REPRESENTING ERROR CORRECTING CODES AS THE IDEALS OF RINGS

EMMA CARDWELL, KATHERINE TAYLOR

1. Introduction

In a perfect world, we would have perfect data transmission. In our world, this is not the case. Every time we send a message, there is a chance of something going wrong. For the transmission of digital information in particular, the accidental swapping of a single 1 or 0 could alter the meaning of the message. Of course, we want to minimize the likelihood of misunderstandings. When communicating, we often do this unconsciously. For example, if you are talking to someone standing a distance away, that person could mishear your message. To counteract this, you might move closer to the person before talking. Additionally, if the other person doesn't understand what you're saying, they can ask you to repeat yourself. In digital transmissions, where the only information exchanged is packaged in binary strings, it can be much more difficult to notice when an error has occurred. For example, if you are sending navigation directions to a drone, and the drone receives a message saying "turn right" when you actually sent a command to turn left, the drone doesn't have any way of determining that the message it received was erroneous. What we *can* do is encode our messages in such a way that the the receiving end has a better chance of noticing any errors. This is where error correcting codes are very useful. In this paper, we will discuss cyclic error correcting codes, their relation to rings, and explain how to represent Hamming codes as ideals of a ring.

Definition 1.1 (code)**.** *A code of blocklength n is a set of m* codewords *which are n-tuples of elements of some finite field $\mathbb{F}_q$, along with a one-to-one correspondence between the codewords and a set of m possible* messages*.*

Remark 1.2. Note the distinction between "codeword" (an individual n-tuple representing some message) and "code" (some set of related codewords).

Messages will usually also be tuples of elements of $\mathbb{F}_q$, of some length $k < n$. Codewords look like this: $(s_0, s_1, s_2, \ldots, s_n)$, where $s_i \in \mathbb{F}_q$. We may also write $s_0 s_1 \ldots s_n$ to denote the same codeword.

A code is useful because it enables us to send information in a different form. An especially useful sort of code is one that can resist errors in its transmission. That is, supposing that at most d bits of a given codeword might be corrupted between sending and receiving the codeword, we'd like to make a code such that the receiver can still determine with certainty which codeword was originally sent. Some types of errors that can occur include the insertion/deletion of bits and the flipping of bits. In our paper, we will focus on codes that can detect and correct the flipping of bits only.

Definition 1.3 (error-correcting code)**.** *An error-correcting code is a code with the property that, if up to a certain number d of bits are incorrect, the receiver can still be certain of which codeword was sent.*

A very simple error-correcting code is the repetition code of length 3. In this code, we have a set of q^k messages, where each message is a k-tuple of elements of $\mathbb{F}_q$. (Think of 2^k binary messages

Date: June 34, 2020.

for a simple example.) The codewords are each $3k$-tuples of elements of $\mathbb{F}_q$, where the codeword corresponding to a message is just that message with each bit repeated 3 times. The one-to-one correspondence is:

$$s_0 s_1 \dots s_k \to s_0 s_0 s_0 s_1 s_1 s_1 \dots s_k s_k s_k$$

For example, take $k = 6$ and $q = 2$. We have 2^6 possible messages which are all 6-tuples of 1s and 0s. The message 010001 is sent to the codeword 000111000000000111.

This code can correct at most one error ($d = 1$). For example, if we receive 000111001000000111 while using the example code from above and we know that at most one bit is incorrect, then the original codeword can only have been 000111000000000111. If there are at most two incorrect bits, we can no longer always correct errors, but we can still detect them: we can't know whether 000111001000000111 is a one-bit corruption of 000111000000000111 or a two-bit corruption of 000111111000000111, but we know that something has gone wrong.

We consider an error-correcting code "good" when it can correct a high number of errors without being too long relative to the length of the messages. By this standard, repetition codes are pretty bad: this one triples the length of the message and only corrects one error.

Codes are fine by themselves, but it's more fun when they have some algebraic structure. We'll get into this more with cyclic codes, but here are some preliminary definitions.

Definition 1.4 (linear code)**.** *An error-correcting code where codewords form a vector space is called a linear code.*

In a linear code, we have a notion of "adding" codewords, and we must also be able to scalar multiply them by the elements of some (usually finite) field. The obvious way of making a code consisting of n-tuples of elements of $\mathbb{F}_q$ into a vector space is by treating the n-tuples as vectors of dimension n over $\mathbb{F}_q$, and that's exactly what we do. More formally, addition is defined by

$$(s_0, s_1, \dots, s_n) + (r_0, r_1, \dots r_n) = (s_0 + r_0, s_1 + r_1, \dots, s_n + r_n)$$

and scalar multiplication is defined by

$$r(s_0, s_1, \dots, s_n) = (rs_0, rs_1, \dots, rs_n)$$

Just like in a usual vector space, there is a "zero" codeword $(0, 0, \dots, 0)$ and each codeword has an additive inverse.

