_0, x_1, x_2, and x_3 are in positions 3, 5, 6, and 7 (as in the Reed-Muller code). For a syndrome s_0, s_1, s_2, s_3, the following tensor network is the projection onto the set of vectors y^n which have the syndrome s:



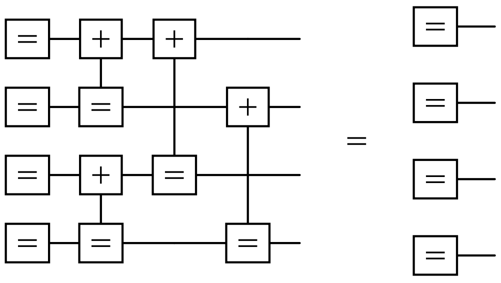
Just as we did in belief propagation, we can now try to compute the probability of a single variable on the input side using the probability distribution of errors on the righthand side. Given an estimate for the variables on the left, we can recover the error pattern by propagating their values through the circuit / tensor network. Some of the variables on the input side are syndromes, of course, so their values are known. Differently to what we considered in BP, we may compute the probability of input u_i given the values for $u_0, \dots, u_{i-1}$, marginalizing over all $u_{i+1}, \dots, u_{n-1}$. So for instance, the following tensor network describes the probability of u_3 given $(u_0, u_1, u_2) = (s_0, s_1, s_2)$

and marginalizing over $u_4, \ldots, u_7$:

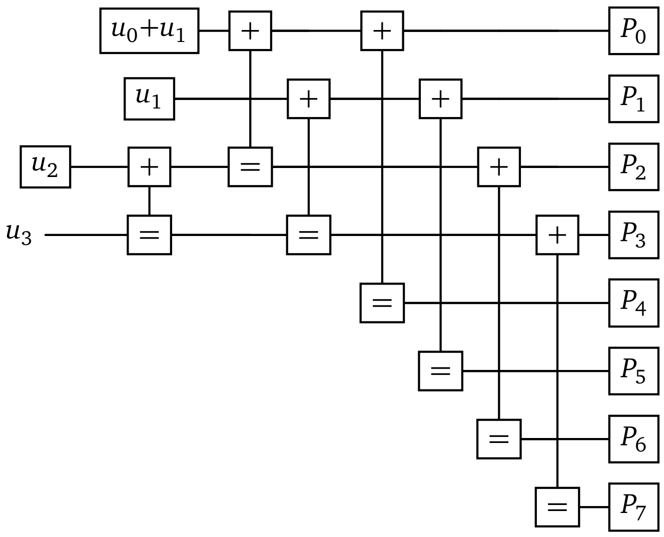


Here the P_i tensors are the distributions of individual qubit errors, and the u_i tensors are delta functions specifying that the outgoing edge has the value u_i. Equality tensors with a single edge just represent the constant function 1 for all inputs, so that both values of the associated edge are possible and receive the same weight.

Contraction of the resulting tensor network can be done efficiently by making some simplifications and choosing a suitable contraction order. First observe that the first two layers in the final four inputs (lines 4, 5, 6, and 7) just produce four single-edge equality nodes:

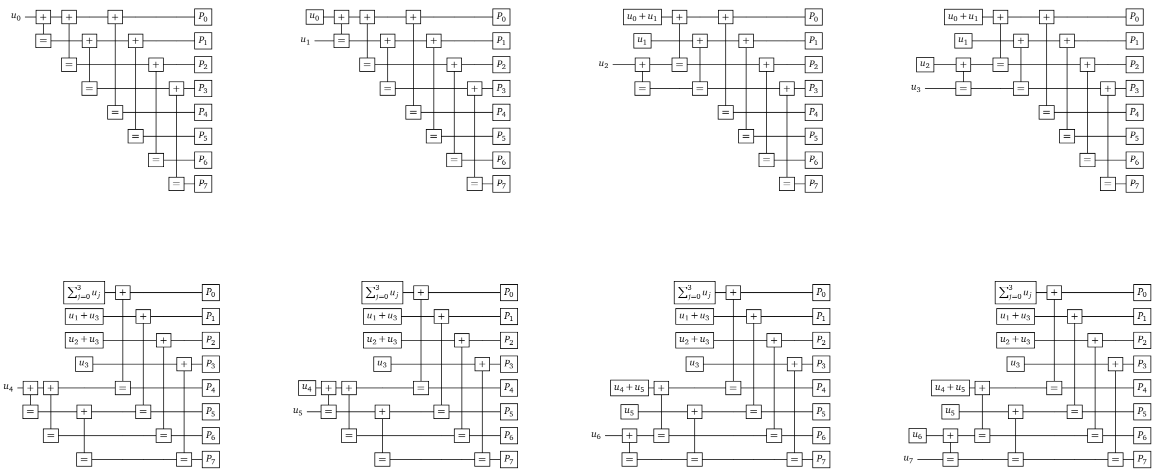


Next, for the fixed input values we can just propagate their values as far to the right as possible, through parity tensors if all but one edge is deterministic and equality tensors as long as one edge is deterministic. These two simplifications produce the following tensor network:



It can be verified that the graph is a tree, and therefore we may apply the BP rules (8.4) and (8.5) to contract the network to find the more likely value of u_3. In fact, the calculation for every u_i reduces to a tree. Performing this calculation for every unknown value of u_i gives an estimate for u^n, which can then be mapped to an estimate for the bit error pattern by the original circuit. Nominally this sequential procedure would require order n operations for each bit, for an overall complexity of n^2. But by recycling intermediate calculations, the computational cost can be reduced to $O(n \log n)$.

