Reed-Solomon codes II, GF(2^m) arithmetics

Coding Technology

Illés Horváth

2025/10/10

Reminder: cyclic codes

Reminder: cyclic codes.

Cyclic linear codes can be generated by a single generator polynomial.

Reed-Solomon codes are MDS codes ($d_{\min} = n - k + 1$) over GF(q). RS codes are cyclic linear codes generated using a primitive element α. We assume $n = q - 1$. RS codes can be obtained either by generator matrix

$$G = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \alpha & \alpha^2 & \dots & \alpha^{n-1} \\ \vdots & & & \ddots & \vdots \\ 1 & \alpha^{k-1} & \alpha^{2(k-1)} & \dots & \alpha^{(n-1)(k-1)} \end{bmatrix}$$

or by generator polynomial $g(x) = \prod_{i=1}^{n-k}(x - \alpha^i)$ (which gives the same set of codewords but a different message$\rightarrow$codeword mapping).

Decoding: error locator algorithm.

Binary linear cyclic codes

The result that linear cyclic codes can be generated using a generator polynomial is valid for binary codes as well.

So are any of the binary codes we have seen previously cyclic?

- repeater codes are clearly cyclic; the C$(n, 1)$ repeater code has generator polynomial $g(x) = 1 + x + \cdots + x^{n-1}$.
- Hamming codes are also cyclic (at least for some orderings of the columns of the parity check matrix H). Generator polynomials:
  - C(7,4): $g(x) = 1 + x + x^3$;
  - C(15,11): $g(x) = 1 + x + x^4$ or $g(x) = 1 + x^3 + x^4$;
  - C(31,26): $g(x) = 1 + x^2 + x^5$.
  - etc.
- the C(23,12) Golay code is also cyclic with either $g(x) = 1 + x^2 + x^4 + x^5 + x^6 + x^{10} + x^{11}$ or $g(x) = 1 + x^1 + x^5 + x^6 + x^7 + x^9 + x^{11}$.

Not cyclic: Hadamard.

Code generation with generator polynomials

What do we gain by using polynomials to describe codes?

- Polynomials are smaller than matrices. E.g. the C(6,4) RS code over GF(7) generated by the primitive element 5 has

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 5 & 4 & 6 & 2 & 3 \\ 1 & 4 & 2 & 1 & 4 & 2 \\ 1 & 6 & 1 & 6 & 1 & 6 \end{bmatrix}$$

and

$$g(x) = 6 + 5x + x^2.$$

- The computational cost of polynomial multiplication is comparable to matrix-vector multiplication.
- Efficient decoding methods for polynomials (coming soon).

Brief summary of code polynomials (so far)

Every cyclic linear C(n, k) code over GF(q) has a generator polynomial $g(x)$ over GF(q) such that

- $c(x)$ is a code polynomial $\iff$ $g(x)|c(x)$;
- $g(x)|x^n - 1$;
- $\deg(g(x)) = n - k$, and
- the main coefficient of $g(x)$ is $g_{n-k} = 1$.

We can generate codewords from message vectors with

$$c(x) = u(x)g(x).$$

This gives a $u \to c$ mapping different from matrix-vector multiplication.

Decoding with generator polynomials

For encoding in polynomial form, we will stick to using

$$c(x) = u(x)g(x).$$

Next we prepare for decoding using code polynomials.

The syndrome polynomial assigned to a received code polynomial $v(x)$ is

$$s(x) = v(x) \mod g(x)$$

A received polynomial $v(x)$ is a codeword $\iff$ $s(x) = 0$.

Error trapping algorithm

Next we present the Error trapping algorithm for decoding RS codes. It has more limited error correction capability than the Error locator algorithm, but it is much faster at any parameter setup.

Notably the Error trapping algorithm can correct $\left\lfloor \frac{n-k}{2} \right\rfloor$ errors for a $C(n, k)$ RS code as long as all of the errors fall close to each other.

This restriction is outside our usual definition of error correction capabilities over a symmetric channel, but for some physical channels, errors typically occur in bursts, and the Error trapping algorithm is very relevant in such situations.

Error trapping algorithm

We have a $C(n, k)$ RS code over $GF(q)$. Assume that the error vector e is such that

- $w(e) \leq \left\lfloor \frac{n-k}{2} \right\rfloor$;
- all errors in e occur within an interval of $(n-k)$ consecutive digits.

The received polynomial is

$$v(x) = u(x)g(x) + e(x)$$

due to the channel model.

Dividing the polynomial $v(x)$ by $g(x)$ yields

$$v(x) = a(x)g(x) + r(x)$$

where $\deg(r(x)) < \deg(g(x)) = n - k$.

Do the two equations imply $r(x) = e(x)$?

Error trapping algorithm

No, $r(x) = e(x)$ is not implied. We know that $\deg(r(x)) < n - k$, but $\deg(e(x))$ can be higher, up to $n - 1$.

However, we assumed that e has all errors within coordinates $[i, i + n - k - 1]$ for some i. Assume for a moment that the value of i is available. Then applying S^{-i} in advance for the previous calculations shifts the nonzero terms of $e(x)$ to the front; in other words, $\deg(S^{-i}e(x)) < n - k$.

Then

$$S^{-i}v(x) = S^{-i}u(x)g(x) + S^{-i}e(x)$$
$$S^{-i}v(x) = S^{-i}a(x)g(x) + r^{(i)}(x),$$

and now $S^{-i}e(x) = r^{(i)}(x)$ is implied since $S^{-i}e(x) - r^{(i)}(x)$ is a codeword, so it is a multiple of $g(x)$, but $\deg(S^{-i}e(x) - r^{(i)}(x)) < \deg(g(x))$, so $S^{-i}e(x) - r^{(i)}(x)$ must be 0.

Error trapping algorithm

Based on the previous calculation, if we knew the position i where the errors start in $e(x)$, we could do the following:

- compute the polynomial division $S^{-i}v(x) = S^{-i}a(x)g(x) + r(x)$, and
- the detected error polynomial is $e'(x) = S^i r(x)$.