2. Construction of Hamming Codes

We will look at a common type of error correcting code, the Hamming code. Hamming codes can correct 1 error and possibly detect up to 2 errors (this depends on the length of the codewords). Some places where Hamming codes are implemented include computer memory chips and satellite communication hardware. We can generate a Hamming code from any number of bits. Since Hamming codes correct/detect a set number of errors, regardless of their length, shorter Hamming codes are more likely to fix errors. We're using the Hamming code as an example because it can be defined both with and without abstract algebra. First, let's define bitwise addition:

Definition 2.1 (bitwise sum)**.** *The bitwise sum (or bitwise XOR) of two bits is defined as follows:*

$$a \oplus b = \begin{cases} 0 & \text{if } a = b = 0 \text{ or if } a = b = 1 \\ 1 & \text{if } a = 1, b = 0 \text{ or if } a = 0, b = 1 \end{cases}$$

The bitwise sum is essentially addition modulo 2.

Here's how we can construct a Hamming code for a message in binary:

(1) Label each bit position in the codeword with the binary representation of its position. (Label the bit in position 1 as 01, position 2 as 10, position 3 as 11, etc). We will denote the n^{th} bit in our Hamming codeword as h_n.
(2) Reserve the bits that correspond to powers of 2 (these have binary representation 100...) for *parity bits*. The bit positions that don't correspond to powers of 2 are our *data bits*.
(3) We "fill in" our data bits by slotting in the bits we want to transmit. If we let d_n denote the n^{th} bit in the message we want to transmit, then $d_1 = h_3$, $d_2 = h_5$, $d_3 = h_6$, $d_4 = h_7$, $d_5 = h_9$, etc.
(4) We construct the parity bits as following:
    (a) For parity bit 1, take the bitwise sum of the other bits in positions whose binary representations end in 1: $p_1 = h_3 \oplus h_5 \oplus h_7 \oplus \oplus h_9 \oplus \ldots$.
    (b) For parity bit 2, take the bitwise sum of the other bits in positions whose binary representations have a 1 in the second to last position: $p_2 = h_3 \oplus h_6 \oplus h_7 \oplus h_{10} \oplus h_{11} \oplus \ldots$
    (c) For parity bit 3, take the bitwise sum of the other bits in positions whose binary representations have a 1 in the third to last position: $p_3 = h_5 \oplus h_6 \oplus h_7 \oplus h_{12} \oplus h_{13} \oplus \ldots$
    (d) For parity bit n, take the bitwise sum of the other bits in positions whose binary representations have a 1 in the n^{th} to last position.
(5) We fill in our parity bits by assigning parity bit n to the position corresponding to 2^{n-1}, so $p_1 = h_1$, $p_2 = h_2$, $p_3 = h_4$, $p_4 = h_8$, etc.

Let's actually construct a Hamming code for a message with length 4: 1011.

First, let's put in our data bits:

h_1	h_2	h_3	h_4	h_5	h_6	h_7
		1		0	1	1

Next, we have to construct our parity bits:

$$p_1 = h_3 \oplus h_5 \oplus h_7 = 1 \oplus 0 \oplus 1 = 0$$
$$p_2 = h_3 \oplus h_6 \oplus h_7 = 1 \oplus 1 \oplus 1 = 1$$
$$p_3 = h_5 \oplus h_6 \oplus h_7 = 0 + 1 + 1 = 0$$

Plugging in our parity bits, we get:

h_1	h_2	h_3	h_4	h_5	h_6	h_7
0	1	1	0	0	1	1

So our codeword is 0110011.