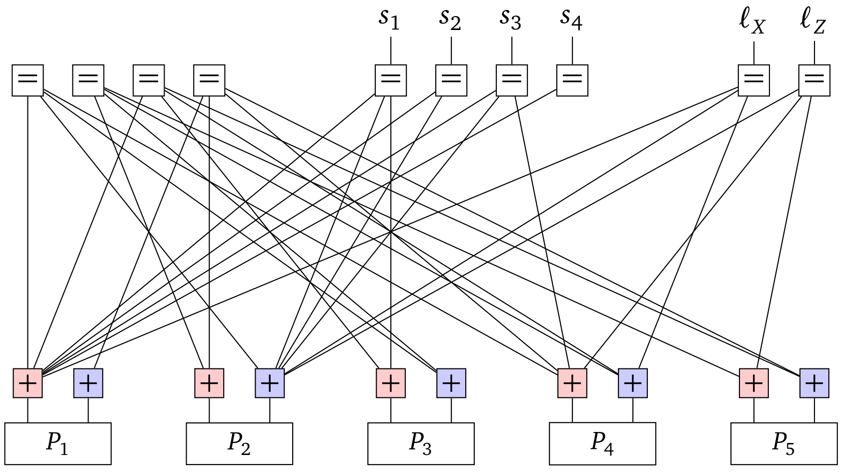
Here are all the tensor networks for the $n = 8$ case:



8.3 Approximating MLL decoding of the surface code

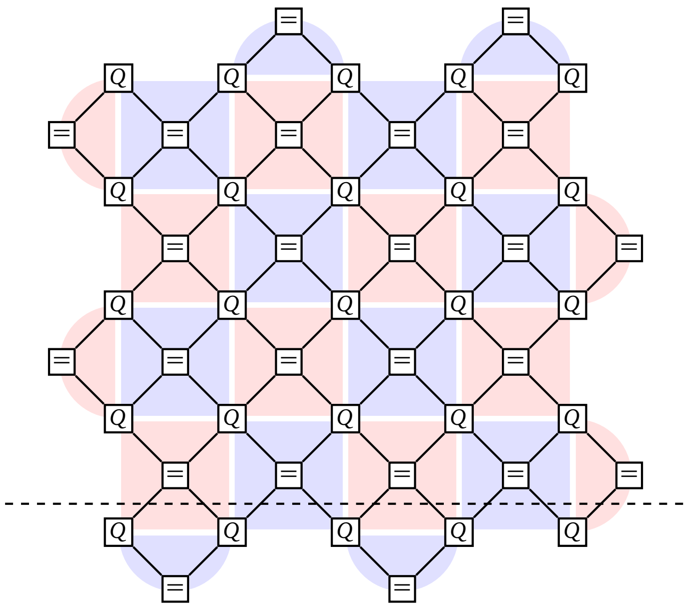
The quantity $\sum_{\xi \in S} P_E[\xi \eta(s) \tau]$ in (5.3) can be expressed as a tensor network for any stabilizer code. The idea is that argument to the summation can be written as a product of the tensor (or tensors) describing P_E and tensors describing the Pauli operator $\xi \eta(s) \tau$. When the code is such that the qubits and stabilizers have a planar layout, as in the surface code, the resulting tensor network can be approximately contracted by an efficient scheme based on “matrix-product states” (MPS).

Let us first see how to realize the value of $\sum_{\xi\in S} P_E[\xi\eta(s)\tau]$ as a tensor network. We work in the vector representation of the Pauli operators, and choose a stabilizer tableau with stabilizer generators given by the stabilizers actually measured in the error-correction procedure, along with associated destabilizers and logicals. This way, there is a single destabilizer operator for each component of the syndrome s. Since we will be multiplying the Pauli operators, which in the vector representation is addition over $\mathbb{F}_2^{2n}$, we start with $2n$ parity tensors to perform the addition. Each one has an "outgoing" edge and several "incoming edges". Connected to the parity tensors are tensors which determine the ξ, $\eta(s)$, and τ contributions. For each stabilizer generator, we associate an equality tensor. It is connected to the incoming edges of the appropriate parity tensors as determined by the vector representation of the stabilizer. Similarly, for each destabilizer generator we associate an equality tensor, again connected to the appropriate parity tensors. The destabilizer tensors have an additional edge whose value is s_j for the jth destabilizer. Equality tensors are also associated to the logical generators and have an external edge to specify the precise logical operator. Finally, the outgoing edges of the parity tensors are connected to the P_E tensors. Let us suppose that P_E is an i.i.d. model for each qubit, but that X and Z errors may be correlated. Therefore P_1 has two edges, one for the X component and one for the Z component. It is more convenient to reorder the parity tensors so that X and Z components of individual qubits are next to each other. Then we obtain a diagram like the following, for the five-qubit code (using Table 3.2).



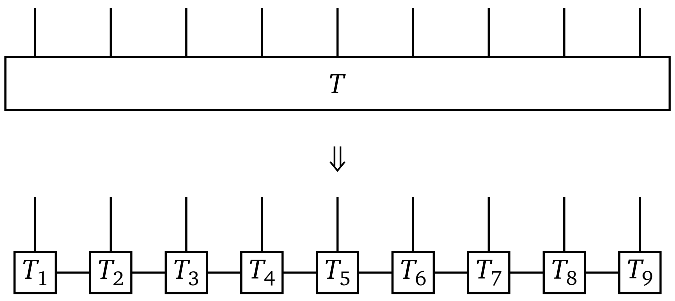
Since the syndrome s_1, s_2, s_3, s_4 and logical operator, specified by ℓ_X and ℓ_Z, are connected to equality nodes, we can incorporate their values into the corresponding parity tensors directly and remove the associated edges. This leaves just the edges corresponding to the stabilizer. The tensor network can be contracted to find the probability of the equivalence class of errors, but there is no real advantage in doing this calculation on the tensor network over a more direct algebraic approach.

