The finite field with 2 elements

The simplest **finite field** is

$$GF(2) = \mathbf{F}_2 = \{0, 1\} = \mathbf{Z}/2$$

It has addition and multiplication $+$ and $\times$ defined to be

$$\begin{array}{cccc} 0+0=0 & 0+1=1 & 1+0=1 & 1+1=0 \\ 0\times 0=0 & 0\times 1=0 & 1\times 0=0 & 1\times 1=1 \end{array}$$

Notation $\mathbf{Z}_2$ is sometimes used, but this is ill-advised since in closely-related contexts $\mathbf{Z}_2$ is the 2-adic integers, an entirely different thing.

Remark: Since $1 + 1 = 0$,

$$-1 = 1$$

if by '-1' we mean something which when added to 1 gives 0. Similarly,

$$-0 = 0$$

since $0 + 0 = 0$.

Remark: It is not possible to tell without context whether symbols 1 and 0 refer to elements of $\mathbf{F}_2$ or something else.

Remark: With 0 and 1 viewed as *bits*, the addition in $\mathbf{F}_2$ is **exclusive or**. This is worth noting, but there are important and useful other viewpoints which we will pursue. One is the construction of $\mathbf{Z}/p$, **the integers modulo** p. Here is $p = 2$. The further abstraction is to **finite fields**.

Remark: In $\mathbf{F}_2$

$$2 = 1 + 1 = 0$$

$$3 = 1 + 1 + 1 = 2 + 1 = 0 + 1 = 1$$

$$4 = 1 + 1 + 1 + 1 = 2 + 2 = 0 + 0 = 0$$

and so on. Yes, here

$$2 = 0$$

$$3 = 1$$

$$4 = 0$$

etc.

Polynomials over $GF(2)$

Basic algebra with polynomials having coefficients in the finite field $\mathbf{F}_2$ is in almost exactly the same as polynomials having real or complex coefficients.

A **polynomial** $P(x)$ in the **indeterminate** x with **coefficients** which are *real numbers* is a sum of powers of x with real numbers in front of them, like

$$P(x) = 2.077 \cdot x^7 - 9.11 \cdot x^2 + 17$$

The numbers in front of the powers of x are the **coefficients**. The **degree** of a polynomial is the highest power of x that appears with a non-zero coefficient. By convention, to avoid making exceptions, assign the 0-polynomial degree $-\infty$.

Proposition: The degree of the product of two polynomials is the sum of their degrees.

Proof: The coefficient of the highest-degree term of the product is the product of the highest-degree terms of the factors. ///

A polynomial gives rise to a **polynomial function**, denoted by the same symbol, into which we can plug numbers in place of the indeterminate x.

Analogously, a **polynomial** in the indeterminate x with **coefficients** in the finite field $\mathbf{F}_2$ looks like a polynomial with real coefficients but with coefficients only 0 or 1.

Remark: The exponents of x are **always** ordinary integers, regardless of what kind of coefficients we use.

For example,

$$P(x) = 1 \cdot x^4 + 0 \cdot x^3 + 0 \cdot x^2 + 0 \cdot x + 1 \cdot x^0 = x^4 + 1$$

is such a polynomial.

Remark: Since the only coefficients are 0 and 1, if the coefficient of a power of x is 0, don't write it at all, and if the coefficient is 1 write the power of x.

Such a polynomial gives a **polynomial function** from $\mathbf{F}_2$ to $\mathbf{F}_2$, by **evaluation**:

$$\begin{aligned} P(0) &= 0^4 + 1 = 1 \\ P(1) &= 1^4 + 1 = 0 \end{aligned}$$

Unlike with real numbers, different polynomials can give rise to the same function. For example, $P(x) = x^2 + x + 1$ and $Q(x) = 1$ have the same values for any input in $\mathbf{F}_2$.

