CO 639 — Quantum Error Correcting Codes
CSS Codes & $GF(4)$ codes

Scribe: Niel de Beaudrap
Edited by Daniel Gottesman

January 29, 2004

1 CSS codes, continued

As we saw in the last lecture, a CSS code uses two classical linear codes, and uses them to correct different kinds of Pauli errors. We come up with two classical linear codes $\mathcal{C}_1$ (an $[n, k_1, d]$ code) to correct bit-flip and phase-flip errors, respectively. We do this by taking the parity check matrix H_1 from the code $\mathcal{C}_1$, and replacing all the entries with the rules $0 \mapsto I$, $1 \mapsto Z$, and we do the same with the parity check matrix H_2 from the code $\mathcal{C}_2$, except we replace $1 \mapsto X$. $\mathcal{C}_1$ then contributes $n - k_1$ stabilizer generators, and $\mathcal{C}_2$ gives us $n - k_2$ generators, for a total of $2n - k_1 - k_2$. In order for the generators contributed by the two codes to commute, we also require ${\mathcal{C}_2}^\perp \leqslant \mathcal{C}_1$, so that the rows of H_2 (which is the generator for ${\mathcal{C}_2}^\perp$) lie in the kernel of H_1. The *quantum* error correcting code that results will then be a stabilizer code encoding $k_1 + k_2 - n$ qubits.

1.1 Distance of CSS codes

At the end of the last class, we convinced ourselves that $d = \min\{d_1, d_2\}$, but we then saw that this produced a contradiction: the 9-qubit code has a CSS construction, and has distance 3 (because it corrects any single qubit error), but the phase-correction part of the code only has distance 2. How do we resolve this apparent contradiction? The answer is that, in fact, we can only prove $d \geqslant \min\{d_1, d_2\}$.

Is there a more fundamental way to understand why $d \neq \min\{d_1, d_2\}$ is not necessarily true? The argument we used last lecture was based on what the CSS code could hypothetically correct, but remember that the definition of distance is more in terms of *detection* than correction. If our original argument was flawed, it may be better to talk about things more in line with this definition.

First of all, it is clear that $d \geqslant \min\{d_1, d_2\}$ is true: if an error of weight less than $\min\{d_1, d_2\}$ errors occurs, that error operator will anticommute with stabilizer generators contributed by $\mathcal{C}_1$ and also ones contributed by $\mathcal{C}_2$, and so it will be detected. What about an error with weight exactly $\min\{d_1, d_2\}$? For the sake of argument, suppose $d_2 < d_1$. Even if we have an error E of weight d_2 which $\mathcal{C}_2$ cannot detect, if it contains any X or Y operators, $\mathcal{C}_1$ will still be able to detect it, because it will anticommute with one or more stabilizer generators from $\mathcal{C}_1$. Even if E contains only Z operators (and thus commutes with the stabilizer generators from $\mathcal{C}_1$), it might be the case that E is *generated* by the generators from $\mathcal{C}_1$, in which case it would be in the stabilizer, and have no effect on encoded states.

(This is exactly what happens in the case of the 9-qubit code.) So, we cannot prove anything definitive in this case beyond just $d \geqslant \min\{d_1, d_2\}$.

1.2 Constructing codewords

The stabilizer notation is quite useful, but it is also useful to know what the actual encoded states look like. For stabilizer codes, we can find a state that lies in the codespace in a straightforward way. Recall that the projection operator Π onto the the codespace can be given by

$$\Pi = \frac{1}{|\mathcal{S}|} \sum_{M \in \mathcal{S}} M \tag{1}$$

(where $\mathcal{S}$ is the stabilizer). Taking an arbitrary standard basis state $|a\rangle$, we can calculate the projection of $|a\rangle$ into the codespace,

$$\Pi |a\rangle = \frac{1}{|\mathcal{S}|} \sum_{M \in \mathcal{S}} \Big[M |a\rangle \Big] \ . \tag{2}$$

In the unlikely event that $|a\rangle$ is orthogonal to the codespace, one keeps trying until obtaining a basis state $|a'\rangle$ that isn't orthogonal to the codespace. When one finally obtains a non-zero vector as the result of applying the projection operator, the normalized vector will be a state in the codespace.