For the surface code, however, the tensor network approach is advantageous. In particular, it suggests a certain approximation scheme which seems to work well in practice. Suppose, as just mentioned, we fix the value of the syndrome and the particular logical operator and then locally contract the probability tensor P_i with the two associated (now modified) parity tensors. Call the resulting tensor Q_i. For the surface code, the result is the following



Each stabilizer equality node connects to the X or Z parity node of surrounding Q nodes depending on its type, indicated by the background color. The Q tensors are not all identical; each one contains the contribution of the probability tensor and the specific destabilizer and logical values.

The 2D lattice arrangement allows for the following approximate contraction scheme. Consider all the tensors below the dotted line in the diagram. These can be contracted together to form a single tensor with 9 outgoing edges. Suppose this degree-9 tensor, call it T, can be split up into a 1D sequence of 9 tensors like so:



This kind of 1D tensor network is called a “matrix-product state” (MPS, for its use in representing pure quantum many-body states) or “tensor train”. If we can manage this conversion, then we can contract the next layer of tensors onto it and repeat the splitting process. The trick is to make the MPS in such a way that the horizontal edges do not have a very large size. This size is the “bond dimension” of the MPS.

To create the MPS, we can make use of the singular-value decomposition of matrices. Consider the leftmost edge in T and the remaining 8 edges on the right. Considering these 8 edges as defining a compound index, T then becomes a matrix. Applying the singular value decomposition will split the matrix T into a product of three matrices USV such that S is diagonal and positive. If we now keep only the D largest entries in S and replace the remainder with zero, we can define T_1 from US and the remainder of T from V. Repeating the process on V will then generate the MPS with fixed bond dimension D.

Somewhat amazingly, the MPS approximation to the MLL decoder for the surface code is already extremely good even for $D = 4$. The resulting decoder has a threshold for independent bit and phase errors of about 11%. This means that for noise rates above 10% the logical error rate of the matching decoders is getting worse and worse as the code size increases, whereas the logical error rate of the MPS tensor network decoder is getting better and better. This is a striking example of the importance of degeneracy in decoding (though one might well argue that the difference is not that important in practice).

Framework of fault-tolerance 9

9.1 Setup and basic idea

Up till now we have been considering error correction in the "communication scenario", where the noise is just present between sender and receiver, but their own encoding and decoding operations are presumed to be error-free. This is a good model for how mobile phones communicate with cell towers, or computers with each other via the internet. But it's not going to be a good model for quantum computers, where all operations we can currently perform are actually quite noisy. So we turn to the question of fault-tolerance: Is it possible to build a reliable computer out of unreliable parts? And if so, how?

Do-nothing error scaling The goal would be to execute a quantum circuit such that the fidelity of the actual output with the desired output is $1-\varepsilon$ for some small ε. One option is to just use the noisy gates that we have. If we think of the infidelity as a probability of error and imagine that each gate fails with some probability p, then in a computation of T operations we'll need to have $Tp \sim \varepsilon$, i.e. p decreasing with T. This isn't a terribly rigorous argument, but the point is made that if we "do nothing" then arbitrarily long computation will require correspondingly small gate imperfections. This is not really an option, though, as at best we can make these gate imperfections independent of the size of the computation. It is probably difficult enough to achieve this; after all imperfections may increase in larger computations due to cross-talk or interactions between components.

Basic idea of fault-tolerant compilation The alternative is to use error correction, somehow. Of course now we need to be more careful in how it is implemented, since the implementation will be noisy. Specifically, our goal is to be able to take a description of a circuit that we would like to implement and be able to construct a circuit involving error correction that will simulate the desired circuit even though its own components are noisy. A useful way to phrase this is that we would like to design a fault-tolerant *compiler*, the algorithm or recipe for constructing reliable circuits to implement any desired reference circuit. *The basic idea is that we should implement error correction and all the desired gates of the circuit so that we correct more errors than we create, or at least just as many.* It won't be possible to reliably correct *all* errors in a given correction round, instead we want to keep the number of errors manageable. In particular, none of the elements in the circuit should cause errors to spread in an uncontrolled manner, else we cannot recover in subsequent correction rounds.

A word on terminology: We will speak of *errors*, which afflict the state of the qubits, and *faults*, which are the production of errors in a circuit. The reason to keep these distinct is that it's then easier to talk about some number of faults creating and spreading some (possibly larger) number of errors via the circuit.

9.2 Fault-tolerant quantum memory

Here we will focus just on the question of preserving quantum information, not on performing any particular computation. So the compiler we need is fairly simple: We just a reliable way of implementing the operations needed for error correction. We will even leave out the question of how to reliably encode and decode the quantum information. The setup is shown below. The goal is to preserve the quantum information for a time longer than can be achieved by not encoding.

$$\begin{tikzcd}\boxed{\text{ideal encoder}} \arrow[r] & \boxed{\text{EC}} \arrow[r] & \boxed{\text{EC}} \arrow[r] & \boxed{\text{EC}} \arrow[r] & \boxed{\text{EC}} \arrow[r] & \boxed{\text{ideal decoder}}\end{tikzcd}$$

9.2.1 Sketch of requirements on error correction

Now we can investigate in more detail what is needed from the individual error-correction step in order to meet this goal. Suppose, for simplicity, that we would like to use a code which can correct one error. We need to measure the stabilizers of the code, perform a classical computation to determine the correction operation, and then apply the correction operation. In repeated rounds of error correction, if errors occur too frequently, there will be no way to remove errors before more are created. So suppose we have a procedure for making the stabilizer measurement such that a single fault in any of its internal gates would lead to an output with some error pattern that can be corrected by an ideal decoder. No faults in the procedure result in ideal decoding. We will take up the question of how to design such a procedure in the following chapter.

