8.5 Cyclic Codes

- a subclass of linear block codes, easier to implement for long block lengths

Polynomial representation of sequences

represent the length l sequence $a_{l-1}, a_{l-2}, \cdots a_1, a_0$ as a polynomial of degree $l-1$

$$a(x) = a_{l-1}x^{l-1} + a_{l-2}x^{l-2} + \cdots + a_1 x + a_0$$

note a_i are binary.

ex $\quad 10110 \quad$ is $\quad x^4 + x^2 + x$

codewords will be polys of degree $\leq n-1$

we can multiply and divide sequence polys.
ex $\quad x\,a(x)$ shifts it to the left.

generator polynomial

a generator polynomial $g(x)$ of degree m is a divisor of $x^n - 1$.

ex $\quad g(x) = x^3 + x + 1$ is a divisor of $x^7 - 1 = x^7 + 1$

$$
\begin{array}{r|l}
x^3 + x + 1 & x^4 + x^2 + x + 1 \\
\hline
x^7 & \\
x^7 + x^5 + x^4 & \\
\hline
x^5 + x^4 & +1 \\
x^5 + x^3 + x^2 & \\
\hline
x^4 + x^3 + x^2 & +1 \\
x^4 + x^2 + x & \\
\hline
x^3 + x & +1 \\
x^3 + x & +1 \\
\hline
0 &
\end{array}
$$

ex more concisely $1\,0\,1\,1$) $1\,0\,0\,0\,0\,0\,0\,1$

$1\,0\,1\,1$

$1\,1\,0\,0\,0\,1$

$1\,0\,1\,1$

$1\,1\,1\,0\,1$

$1\,0\,1\,1$

$1\,0\,1\,1$

$1\,0\,1\,1$

$0\,0\,0\,0$

so $x^n - 1 = g(x)\,p(x)$, where $p(x)$ is the parity-check polynomial

$p(x)$ degree k

ex $p(x) = x^4 + x^2 + x + 1$ above

ex CRC-CCITT $x^{16} + x^{12} + x^5 + 1 = g(x)$

a cyclic code consists of all multiples of the generator polynomial by polys of degree $\le k-1 = n-m-1$

$$c(x) = i(x)\,g(x)$$

$c(x)$: degree $\le n-1$; $i(x)$: degree $\le k-1 = n-m-1$; $g(x)$: degree $m = n-k$

called cyclic because all cyclic shifts of a code word are also in the code.

Proof of cyclic property. Consider 1-step left rotation of $c(x)$:

$$\begin{aligned} \text{ROTL}(c(x)) &= x\,c(x) - c_{n-1}x^n + c_{n-1} \\ &= x\,c(x) - c_{n-1}(x^n - 1) \\ &= x\,i(x)\,g(x) - c_{n-1}\,p(x)\,g(x) \\ &= \left[x\,i(x) - c_{n-1}\,p(x)\right] g(x) \end{aligned}$$

But $[x\,i(x) - c_{n-1}\,p(x)]$ is a poly of degree $k-1$, since the coeff of x^k is:

$$i_{k-1} - c_{n-1} = 0$$

since $c(x) = i(x)\,g(x)$. Therefore $\text{ROTL}(c(x))$ is also a codeword

ex The Hamming (7, 4) code is cyclic

Table 1.1 Hamming (7, 4) code.

0	0	0	0	0	0	0
0	0	0	1	0	1	1
0	0	1	0	1	1	0
0	0	1	1	1	0	1
0	1	0	0	1	1	1
0	1	0	1	1	0	0
0	1	1	0	0	0	1
0	1	1	1	0	1	0
1	0	0	0	1	0	1
1	0	0	1	1	1	0
1	0	1	0	0	1	1
1	0	1	1	0	0	0
1	1	0	0	0	1	0
1	1	0	1	0	0	1
1	1	1	0	1	0	0
1	1	1	1	1	1	1

a cyclic code can also be represented by the generator matrix

$$\underline{c} = (i_{k-1} \cdots i_1, i_0) \begin{bmatrix} g_n & g_{n-1} & \cdots & g_1 & g_0 & 0 & 0 & 0 \\ 0 & g_n & & g_2 & g_1 & g_0 & 0 & 0 \\ 0 & 0 & g_n & \cdots & g_2 & g_1 & g_0 & 0 \\ & & & & & & & \\ 0 & 0 & 0 & 0 & & g_n & \cdots & g_1 g_0 \end{bmatrix}$$

(k rows, $k-1$ zeros, n columns)

note nonsystematic

which follows from the way we perform $i(x)g(x)$ as explicit multiplication.

The <u>syndrome polynomial</u> of any word $w(x)$ is the remainder when divided by $g(x)$:

$$s(x) = R_g[w(x)]$$

and has degree $\leq m-1$.

