Math Seminar note 2

14.0 Quantum error correction

We have seen a way of dealing with the computational errors introduced by the physical problem of decoherence, namely the Shor [[9,1,3]] code, but this is just the start of the story. There is a vast body of work on classical error correction, so it's sensible to ask if we can adapt this to help us in the world of quantum computation. As we shall see, we can actually use quite a lot of the theory of classical error-correction codes, and in doing so we will start to really make use of the stabiliser formalism introduced all the way back in Chapter 7. But note that this still isn't the end of the story: our goal is so-called fault-tolerant computation, which we come to in Chapter 15.

14.1 the Hamming code

A little background:

In the late 1940s, Richard Hamming, working at Bell Labs, was frustrated by the fact that the machines running his punch cards (back then punch cards were used for data storage) were good enough to notice when there was an error (and halting) but not good enough to know how to fix it.

The challenge in designing efficient error-correcting codes resides in the trade-off between **rate** and **distance**.
Ideally, both quantities should be high:

- A high rate signifies low overhead in the encoding process (i.e. requiring only a few redundant bits),
- A high distance means that many errors can be corrected.

So can we optimize both of these quantities simultaneously? Unfortunately, various established bounds tell us that there is always a trade off, so high-rate codes must have low distance, and high-distance codes must have a low rate. (They have an inverse relationship). Still, there is a lot of ingenuity that goes into designing good error-correction codes, and some are still better than others!

Before looking at quantum codes in more depth, we again start with classical codes. For example, in Section 13.6 we saw the three-bit repetition code, which has a rate of $R=1/3$ and distance 3. However, the Hamming [7,4,3] code has the same distance, but a better rate of $R=4/7>1/3$.

[7,4,3] code is [n, k, d] code

Definition:

1. Linear code: A code where the codewords for, a commutative group under addition
2. Linear code over F2: The code words are only composed of {0, 1}
3. [n, k] code:
    - Let $1 \leq k < n$, a [n,k] code over F_2 is a k-dimensional subspace of F_2^n
4. [n, k, d] code:
    - Let $1 \leq k < n$, $1 \leq d$, it is a [n, k] code where the hamming distance of the [n, k] code is d
5. Hamming distance of a code C:
    - It is the minimal distance between any two different code words.

Example of constructing a [n, k, d] code:

Construction:

$1 \leq k < n$, generator matrix $G = [I_k | P]$

$\rightarrow I_k = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ 0 & 0 & \cdots & 1 \end{bmatrix} = k \times k$

$\rightarrow P = k \times (n-k)$ matrix with coeff. in $\mathbb{F}_2$.

Ex. [8, 3] code

$k=3,\ n=8,\ I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},\ P = \begin{bmatrix} * & * & * & * & * \\ * & \cdots & \cdots & \cdots & * \\ * & \cdots & \cdots & \cdots & * \end{bmatrix} 3$

5

- there will be 2^{15} possible P matrices

Let $P = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix} \Rightarrow G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$

Code word: $(u_1, u_2, u_3) \cdot G$, $u_1, u_2, u_3 \in \mathbb{F}_2$

There will be 8 code words

$(0\,0\,0) \cdot G = (0\,0\,0\,0\,0\,0\,0\,0)$

$(1\,0\,0) \cdot G = (1\,0\,0\,0\,1\,1\,0\,1)$

$(0\,1\,0) \cdot G = (0\,1\,0\,1\,0\,1\,1\,0)$

$(1\,1\,1) \cdot G = (1\,1\,1\,1\,0\,0\,0\,1)$

$(0\,0\,1) \cdot G = (0\,0\,1\,0\,1\,0\,1\,0)$

$(1\,1\,0) \cdot G = (1\,1\,0\,1\,1\,0\,1\,1)$

$(1\,0\,1) \cdot G = (1\,0\,1\,0\,0\,1\,1\,1)$

$(0\,1\,1) \cdot G = (0\,1\,1\,1\,1\,1\,0\,0)$

[7,4,3] code

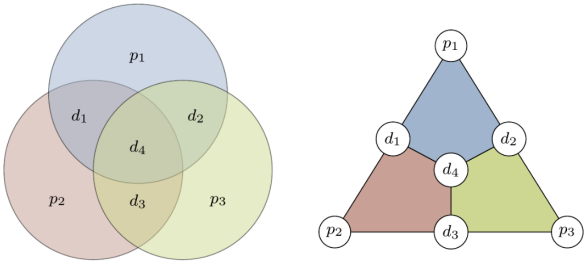


Figure 14.1: *Left:* The Venn diagram for the Hamming $[7, 4, 3]$ code. *Right:* The **plaquette** (or **finite projective plane**) diagram for the same code. In both, d_i are the **data bits** and the p_i are the **parity bits**. The coloured circles (resp. coloured quadrilaterals) are called **plaquettes**.