3. Matrix Generation of Hamming Codes

While it is relatively simple to construct Hamming codes manually, we can use some matrices to make it even easier. For the (7,4) Hamming code, we can construct a codeword for the message $d_1d_2d_3d_4$ by taking the product of the following matrices:

$$(1) \qquad \begin{bmatrix} d_1 & d_2 & d_3 & d_4 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Notice that evaluating this matrix results in a 7 bit codeword[1]:

$$[d_1, d_2, d_3, d_4, d_1 + d_2 + d_4, d_1 + d_3 + d_4, d_2 + d_3 + d_4] \tag{2}$$

The last three bits are equivalent to our three parity check bits! Recall that $h_3 = d_1, h_5 = d_2, h_6 = d_3$ and $h_7 = d_4$, so we have

$$p_1 = h_3 \oplus h_5 \oplus h_7 = d_1 \oplus d_2 \oplus d_4$$
$$p_2 = h_3 \oplus h_6 \oplus h_7 = d_1 \oplus d_3 \oplus d_4$$
$$p_3 = h_5 \oplus h_6 \oplus h_7 = d_2 \oplus d_3 \oplus d_4$$

To construct a generator matrix for any binary Hamming code, start by constructing an identity matrix with the same number of rows as the number of bits in the message to be encoded. Next, for each parity bit, adjoin a column to the right of the matrix. For each column $\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_j \end{bmatrix}$, let $c_i = 1$ if d_i is part of the XOR sum of message bits for the corresponding parity check bit and let $c_i = 0$ if it isn't.

4. Decoding Hamming Codes

We know the positions of the data bits and the parity bits, so, in theory, we could just look at the data bits. The issue is, the whole purpose of error correcting codes is to fix errors. So, while we could just observe the relevant data bits and ignore the parity bits, this method of "decoding" doesn't help us detect or correct errors.

There are actually several ways to decode Hamming codes. The first way is to compare the received codeword to a list of all possible Hamming codewords of the same length and their decoded versions. If the received codeword matches a codeword on the list, then great! If the received codeword doesn't match any of the codewords on the list, then just choose the codeword that is most similar to it, and assume that the decoded version of that codeword is the original message. This is not often used because, especially for longer Hamming codes, checking through long lists of every possible codeword is rather time consuming.

The second way to decode Hamming codes is a bit more magical. Let's look at the decoding matrix for (7,4) Hamming codes. Suppose we have a codeword $h_1h_2h_3h_4h_5h_6h_7$, where h_5, h_6, h_7 are the

[1]The order of the data/parity bits is different here - instead of placing the parity bits at positions corresponding to powers of two, this matrix places them all after the data bits. We can swap the columns of the matrix in order to produce the correct arrangement of data/parity bits, but for the sake of keeping things simple, more rigorous confirmation of this is left as an exercise to the reader.

parity bits[2]. To decode it, evaluate the product of the following matrices:

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix} \tag{3}$$

Evaluating this gives us a 3-bit vector, $s_1s_2s_3$, called the *error syndrome* (or *syndrome*). Here's the really magical part: if exactly one error occurred, the syndrome will tell us exactly what position the faulty bit is located at in our encoded word. Since we're working in binary, we can then fix the error by fixing the bit in the specified position. After that, we can reconstruct the original message by just considering the data bits in the encoded message.

Let's look at an example. Suppose we want to decode 1011010 (this is the codeword we constructed earlier, just with the bits rearranged so the parity bits are all at the end). But suppose one bit got flipped, and instead of receiving 1011010, we received 1001010 (the third bit got flipped). We can use a check matrix to help us find the syndrome:

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} s_1 = h_1 + h_2 + h_4 + h_5 \\ s_2 = h_1 + h_3 + h_4 + h_6 \\ s_3 = h_2 + h_3 + h_4 + h_7 \end{bmatrix}$$

We find that

$$\begin{aligned} s_1 &= 1 \oplus 0 \oplus 1 \oplus 0 = 0 \\ s_2 &= 1 \oplus 0 \oplus 1 \oplus 1 = 1 \\ s_3 &= 0 \oplus 0 \oplus 1 \oplus 0 = 1 \end{aligned}$$

Guess what? 011 is the binary representation of 3! This tells us that the error occurred in the 3rd spot, so we can change 1001010 $\Rightarrow$ 1011010. This tells us the correct encoded version of our message, and we can decode it simply by ignoring the parity bits.

The reason why this check matrix works is very similar to why the generator matrix works. Just evaluate the "hard" way and it will make sense.

5. Finite Fields

In order to focus on codes and rings, we've left out some proofs about finite fields. But... Here Are Some FaCtS[3]:

[2]Just as before, we can rearrange this matrix to fit the convention of placing parity bits at positions corresponding to powers of two. Again, we leave the verification of this as an exercise for the reader.

[3]The verification of these facts is left as an exercise to the reader. Alternatively, they can be found in [1] and [2].