The important observation is that if there is only one fault in any two subsequent rounds, then the resulting error pattern can be corrected before another fault causes more errors and ruins any possibility of recovery. If no faults occur, then everything is ok, of course. If a fault occurs in the first round, then by assumption the second round will clean up whatever error pattern it created. If a fault occurs in the second round, then it creates an error that will have to be dealt with in the following third round.

Let us adopt a simple model in which all gates take the same amount of time, and faults occur at rate p on individual gates, including the identity gate (i.e. doing nothing). Moreover, assume also that there are G gates needed in each round of error correction. The overall scheme will (potentially) fail if there are more than two faults in any subsequent rounds. The probability of this event is $\sum_{k=2}^{2G}\binom{2G}{k}p^k(1-p)^{2G-k} = O(G^2p^2)$. For T rounds the probability of an unrecoverable event is not larger than $O(TG^2p^2)$ by treating each pair independently (and therefore overestimating). Now suppose we want the overall probability of failure to be $\varepsilon \approx O(TG^2p^2)$. Then with error correction we can sustain the qubit to this level for a number of rounds T which scales like $T \approx \varepsilon/G^2p^2$. This is an improvement over the no-encoding case of $T \approx \varepsilon/p$, provided $G^2p^2 \leq p$. Thus, the lifetime of the qubit can be improved from $O(1/p)$ to $O(1/p^2)$, but only for $p \leq 1/G^2$. If the correction scheme requires many gates, then there are many places for faults to occur, and so the breakeven noise rate for error correction is lower.

9.2.2 Detailed conditions

In the above we assumed that the error-correction mechanism, which is sometimes referred to as a "gadget", was such that single faults in its execution could be recovered by a subsequent round of ideal correction. A related condition, suitable for considering multiple faults, is

FTEC An error correction gadget is fault-tolerant to distance $d = 2t + 1$ when the output has error weight no larger than s_{int} whenever the input has no more than s_{ext} errors and no more than s_{int} faults occur during its execution, for $s_{\text{ext}} + s_{\text{int}} \leq t$.

A more common condition found in the literature has two parts: the error-correction correctness property and the error-correction recovery property.

ECCP Provided $s_1 + s_2 \leq t$, for any input codeword with weight-s_{ext} error and s_{int} faults in the gadget, an ideal decoding of the gadget output gives the same result as ideally decoding its input.

ECRP No matter the input, the output of the gadget is within s errors of a codeword, provided the number of faults s in the gadget is less than t.

We refer to these conditions together as AGP, for Aliferis-Gottesman-Preskill, who formulated them. It turns out that AGP implies FTEC, but the converse does not hold (except in special cases).

Let us see that FTEC (and therefore AGP) is sufficient for the above argument regarding repeated correction to go through. Suppose that no more than t faults occur in any two subsequent rounds of error correction. Since the input to the first round is error-free, $s_1 \leq t$ faults in the first round lead to at most s_1 errors on the input to the second round. The $s_2 \leq t - s_1$ faults in the second round then ensure that the input to the third round has only s_2 errors. And so on, till the last round, which will output a state with t errors or fewer. This state will be correctly decoded by the ideal decoder at the end.

Hence, the procedure will (potentially) fail if there are more than t faults in two subsequent rounds. Following the probabilistic analysis above, the encoded qubits can be sustained at a logical error of ε for $T \approx \varepsilon/\binom{2G}{t+1}p^{t+1}$ rounds. This will beat the no-encoding strategy provided $p^t \leq 1/\binom{2G}{t+1}$, which can be approximated by $p \leq (\frac{t+1}{2G})^{(t+1)/t}$. (So the dependence on G is less severe for larger t, and there is an additional factor of $t+1$, but we can expect that a code with a larger t requires a gadget with a larger G...)

Both AGP and FTEC are sufficient conditions for a gadget to be fault-tolerant, in the sense that the above argument will go through. However, both are formulated in terms of the code distance, which we know from Chapters 5 and 7 is not a good indication of code performance in the probabilistic setting. We will later see that a simple decoding gadget for the surface code is sufficient to keep errors from accumulating even though FTEC is not satisfied.

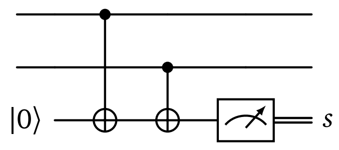
The AGP conditions were formulated to handle the analysis of concatenated codes, specifically to show that by concatenating distance-3 codes such as the Steane code with itself sufficiently many times, one can eventually reach any desired logical error rate provided the physical error rate is below a certain threshold value. The analysis is considerably simplified by making use of concatenated decoding. Then we can treat the inner encoding, noise, and inner decoding steps as defining a modified single-qubit noise channel, at which point we can just reuse the result we have for such channels.

For example, concatenating the Steane code with itself yields a code of 49 qubits. Say that the logical error rate of the Steane code itself for depolarizing noise channels of rate p is cp^2 for some constant c. Under concatenated decoding we can just examine the outer layer and treat the noise rate as cp^2 (note that the noise model might no longer precisely be the depolarizing channel, but one can get around that problem). Hence the noise rate after two levels of concatenation should be $\sim c^3p^4$. The catch in the setting of noisy syndrome extraction is that the error correction at the inner layer will not necessarily return the quantum state to the codespace. But it needs to get close enough so that the next layer of error correction has a chance of succeeding. Hence the ECRP condition.