In general, the above procedure will require a lot of time to calculate, because the stabilizer grows exponentially with the number of generators. For CSS codes, however, there is a much simpler way of determining states in the codespace. As we saw, the number of encoded qubits in a CSS code is $k = k_1 + k_2 - n$. Re-expressing this in terms of the dimensions of $\mathcal{C}_1$ and $\mathcal{C}_2$, we have

$$k = \dim \mathcal{C}_1 + \dim \mathcal{C}_2 - n = \dim \mathcal{C}_1 - \dim \mathcal{C}_2{}^{\perp} \ .$$

Recall that $\mathcal{C}_2{}^{\perp} \leqslant \mathcal{C}_1$: then, the number of cosets of $\mathcal{C}_2{}^{\perp}$ in $\mathcal{C}_1$ is the same as the number of standard basis states that our CSS code has to encode. We can use this to motivate an idea of how to construct the encoded standard basis states in a CSS code.

Consider an arbitrary codeword $\mathbf{u} \in \mathcal{C}_1$, and construct the quantum state

$$|\overline{\mathbf{u}}\rangle = \frac{1}{\sqrt{|\mathcal{C}_2{}^{\perp}|}} \sum_{\mathbf{w} \in \mathcal{C}_2{}^{\perp}} |\mathbf{u} + \mathbf{w}\rangle \tag{3}$$

Because $\mathcal{C}_2{}^{\perp} \leqslant \mathcal{C}_1$, the vector $\mathbf{u} + \mathbf{w}$ is in $\mathcal{C}_1$ for all $\mathbf{w} \in \mathcal{C}_2{}^{\perp}$. As a result, $\mathbf{u} + \mathbf{w}$ will satisfy all of the parity checks of $\mathcal{C}_1$: the stabilizer generators from $\mathcal{C}_1$ all have eigenvalue $+1$ for $|\mathbf{u} + \mathbf{w}\rangle$, and so it will leave the encoded state $|\overline{\mathbf{u}}\rangle$ undisturbed.

If $\mathbf{u} \in \mathcal{C}_2{}^{\perp}$, the state $|\overline{\mathbf{u}}\rangle$ will actually be the same as $\big|\overline{00\cdots0}\big\rangle$: adding $\mathbf{u}$ to the vectors $\mathbf{w}$ in the sum of equation 3 just permutes the terms, which is the same as setting $\mathbf{u} = 00\cdots0$. In fact, for any two vectors $\mathbf{u}, \mathbf{v} \in \mathcal{C}_1$, we have

$$|\overline{\mathbf{u}}\rangle = |\overline{\mathbf{v}}\rangle \quad \Longleftrightarrow \quad \mathbf{u} + \mathcal{C}_2{}^{\perp} = \mathbf{v} + \mathcal{C}_2{}^{\perp} \quad \Longleftrightarrow \quad \mathbf{u} - \mathbf{v} \in \mathcal{C}_2{}^{\perp} \tag{4}$$

So, we can't just take codewords $u \in \mathcal{C}_1$ to encode standard basis states. However, we can take (representative elements of) cosets in the quotient set $\mathcal{C}_1 \,/\, \mathcal{C}_2{}^{\perp}$ to encode standard basis states. As we

noted above, there are exactly the right number of these cosets to encode the 2^k standard basis states that we need to encode.