The issue is that the position i is not known in advance.

So how can we obtain i?

Let i be any position (not necessarily where the errors in $e(x)$ start), and compute the polynomial division

$$S^{-i}v(x) = S^{-i}a(x)g(x) + r^{(i)}(x)$$

anyway.

The polynomial $S^{-i}e(x) - r^{(i)}(x) = S^{-i}v(x) - r^{(i)}(x)$ is a code polynomial, which means that either

$$S^{-i}v(x) - r^{(i)}(x) = 0 \quad \text{or} \quad w(S^{-i}v(x) - r^{(i)}(x)) \geq n - k + 1$$

because the RS code has minimal codeword distance $n - k + 1$.

Error trapping algorithm

Based on this, the Error trapping algorithm is the following:

- For each value of i from 0 to $n-1$, compute the polynomial division $S^{-i}v(x) = S^{-i}a(x)g(x) + r^{(i)}(x)$.
- Compute $w(S^{-i}v(x) - r^{(i)}(x))$.
- If $w(S^{-i}v(x) - r^{(i)}(x)) \geq n-k+1$, move on to the next value of i.
- If $w(S^{-i}v(x) - r^{(i)}(x)) \leq n-k$, then we stop, and the detected error polynomial is

$$e'(x) = S^i r^{(i)}(x).$$

If $e(x)$ indeed contains all $\leq \left\lfloor \frac{n-k}{2} \right\rfloor$ errors within an interval of length $\leq n-k$, then the algorithm is guaranteed to stop for at least one choice of i. (It is possible that several i positions are good, then we can use either of them.)

Decoding for cyclic codes other than RS

We have discussed two different decoding methods for RS codes: the Error locator method and the Error trapping method.

Both methods are general enough that they can be adapted to other cyclic linear codes, not just RS.

Error trapping methods are typically used for burst error situations, when the error digits are close to each other.

Architectures for polynomial calculations

Coding and decoding using code polynomials rely heavily on calculations with polynomials.

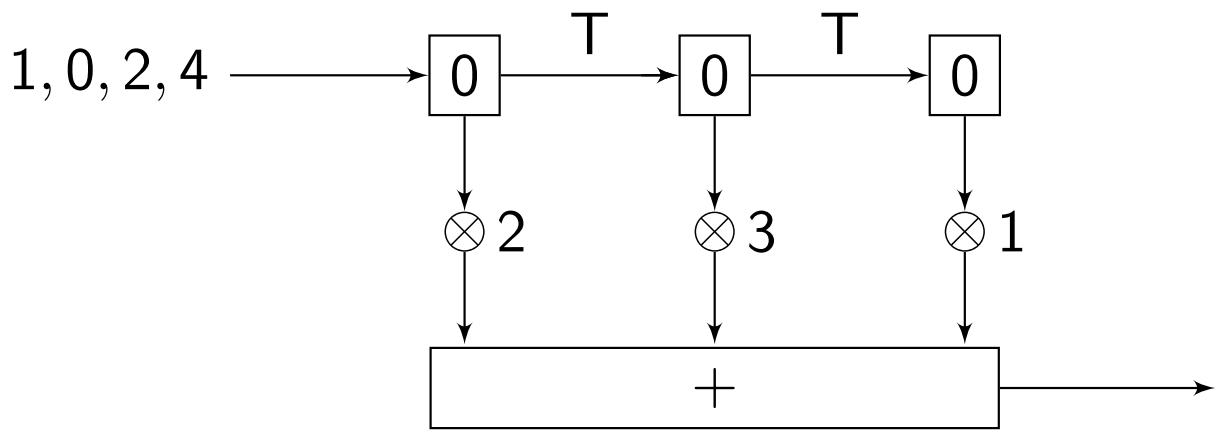
We want to make sure that polynomial calculations are fast.

Some of the operations (like addition) are straightforward; next we look at architectures for polynomial multiplication and polynomial division with remainder.

Polynomial multiplication by LFFSR

The Linear FeedForward Shift Register architecture for multiplication by $2 + 3x + x^2$.

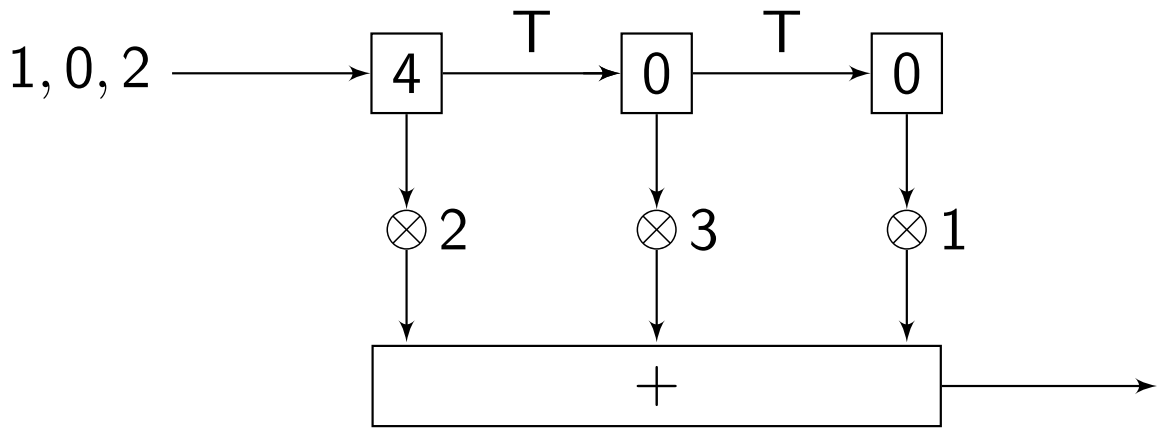
Compute $(4 + 2x + x^3)(2 + 3x + x^2)$ over GF(5).



Polynomial multiplication by LFFSR

The Linear FeedForward Shift Register architecture for multiplication by $2 + 3x + x^2$.