(1) Every finite field is of the form $\mathbb{F}_{p^k}$, meaning it has p^k elements, with p prime. For every prime p and integer k, there is a finite field $\mathbb{F}_{p^k}$. We will sometimes denote finite fields by $\mathbb{F}_q$, where q might be prime or a prime power.
(2) All finite fields of the same order are isomorphic.
(3) The *characteristic* of $\mathbb{F}_{p^k}$ is p, which is also the size of its smallest subfield $\mathbb{F}_p$. For any $\alpha \in \mathbb{F}_{p^k}$, $p\alpha = 0$.
(4) $\mathbb{F}_{p^n}$ is a subfield of $\mathbb{F}_{p^k}$ iff $n|k$.
(5) Every finite field $\mathbb{F}_{p^k}$ has at least one primitive element (element of multiplicative order $p^k - 1$).
(6) The elements of $\mathbb{F}_{p^k}$ are precisely the roots of $x^{p^k} - x$.
(7) If $\alpha \in \mathbb{F}_{p^k}$ and $\mathbb{F}_{p^n}$ is a subfield of $\mathbb{F}_{p^k}$, then α has an irreducible minimal polynomial over $\mathbb{F}_{p^n}$ with coefficients in $\mathbb{F}_{p^n}$.
(8) A finite field forms a vector space over any of its subfields.
(9) Finite fields have a Galois theory. The Galois group of the extension $\mathbb{F}_{p^k}/\mathbb{F}_p$ is cyclic and generated by the automorphism $\sigma(\alpha) = \alpha^p$. The Galois group of the extension $\mathbb{F}_{p^k}/\mathbb{F}_{p^m}$ is cyclic and generated by the automorphism $\sigma(\alpha) = \alpha^{p^m}$.

6. Cyclic Codes with Rings

Cyclic codes are a type of linear error-correcting code with an interesting representation that involves polynomial rings over finite fields.

Cyclic codes are defined as codes in which any cyclic shift of a codeword is also a codeword. A cyclic code of block length n can be represented using the ring $R_n = \mathbb{F}_q[x]/(x^n - 1)$. Each codeword corresponds to a polynomial in the ring, with the correspondence given by

$$c_0c_1c_2 \dots c_{n-1} \iff c_0 + c_1x + c_2x^2 + \cdots + c_{n-1}x^{n-1}$$

Notice that, because we are quotienting by $(x^n - 1)$, multiplication by x corresponds to a rightward shift $c_0c_1 \dots c_n \to c_nc_0 \dots c_{n-1}$:

$$\begin{aligned} &x(c_0 + c_1x + c_2x^2 + \cdots + c_{n-1}x^{n-1}) \\ &= c_0x + c_1x^2 + c_2x^3 + \cdots + c_{n-1}x^n \\ &= c_n + c_0x + c_1x^2 + \cdots + c_{n-2}x^{n-1} \end{aligned}$$

A cyclic code of block length n, since it is a linear code, is closed under addition. Since it is closed under multiplication by any power of x and any element of $\mathbb{F}_q$, a cyclic code is also closed under multiplication by any polynomial in R_n. Thus, every cyclic code over $\mathbb{F}_q$ of block length n is an ideal of R_n. But what sort of ideal?

Every cyclic code C contains a unique monic polynomial $g(x)$ of least degree (if there were two, we could subtract them in such a way to get a polynomial of lesser degree). Now consider some codeword $f(x) \in C$. We can use polynomial division to write $f(x) = h(x)g(x) + r(x)$, where the remainder $r(x)$ has degree less than $g(x)$. Since $g(x)$ is of minimal degree, that means $r(x) = 0$, and we conclude that the polynomial representation of every codeword in C is divisible by $g(x)$. Thus, C can be represented by a polynomial ideal, $(g(x))$. We call $g(x)$ the *generator* of C because every codeword in C can be generated from it. We know that the code is exactly the ideal $(g(x))$ because cyclic codes are closed under multiplication.

A natural question to ask is, what sort of polynomials can be generators? That question determines exactly what the cyclic codes of blocklength n over $\mathbb{F}_q$ are, and fortunately it can be answered fully.

We use polynomial division again: $x^n - 1 = h(x)g(x) + r(x)$, where $r(x)$ has degree less than $g(x)$ and is therefore 0. So, a generator must be a factor of $x^n - 1$. We start by developing some theory that will help us factor $x^n - 1$.

The q-cyclotomic cosets modulo n partition the integers modulo n, meaning they split the integers into disjoint sets sort of like the cosets of a subgroup. For q relatively prime to n, the cyclotomic cosets mod n are the "cosets" of the sequence $1, q, q^2, q^3, \ldots q^{r-1}$ where $q^r \equiv 1 \pmod{n}$. The q-cyclotomic coset of $m \pmod{n}$ is $C_m = \{m, mq, mq^2, \ldots mq^{r-1}\}$, where each term in the sequence is evaluated modulo n.