We haven't yet shown that this encoding is preserved by the stabilizer generators contributed by $\mathcal{C}_2$. We can do this most easily by observing what happens when we apply Hadamards to each qubit:

$$\begin{aligned}\frac{1}{\sqrt{|\mathcal{C}_2{}^{\perp}|}}\sum_{\mathbf{w}\in\mathcal{C}_2{}^{\perp}}|\mathbf{u}+\mathbf{w}\rangle &\xrightarrow{H^{\otimes n}} \frac{1}{\sqrt{2^n\,|\mathcal{C}_2{}^{\perp}|}}\sum_{\mathbf{h}\in\mathbb{Z}_2{}^n}\sum_{\mathbf{w}\in\mathcal{C}_2{}^{\perp}}(-1)^{\mathbf{h}\cdot(\mathbf{u}+\mathbf{w})}\,|\mathbf{h}\rangle \\ &= \frac{1}{\sqrt{2^n\,|\mathcal{C}_2{}^{\perp}|}}\sum_{\mathbf{h}\in\mathbb{Z}_2{}^n}\left[(-1)^{\mathbf{h}\cdot\mathbf{u}}\sum_{\mathbf{w}\in\mathcal{C}_2{}^{\perp}}(-1)^{\mathbf{h}\cdot\mathbf{w}}\,|\mathbf{h}\rangle\right] \qquad (5)\\ &= \frac{1}{\sqrt{|\mathcal{C}_2|}}\sum_{\mathbf{h}\in\mathcal{C}_2}(-1)^{\mathbf{h}\cdot\mathbf{u}}\,|\mathbf{h}\rangle\end{aligned}$$

To check that the original state was preserved by any stabilizer generator M contributed by $\mathcal{C}_2$, we can test whether this state is preserved by the operator $M' = H^{\otimes n}MH^{\otimes n}$, which will be the same operator except with Pauli Z operations at each qubit rather than X operations. Because each codeword $k \in \mathcal{C}_2$ satisfies the parity checks of $\mathcal{C}_2$, the state $H^{\otimes n}\,|\overline{\mathbf{u}}\rangle$ will have $+1$ eigenvalue with each operator M'; then, the generators from $\mathcal{C}_2$ also preserve codewords. Then, the states $|\overline{\mathbf{u}}\rangle$ are code states of the CSS code.

It may seem that the construction presented here is unbalanced, in that $\mathcal{C}_1$ and $\mathcal{C}_2$ play significantly different roles. This is not quite true, however: if we take a state in the conjugate basis of this construction (that is, where the encoded state is just $\frac{1}{\sqrt{2}}\big(\,|0\rangle \pm |1\rangle\big)^{\otimes k}$), the resulting state will be exactly the same as a codeword created in the standard basis, where we switch the roles of $\mathcal{C}_1$ and $\mathcal{C}_2$ in the construction. So, the two "different" constructions can be related to one another by a change in basis. This is a nice symmetry of the construction, although it would not have been necessary to have it in order for the code to work.

2 Constructing more general stabilizers using $GF(4)$

The CSS construction for quantum codes is very useful, but are the any others? It turns out that another useful construction can be found by considering classical error correction codes — but instead of using binary vectors for codewords, we use vectors over the finite field $GF(4)$.

2.1 Describing $GF(4)$

A finite field $\mathbb{F}$ is a finite set with two operations, addition and multiplication. The set $\mathbb{F}$ is an abelian group under the addition operation (we call the additive identity 0), and the set $\mathbb{F} \setminus \{0\}$ is also an abelian group under the multiplication operaton (we call the multiplicative identity 1). We also have the requirement that multiplication distributes over addition in the usual way. It turns out that we can create a finite field of size p^r, for any prime p and any $r \in \mathbb{Z}^+$, and that this field will be unique; we call such a field $GF(p^r)$. Whenever $r = 1$, the resulting field is just the integers modulo p, $\mathbb{Z}_p$.