Addition of polynomials with coefficients in $\mathbf{F}_2$ is to add the coefficients of corresponding powers of x, inside $\mathbf{F}_2$. For example,

$$(x^3 + x^2 + 1) + (x^3 + x + 1)$$

$$= (1+1) \cdot x^3 + (1+0) \cdot x^2 + (0+1)x + (1+1)$$

$$= x^2 + x$$

Multiplication of polynomials is as usual, much like multiplying decimal integers, keeping track of powers of x instead of decimal places. Multiplication of polynomials is simpler in that there is *no carry*. Integer multiplication is

$$\begin{array}{rrrr} & & 8 & 3 \\ & \times & 2 & 3 \\ \hline & 2 & 4 & 9 \\ 1 & 6 & 6 & \\ \hline 1 & 9 & 0 & 9 \end{array}$$

Polynomial multiplication is similar. For example, with real coefficients

$$\begin{array}{llllll}
 & & 2x^3 & +3x^2 & +x & -3 \\
 & & & x^2 & -2x & +1 \\
\hline
 & & +2x^3 & +3x^2 & +x & -3 \\
 & -4x^4 & -6x^3 & -2x^2 & +6x & \\
2x^5 & +3x^4 & +1x^3 & -3x^2 & & \\
\hline
2x^5 & -x^4 & -3x^3 & -2x^2 & +7x & -3
\end{array}$$

Each term in the first polynomial multiplies each term in the second polynomial.

Analogously, multiply polynomials with coefficients in $\mathbf{F}_2$: each term in the first multiplies each term in the second, and add them. This is easier because the arithmetic of $\mathbf{F}_2$ is so easy.

For example, keeping in mind that $1+1=0$ in the finite field $\mathbf{F}_2$:

$$\begin{array}{llllll}
 & & x^3 & & +x & +1 \\
\times & & & x^2 & +x & +1 \\
\hline
 & & +x^3 & & +x & +1 \\
 & +x^4 & & +x^2 & +x & \\
x^5 & & +x^3 & +x^2 & & \\
\hline
x^5 & +x^4 & & & & +1
\end{array}$$

Preserve the vertical alignment of like powers of x against miscopying errors.

Remark: With coefficients in $\mathbf{F}_2$, polynomials which look different may be the same if we do not *reduce* the coefficients to be either 0 or 1. For example,

$$x-1 = x+1 = -x+1$$

$$2x+1 = 1 = -2x+3$$

Division of one polynomial by another is analogous to long division (with remainder) of integers, except there is no borrowing nor carrying.

First we do an example with coefficients viewed as being ordinary integers or real numbers:

$$
\begin{array}{r|rrrrrrrrr}
 & \multicolumn{9}{l}{x^3 + x^2 - x^1 - 1 \text{ R } x^4 + 3x + 2} \\
\cline{2-10}
x^5 + x^3 + x + 1 & x^8 & +x^7 & +0 & +0 & +x^4 & +x^3 & +0 & +x & +x^0 \\
 & x^8 & +0 & +x^6 & +0 & +x^4 & +x^3 & +0 & +0 & +0 \\
\cline{2-10}
 & & x^7 & -x^6 & +0 & +0 & +0 & +0 & +x & +x^0 \\
 & & x^7 & +0 & +x^5 & +0 & +x^3 & +x^2 & +0 & +0 \\
\cline{2-10}
 & & & -x^6 & -x^5 & +0 & -x^3 & -x^2 & +x & +x^0 \\
 & & & -x^6 & +0 & -x^4 & +0 & -x^2 & -x & +0 \\
\cline{2-10}
 & & & & -x^5 & +x^4 & -x^3 & +0 & +2x & +x^0 \\
 & & & & -x^5 & +0 & -x^3 & +0 & -x & -x^0 \\
\cline{2-10}
 & & & & & x^4 & +0 & +0 & +3x & +2
\end{array}
$$

x^3 x^5s go into x^8: multiply the divisor by x^3 and subtract from dividend, yielding $x^7 - x^6 + x + 1$. x^2 x^5s go into x^7: multiply the divisor by x^2 and subtract, giving new remainder $-x^6 - x^5 - x^3 - x^2 + x + 1$. Continue until **degree** of the remainder < degree of divisor.

Now view the coefficients as in $\mathbf{F}_2$, so $-1 = 1$, $1 + 1 = 0$, etc:

