Error Correction Coding: Weeks 3-4

Finite Fields & Algebraic Coding Theory

Eric DeFelice
Department of Electrical Engineering
Rochester Institute of Technology
ebd9423@rit.edu

***Abstract*—Finite field theory is the basis for many of the algebraic coding schemes. This paper will provide an overview of finite field theory and its use in error correction coding. Finite fields (also called Galois fields) will be defined and examples will be presented showing many of their properties. Generator polynomials will be introduced and their importance will be discussed. Finally, the connection between finite field theory and algebraic codes will be analyzed using a few different examples.**

I. Introduction

As we already know, a field is a set of elements that is closed under two binary operations, which are addition and multiplication. Some examples of fields are the real numbers, R, or the complex number set, C, which does satisfy the addition and multiplication requirements. Now if we have a code set, A, we can morph this set into a field by defining these two operations for A [1]. In doing this, we would have a field A of code words, a very useful set for channel coding. The only difference between this set, A, and the sets of real or complex numbers is that those two sets contain infinite elements, while our code set A only has a finite number of elements. Because of this fact, the set A is considered a finite field. This paper will investigate and point out differences of this type of field.

The theory of finite fields has been around for quite a while, but interest in its study has intensified in the recent past because of its applications in mathematics, computer science and communication theory [1]. In the remainder of this paper, we will look at finite fields and how they are applicable to error correction coding. The definition of fields will be introduced, and the subset of that group, finite fields, will be discussed. Properties of finite fields will then be looked at, and some examples that are useful for error correction coding will be presented. Finally, generator polynomials will be analyzed when a CRC example is reviewed.

II. Fields

A *field* F is a set of elements which is closed under two binary operations, which we denote by "+" and "•", such that nine axioms are satisfied for all a,b,c ∈ F. This basically states that any set of elements in which multiplication and addition can be performed between any of the two elements, and which abides by a set of nine rules, can be considered a field. The nine axioms that define a field are as follows:

(i) $a + (b + c) = (a + b) + c$

(ii) $a + b = b + a$

(iii) there exists an element $0 \in F$ such that $a + 0 = a$

(iv) there exists an element $a \in F$ such that $a + (-a) = 0$

(v) $a \cdot (b \cdot c) = (a \cdot b) \cdot c$

(vi) $a \cdot b = b \cdot a$

(vii) there exists an element $1 \in F$ such that $a \cdot 1 = a$

(viii) for each $a \neq 0$, there exists an element $a^{-1} \in F$ such that $a \cdot a^{-1} = 1$

(ix) $a \cdot (b + c) = a \cdot b + a \cdot c$

It is easily seen in the axioms that both multiplication and addition in a field are associative and commutative. Both multiplication and addition have inverses, meaning that an element plus its inverse equals 0 and an element multiplied by its inverse equals 1 (except for the 0 element, which has no multiplicative inverse).

Most of these operations are already known with a familiar set, the set of real numbers. In that set, all of the axioms are satisfied and it is considered a field. This is an example of an infinite field however, as the set on real numbers goes from $-\infty$ to ∞. If the field contains only a finite number of elements, then it is considered a *finite field*. Since there are only a finite number of elements, then the operations of addition and multiplication must then be performed modulo n, where n is the number of elements in the finite field. We will now denote a finite field of n elements as Z_n.

Given this definition of a finite field, it appears that there are some finite fields that would not satisfy all of the required axioms. There is another key requirement that would make Z_n a finite field using modulo n arithmetic. Let us theorize that Z_n is a finite field if and only if *n* is a prime number. To prove this theory, we can first look at the case where *n* is not prime. If this is the case, then *n* has a factorization $n = ab$. In this case, the element b has no multiplicative inverse, since $bc \equiv 1\ (mod\ n)$ would imply that $abc \equiv a\ (mod\ n)$, and then $0 \equiv a\ (mod\ n)$, which would contradict the factorization of *n*. Therefore, Z_n is only a finite field when n is prime.

If we look at a brief example of this point, it can be seen that Z_2 or Z_5 are finite fields, as they both satisfy all of the axioms under modulo *n* arithmetic. However, if we look at Z_9, it does not qualify as a finite field. If we take the element 3 in that field, it has no multiplicative inverse, ie. $3 \cdot 3^{-1} = 1\ (mod\ 9)$ would equate to $1 = 3 \cdot b\ (mod\ 9)$. There is no element b in Z_9 that satisfies the multiplicative inverse for element 3, so Z_9 is not a finite field.