Syndrome extraction gadgets 10

10.1 Back action

This is one major additional complication for reliable quantum error correction relative to classical error correction: Classical syndrome extraction cannot *directly* spread errors from the ancilla into the data. Consider the following example. Suppose we want to measure the value of the Z_1Z_2 stabilizer in the Shor code. Nominally, we would do this with the following circuit, involving one ancilla:



In the classical case, the ancilla could be subject to a bit flip fault somewhere in the circuit. This would change the value of s, but not by itself change any of the input data bits. A CNOT gate could also be faulty, which could cause an error on both the data and the ancilla, but still it is only one error on the data. Errors on the ancilla cause the syndrome to change, and since this could lead to an incorrect recovery operation, ancilla errors do spread indirectly to the data block. Something will anyway have to be done about this, a point we'll come back to below, so a faulty CNOT gate causing single errors in both data and ancilla isn't an additional problem.

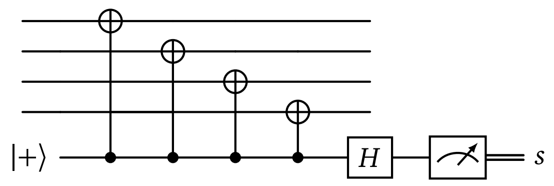
All of this holds as well for bit errors in the quantum case, but quantumly we must also consider phase errors. An unwanted Z on the ancilla will propagate through the CNOT gate and result in a phase error in the outgoing data. So now there are more ways error correction itself can corrupt the encoded information!

Problematic faults Let's examine this particular example more carefully. Note that a phase flip immediately after the ancilla preparation has no effect, since it's an eigenstate of Z. And a Z flip prior to measurement in the Z basis doesn't change the measurement result. So the circuit is immune to Z faults in these locations. On the other hand, a Z fault between the two CNOT gates will result in Z_2 on the data block. Moreover, it will be undetected by the measurement. So this circuit can potentially spread errors from the ancilla to the data.

On the other hand, in this example, a single fault in the ancilla causes at most a single error in the data, so the errors are not really increasing. But this was the simplest stabilizer we could possibly measure. Consider the same setup for one of the stabilizers from the Steane code, or just the X type stabilizers of the Shor code. These have weight four and six, respectively. An ancilla error in the middle of syndrome extraction then causes a weight two or weight three error. (This is the worst case since nominally larger weight errors from faults earlier in the circuit are equivalent to lower weight errors by multiplying by the stabilizer.) We now turn to several methods to work around this problem and deal with the problem of unreliable measurement results.

10.2 Shor error correction (gadget)

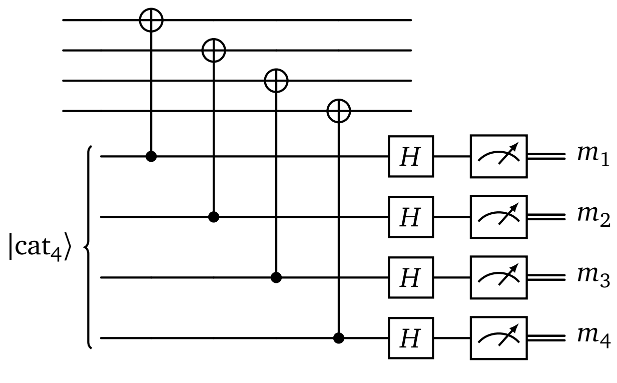
The first correction gadget we'll look at is the original proposal by Shor. Looking back at the example of error propagation from the ancilla, the problem was that the single ancilla talks to several qubits in the data block, opening a route for the error of a single fault to spread to multiple qubits. Shor's idea was to suitably encode the ancilla. It'll be more convenient to switch the example to measuring an X-type stabilizer, say on four qubits:



This circuit comes from interchanging X and Z everywhere by using Hadamard gates, and can be confirmed by propagating the X operator associated with the measurement result backwards through the circuit. In fact, we can measure any Pauli operator we want this way, simply by changing the target unitary operation. Here we wish to measure $XXXX$, so the target of the CNOT gate is the unitary X. If instead we want to measure $XZXZ$, we would just switch the target operation on the second and fourth qubits to be Z instead, i.e. perform a CPHASE gate at those positions instead.

10.2.1 Encode the ancilla with the cat state

The problem in this circuit is that a single X fault on the ancilla after the first two CNOTs can become a weight-2 X error in the data block. Instead of using a bare ancilla, we can instead use a *cat*[1] *state*, the logical $|+\rangle$ state of the repetition code: $|\text{cat}_4\rangle = |0000\rangle + |1111\rangle$. We'll need the number of qubits in the cat state to be equal to the weight of the stabilizer we're trying to measure. The qubits are interchangeable, so now we can couple the data and ancilla *transversally*:



What we really want to measure at the end is the logical $\bar{X}$ value, since that's the analog of what we would have measured in the bare ancilla case. Moreover, by propagating $\bar{X}$ prior to measurement backwards through the circuit and using the fact that the input state is a $+1$ eigenstate, it is apparent that this circuit will measure $XXXX$ on the data. Because $\bar{X} = XXXX$, we can obtain the desired measurement result s by computing the parity $s = \oplus_{j=1}^{4} m_j$ of the individual X measurement results.