The idea is that we have a four-bit string d1d2d3d4 consisting for the four-data bits, and we encode into a seven-bit string d1d2d3d4p1p2p3 by appending three parity bits p1, p2, and p3, which re defined by:

$$
\begin{aligned}
p_1 &= d_1 + d_2 + d_4 \bmod 2 \\
p_2 &= d_1 + d_3 + d_4 \bmod 2 \\
p_3 &= d_2 + d_3 + d_4 \bmod 2.
\end{aligned}
$$

We can also express this encoding in matrix notation, defining the **data vector** $\mathbf{d}$ by

$$
\mathbf{d} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \end{bmatrix}
$$

and the **generator matrix**[281] G by

$$
G = \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \hline 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{array}\right]
$$

The encoding process is then given by the matrix G acting on the vector d. Indeed, since the top (4×4) part of G is the identity, the first four rows of the output vector Gd will simply be a copy of; the bottom (3×4) part of G is chosen precisely so that the last three rows of Gd will be exactly

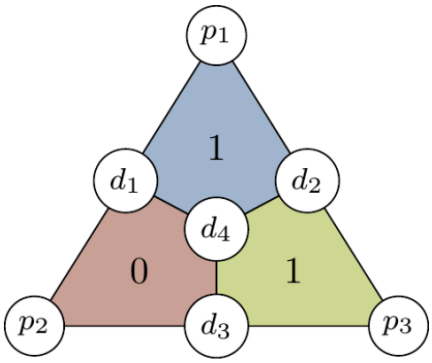
$$
G\mathbf{d} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} \mathbf{d} \\ p_1 \\ p_2 \\ p_3 \end{bmatrix}.
$$

By construction, the sum of the four bits in any single plaquette of the code sum to zero.

$$
\begin{aligned}
p_2 + d_1 + d_3 + d_4 &= d_1 + d_3 + d_4 + d_1 + d_3 + d_4 \\
&= 2(d_1 + d_3 + d_4) \\
&= 0
\end{aligned}
$$

and the same argument holds for the other two plaquettes. This incredibly simple fact is where the power of the Hamming code lies, since it tells the receiver of the encoded string a lot of information about potential errors.

Concrete example. Say that Alice encodes her data string d1d2d3d4 and sends the result Gd to Bob, who takes this vector and looks at the sum of the bits in each plaquette, and obtains the following:



If we make the assumption that at most one error occurs then this result tells us exactly where the bit-flip happened: it is not in the bottom-left (red) plaquette, but it is in both the top (blue) and bottom-right (yellow) plaquettes. Looking at the diagram we see that it must be d2 that was flipped, and so we can correct for this error by simply flipping it back before unencoding (where the unencoding process is given by simply forgetting the last three bits of the received string).

We can describe the error location process in terms of matrices as well, using the parity-check matrix H, given by

$$H = \left[\begin{array}{cccc|ccc} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right].$$

Note that the rows of H are exactly the coefficients of the parity-check equations for each plaquette, where we order them top–left–right (blue–red–yellow). For example, to get the sum corresponding to the bottom-left (red) plaquette, we need to sum the first, third, fourth, and sixth bits of the encoded string d1d2d3d4p1p2p3 and these are exactly the non-zero entries of the second row of H. The columns of the parity-check matrix H are known as the error syndromes, for reasons we will now explain.

The parity-check matrix H is defined exactly so that

$$H\mathbf{c} = 0 \iff \mathbf{c} \text{ is a codeword.}$$

Now we can see a bit more into how things work, since linearity of matrix multiplication tells us that, if a receiver receives $\mathbf{c} + \mathbf{e}$ where $\mathbf{e}$ is the error,

$$\begin{aligned} H(\mathbf{c} + \mathbf{e}) &= H\mathbf{c} + H\mathbf{e} \\ &= H\mathbf{e}. \end{aligned}$$

Decoding the message then consists of finding the most probable error e that yields the output He. If e is a single bit-flip error, then He is exactly a column of H, which justifies us describing the columns as error syndromes. We can construct a table describing all of the possible error syndromes, and which bit they indicate for us to correct:

Syndrome	000	110	101	011	111	100	010	001
Correction	-	d_1	d_2	d_3	d_4	p_1	p_2	p_3

14.4 Logical operators

- We have been slowly building up towards constructing quantum error-correction codes using the stabilizer formalism, but there is one major detail that we have yet to mention.
- We haven't written out what the stabilizer states actually are, nor what the encoding circuits look like. There is a simple reason for this: at this point, we don't actually know!
- There's a little more work to be done — the stabilizers have provided us with a two-dimensional space, but if we have $|0\rangle$ and $|1\rangle$ to encode, how are they mapped within the space? So far, it's undefined, and there is a lot of freedom to choose, but the structures provided by group theory are quite helpful here in providing some natural choices.