Now that we know what a finite field is, we can define a field of polynomials having some characteristic value, meaning that the finite field can be reduced to p or a multiple of p values for a field F_p, where p is a prime number. This tells us that there can be no finite field having 6 elements, for instance. It can also be proven that for any prime number p and any positive integer n there exists a finite field that contains p^n elements. As an example, let us look at the polynomial field for n=1 and p=2:

$$Z_2[x] = \{0,1,x,1+x,1+x^2,x^2,x+x^2,1+x+x^2,\dots\}$$

From this field, we can select an irreducible polynomial, *f(x)*, of degree n. An irreducible polynomial is one that cannot be written as the product of two separate polynomials of positive degree. Once we select an irreducible polynomial from the original field, which can be used as the basis for a finite field that could be used as a generator polynomial. To further investigate this, let us construct a finite field with 4 elements. This finite field could be constructed using $f(x) = x^2 + x + 1 \in Z_2[x]$, which is irreducible over Z_2. This polynomial defines the field:

$$F = \{0, 1, x, 1 + x\}$$

Addition over this field is standard polynomial addition, for example, $[1 + x] + [x] = [1 + 2x] = [1]$. This is because $2 \equiv 0 \ (mod\ 2)$ in the Z_2 space, the field with characteristic 2. The following is the addition table for this field:

+	*0*	*1*	*x*	*1+x*
0	0	1	x	1+x
1	1	0	1+x	x
x	x	1+x	0	1
1+x	1+x	x	1	0

Multiplication also uses standard polynomial multiplication over modulo 2. In the Z_2 field, the degree of the product can be reduced back into the modulo 2 space by using the relation $x^2 \equiv x + 1$. For example, $[1 + x] \cdot [1 + x] = [1 + 2x + x^2] = [1 + x^2] = [1 + (1 + x)] = [x]$. The following is the multiplication table for this field:

·	*0*	*1*	*x*	*1+x*
0	0	0	0	0
1	0	1	x	1+x
x	0	x	1+x	1
1+x	0	1+x	1	x

Now that we have some examples of finite fields and their elements, we can look into the properties of these finite fields. In doing this, we can also explore how they might be used in the field of error correction coding.

III. Properties of Finite Fields

Now that finite fields have been defined and an example field has been introduced, we can introduce the concept of *primitive elements*. Firstly, we will let F^* denote the set of non-zero elements in the field F. An element $\propto$ in the finite field F is said to be a generator, or primitive element, of F^* if:

$$\{\propto^i : i \geq 0\} = F^*.$$

This basically means that all elements in F^* can be created from the generator,$\propto$. This concept can be easily seen in an example. Let us first take a finite field of 9 elements, which was constructed using the irreducible polynomial, $f(x) = x^2 + 1$ from Z_3[x]. The following is all of the elements in the field $f(x)$:

$$F = \{0, 1, 2, x, 2x, 1 + x, 1 + 2x, 2 + x, 2 + 2x\}$$

Note that in this field$x^2 = -1 = 2$. Next, we can find the primitive element from trial and error. $\propto = x$ is not a generator as it cannot produce any of the elements containing $1 + x$. Since that does not work, we can try $\propto = 1 + x$, which gives us the following:

$$\begin{array}{ll} (1+x)^0 = 1 & (1+x)^4 = 2 \\ (1+x)^1 = 1+x & (1+x)^5 = 2+2x \\ (1+x)^2 = 2x & (1+x)^6 = x \\ (1+x)^3 = 1+2x & (1+x)^7 = 2+x \end{array}$$

Since all of the non-zero elements could be created using $\propto = 1 + x$, it is a generator for $f(x)$. Now that we have this generator function, it can be used to represent all of the elements in the field. One way of doing this, for example, is to use the exponent in which the generator function is raised as the code word. Another convenient way to write these elements is to signify each element by ordering their coefficients from low order to high order. For example, the field element $a_0 + a_1x + a_xx^2$ could be written as $(a_0a_1a_2)$.