Remaining problems with the gadget Due to the transversal coupling of data and ancilla, single X faults in the ancilla can only cause weight-1 X errors on the data. Single Z faults, meanwhile, in the ancilla remain in the ancilla. Faults in the CNOT gates themselves could cause any kind of error on the two outgoing qubits, but one of them is in the data block and the other will not interact with other ancilla qubits before being measured. Moreover, the recovery operation of applying a Pauli

[1]Schrödinger's

operator does not couple qubits, so single faults in the recovery operation just become single qubit errors.

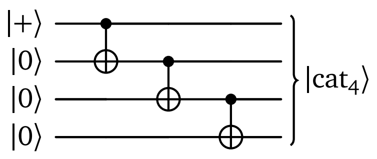
That's good, but there are still two problems.

1. What about measurement errors or, equivalently, Z faults on the ancilla? A single Z error will flip the syndrome value and cause an incorrect recovery operation to be applied, so we must avoid this.

2. Where do we get the $|\mathrm{cat}_4\rangle$ state and how do we know it does not contain too many errors? Maybe a single fault in its creation can lead to multiple errors in the cat state, which then spread to the data.

10.2.2 Creating the cat state

It's a little bit easier to handle the second problem first. To encode $|\mathrm{cat}_4\rangle$ we can use the following circuit:

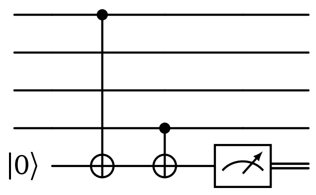


First let us consider which error patterns on the output need to be avoided, and if any of these can be caused by single faults.

Bit flip errors, no matter their number, will spread to the data block when measuring X type stabilizers (they will spread as Z type errors when using CPHASE). Due to the cascade nature of the circuit, a single fault can very well cause errors of weight one, two, three, or four. However, since $|\mathrm{cat}_4\rangle$ is an eigenstate of $XXXX$, there is no error of weight four and errors of weight three are equivalent to weight one. So weight two is the only worry. More specifically, the weight-four error comes from the first CNOT, producing two flips immediately afterward. But this is equivalent to a single flip on the control qubit prior to the gate, whose action is trivial on the input $|+\rangle$ state. Hence no weight-four flip error can occur. Weight-three errors come from failure of the second CNOT, essentially a bit flip on the second qubit after the first CNOT and before the second. But this is equivalent to two bit flips on the first qubit, one prior to the first CNOT and one afterward (this can be seen by introducing two bit flips on the first qubit after the first CNOT and using one of them to propagate the flip on qubit two backwards through the first CNOT). Again the flip on the state $|+\rangle$ has no effect, leaving just an outgoing bit flip on the first qubit.

Meanwhile, it turns out that phase flips are not a worry here. They cannot spread uncontained through the preparation circuit, so a single fault could cause at most two phase errors. And phase errors will not propagate into the data block, so the only concern is with their effect on the syndrome measurement. Then, since a weight-two error will leave the computed syndrome value unchanged, only single phase flips matter. We will anyway have to deal with erroneous syndromes, so this issue need not bother us here.

The only fault that can cause two bit flip errors is a single X fault on the third qubit prior to the final CNOT gate (or just the final CNOT gate itself). Thus the last two qubits would be in error (equivalently, the first two qubits). To avoid this problem, we can simply check if the parity of, say, the first and last qubits match. If not, we can discard the state and start over. In particular, we execute the cat state preparation circuit above and then test its output as follows:



If the measurement outcome is 1, the parity is incorrect, and we have to start over and repeat the procedure. If the measurement outcome is 0, we keep the cat state. Of course, the verification circuit could have a fault, and our confidence that the cat state is correct in the latter case could be misplaced. However, no single fault can lead to a 0 outcome and more than one bit flip error on the cat state, so everything is ok (not correct, just ok).

Nominally we also need to worry about phase errors propagating from the verification ancilla to the cat state, but in this case we don't need to do anything further about it. Only a single phase error could appear on the cat state, which would then ruin the measurement result, but as mentioned above we need to deal with untrustworthy syndromes anyway.

10.2.3 Repeating the syndrome measurement

Now we come to this issue of untrustworthy syndromes. We already encountered it in the discussion of measuring Z_1Z_2, and it would plague a classical reversible computer as well. The solution is yet more repetition, in this case of the entire syndrome measurement procedure. By doing so, we can eventually be confident that we have the correct syndrome.

However, a subtlety arises because there are several syndrome bits to determine. As we cannot measure the stabilizers simultaneously, since we need to interact with the data qubits, a fault could creep in after we have measured some of the stabilizers but before we have measured all of them. In light of this, how should we interpret the syndrome? Part of it was measured before the fault and part of it after; does it give anything useful?

Let's examine the simpler case of just correcting phase errors to get some intuition without dealing with the full problem. Suppose we have improved the quantum hardware to the point that bit flip errors are not a problem, and we just use the three-bit repetition code in the X basis to correct single phase errors. This code has stabilizers X_1X_2 and X_2X_3, as well as logical operators $\bar{X} = X_1X_2X_3$ and $\bar{Z} = Z_1Z_2Z_3$, and we want to use the Shor gadget to measure the syndrome. Our procedure to ensure reliability of the syndrome is as follows.

1. Measure the full (2 bit) syndrome using bare ancillas.
2. If the syndrome is trivial, apply no recovery and finish.
3. If the syndrome is nontrivial, measure the syndrome again and apply recovery operation implied by the *second* syndrome.