$$\begin{array}{r|lllllllll}
\multicolumn{1}{r}{} & x^3 & +x^2 & +x & +1 & \multicolumn{5}{l}{\text{R } x^4+x} \\ \cline{2-10}
x^5+x^3+x+1 & x^8 & +x^7 & +0 & +0 & +x^4 & +x^3 & +0 & +x & +x^0 \\
 & x^8 & +0 & +x^6 & +0 & +x^4 & +x^3 & +0 & +0 & +0 \\ \cline{2-10}
 & & x^7 & +x^6 & +0 & +0 & +0 & +0 & +x & +x^0 \\
 & & x^7 & +0 & +x^5 & +0 & +x^3 & +x^2 & +0 & +0 \\ \cline{2-10}
 & & & +x^6 & +x^5 & +0 & +x^3 & +x^2 & +x & +x^0 \\
 & & & +x^6 & +0 & +x^4 & +0 & +x^2 & +x & +0 \\ \cline{2-10}
 & & & & +x^5 & +x^4 & +x^3 & +0 & +0 & +x^0 \\
 & & & & +x^5 & +0 & +x^3 & +0 & +x & +x^0 \\ \cline{2-10}
 & & & & & x^4 & +0 & +0 & +x & +0
\end{array}$$

Remark: Because in $\mathbf{F}_2$ we have $-1 = +1$, addition and subtraction are the same. We can add, which is easier, rather than subtract.

Remark: Keep in mind things like $2 = 0$, $3 = 1$ in $\mathbf{F}_2$.

Remark: It's impossible to tell what the coefficients of a polynomial are without context.

Remark: Regardless of the *coefficients*, the *exponents* of x are non-negative integers.

Cyclic redundancy checks (CRCs)

Cyclic redundancy checks (**CRC**s) *detect* but do not *correct* errors, generalizing **parity check bits**, but better.

An optimal n-bit CRC will detect any 2 bit errors in $2^n - 1$. Common values of n are 12, 16, and 32.

With data a stream of bits like 1100010100, create a **data polynomial** with coefficients in $\mathbf{F}_2$ by using the 0s and 1s as coefficients:

$$1100010100 \to x^9 + x^8 + x^4 + x^2$$

A CRC-computing algorithm is specified by its **generating polynomial**, a polynomial with coefficients in $\mathbf{F}_2$. For example, take as generating polynomial

$$x^3 + x + 1$$

The CRC **value** of the data is computed by **finding the remainder when the data polynomial is divided by the generating polynomial.**

With data polynomial 1100010100 $= x^9 + x^8 + x^4 + x^2$ and generating polynomial $1101 = x^3 + x + 1$

$$
\begin{array}{r|l}
 & x^6 + x^5 + x^4 + 0 + 0 + 0 + 0 \; R x^2 \\
\hline
x^3 + x + 1 & x^9 + x^8 + 0 + 0 + 0 + x^4 + 0 + x^2 + 0 + 0 \\
 & x^9 + 0 + x^7 + x^6 \\
\cline{2-2}
 & \qquad x^8 + x^7 + x^6 + 0 + x^4 + 0 + x^2 + 0 + 0 \\
 & \qquad x^8 + 0 + x^6 + x^5 \\
\cline{2-2}
 & \qquad\qquad x^7 + 0 + x^5 + x^4 + 0 + x^2 + 0 + 0 \\
 & \qquad\qquad x^7 + 0 + x^5 + x^4 \\
\cline{2-2}
 & \qquad\qquad\qquad\qquad\qquad\qquad x^2
\end{array}
$$

Thus, the remainder is x^2, which we translate back to bits as 100 (descending degree). That is

CRC with gen poly $x^3 + x + 1$ computed for

data 1100010100 is 100

Remark: Division can be implemented cleverly so the CRC of large data can be computed quickly.

Remark: In some cases division is run in the opposite direction bit-wise, meaning that the bit string is interpreted as coefficients in *ascending* rather than *descending* order. This doesn't change the idea but changes the interpretation.

Remark: Computation of a **single parity bit** is computation of CRC with generating polynomial $x + 1$.

Remark: For CRC generating polynomials of degree n there are 2^n possible values of CRC of data, since the remainder after division can be any polynomial of degree $n - 1$ or less. If the remainders are uniformly distributed then the CRC misses only 1 in 2^n bit errors.

Remark: A common but deficient type of **redundant information** computed to detect errors or changes in data is an **XOR checksum**. meaning **exclusive-or**. These are easy in typical computer operations: XOR all the *bytes* to produce a single-byte checksum value. This is appealing because it is easy and fast. With 2^8 different possible checksum values (because of the 8-bit ASCII bytes) it would *seem* that these checksums should be good at detecting errors. **However**, since some 8-bit bytes are much more common than others, these checksum values are *not* uniformly distributed among the 2^8 possibilities.