The syndrome of any code word is 0, since

$$R_g[c(x)] = R_g[i(x)g(x)] = 0$$

<u>decoding</u> receive $r(x) = c(x) + e(x)$

calculate $s(x) = R_g[r(x)] = R_g[c(x) + e(x)]$

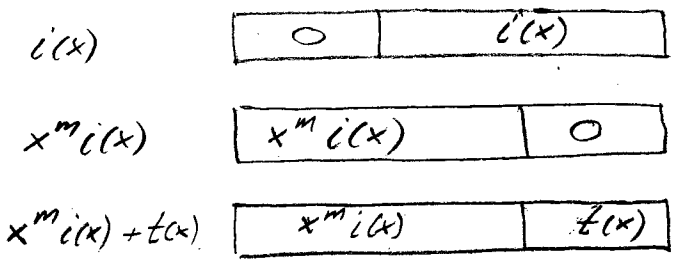
$$= R_g[e(x)]$$

if $s(x) \neq 0$ then there's an error. look up table indexed by $s(x)$ to get lowest weight $\hat{e}(x)$, and perform the correction.

$$\hat{c}(x) = r(x) + \hat{e}(x)$$

for long codes, the table becomes enormous. for cyclic codes, though, there are other techniques — much more complex, but practical, even for very long codes.

a <u>systematic form</u> of cyclic code is obtained by



where $t(x) = R_g[x^m i(x)]$ degree $\leq m-1$

supplies the check bits

and $c(x) = x^m i(x) + t(x)$ is a valid code word, since

$$x^m i(x) = g(x) q(x) + t(x)$$

$$x^m i(x) + t(x) = g(x) q(x)$$

this systematic form is almost universal in data comm.

implementation — Strangely, polynomial division turns out to be easy in hardware. Consider the (15,11) code generated by $g(x) = x^4 + x + 1$

example $i(x) = 1\,0\,1\,1\,0\,0\,1\,0\,0\,0\,1$

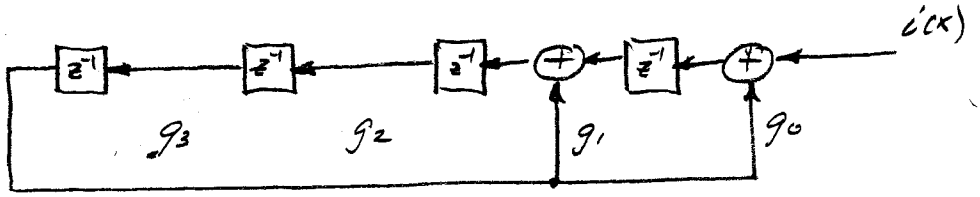
$$t(x) = R_g[x^4 i(x)]$$

```
                1 0 1 0 1 0 1 0 1 0 1
          ------------------------------
1 0 0 1 1 ) 1 0 1 1 0 0 1 0 0 0 1 0 0 0 0
            1 0 0 1 1
            -----------
              0 1 0 1 0
              0 0 0 0 0
              ---------
                1 0 1 0 1
                1 0 0 1 1
                ---------
                  0 1 1 0 0
                  0 0 0 0 0
                  ---------
                    1 1 0 0 0
                    1 0 0 1 1
                    ---------
                      1 0 1 1 0
                      1 0 0 1 1
                      ---------
                        1 0 1 1 0
                        1 0 0 1 1
                        ---------
                          1 0 1 0 0
                          1 0 0 1 1
                          ---------
                            0 1 1 1 0
                            0 0 0 0 0
                            ---------
                              1 1 1 0   <- t(x)
```

if leading bit is 1, then XOR g(x) with current remainder

send $c(x)$ = 1 0 1 1 0 0 1 0 0 0 1 1 1 1 0

this circuit does the same thing:



if leading bit is 1, then XOR g(x) with current remainder.

when i(x) is completely clocked in, the shift register contains t(x) !

so use it for encoding (to calculate t(x)) and decoding (to calculate s(x) from r(x)).

Software implementations are possible, but time consuming if they are bit by bit. However, for CRC 16 or CRC CCITT and messages which are in bytes, there is a fast byte by byte implementation which uses only 512 bytes of table storage.

error detection Most data systems are built on error detection and retransmission. Cyclic codes detect far more errors than their min distance guarantees. Also codewords don't have to be cyclic, so the length is unconstrained. In practice, use i(x) of arbitrary length, run it through the feedback shift register and append t(x).

- BCH Codes – a class of cyclic codes with straightforward decoding and many combinations of (n, k, t). Standard lengths are of the form

$n = 2^u - 1$ codeword length

$k = n - ut$ info length (t is #correctable errors)

$d_{min} = 2t - 1$

so, for example,

$n = 63$ $u = 6$

k	t	
57	1	
51	2	
45	3	etc

the $(63, 45, 3)$ BCH code has rate $\frac{45}{63}$, corrects 3.

See P+S Table 10-1.

- Reed Solomon codes — not binary, alphabet is typically $0..q-1$ where q is power of 2: $q = 2^{k'}$. This is also the length of the codeword in these q-ary symbols.

e.g. $k' = 6$ so 6-bit symbols, $q = 64$

$N = 63$ symbols in codeword

- flexible: choose K info symbols and correct

$t = \left\lfloor \frac{N-K}{2} \right\rfloor$ symbols e.g. $N = 63$, $K = 45$

correct $t = \left\lfloor \frac{63-45}{2} \right\rfloor = 9$

- good for burst errors, since any number of bit errors in a symbol still counts as only one symbol error.

- Shortened codes. Sometimes you want a codeword of a specific length n that isn't conveniently a power of 2 (minus 1). Common practice to go to the next higher standard length, but set the unnecessary extra info bits to 0 and don't send them. This retains at least the error correcting power of the base code.

- e.g. short packets accommodate only 7 bytes for the codeword. Use BCH, $n = 56 \rightarrow n' = 63$. Suppose $t = 2$ error correction is sufficient, so $m = nt = 6 \times 2 = 12$ check bits, $k' = n' - m = 51$

  base code $(63, 51, 2)$ (n k t)