Analysis of the repeated syndrome measurement Let us show that this scheme satisfies the FTEC criterion for $t = 1$. In the case of no input errors and no faults, the first syndrome result is trivial, and so the correct recovery of no action is taken. If the input has a single error, then we only consider the case of fault-free syndrome measurements. The two syndromes now agree, and the correct recovery will be applied. Finally, we are left with the case of an error-free input and one fault in the correction procedure. We do not need to consider a fault in the correction operation itself, since the correction will only be needed if there were a fault earlier in the syndrome extraction step, and so there would then be two faults in total. If the syndrome from the first round is nontrivial due to a fault immediately prior to the syndrome extraction, then it just as well counts as an error on the input, which we already dealt with. If it occurs on the data after the syndrome extraction, then it just causes a single outgoing error. Faults that occur between the two steps of a single round are the reason the gadget has the structure it does. If a phase error afflicts the second qubit after the X_1X_2 measurement but before the X_2X_3 measurement, then the first syndrome is $(0,1)$. Since this is nontrivial we measure again and now find a syndrome of $(1,1)$. So the recovery operation (which we now assume has no faults) corrects the phase error Z_2. Had we accepted the first syndrome result, the correction would be Z_3, at which point the outgoing error would be Z_2Z_3. Avoiding this single fault causing two errors necessitates a second round of syndrome measurement.

For larger t, the strategy is to perform syndrome extraction up to $(t+1)^2$ times, stopping when the same syndrome is observed in $t+1$ consecutive rounds. The resulting gadget can be shown to satisfy the AGP conditions (and that in t faults there will be a sequence of $t+1$ identical syndromes in $(t+1)^2$ rounds).

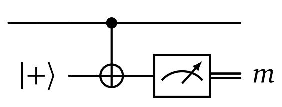
10.2.4 Discussion of the Shor gadget

Altogether we have a rather Rube Goldberg construction, to be sure. It is almost comical how the gadget construction continually piles on extra patches in order to finally meet the fault tolerant conditions, and this just to protect against one error! On the other hand, this is sort of a triumph of the usefulness of error correction. The logic of the construction is almost to just bash our way through by applying the repetition code whenever we have a problem. For instance, employing the Shor gadget on the Shor code amounts to five instances of repetition: one for X errors and one for Z errors in the code itself, one for the cat state itself, one to ensure reliable measurement using the cat state, and one to ensure reliable cat state preparation.

Although it might seem like all the repetitions will result in a gadget that is sure to fail because it will encounter more than one fault by the time it's all done, actually this statement only implies that the threshold must be very small. We can push the error rate as low as we want by concatenation, adding more layers to our Rube Goldberg machine, but only if the bare gates, state preparation, and measurements have error rates below the threshold.

10.3 Steane error correction gadget

For CSS codes we can use a somewhat more straightforward gadget due to Steane. It relies on the fact that a logical CNOT gate between two blocks of a CSS code can be implemented transversally. First consider the funny-looking circuit on two qubits.

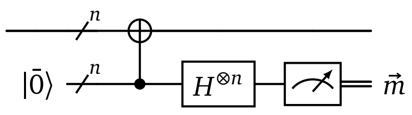


Nominally this will just produce a random measurement outcome and won't do anything at all to the first qubit: The target of the CNOT is a +1 eigenstate of X, so nothing happens. Note that we definitely don't want to start with $|-\rangle$. This is equivalent to a Z acting after preparing $|+\rangle$, which then propagates via the CNOT to Z acting on the first qubit.

If we lift this to an encoded qubit (the first) and encoded $|\bar{+}\rangle$ ancilla (the second), implementing a logical CNOT, the above conclusion still holds at the level of the logical qubits. Nothing happens to the logical value of the data block. However, when the CNOT is implemented transversally, X

errors on the data propagate to the ancilla. In fact, the errors are just copied onto the ancilla. Then we can measure the Z type stabilizers of the ancilla to determine the position of X errors on the data. Moreover, we only need to measure the Z value of each qubit individually, since with this information we can reconstruct the stabilizers.

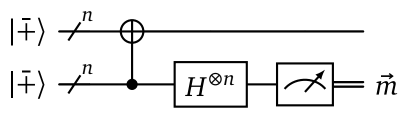
Swapping X and Z and turning the CNOT around, we have the circuit to determine Z errors by measurement of X on the ancilla:



Here the dash n is meant to indidate that the wire represents n qubits. The CNOT is transversal, the transversal Hadamard is also indicated, and the only symbol that doesn't quite make sense is the measurement. In this circuit it indicates individual measurement on the qubits.

Note that a single circuit is sufficient to obtain *all* of the Z syndromes, unlike the Shor gadget where each stabilizer necessitates its own circuit. This also simplifies the problem of untrustworthy syndromes, now there are no half correct and half incorrect syndromes to consider. In fact, the syndrome measurement does not need to be repeated when using the Steane gadget. A single X error on the ancilla in Z syndrome extraction will cause the syndrome to diagnose one more error, and so the correction will only be off by that single error. Similarly, if a CNOT gate fails and produces an X error in the data and the ancilla, then the error in the data will actually be corrected. If a CNOT fails and produces a Z error in the data and an X error on the ancilla, then the recovery operation will create an X error in the data at the same location as the Z error. Hence, the overall effect will still be a single-qubit error. Note that these considerations apply whether the input is a codeword or not, and the argument extends to codes correcting t errors as well. For a full discussion, see Gottesman's review.