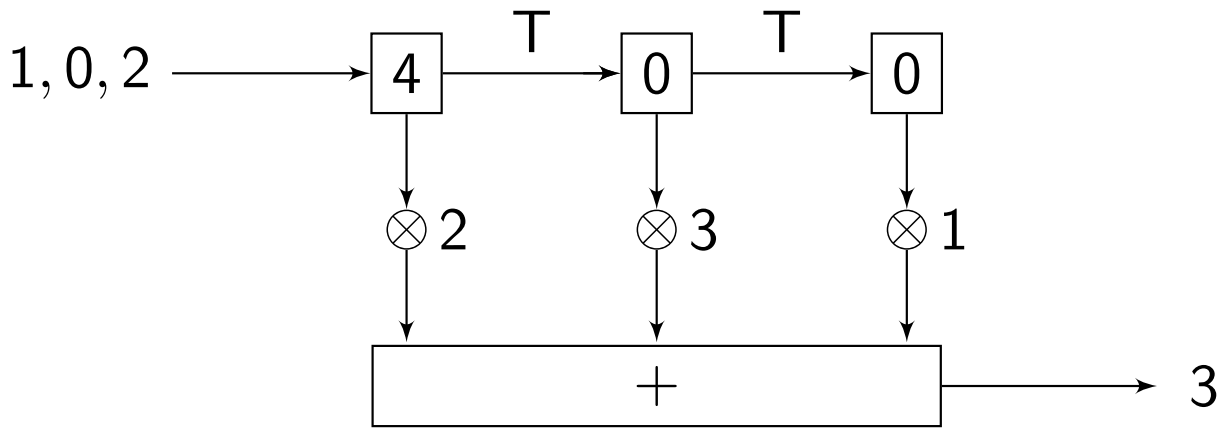
Compute $(4 + 2x + x^3)(2 + 3x + x^2)$ over GF(5).



Polynomial multiplication by LFFSR

The Linear FeedForward Shift Register architecture for multiplication by $2 + 3x + x^2$.

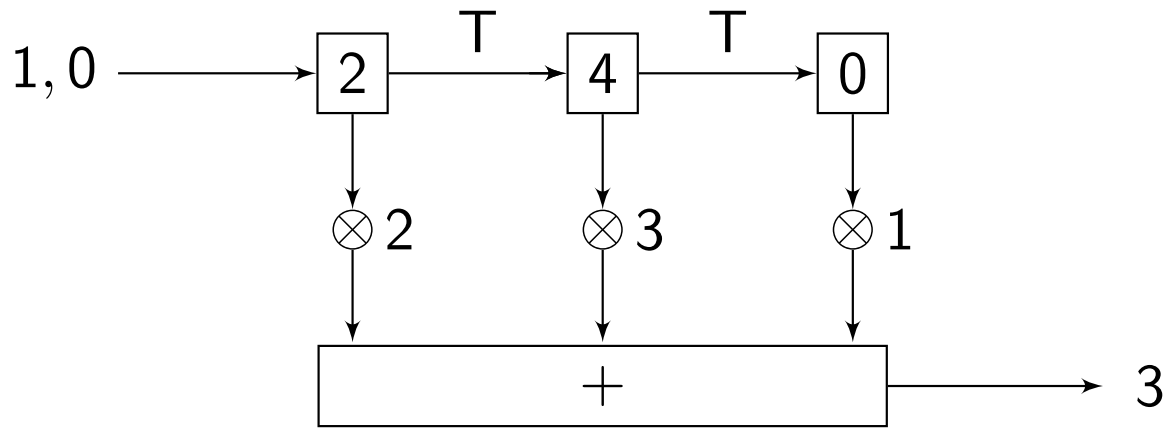
Compute $(4 + 2x + x^3)(2 + 3x + x^2)$ over GF(5).



Polynomial multiplication by LFFSR

The Linear FeedForward Shift Register architecture for multiplication by $2 + 3x + x^2$.

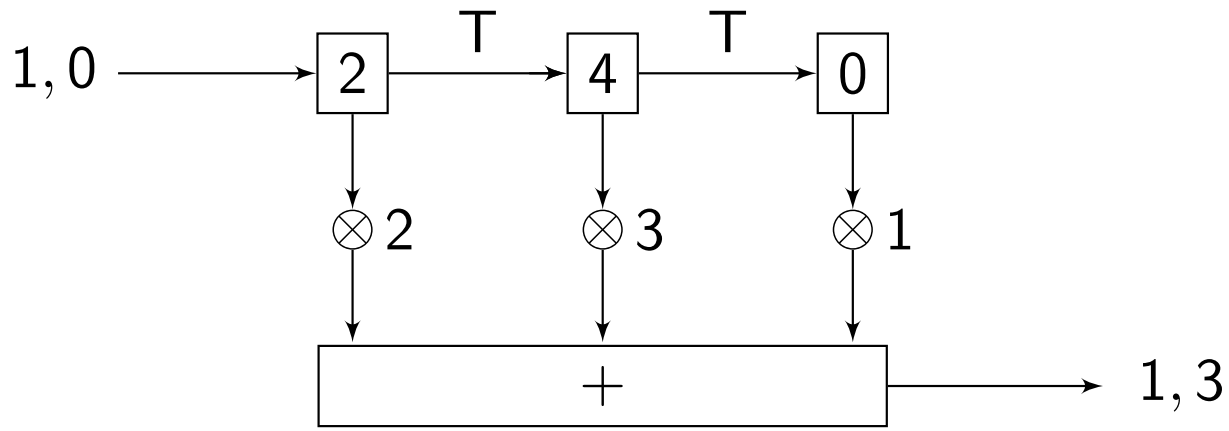
Compute $(4 + 2x + x^3)(2 + 3x + x^2)$ over GF(5).



Polynomial multiplication by LFFSR

The Linear FeedForward Shift Register architecture for multiplication by $2 + 3x + x^2$.