The field $GF(4)$ is fairly easy to describe. The four elements are 0 and 1 (which behave much the same as in $\mathbb{Z}_2$), and two additional elements ω and ω^2. The arithmetic of the field can be described

entirely with two equations. First of all, just as in $\mathbb{Z}_2$, we have $1+1=0$: from this, we can also obtain $x+x=0$ for any x in the field.[1] Aside from that, we have one more equation:

$$1+\omega+\omega^2=0\,.$$

These two equations characterize $GF(4)$ entirely: for instance, we can also deduce

$$\begin{aligned} 1+\omega &= \omega^2 & \qquad \omega+\omega^2 &= 1 \\ 1+\omega^2 &= \omega & \qquad \omega^3 &= 1\,. \end{aligned}$$

2.2 Using $GF(4)$ to describe Pauli operators

The reason for choosing $GF(4)$ is simple: there are the same number of elements of $GF(4)$ as there are single-qubit Pauli operators. Then, instead of using classical linear codes over $\{0,1\}=\mathbb{Z}_2$, we can use a classical linear code over $GF(4)$, and map

$$\left.\begin{aligned} 0 &\longmapsto I \\ 1 &\longmapsto X \\ \omega &\longmapsto Z \\ \omega^2 &\longmapsto Y \end{aligned}\right\} \text{(arbitrary order)}$$

to form Pauli operators in the same way as we have been doing for parity check matrices to obtain Pauli operators. Just as with binary vectors, adding two vectors over $GF(4)$ corresponds to multiplying two Pauli operations.

In the $2n$-dimensional representation of Pauli operators, we used the symplectic inner product to describe when two operators would commute. In this $GF(4)$ representation, we have something which plays the same role, and which makes good use of the algebraic machinery of the field.

Because $x+x=0$ for any $x\in GF(4)$, the map $x\longmapsto x^2$ is a linear transformation (in the sense of respecting additions, and scalar multiplication by elements of $\mathbb{Z}_2$). We sometimes refer to x^2 as the *conjugate* of x, and write $\overline{x}=x^2$. We can extend this in a natural way to vectors over $GF(4)$: for instance, if we have a vector $\mathbf{v}=[v_1\ \ v_2\ \ \cdots\ \ v_n]$, we would define

$$\overline{\mathbf{v}} \;=\; [\overline{v_1}\ \ \overline{v_2}\ \ \cdots\ \ \overline{v_n}] \;=\; \left[{v_1}^2\ \ {v_2}^2\ \ \cdots\ \ {v_n}^2\right]\,. \tag{6}$$

This is also clearly a linear transformation. As well, for any $x\in GF(4)$, we define the *trace* of x by the formula

$$\mathrm{tr}(x)=x+\overline{x}=x+x^2\,. \tag{7}$$

Then, the symplectic inner product can be described by the formula

$$\left(\mathbf{u}\,\middle|\,\mathbf{v}\right) \;=\; \mathrm{tr}(\mathbf{u}\cdot\overline{\mathbf{v}}) \;=\; \sum_j \mathrm{tr}(u_j\overline{v_j})\,, \tag{8}$$

so that the Pauli operators arising from $\mathbf{u}$ and $\mathbf{v}$ commute exactly when $\mathrm{tr}(\mathbf{u}\cdot\overline{\mathbf{v}})=0$.

[1] This is actually common to all fields of size 2^r: it's a special case of the rule $[\forall x\in GF(p^r)]\,:\,px=0$, which holds for prime p in general.

To show this, first notice that $\mathrm{tr}(0) = \mathrm{tr}(1) = 0$, and $\mathrm{tr}(\omega) = \mathrm{tr}(\omega^2) = 1$. Then, for the j^{th} component of two vectors $\mathbf{u}$ and $\mathbf{v}$, $\mathrm{tr}(u_j \overline{v_j}) = 0$ if the Pauli operators corresponding to u_j and v_j commute. On the one hand, if the Pauli operators *do* commute, then either one of them is the identity and $u_j \overline{v_j} = 0$, or $u_j = v_j \neq 0$, in which case $u_j \overline{v_j} = {u_j}^3 = 1$: either way, the trace will be zero. Otherwise, it is easy to show that $u_j \overline{v_j}$ will be either ω or ω^2, both of which have trace one. In order for the Pauli operators arising from u and v to commute, we require that the parity of these component-wise traces be even, or equivalently, that the sum in $GF(4)$ be equal to zero. Then, the formula of Equation 8 plays the same role as the symplectic product for the $2n$-bit representation of the syndrome generators.