For example, the 3-cyclotomic cosets mod 14 are:

$$C_1 = \{1, 3, 9, 13, 11, 5\}$$
$$C_2 = \{2, 6, 4, 12, 8, 10\}$$
$$C_7 = \{7\}$$
$$C_0 = \{0\}$$

The 2-cyclotomic cosets mod 15 are:

$$C_1 = \{1, 2, 4, 8\}$$
$$C_3 = \{3, 6, 12, 9\}$$
$$C_5 = \{5, 10\}$$
$$C_7 = \{7, 14, 13, 11\}$$
$$C_0 = \{0\}$$

Now, let's see why cyclotomic cosets are useful. Suppose we have the polynomial $x^n - 1$ in $\mathbb{F}_q[x]$ where q and n are relatively prime. To learn more about how to factor it, we want to find some field extension $\mathbb{F}_{q^t}/\mathbb{F}_q$ such that $x^n - 1$ has a root α in $\mathbb{F}_{q^t}$. This happens precisely when $\mathbb{F}_{q^t}$ contains an nth root of unity, which, since nonzero elements of $\mathbb{F}_{q^t}$ are roots of $x^{q^t-1} - 1$, happens precisely when $n|(q^t - 1)$. Suppose we've found such an $\mathbb{F}_{q^t}$, so let $\alpha \in \mathbb{F}_{q^t}$ be an nth root of unity. Then, take some $m < n$ and construct the set

$$S_m = \{\alpha^m, \alpha^{mq}, \alpha^{mq^2}, \alpha^{mq^3}, \ldots, \alpha^{mq^{n-1}}\}$$

where we raise α to every element of C_m. If we do this construction for every q-cyclotomic coset mod n, the S_m's will partition the powers of α.

Now consider the minimal polynomial $f(x)$ with a root α^m and coefficients in $\mathbb{F}_q$. We can write

$$f(\alpha^m) = a_0 + a_1\alpha^m + \cdots + a_r\alpha^{mr} = 0.$$

Now recall that $\sigma(x) = x^q$ for all $x \in \mathbb{F}_{q^t}$ is an automorphism of $\mathbb{F}_{q^t}$ which fixes $\mathbb{F}_q$[4] and therefore all of the a_i's. We have

$$\sigma(f(\alpha^m)) = \sigma(a_0) + \sigma(a_1\alpha^m) + \cdots + \sigma(a_r\alpha^{mr}) = a_0 + a_1\alpha^{mq} + \cdots + a_r\alpha^{mrq} = f(\alpha^{mq})$$

but since that's all still equal to 0, we can conclude that α^{mq} is also a root of $f(x)$. In fact, through repeated application of σ, we see that the roots of $f(x)$ are precisely the elements of S_m. We have

$$f(x) = \prod_{i \in C_m} (x - \alpha^i).$$

This means $f(x)$ divides $x^n - 1$, since every α^i is a root of $x^n - 1$. We can construct a minimal polynomial like this for the elements of every S_m, so $x^n - 1$ can be factored into minimal polynomials

[4]Following from one of our Field FaCtS, $x^q = x$ for all $x \in \mathbb{F}_{q^t}$

where each minimal polynomial corresponds to some C_m and has roots equal to the elements S_m. For example, the factorization of $x^{15}-1$ in $\mathbb{F}_2$ is

$$x^{15}-1=(x+1)(x^4+x+1)(x^4+x^3+x^2+x+1)(x^2+x+1)(x^4+x^3+1)$$

where each factor corresponds to a 2-cyclotomic coset mod 15, with the correspondence given below.

Coset	Minimal Polynomial	Roots
C_1	(x^4+x+1)	$\alpha, \alpha^2, \alpha^4, \alpha^8$
C_3	$(x^4+x^3+x^2+1)$	$\alpha^3, \alpha^6, \alpha^{12}, \alpha^9$
C_5	(x^2+x+1)	α^5, α^{10}
C_7	(x^4+x^3+1)	$\alpha^7, \alpha^{14}, \alpha^{13}, \alpha^{11}$
C_0	$(x+1)$	α^0

TABLE 1. The 2-cyclotomic cosets mod 15.