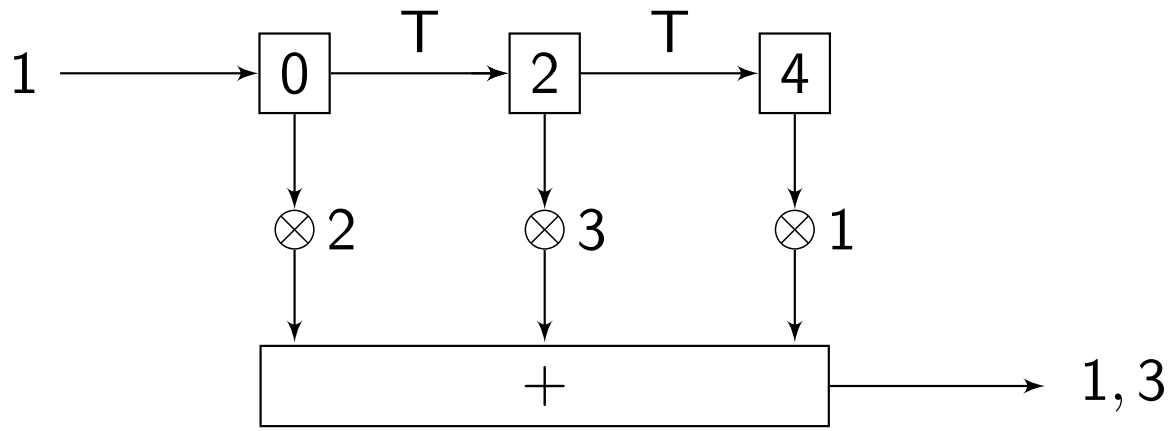
Compute $(4 + 2x + x^3)(2 + 3x + x^2)$ over GF(5).



Polynomial multiplication by LFFSR

The Linear FeedForward Shift Register architecture for multiplication by $2 + 3x + x^2$.

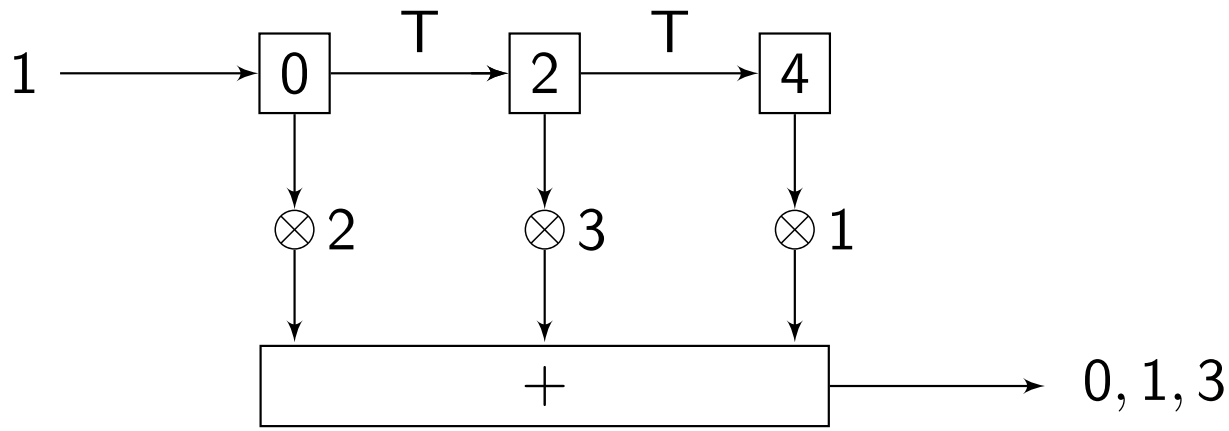
Compute $(4 + 2x + x^3)(2 + 3x + x^2)$ over GF(5).



Polynomial multiplication by LFFSR

The Linear FeedForward Shift Register architecture for multiplication by $2 + 3x + x^2$.

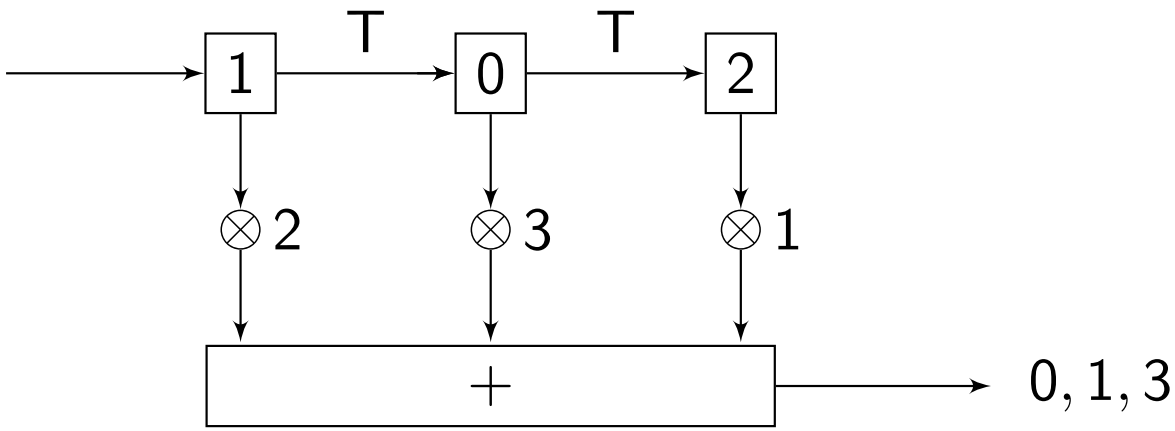
Compute $(4 + 2x + x^3)(2 + 3x + x^2)$ over GF(5).



Polynomial multiplication by LFFSR

The Linear FeedForward Shift Register architecture for multiplication by $2 + 3x + x^2$.