2.3 Building the stabilizer

The stabilizer is again built from the parity check matrix of the classical code: however, this time there is only one classical code, and it's based on the field $GF(4)$. This has a significant impact on how the stabilizer generators can be found. To illustrate how stabilizer generators could be chosen, we'll consider the case where we want to build a $[[5, 1, 3]]$ quantum code (which we know does exist) from a $GF(4)$ code.

Recall from the last lecture that the distance of the code is equal to the size of the *smallest* linearly dependent collection of columns from the stabilizer. Then, we want to come up with a parity check matrix over $GF(4)$ such that any two columns are linearly independent over $\mathbb{Z}_2$, but where some three column collection is linearly dependent. Because we need the resulting stabilizer to be abelian, we also require the two rows to have a symplectic inner product of zero. For reasons that we will see below, we actually require the two row vectors $\mathbf{u_1}$ and $\mathbf{u_2}$ to be orthogonal to each other *and to themselves*: that is, to have $\mathbf{u_i} \cdot \overline{\mathbf{u_j}} = 0$ for $i, j \in \{1, 2\}$. As a result, the symplectic inner product of the two rows will automatically be zero.

Perhaps the simplest way to try to fulfill these requirements is to start with only two rows: because the row-space will have dimension at most 2, certainly there exists a collection of three columns which is linearly dependent (any three columns will do in this case). We can then focus on the independance of any pair of columns, and on the orthogonality conditions. The following two rows satisfy our requirements:

$$\begin{aligned} \mathbf{u_1} &= \left[\begin{array}{ccccc} 0 & 1 & 1 & 1 & 1 \end{array}\right], \\ \mathbf{u_2} &= \left[\begin{array}{ccccc} 1 & 0 & 1 & \omega & \omega^2 \end{array}\right]. \end{aligned}$$

Because $n = 5$ is relatively small, we can essentially do this by trial and error, although some more systematic approaches do exist.[(2)] Because we're constructing a $GF(4)$ code, the two rows will be linearly independent over $GF(4)$ as well as being additively independent — this will be useful in a moment.

The rows $\mathbf{u_1}$ and $\mathbf{u_2}$ alone form the complete parity check matrix for the classical $GF(4)$ code, and also form a basis for the dual code. However, the dual code contains (amoung other things) the vectors $\omega\,\mathbf{u_1}$, $\omega\,\mathbf{u_2}$, $\omega^2\,\mathbf{u_1}$, and $\omega^2\mathbf{u_2}$. It's easy to verify that scalar multiplication of vectors by ω or ω^2 doesn't correspond to any Pauli operation: so, in order to have a one-to-one correspondance between the dual code and stabilizer elements in the quantum code, we need to include the Pauli operations arising from

[(2)] For example, the above rows can be generated by considering polynomials over $GF(4)$ with leading coefficient 1 and degree less than 2, until we produce enough of them for the number of qubits we have in the code.

$\omega\,\mathbf{u_1}$, $\omega\,\mathbf{u_2}$, $\omega^2\,\mathbf{u_1}$, and $\omega^2\,\mathbf{u_2}$. So, consider the vectors

$$\begin{aligned}
\mathbf{u_3} &= \omega\,\mathbf{u_1} &&= \begin{bmatrix} 0 & \omega & \omega & \omega & \omega \end{bmatrix} \\
\mathbf{u_4} &= \omega\,\mathbf{u_2} &&= \begin{bmatrix} \omega & 0 & \omega & \omega^2 & 1 \end{bmatrix} \\
\mathbf{u_5} &= \omega^2\,\mathbf{u_1} &&= \begin{bmatrix} 0 & \omega^2 & \omega^2 & \omega^2 & \omega^2 \end{bmatrix} \\
\mathbf{u_6} &= \omega^2\,\mathbf{u_2} &&= \begin{bmatrix} \omega^2 & 0 & \omega^2 & 1 & \omega \end{bmatrix}.
\end{aligned}$$