Let's start with a brief recap.

- The n-qubit Pauli group, denoted as Pn, includes all possible combinations (n-fold tensor products) of Pauli matrices (1, X, Y, and Z), along with potential global phase factors of ±1 and ±i. When we have an operator, s, from this Pauli group, and if this operator, when applied to a non-zero n-qubit quantum state $|\psi\rangle$, results in the same state $|\psi\rangle$ (i.e., $s|\psi\rangle = |\psi\rangle$), we say that the operator stabilizes the state $|\psi\rangle$. This means the state $|\psi\rangle$ is an eigenstate of the operator s with an eigenvalue of +1.
- **Stabilizer group**: A stabilizer group is defined as the set of all operators that have the property of stabilizing every state within a particular subspace, denoted as V.
- By applying some basic principles of group theory, we were able to identify all potential stabilizer groups, showing that they correspond precisely to the abelian subgroups of the n-qubit Pauli group (Pn) that exclude the element -1. Following this, we explored the group structure of the Pauli group itself and examined the relationship between any stabilizer group, S, and how it is incorporated within the Pauli group. We discovered that the **normalizer:**

$$N(\mathcal{S}) = \{g \in \mathcal{P}_n \mid gsg^{-1} \in \mathcal{S} \text{ for all } s \in \mathcal{S}\}$$

of $\mathcal{S}$ in $\mathcal{P}_n$, and the **centraliser**

$$Z(\mathcal{S}) = \{g \in \mathcal{P}_n \mid gsg^{-1} = s \text{ for all } s \in \mathcal{S}\}$$

of $\mathcal{S}$ in $\mathcal{P}_n$ actually agree, because of some elementary properties of the Pauli group. Furthermore, we showed that the normaliser (or centraliser) was itself normal inside the Pauli group, giving us a chain of normal subgroups

$$\mathcal{S} \triangleleft N(\mathcal{S}) \triangleleft \mathcal{P}_n.$$

This lets us arrange the elements of $\mathcal{P}_n$ into cosets by using the two quotient groups

$$N(\mathcal{S})/\mathcal{S} \qquad \text{and} \qquad \mathcal{P}_n/N(\mathcal{S}).$$

How does this help us with our stabiliser error-correction codes? Let's look first at the former: cosets of $\mathcal{S}$ inside its normaliser $N(\mathcal{S})$.

If $|\psi\rangle \in V_{\mathcal{S}}$ is a state in the stabilised subspace, then any element $g \in \mathcal{S}$ always satisfies

$$g|\psi\rangle = |\psi\rangle$$

whereas any element $g \in N(\mathcal{S}) \setminus \mathcal{S}$ merely satisfies

$$g|\psi\rangle \in V_{\mathcal{S}}$$

and, for any such g, there are always states in V_S that are not mapped to themselves. However, if we look at cosets of $\mathcal{S}$ inside $N(\mathcal{S})$ then we discover an incredibly useful fact: all elements of a given coset act on $|\psi\rangle$ in the same way. To see this, take two representatives for a coset, say $g\mathcal{S} = g'\mathcal{S}$ for $g, g' \in N(\mathcal{S})$. By the definition of cosets, this means that there exist $s, s' \in \mathcal{S}$ such that $gs = g's'$. In particular then,

$$gs|\psi\rangle = g's'|\psi\rangle$$

but since $s, s' \in \mathcal{S}$ and $|\psi\rangle \in V_{\mathcal{S}}$, this says that

$$g|\psi\rangle = g'|\psi\rangle$$

as claimed.

Since the cosets of $\mathcal{S}$ inside $N(\mathcal{S})$ give well defined actions on stabiliser states, preserving the codespace, we can treat them as operators in their own right.

Remark:

The cosets of S inside N(S) are called logical operators, and any representative of a coset is an implementation for that logical operator.