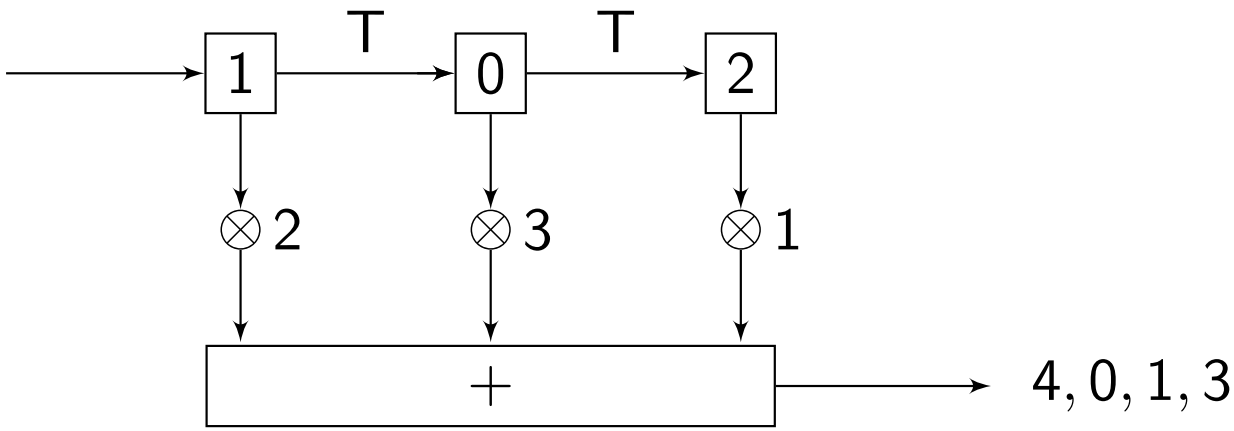
Compute $(4 + 2x + x^3)(2 + 3x + x^2)$ over GF(5).



Polynomial multiplication by LFFSR

The Linear FeedForward Shift Register architecture for multiplication by $2 + 3x + x^2$.

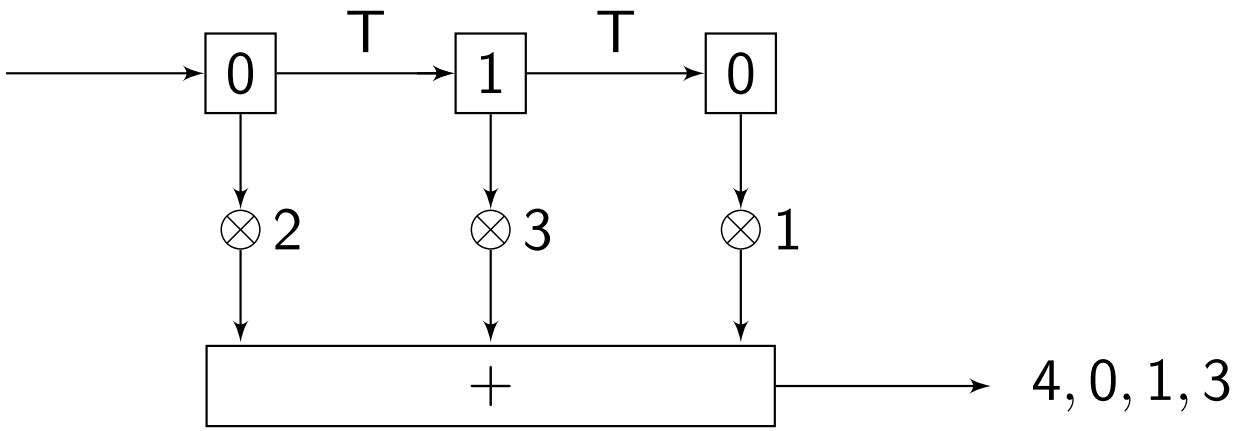
Compute $(4 + 2x + x^3)(2 + 3x + x^2)$ over GF(5).



Polynomial multiplication by LFFSR

The Linear FeedForward Shift Register architecture for multiplication by $2 + 3x + x^2$.

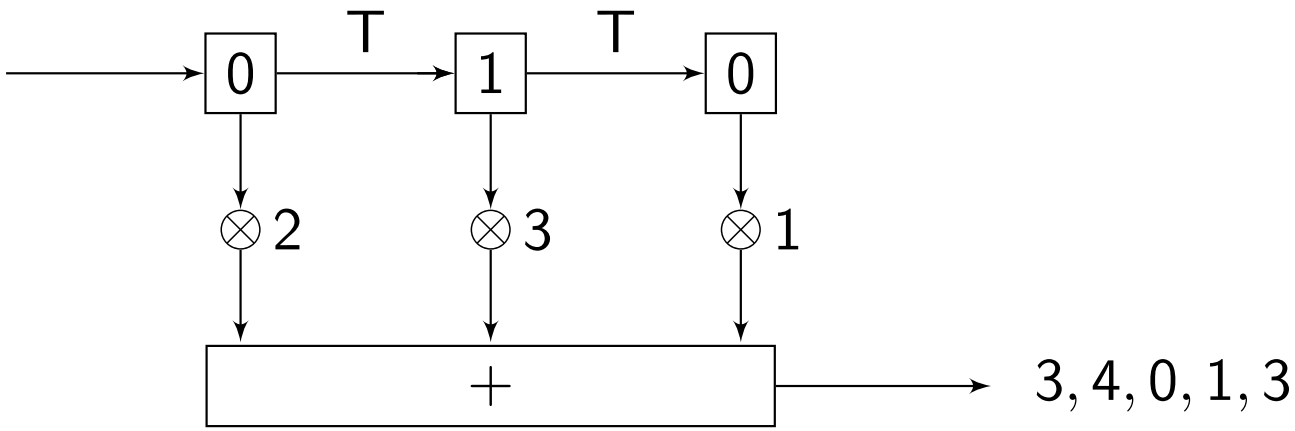
Compute $(4 + 2x + x^3)(2 + 3x + x^2)$ over GF(5).



Polynomial multiplication by LFFSR

The Linear FeedForward Shift Register architecture for multiplication by $2 + 3x + x^2$.