Because we chose $\mathbf{u_1}$ and $\mathbf{u_2}$ to be linearly independent over $GF(4)$, it's fairly easy to show that $\mathbf{u_1}$ through $\mathbf{u_4}$ are additively independent. However, because $\omega^2 = 1 + \omega$, we have $\mathbf{u_5} = \mathbf{u_1} + \mathbf{u_3}$ and $\mathbf{u_6} = \mathbf{u_2} + \mathbf{u_4}$; we don't need to explicitly consider the last two vectors, as they're already in the span of the first four. It's also easy to verify that $\mathbf{u_i} \cdot \overline{\mathbf{u_j}} = 0$ for any $1 \leqslant i, j \leqslant 4$, given that the first two are orthogonal to each other and to themselves.

Having obtained these rows, we obtain the stabilizer generators

$$\left.\begin{array}{ccccc} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & \omega & \omega^2 \\ 0 & \omega & \omega & \omega & \omega \\ \omega & 0 & \omega & \omega^2 & 1 \end{array}\right\} \longmapsto \left\{\begin{array}{ccccc} I & X & X & X & X \\ X & I & X & Z & Y \\ I & Z & Z & Z & Z \\ Z & I & Z & Y & X \end{array}\right.$$

Note that there are four of these: the resulting *quantum* code then does encode one qubit.

2.4 Distance of the quantum code

It would be nice to verify that the distance of the quantum code that we have built is the correct size: for instance, in the example above, that fixing the distance of the classical $GF(4)$ code to be 3 is enough to give us a quantum code of distance at least 3. To prove this, we must consider the minimum weight of an element of $N(\mathcal{S}) \setminus \mathcal{S}$. In the $GF(4)$ code, this corresponds to finding the minumum Hamming weight of a vector $\mathbf{v}$ such that $\mathrm{tr}(\mathbf{u_j} \cdot \overline{\mathbf{v}}) = 0$ for each of the parity check rows $\mathbf{u_j}$.

Let $\mathcal{C}$ be the classcial $GF(4)$ code, and consider any $\mathbf{u} \in \mathcal{C}^\perp$ (i.e. a vector generated by the rows of the parity check matrix). Let $\mathbf{v}$ be a vector which produces a Pauli operator in $N(\mathcal{S})$: then, we have $tr(\mathbf{u} \cdot \overline{v}) = 0$. Let $\mathbf{u} \cdot \overline{\mathbf{v}} = \alpha + \omega\beta$: then, we know that

$$0 = \mathrm{tr}(u \cdot \overline{v}) = \mathrm{tr}(\alpha) + \mathrm{tr}(\beta\omega) = \beta\,, \tag{9}$$

because of our assumption on $\mathbf{v}$. However, $\omega^2\mathbf{u}$ is also an element of the dual code $\mathcal{C}^\perp$, so we also know that $\mathrm{tr}(\omega^2\mathbf{u} \cdot \overline{\mathbf{v}}) = 0$. Then, we have

$$0 = \mathrm{tr}(\omega^2\mathbf{u} \cdot \overline{\mathbf{v}}) = \mathrm{tr}(\omega^2\alpha) + \mathrm{tr}(\beta) = \alpha\,. \tag{10}$$

Thus, we have $\mathbf{u} \cdot \overline{\mathbf{v}} = 0$, so that $\mathbf{v}$ is orthogonal to $\mathbf{u}$. Then, a vector $\mathbf{v}$ whose corresponding Pauli operator is an element of $N(\mathcal{S})$ must be orthogonal to all of the rows of the parity check matrix for the code.[3] That is, $\mathbf{v}$ must be an element of the $GF(4)$ code itself. Because the code $\mathcal{C}$ has distance d, we can deduce that the minimum weight of $E \in N(\mathcal{S})$ is d, so the distance of the quantum code is at least d.

[3] Because the stabilizer $\mathcal{S}$ is itself a subset of $N(\mathcal{S})$, we can deduce that all the vectors of the parity check matrix must be orthogonal to every other row, and orthogonal to themselves: this is where that condition came from in producing the parity check matrix earlier.