There's no free lunch, though. For one thing, we need bigger ancillas, as large as the data block instead of the stabilizer weight. And we still have to verify the ancilla. This is more complicated than in the Shor gadget, but like there, we use the fact that we're preparing a known state (instead of encoding an unknown state). As ever, the trouble is single or small numbers of faults causing high weight error, which could then lead to the ancilla looking more like $|\bar{-}\rangle$ than $|\bar{+}\rangle$. Consider the following circuit for verification of $|\bar{+}\rangle$:



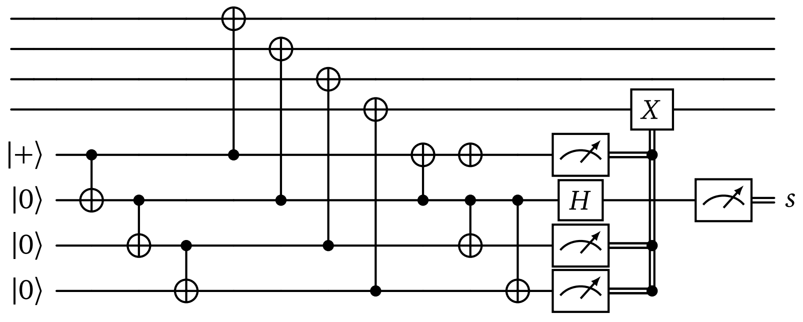
Again we measure each qubit in the second block individually, here in the X basis. From this we can infer the value of the logical $\bar{X}$ by using the syndromes to determine errors and subsequently computing the logical value. If we find $\bar{X} = +1$, we accept, otherwise we reject and start over. If a fault occurs in the ancilla block which causes such a high weight error that $|\bar{+}\rangle$ would be corrected to $|\bar{-}\rangle$, then this will be detected in the ancilla-ancilla block. A similar fault in the ancilla-ancilla block will also cause a rejection, but it won't damage the data. Continuing this train of thought, it is apparent that if we prepare three ancillas and use the second two to test the first one, then a single fault in the first will show up as $\bar{X} = -1$ in both the verification ancillas. In this case we can just apply a logical $\bar{X}$ to the actual ancilla and continue.

10.4 Alternatives to ancilla verification

The overhead of ancilla verification is quite large. Most calculations of the number of qubits needed to execute given quantum algorithms, e.g. Shor's algorithm, show that most of the time is spent on error correction and most of the time spent in error correction is spent on ancilla verification. So it's useful to think about alternatives. Here I mention two, one of which you'll examine in more detail in the problem sets. The broad point of both is to use the spare syndromes that are present in most coding schemes, i.e. for non-perfect codes. For instance, though unrelated to fault tolerance, the Shor code can correct some weight two Pauli operators, provided they occur in different rows. By cleverly modifying a given correction circuit with some catastrophic faults (i.e. which cause too many errors), the extra syndromes can be used to diagnose the catastrophic faults and correct the resulting errors.

10.4.1 Ancilla decoding

DiVincenzo and Aliferis observed that by decoding the cat state with a different circuit than used for preparation, single faults which cause multiple errors can be caught by the measurement.[2] Here's the whole circuit, for illustration:



The ancilla is not verified before use, it is just applied directly to the data. But instead of wasting the ancilla qubits by measuring each in the X basis and computing the logical $\bar{X}$ value, here it is decoded back to individual qubits. But not with the same circuit which created the cat state to begin with. The syndrome value is now contained in the measurement of one qubit, and the other three give information about multiqubit errors that may have spread to the data block. In our example, we are worried about weight-two errors on the second two qubits. This problem is addressed by flipping one of the qubits if the additional outputs of the ancilla decoding circuit are all +1 (note that there's an extra NOT gate on the first ancilla qubit relative to a usual cat state encoder). We could also flip the last two qubits, but the point is now that a single fault can cause at most one error in the data.

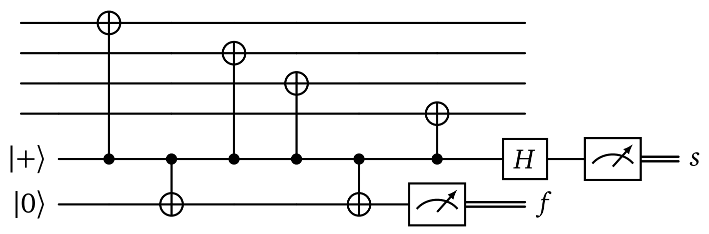
The method can also be adapted to Steane correction.

10.4.2 Flag qubits

An even simpler and more recent scheme is the "flag method" invented by Chao and Reichardt.[3] Well, it is simpler to implement, but harder to analyze. Here we go back to the unencoded ancilla interacting with the data, but we add an additional ancilla to check for the fault which causes the

[2]See the supplement of arXiv:quant-ph/0607047.
[3]See arXiv:1705.02329 [quant-ph].

weight-two error. Originally it was designed for distance-three codes, but has been extended to arbitrary stabilizer codes. Here's the circuit:



An X error between the two middle CNOTs will flip the flag qubit (raise the flag, I guess) and the measurement result f. This can be used to modify the circuits for extracting the remaining syndromes and subsequent rounds of syndrome extraction such that the error can be uniquely identified. The analysis is more complicated than ancilla decoding. For one thing, note that the flag is not only raised if an X fault occurs between the middle CNOTs, but also right before them. This event causes a weight-one error, so does not need to be addressed. Distinguishing these cases is done with the remaining syndrome measurements in this cycle of syndrome measurements as well as subsequent cycles.

10.5 Surface code decoding

We didn't have time to cover this. It turns out that matching can be used to decode the surface code when multiple rounds of syndrome extraction are performed with bare ancillas. The resulting threshold when the noise model is that only qubits or measurement results themselves may flip (so no back-action from the ancilla to the data) is around 3% for independent bit and phase errors. If we consider noise in all parts of the syndrome extraction circuit (i.e. including back-action on the data), the threshold drops to around 0.67%.