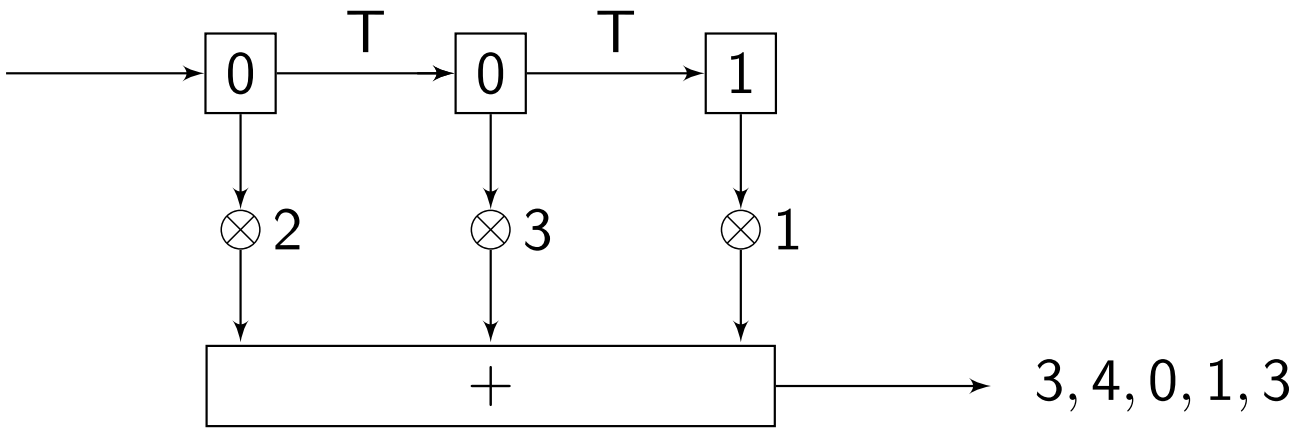
Compute $(4 + 2x + x^3)(2 + 3x + x^2)$ over GF(5).



Polynomial multiplication by LFFSR

The Linear FeedForward Shift Register architecture for multiplication by $2 + 3x + x^2$.

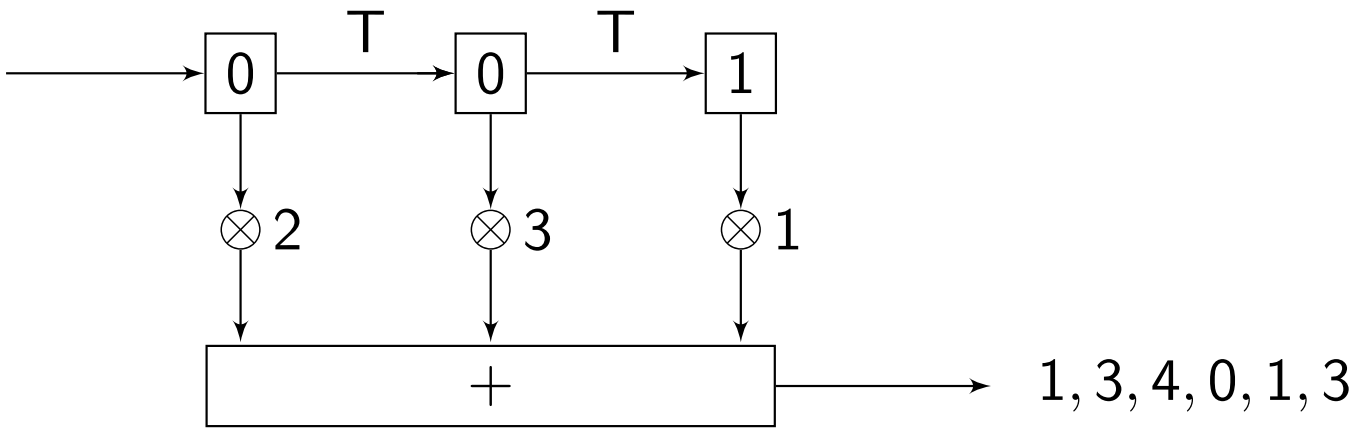
Compute $(4 + 2x + x^3)(2 + 3x + x^2)$ over GF(5).



Polynomial multiplication by LFFSR

The Linear FeedForward Shift Register architecture for multiplication by $2 + 3x + x^2$.

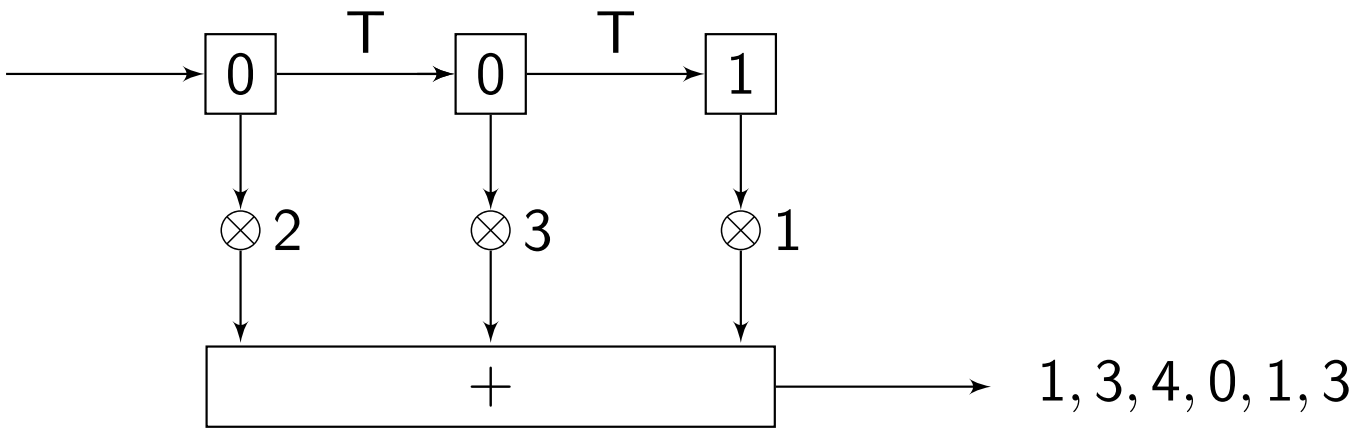
Compute $(4 + 2x + x^3)(2 + 3x + x^2)$ over GF(5).