In general, the logical states will be superpositions of states, but we still sometimes refer to them as codewords.

In our example of the three-qubit code, we have the two logical states logical 0 and logical 1, which we denote by

$$|0\rangle_L := |000\rangle$$
$$|1\rangle_L := |111\rangle.$$

The reason for these names comes down to two main points: first, |0⟩_L is the encoded form of |0⟩, the actual zero state; second, this state |0⟩_L acts just like the zero state would when logical operators are used on it. For instance, the operator X changes |0⟩ to |1⟩, so in the same way, the logical X should change |0⟩_L to |1⟩_L.

The normaliser of $\mathcal{S}$ inside $\mathcal{P}_3$ is

$$N(\mathcal{S}) = \{\mathbf{1}, XXX, -YYY, ZZZ\} \times \mathcal{S}$$

which we have written in such a way that we can just read off the cosets: there are four of them, and they are represented by $\mathbf{1}$, XXX, $-YYY$, and ZZZ. These four (implementations of) logical operators all get given the obvious names:

$$\mathbf{1}_L := \mathbf{1}$$
$$X_L := XXX$$
$$Y_L := -YYY$$
$$Z_L := ZZZ$$

14.6 Logical operators

We already know that the Pauli matrices provide a useful basis with respect to which we can decompose the effects of any quantum channel300, so we should carefully understand how the Pauli operators P∈Pn interact with any error-correcting code. For this, we introduce the notation:

$$c(P,\sigma) := \begin{cases} 0 & P \text{ and } \sigma \text{ commute} \\ 1 & P \text{ and } \sigma \text{ anticommute} \end{cases}$$

for any Pauli operator P and any other operator σ. A particularly nice thing about this choice of definition (as opposed to taking $c(P,\sigma) \in \{\pm 1\}$, say) is that we can write

$$P\sigma = (-1)^{c(P,\sigma)}\sigma P.$$

Furthermore, this function has a nice relation on products: writing $\oplus$ to mean addition $\mod 2$, we can see that

$$c(P,\sigma\tau) = c(P,\sigma) \oplus c(P,\tau)$$

which reminds us of the fact that two anticommuting operators multiplied together produce a commuting operator.

Now fix some stabiliser group[301] $\mathcal{S} = \langle g_1, \ldots, g_{n-k} \rangle$. We define the **error syndrome** $\underline{e}_P$ of a Pauli operator P to be the vector of all the values $c(P, g_i)$, i.e.

$$\underline{e}_P = \big(c(P,g_1), \ldots, c(P,g_{n-k})\big).$$

It follows from the the above relation of how $c(P,-)$ turns products into sums that

$$\underline{e}_{P\sigma} = \underline{e}_P \oplus \underline{e}_\sigma.$$

The set of Pauli operators that have *zero* syndrome are special, and form a set known as the **normaliser**:

$$N(\mathcal{S}) := \big\{P \in \mathcal{P}_n \mid c(P,\sigma) = 0 \text{ for all } \sigma \in \mathcal{S}\big\}.$$

So the Pauli group is subdivided into error cosets. by the normaliser, and every Pauli in the same coset has the same error syndrome. If we perform a syndrome measurement after passing through some noisy channel and get the result $\underline{e}$, then the effect of the channel collapses to being a linear combination of the terms inside the error coset corresponding to the error syndrome $\underline{e}$. By applying any element of that error coset, we are mapped back to the normaliser.

In fact, there is further substructure[303] within the normaliser, and this is also reflected in each error family. Given a Pauli operator $P \in N(\mathcal{S})$ in the normaliser, we define its **logical syndrome** to be the vector $\underline{\ell}_P$ of all the values $c(P, \sigma)$, where σ ranges over $N(\mathcal{S})$. Note that, for any $\tau \in \mathcal{S}$, we have $\underline{\ell}_P = \underline{\ell}_{P\tau}$. Again, this splits the normaliser Pauli operators into sets, which we call the **logical cosets**, each being defined by having the same logical syndrome.

Final

remark:
While we can measure the error syndrome (all the stabilizers commute), we cannot measure the logical syndrome (not all the logical operators commute). Indeed, we must not even try — measuring just one such value is equivalent to performing a measurement of the logical qubit, destroying the superposition of the very state with which we're trying to compute!