The XOR checksum is simply the (bad) CRC with generating polynomial

$$x^8 - 1$$

Remark: We should realize that if we are unhappy with the statistical (error-detecting) properties of a CRC then we can try a different generating polynomial.

Better CRC's in real life are among the following standard ones:

$$x^{12} + x^{11} + x^3 + x + 1$$

$$x^{16} + x^{12} + x^5 + 1$$

and

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10}$$

$$+x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

These catch all two-bit errors in very long strings: that degree 12 CRC catches all two-bit errors up to distance $2^{12} - 1$ apart, the degree 16 CRC catches all two-bit errors up to distance $2^{16} - 1$ apart, and the degree 32 CRC catches all two-bit errors up to distance $2^{32} - 1$ apart.

Remark: This is so because these polynomials are all *primitive*, discussed subsequently.

What errors does a CRC catch?

Algebra can explain what errors will be detected by a CRC, and how choices of the generating polynomial affect performance.

For CRC with generator g and for data a polynomial d (both with coefficients in $\mathbf{F}_2$), suppose that d is transmitted or played back with some errors, and becomes $\tilde{d}$.

The **error vector** or **error polynomial** is the difference

$$e = d - \tilde{d} = d - \tilde{d}$$

The number of non-zero coefficients in e is its **Hamming weight**, and is the number of bit errors.

Let r be the CRC value of d, the remainder when d is divided by g.

$$d = q \cdot g + r$$

where q is the quotient. Let $\tilde{r}$ be the CRC value of $\tilde{d}$, and

$$\tilde{d} = \tilde{q} \cdot g + \tilde{r}$$

where $\tilde{q}$ is the quotient dividing $\tilde{d}$ by g.

Then the error is

$$e = d - \tilde{d} = (q \cdot g + r) - (\tilde{q} \cdot g + \tilde{r})$$

$$= (q - \tilde{q}) \cdot g + r - \tilde{r}$$

The remainder upon dividing e by g is $r - \tilde{r}$.

The CRC **fails** *to detect error $e = d - \tilde{d}$ the remainder $r - \tilde{r}$ must be* 0*, that is if g* **divides** *e (with remainder* 0*).*

Remark: Take for granted for now that divisibility properties of polynomials with coefficients in $\mathbf{F}_2$ are reasonable and consistent with divisibility properties of polynomials with real or complex coefficients.

• If there is **just one bit error**, at i^{th} position, the error polynomial is

$$e(x) = x^i$$

undetected by the CRC if and only if $g(x)$ divides $e(x) = x^i$. Since x^i is the product of i copies of x, $g(x)$ cannot divide x^i unless $g(x) = x^j$ for some $j \le i$. So $g(x) = x + 1$ detects single bit errors.

• With **two bit errors**, at m^{th} and n^{th} positions (with $m < n$), the error is

$$e(x) = x^m + x^n$$

undetected if and only if $g(x)$ divides

$$e(x) = x^m + x^n = x^m(1 + x^{n-m})$$

If $g(x)$ has **constant term** 1, then g has no factor of x and $g(x)$ must divide $1 + x^{n-m}$.

Example: About detectability of two-bit errors: the XOR checksum, the CRC with $g(x) = x^8 - 1$. Recall high-school algebra

$$\begin{array}{rcl} x^2 - 1 & = & (x-1)(x+1) \\ x^3 - 1 & = & (x-1)(x^2+x+1) \\ x^4 - 1 & = & (x-1)(x^3+x^2+x+1) \\ x^5 - 1 & = & (x-1)(x^4+x^3+x^2+x+1) \\ & \ldots & \\ x^N - 1 & = & (x-1)(x^{N-1}+\ldots+x+1) \end{array}$$

Replaceng x by x^8

$$\begin{array}{rcl} x^{16} - 1 & = & (x^8-1)(x^8+1) \\ x^{24} - 1 & = & (x^8-1)(x^{16}+x^8+1) \\ x^{32} - 1 & = & (x^8-1)(x^{24}+x^{16}+x^8+1) \\ x^{40} - 1 & = & (x^8-1)(x^{32}+x^{24}+x^{16}+x^8+1) \\ & \ldots & \\ x^{8N} - 1 & = & (x^8-1)(x^{8(N-1)}+\ldots+x^8+1) \end{array}$$

That is, $x^8 - 1$ divides (with remainder 0) any polynomial $x^{8N} - 1$. So two bit errors occur a distance apart which is a **multiple of 8**, the XOR checksum CRC will **not** detect it.