Lastly, we can define the minimal polynomial for a field element. A *minimal polynomial* of an element $\propto \in F_q^m$ with respect to F_q is a nonzero monic polynomial $f(x)$ of the least degree in $F_q[x]$ such that $f(\alpha) = 0$ [1]. The minimal polynomial is found using the following equation:

$$m_a(x) = \prod_{B \in C(\alpha)} (x - \beta)$$

Using the notation from above, we can find the minimal polynomials of the elements in the following field:

$$f(x) = x^4 + x + 1 \ \in \ Z_2[x]$$

The generator for this field is $\alpha = x$. The elements in this field are as follows:

$$\begin{array}{ll} (\alpha)^0 = (1000) & (\alpha)^8 = (1010) \\ (\alpha)^1 = (0100) & (\alpha)^9 = (0101) \\ (\alpha)^2 = (0010) & (\alpha)^{10} = (1110) \\ (\alpha)^3 = (0001) & (\alpha)^{11} = (0111) \\ (\alpha)^4 = (1100) & (\alpha)^{12} = (1111) \end{array}$$

$(\alpha)^5 = (1100)$	$(\alpha)^{13} = (1011)$
$(\alpha)^6 = (0011)$	$(\alpha)^{14} = (1001)$
$(\alpha)^7 = (1101)$	$(\alpha)^{15} = a^0 = 1$

The minimal polynomials of the elements are as follows:

$$m_1(y) = y^4 + y + 1 = m_2(y) = m_4(y) = m_8(y)$$

$$m_3(y) = y^4 + y^3 + y^2 + y + 1 = m_6(y) = m_{12}(y) = m_9(y)$$

$$m_5(y) = y^2 + y + 1 = m_{10}(y)$$

$$m_7(y) = y^4 + y^3 + 1 = m_{14}(y) = m_{13}(y) = m_{11}(y)$$

$$m_0(y) = y + 1$$

IV. CRC Example

As a case study in the application of finite fields in channel coding, we can look at a simple example. The CRC, or cyclic redundancy check, code can detect errors in a data stream, but cannot correct those errors. The CRC consists of a section of redundant bits being transmitted with the data bit stream. These redundant bits are generated from a generator polynomial, by dividing the data stream by this polynomial. The remainder from this operation is the CRC, and is what is sent along with the data. It is easy to see that this is in fact a simple channel code, as it adds redundant bits so that the receiver can tell if there are any errors caused by the channel.

The first step when generating a CRC is to set the generator polynomial. For this example we will use the following generator:

$$g(x) = x^3 + x + 1$$

Now that we have this generator polynomial, we can divide the data stream polynomial by this, to get our remainder, or CRC bits. For the sake of this example, we will use the data stream, 1100010100, which would make the following data polynomial:

$$1100010100 \rightarrow d(x) = x^9 + x^8 + x^4 + x^2$$

The CRC value is computed by finding the remainder when the data polynomial is divided by the generator polynomial. That operation is shown below:

$$\frac{x^9 + x^8 + x^4 + x^2}{x^3 + x + 1} = x^2$$

This remainder then gets converted to a packet, which would be 001 (ascending degree), and added to the end of the data stream, making the transmit data stream 1100010100 001.

On the receive side the data stream that is received now has the CRC bits added to the end. The receiver again finds the remainder when the data polynomial is divided by the generator polynomial. If the remainder is 0, then there are no errors in the data stream.

V. Conclusion

In this paper, finite fields were introduced and were defined by the axioms which are required to be satisfied. The mod n structure of these fields was discussed and prime fields were introduced. The polynomial representation of these fields was used in the discussion, as it is used in many algebraic coding schemes that will be analyzed later on in my studies. Some examples of the addition and multiplication operators in these fields were then presented, and minimal polynomials were looked at. These serve as the basis functions for some coding schemes. Finally, a simple CRC example was used to show how finite fields could be used in a channel coding application, even if it is a simple one.

The next couple of weeks will be spend studying binary block codes, and to begin studying Reed-Solomon codes.

References

[1] San Ling, Chaoping Xing, *Coding Theory: A First Course*. Cambridge University Press, 2004

[2] Andre Neubauer, Jurgen Freudenberger, Volker Kuhn, *Coding Theory: Algorithms, Architectures, and Applications*. John Wiley & Sons Ltd., 2007

[3] Scott A. Vanstone, Paul C. van Oorschot, *An Introduction to Error Correcting Codes with Applications*. Kluwer Academic Publishers., 1989