Polynomial multiplication by LFFSR

The Linear FeedForward Shift Register architecture for multiplication by $2 + 3x + x^2$.

Compute $(4 + 2x + x^3)(2 + 3x + x^2)$ over GF(5).

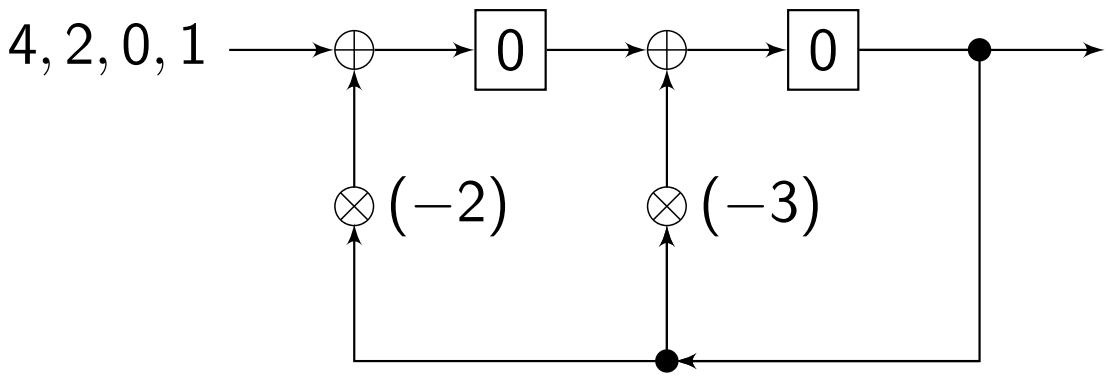


Result: $3 + 1 \cdot x + 0 \cdot x^2 + 4x^3 + 3x^4 + 1 \cdot x^5$.

Polynomial division by LFBSR

The Linear Feedback Shift Register architecture for division by $g(x) = 2 + 3x + x^2$ over GF(5). We assume the main coefficient of $g(x)$ is 1.

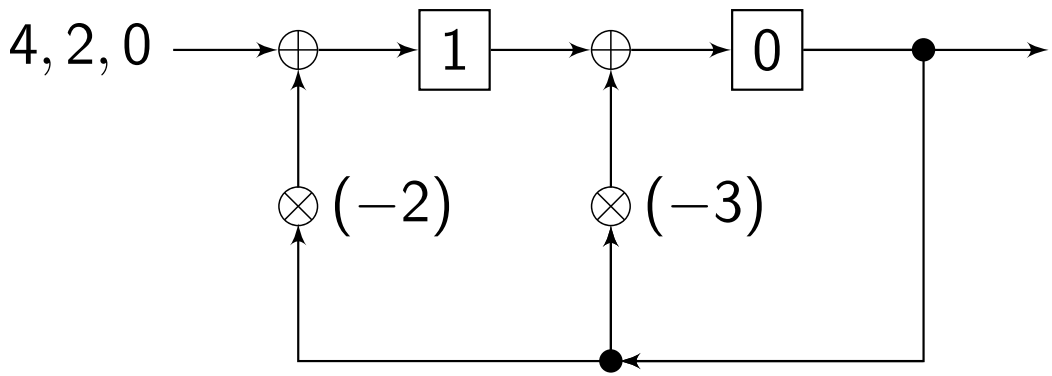
Compute $(4 + 2x + x^3) : (2 + 3x + x^2)$ over GF(5).



Polynomial division by LFBSR

The Linear Feedback Shift Register architecture for division by $g(x) = 2 + 3x + x^2$ over GF(5). We assume the main coefficient of $g(x)$ is 1.

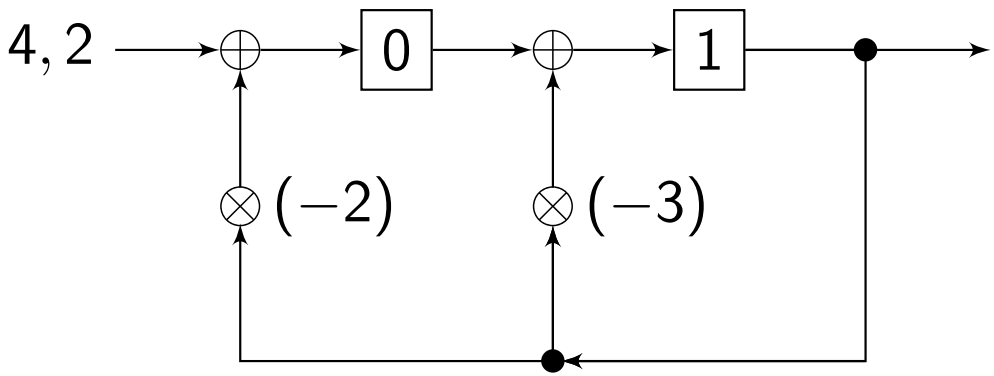
Compute $(4 + 2x + x^3) : (2 + 3x + x^2)$ over GF(5).



Polynomial division by LFBSR

The Linear Feedback Shift Register architecture for division by $g(x) = 2 + 3x + x^2$ over GF(5). We assume the main coefficient of $g(x)$ is 1.

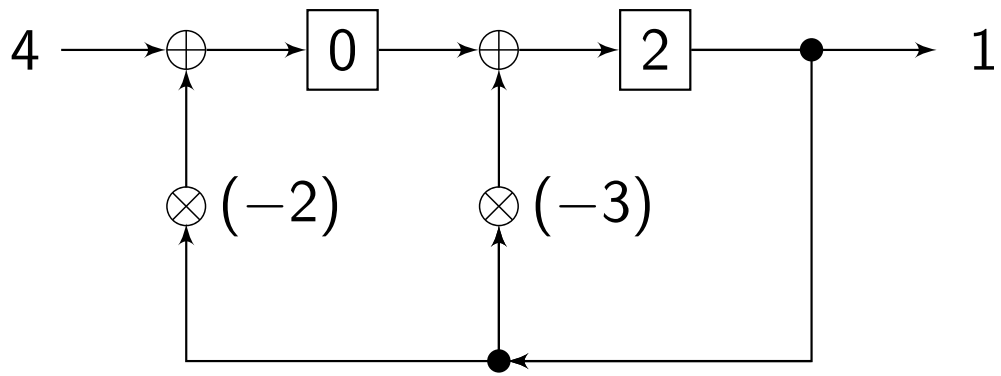
Compute $(4 + 2x + x^3) : (2 + 3x + x^2)$ over GF(5).