Here α is a primitive element of $\mathbb{F}_{16}$. (Remember that $\mathbb{F}_{16}$ has a primitive 15th root of unity because $15|(16-1)$, which satisfies the requirement we set earlier.) Since the minimal polynomials are over $\mathbb{F}_2$, the only values they can take are 0 and 1. That explains why, for example, $(x+1)$ evaluated at α^0 gives $\alpha^0+1=1+1=0$.

Since we have completely factored x^n-1, we have also completely determined the possible generators: they are the polynomials $f(x)$ which are the minimal polynomial of some S_m or the product of such minimal polynomials. Every cyclic code over $\mathbb{F}_q$ of block length n corresponds to the ideal generated by one of these polynomials. Thus, if x^n-1 factors into m irreducible polynomials over $\mathbb{F}_q$, there are a total of 2^m possible generators and 2^m cyclic codes over $\mathbb{F}_q$ of blocklength n. (Here we're counting x^n-1 and 1 as generators, even though they don't produce especially useful codes.)

For example, there is a cyclic code of block length 15 over $\mathbb{F}_2$ where each element is a member of the polynomial ideal (x^4+x+1) of $\mathbb{F}_2[x]/(x^15-1)$, and also a cyclic code of block length 15 over $\mathbb{F}_2$ where each element is a member of the polynomial ideal $((x+1)(x^2+x+1))$ of R_n.

It's nice that C is generated by $g(x)$, but we can actually find a better polynomial than $g(x)$ to generate C with. Remember that we can write $x^n-1=h(x)g(x)$ for some $h(x)\in C$. Since x^n-1 is the product of distinct irreducible polynomials, $\gcd(g(x),h(x))=1$, so for some $a(x),b(x)\in\mathbb{F}_q[x]$,

$$a(x)g(x)+b(x)h(x)=1.$$

Consider $e(x)=a(x)g(x)$ as an element of R_n; specifically, consider

$$(e(x))^2=e(x)(1-b(x)h(x))=e(x)-a(x)b(x)g(x)h(x)=e(x)-a(x)b(x)(x^n-1)=e(x).$$

Thus $e(x)$ is an idempotent element of R_n. But, it has an even more special role as an element of C: take some $f(x)\in C$ and consider $e(x)f(x)$ as an element of C. Recall that if $g(x)$ is the generator of C, we can write $f(x)=g(x)k(x)$ for some $k(x)$, so we have

$$e(x)f(x)=(1-b(x)h(x))g(x)k(x)=g(x)k(x)-b(x)k(x)f(x)g(x)=f(x)-b(x)k(x)(x^n-1)=f(x)$$

Thus, $e(x)$ acts as a multiplicative unit for every element of C (though not necessarily for every element of R_n). This is great, because since $g(x)e(x)=g(x)$ and $(g(x)e(x))\subseteq(e(x))$, we have $(g(x))=C\subseteq(e(x))$. Also, since $e(x)\in C$, we have $(e(x))\subseteq C$, so $C=(e(x))$. In summary, every cyclic code can be expressed as the polynomial ideal generated by some polynomial $e(x)$ which is an idempotent element of R_n and acts as a unit in C.

7. Hamming Codes as Cyclic Codes

Before we start discussing exactly how to represent Hamming codes as cyclic codes, we would like to note that although certain positioning conventions are favored, the positioning of the parity check bits doesn't actually matter for the general Hamming code. However, when discussing Hamming codes as cyclic codes, we typically put all of the parity bits at the beginning of the codeword because not every configuration of parity bits will work. For the (7,4) Hamming code, each seven-bit codeword would be of the form $p_1p_2p_3d_1d_2d_3d_4$, where $p_1p_2p_3$ are the parity bits and $d_1d_2d_3d_4$ are the data bits.

So, if we have any codeword, for example 0101011 (yes, same codeword as before), then 1010110, generated by shifting every bit by one position, is also a codeword. For Hamming codes of specified lengths (e.g. the (7,4) Hamming code), it is easy enough to verify this is actually true for a given code by checking that each codeword shifted by one position is indeed another codeword.

Now that we know Hamming codes can be represented as cyclic codes, we can consider the associated set of polynomials used to define codewords. For example, a generator polynomial of the (7,4) Hamming code is

$$g(x) = x^3 + x + 1 \tag{4}$$

There are other polynomials that can also be used as generators (for example, $x^3 + x^2 + 1$), but we will focus on $g(x)$ for this paper. Most of what we do in the following sections can be done with other minimal polynomials.