14.3 Quantum Codes from Classical

We would like to use the insights gained from our study of classical codes to help us build quantum codes. Let's start with a classical $[n, k, d]$ code (such as the Hamming $[7, 4, 3]$), with parity-check matrix H and generator G. Each row r of H is a binary string $x_r = x_{r,1}x_{r,2}\dots x_{r,n}$, where $x_{i,j}$ is the (i, j)-th element of H. For $1 \leqslant r \leqslant n$, we define a stabiliser generator

$$G_r := X_{x_r} := \otimes_{j=1}^{n} X^{x_{r,j}}.$$

For example, in the case of the Hamming $[7, 4, 3]$ code, we have

$$H = \left[\begin{array}{cccc|ccc} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right].$$

so the three rows define three generators

$$\begin{aligned} G_1 &= X_{1101100} = XX\mathbf{1}XX\mathbf{11} \\ G_2 &= X_{1011010} = X\mathbf{1}XX\mathbf{1}X\mathbf{1} \\ G_3 &= X_{0111001} = \mathbf{1}XXX\mathbf{11}X. \end{aligned}$$

Now consider a state $|\psi\rangle$ that is stabilised by these generators, i.e. such that

$$G_r|\psi\rangle = |\psi\rangle.$$

What happens if a Z error occurs on a particular qubit: what measurement results do we get when we measure the stabilisers? Well, writing Z_j to mean a Z error on the j-th qubit, as usual,

$$\begin{aligned} G_r Z_j|\psi\rangle &= (-1)^{x_{r,j}} G_r|\psi\rangle \\ &= (-1)^{x_{r,j}}|\psi\rangle. \end{aligned}$$

So the measurement outcome directly corresponds to the (r, j)-th entry $x_{r,j}$ of the parity check matrix. Generally, if this Z_j error occurs, then measuring for all rows r will give measurement outcomes that directly correspond to the j-th column of the parity check matrix. This is just the same lookup table as in the classical case: this codespace is a distance d error correcting code for single Z errors. Using the Hamming $[7, 4, 3]$ code as an example again, we get the following table of error syndromes, where we write $\pm$ to mean ± 1:

[Table in the reference]

We can figure out some key properties of combining an $[n, k, d]$ code (G, H) for X-stabilisers and an $[n, k', d']$ code (G', H') for Z-stabilisers without too much difficult. Since our first code encodes k bits, the generator G has k rows, and the parity-check matrix H has $n - k$ rows. Thus there are $n - k$ of the X-type generators, and $n - k'$ of the Z-type generators; in total there are $2n - (k + k')$ generators. Since each generator halves the dimension, the dimension of the Hilbert space defined by the stabilisers is

$$2^{n-2n+k+k'} = 2^{k+k'-n}$$

i.e. it encodes $k + k' - n$ qubits. The combined code has a distance k against Z errors, and k' against X errors; since the two types of errors are correctly independently, the total distance is simply $\min(d, d')$. In summary then, we have created an

$$[[n, k + k' - n, \min(d, d')]]$$

quantum error-correction code, and its decoding is well understood based on the classical decoding methods applied independently for X errors and Z errors. This general construction of quantum error correcting codes is known as the **CSS construction**, for its originators Robert Calderbank, Peter Shor, and Andrew Steane.

Given an $[n, k_1, d_1]$ code $C_1 = (G_1, H_1)$ and an $[n, k_2, d_2]$ code $C_2 = (G_2, H_2)$ such that $\text{range}(H_2^T) \subseteq \text{range}(G_1)$, the **CSS code** $\text{CSS}(C_1, C_2)$ constructed as above is an $[[n, k_1 + k_2 - n, \min(d_1, d_2)]]$ code.

14.5 Error families

All in all, the chain of normal subgroups

$$\mathcal{S} \triangleleft N(\mathcal{S}) \triangleleft \mathcal{P}_n$$

really does describe the full structure of the code: logical states, logical operators, and error families.

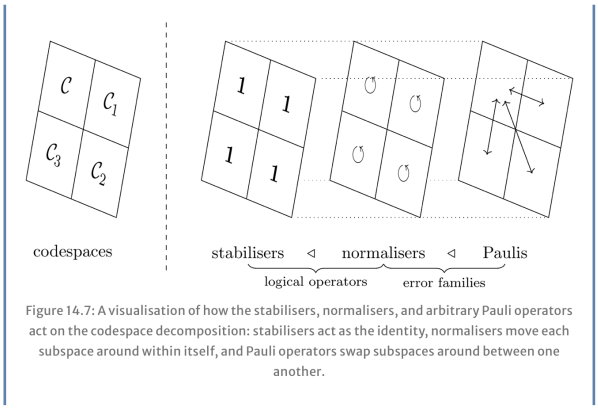


Figure 14.7: A visualisation of how the stabilisers, normalisers, and arbitrary Pauli operators act on the codespace decomposition: stabilisers act as the identity, normalisers move each subspace around within itself, and Pauli operators swap subspaces around between one another.