Example: But almost this performance can already be achieved by a *smaller/cheaper* CRC: the CRC with generating polynomial $x^3 + x + 1$, of degree 3 rather than 8 only fails to detect two-bit errors a multiple of 7 apart. That is, $x^3 + x + 1$ divides $x^N - 1$ (with remainder 0) only when N is a multiple of 7 smaller N works. (How to check?) This property is a consequence of *primitivity*, discussed later.

Example: Still about 2-bit errors: using CRC with generating polynomial $x^4 + x + 1$, even though only degree 4, fails to detect two-bit errors only when they're a multiple of 15 apart. That is, x^4+x+1 divides x^N-1 (with remainder 0) only when N is a multiple of 15. (How to check?) This shows **the XOR checksum is inefficient**.

Example: Still thinking about 2-bit errors: using the CRC with generating polynomial $x^5 + x^2 + 1$, even though it's only of degree 5, fails to detect two-bit errors only when a multiple of 31 apart. $x^5 + x^2 + 1$ divides $x^N - 1$ only when N is a multiple of 32. Can certainly check by trying to divide all candidates that no smaller N works, *but this is not the intelligent way to verify the property.*

Example: Changing the generating polynomial slightly from x^5+x^2+1 to x^5+x+1,. mysteriously, the performance is degraded so that two-bit errors a multiple of 21 apart will pass undetected.

Example: Changing to generating polynomial $x^5 + x^4 + x + 1$ mysteriously causes a further degradation: two-bit errors which are a multiple of 8 apart will pass undetected.

Remark: While the degree of the polynomial is higher, so that the CRC's *reported value* contains more bits, the choice of how these bits are computed is suboptimal.

Example: By contrast, the CRC with generator

$$x^{16}+x^{15}+x^2+1=(x+1)(x^{15}+x+1)$$

will fail to detect two-bit errors only if they are a multiple of $32767=2^{15}-1$ apart! (Obviously this is not discovered by brute force!)

Remark: As in the last example, the most effective CRC's are obtained by taking generating polynomials which have a factor like $x^{15}+x+1$ which is **irreducible**, meaning that it can't be factored further into smaller-degree polynomials with coefficients in $\mathbf{F}_2$. This is the polynomial analogue of being a *prime number.* Further, not only is this polynomial irreducible, it is **primitive**, meaning that the smallest integer N such that the polynomial divides x^N-1 is $N=2^d-1$ where d is the degree of the polynomial.

Remark: It is unclear whether there are many such things, how to find them, how to verify the property, etc.

Remark: The 16-bit CRC above detects all 3-bit errors in data of 32767 bits or less because it is the product of $x+1$ with a primitive degree 15 polynomial. The primitive degree 15 polynomial detects two-bit errors within distance of 32767 while the $x+1$ detects all errors consisting of an odd number of bit errors.

Burst errors are errors close together. A CRC of degree n with non-zero constant term can detect a *any* burst error of length $< n$: the error can be written as

$$e(x) = x^n \cdot p(x)$$

where $p(x)$ is a polynomial of degree $< n$. For the CRC to fail to detect this, it must be that $g(x)$ divides $e(x)$ (with remainder 0). Since $g(x)$ has non-zero constant term it has no factors of x, so for $g(x)$ to divide $x^n \cdot p(x)$ it must be that $g(x)$ actually divides $p(x)$. But if the degree of $p(x)$ is less than the degree of $g(x)$ this is impossible. Thus, the error will be detected.

Remark: We have implicity used **unique factorization** of polynomials with coefficients in $\mathbf{F}_2$. This deserves proof later.

Remark: It is useful that polynomials $f(x)$ with coefficients in $\mathbf{F}_2$ have the funny property

$$f(x^2) = f(x)^2$$

To see this, let $f(x) = x^n + g(x)$ with $g(x)$ the lower-degree terms in $f(x)$. Then

$$f(x)^2 = (x^n)^2 + 2x^n g(x) + g(x)^2$$

$$= (x^n)^2 + 0 + g(x)^2$$

since $2 = 0$ in $\mathbf{F}_2$. By induction on the number of terms, assume $g(x)^2 = g(x^2)$, so

$$f(x)^2 = (x^n)^2 + g(x^2) = (x^2)^n + g(x^2) = f(x^2)$$

as asserted. ///