Consider the polynomial $g(x) = x^3 + x + 1$. $g(x)$ doesn't have any roots in $\mathbb{F}_2$, but if we consider it in $R_7 = \mathbb{F}_8[x]/(x^7 - 1)$, then it has some root α in $\mathbb{F}_8$ and $g(\alpha) = 0$. Additionally, any codeword is a multiple of $g(x)$, so codeword $c(x) = f(x)g(x)$ (for some $f(x) \in R_7$) always satisfies $c(\alpha) = 0$. In fact, since $g(x)$ is equal to the minimal polynomial of α, we know that all of the polynomials $p(x) \in R[x]$ that satisfy $p(\alpha) = 0$ must be multiples of $g(x)$. We can actually think of $(g(x))$ as an ideal of R_7. We purposely choose generator polynomials such that they are the minimal polynomials of primitive roots in our desired field.

Let's formalize this argument:

Theorem 7.1. *Let's define a code, $\mathcal{C}$, with codewords $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$ where $c_0, \ldots, c_{n-1} \in \mathbb{F}_2$ are the individual bits in the codeword $\mathbf{c}$ such that*

$$c_0 + c_1\alpha + \cdots + c_{n-1}\alpha^{n-1} = 0$$

for some $\alpha \in \mathbb{F}_{2^m}$, where α is an nth root of unity. $\mathcal{C}$ is an $(n, n-m)$ binary cyclic code with generator $g(x)$, the minimal polynomial of α. (Where $n = 2^m - 1$). Furthermore, $\mathcal{C}$ is a Hamming code.

Proof. First, every codeword has length n because we defined them them as such. To show that $\mathcal{C}$ is a linear code, we need to show that the sum of any two codewords produces another codeword. Let's add two codewords $\mathbf{c}$ and $\mathbf{d}$. Since $\mathbf{c}(\alpha) = 0$ and $\mathbf{d}(\alpha) = 0$, we know that $(\mathbf{c}+\mathbf{d})(\alpha) = 0$, thus $(\mathbf{c} + \mathbf{d})$ is also a codeword.

Now, let's show that $\mathcal{C}$ is cyclic. We can generate a codeword $\mathbf{c}'$ by evaluating $\mathbf{c}' = \mathbf{c}x$, and we know that $\mathbf{c}'$ is a codeword because it satisfies $\mathbf{c}'(\alpha) = 0$. But recall that $\alpha^n = 1$ since α is an nth root of unity, so $\mathbf{c}'$ is actually equivalent to $\mathbf{c}$ with all of its bits shifted by one position:

$$(c_0 + c_1\alpha + \cdots + c_{n-1}\alpha^{n-1})\alpha = c_{n-1} + c_0\alpha + c_1\alpha^2 + \cdots + c_{n-2}\alpha^{n-2}$$

In other words, we can shift the bits in the codeword by a set position and get another codeword. This shows that $\mathcal{C}$ is cyclic.

Finally, we want to show that $\mathcal{C}$ is a Hamming code. Sorry to disappoint, but we can't do this without introducing some more terminology.

Short Interlude on Distance and Weight
The Hamming distance between two codewords is the number of positions in which their bits differ. For example, 1001 and 0111 have a Hamming distance of 3. The *minimum distance* of a code is defined as the minimum Hamming distance between any two codewords in the code. The *weight* of a codeword is the number of nonzero elements in it. For example, the weight of 0111 is 3. The *minimum weight* of a code is defined as the minimum weight of all of the nonzero codewords in the code. It turns out that the minimum distance of any linear code is equal to the minimum weight of that code[5]. Also, all codes with a minimum distance of 3 are Hamming codes[6].

Back to the Proof
We want to show that $\mathcal{C}$ is a Hamming code. It suffices to show that the minimum distance of $\mathcal{C}$ is equal to 3. To do this, let's determine the minimum weight of $\mathcal{C}$. It turns out that the minimum weight of $\mathcal{C}$ is 3. We can verify this by checking that $\mathcal{C}$ can't have a minimum weight of 1 or 2.

Let's check what happens when the minimum weight is 1. Then 1000..., 0100..., 0010..., etc. are all codewords. But 1000... can't be a codeword, as it has $c_0 = 1$, $c_1 = c_2 = \cdots = c_{n-1} = 0$, so it doesn't satisfy $c_0 + c_1\alpha + \cdots + c_{n-1}\alpha^{n-1} = 0$. Furthermore, if 0100..., 0010..., 0001..., etc. are codewords, then we must have $\alpha = \alpha^2 = \cdots = \alpha^{n-1} = 0$. This is not possible because $0 \in \mathbb{F}_2$, but $\alpha \notin \mathbb{F}_2$.