The quotient group $N(\mathcal{S})/\mathcal{S}$ gave us logical operators, so the next thing to ask is what we get from the quotient group $\mathcal{P}_n/N(\mathcal{S})$.

> The cosets of $N(\mathcal{S})$ inside $\mathcal{P}_n$ are **error families** on the codespace $V_{\mathcal{S}}$. The individual elements of any error family (i.e. the elements of $\mathcal{P}_n$) are called **physical errors**.

Again, we can write $\mathcal{P}_3$ in such a way that we can immediately read off the cosets:

$$\mathcal{P}_3 = \{\mathbf{1}, X\mathbf{11}, \mathbf{1}X\mathbf{1}, \mathbf{11}X\} \times N(\mathcal{S}) \times \{\pm 1, \pm i\}.$$

Ignoring the phases, the three (non-trivial) error families are single bit-flips: $[X\mathbf{11}]$, $[\mathbf{1}X\mathbf{1}]$, and $[\mathbf{11}X]$; these error families X_i map the codespace $\mathcal{C}$ to the subspace $\mathcal{C}_i$, as shown in Figure 14.6.

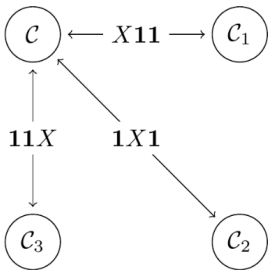


Figure 14.6: The single bit-flip error family X_i maps the codespace $\mathcal{C}$ to the subspace $\mathcal{C}_i$, e.g. $X_2|000\rangle = |010\rangle \in \mathcal{C}_2$.

These errors also let us understand how the structure of the codespace is mirrored across each of the cosets. In other words, we picked $\mathcal{C}$ to be our codespace, but what if we had instead picked $\mathcal{C}_1$? Well, we would get exactly the same code, just expressed in a different way, and this "different way" is described entirely by the error family $[X\mathbf{11}]$. What we mean by this is the following:

- We can write $\mathcal{C}_1$ as the stabiliser space of $\mathcal{S}$ conjugated by $X\mathbf{11}$, i.e.

$$\begin{aligned}(X\mathbf{11})\langle ZZ\mathbf{1}, \mathbf{1}ZZ\rangle(X\mathbf{11})^{-1} &= \langle (X\mathbf{11})(ZZ\mathbf{1})(X\mathbf{11})^{-1}, (X\mathbf{11})(\mathbf{1}ZZ)(X\mathbf{11})^{-1}\rangle \\ &= \langle -ZZ\mathbf{1}, \mathbf{1}ZZ\rangle\end{aligned}$$

  and, indeed, $|100\rangle$ and $|011\rangle$ are both stabilised by this group.
- The logical states of $\mathcal{C}_1$ are, by definition as our chosen basis, the elements $|100\rangle$ and $|011\rangle$, but note that these are exactly the images of the logical states of $\mathcal{C}$ under the error $X\mathbf{11}$, i.e.

$$\begin{aligned}|0\rangle_{L,1} &:= |100\rangle = X\mathbf{11}|000\rangle \\ |1\rangle_{L,1} &:= |011\rangle = X\mathbf{11}|111\rangle\end{aligned}$$

- The logical operators on $\mathcal{C}_1$ are the logical operators on $\mathcal{C}$ conjugated by $X\mathbf{11}$, i.e

$$\begin{aligned}X_{L,1} &:= (X\mathbf{11})(XXX)(X\mathbf{11})^{-1} \\ &= XXX \\ Z_{L,1} &:= (X\mathbf{11})(ZZZ)(X\mathbf{11})^{-1} \\ &= -ZZZ\end{aligned}$$

  and, indeed, $X_{L,1}$ and $Z_{L,1}$ behave as expected on the new logical states, i.e.

$$\begin{aligned}X_{L,1}\colon |0\rangle_{L,1} &\longmapsto |1\rangle_{L,1} \\ |1\rangle_{L,1} &\longmapsto |0\rangle_{L,1} \\ Z_{L,1}\colon |0\rangle_{L,1} &\longmapsto |0\rangle_{L,1} \\ |1\rangle_{L,1} &\longmapsto -|1\rangle_{L,1}\end{aligned}$$

  as you can check by hand.