Let's see what happens if we let the minimum weight equal 2. Then we have codewords that have $c_k = c_j = 1$, all other $c_i = 0$. Since codewords must satisfy $c_0 + c_1\alpha + \cdots + c_{n-1}\alpha^{n-1} = 0$, we have $\alpha^k + \alpha^j = 0$. If we assume without loss of generality that $k > j$, then we have $\alpha^j(\alpha^{k-j} + 1) = 0$. As we explained earlier, α^j can't be equal to 0. $(\alpha^{k-j} + 1) = 0$ is also not possible. Since we're working with bitwise sums, the equation is satisfied when $\alpha^{k-j} = 1$. But the lowest power of α that is equal to 1 is $\alpha^n = 1$. Since $j, k \leq n - 1$, we know that $k - j < n$, so α^{k-j} can't possibly be equal to 1.

Finally, let's verify that everything works properly when we consider a minimum weight of 3. We end up with $\alpha^i + \alpha^j + \alpha^k = 0$ for some $i, j, k \leq n - 1$. We have many codewords with weight 3, and since the weight of a code only considers the *smallest* weight of any codewords, we know that the minimum weight of the code can be equal to 3. □

Let's see what this looks like with our favorite example, the (7,4) Hamming code. Table 2. shows the polynomial representations of codewords for a (7,4) Hamming code:

As shown in the table, every codeword can be represented as a multiple of the generating polynomial.

[5]Here is an explanation from Stackexchange: `https://cs.stackexchange.com/questions/20105/compute-minimum-hamming-distance-of-a-code`

[6]Here is another explanation from Stackexchange: `https://math.stackexchange.com/questions/2490293/proove-that-hamming-distance-is-three-for-hamming7-4`

Polynomial	Binary Codeword	Message
$0(x^3+x+1)=0$	0000000	0000
x^3+x+1	0001011	1011
x^4+x^2+x	0010110	0110
$x^5+x^3+x^2$	0101100	1100
$x^6+x^4+x^3$	1011000	1000
x^5+x^4+1	0110001	0001
x^6+x^5+x	1100010	0010
x^6+x^2+1	1000101	0101
$(x+1)(x^3+x+1)=x^4+x^3+x^2+1$	0011101	1101
$x(x+1)(x^3+x+1)=x^5+x^4+x^3+x$	0111010	1010
$x^2(x+1)(x^3+x+1)=x^6+x^5+x^4+x^2$	1110100	0100
$x^3(x+1)(x^3+x+1)=x^6+x^5+x^3+1$	1101001	1001
$x^4(x+1)(x^3+x+1)=x^6+x^4+x+1$	1010011	0011
$x^5(x+1)(x^3+x+1)=x^5+x^2+x+1$	0100111	0111
$x^6(x+1)(x^3+x+1)=x^6+x^3+x^2+x$	1001110	1110
$(x^3+x^2+1)(x^3+x+1)=x^6+x^5+x^4+x^3+x^2+x+1$	1111111	1111

TABLE 2. Polynomial representation of codewords for a (7,4) Hamming code.

8. Hamming Codes as Ideals of Rings

Earlier, we proved that every cyclic code is an ideal of $\mathbb{F}_q[x]/(x^n-1)$ and every ideal of $\mathbb{F}_q[x]/(x^n-1)$ is a cyclic code. This means that we can consider the generating polynomial $g(x)=x^3+x+1$ as an ideal in the ring $\mathbb{F}_2[x]/(x^n-1)$. This makes some things more clear. For example, it is obvious that $x^n-1=0$ is true. Furthermore, while the Hamming code is simple enough that we can define it without abstract algebra, this is not always the case. For example, codes over the ring $\mathbb{Z}_4$ warranted an entire chapter in Huffman and Pless' *Fundamentals of Error-Correcting Codes*. Being able to represent codes with rings gives us another perspective on them. And, regardless of their complexity, thinking about codes as components of rings, fields, or other structures makes the inherent algebraic structure of the codes themselves much more obvious.

References

[1] Richard E. Blahut. "Chapter 1: Introduction, Chapter 3: Linear Block Codes, Chapter 4: The Arithmetic of Galois Fields, Chapter 5: Cyclic Codes". In: *Algebraic codes for data transmission*. Cambridge University Press, 2003, pp. 1–130.

[2] William C. Huffman and Vera Pless. "Chapter 1: Basic concepts of Linear Codes, Chapter 3: Finite Fields, and Chapter 4: Cyclic Codes". In: *Fundamentals of error-correcting codes*. Cambridge Univ. Press, 2010, pp. 1–162.