Polynomial division by LFBSR

The Linear Feedback Shift Register architecture for division by $g(x) = 2 + 3x + x^2$ over GF(5). We assume the main coefficient of $g(x)$ is 1.

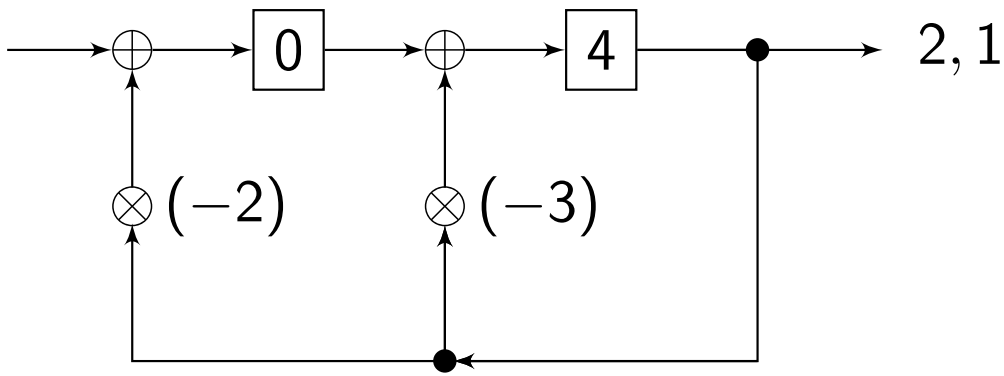
Compute $(4 + 2x + x^3) : (2 + 3x + x^2)$ over GF(5).



Polynomial division by LFBSR

The Linear Feedback Shift Register architecture for division by $g(x) = 2 + 3x + x^2$ over GF(5). We assume the main coefficient of $g(x)$ is 1.

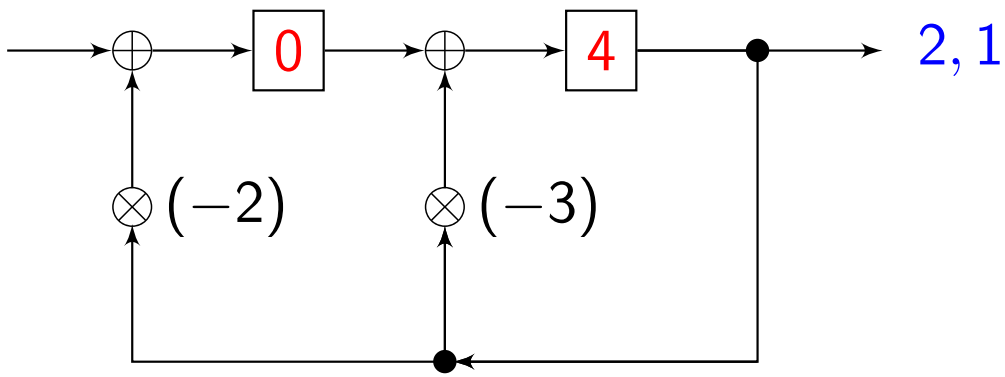
Compute $(4 + 2x + x^3) : (2 + 3x + x^2)$ over GF(5).



Polynomial division by LFBSR

The Linear Feedback Shift Register architecture for division by $g(x) = 2 + 3x + x^2$ over GF(5). We assume the main coefficient of $g(x)$ is 1.

Compute $(4 + 2x + x^3) : (2 + 3x + x^2)$ over GF(5).

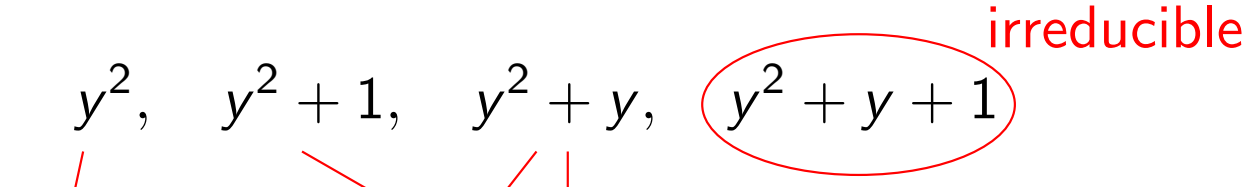


Result: $(2 + 3x + x^2)(2 + x) + 0 + 4x$.

Irreducible polynomials over GF(q)

We say that a polynomial $p(x)$ is irreducible over GF(q) if it cannot be written as the product of two polynomials of smaller degree. (Otherwise it is called reducible.)

Example. The binary polynomials of degree 2 are



Which of these are irreducible?

One way to answer that is to consider all polynomials of degree 1, and list all possible products:

$$y \cdot y, \quad y(y+1), \quad (y+1)y, \quad (y+1)(y+1)$$

and match them to the degree 2 polynomials.

GF(p^m) arithmetics

For a finite field GF(q), q can be either a prime or p^m (with p prime and $m \geq 2$).

Now we focus on the case $q = 2^m$.

$$GF(q) = \{0, 1, \ldots, q-1\}$$

Each element of GF(2^m) has 3 representations:

element	binary	polynomial
0	$(0 \ldots 00)$	0
1	$(0 \ldots 01)$	1
$\vdots$	$\vdots$	$\vdots$
α	$(\alpha_{m-1}, \ldots \alpha_1, \alpha_0)$	$a(y) = \alpha_{m-1} y^{m-1} + \cdots + \alpha_1 y + \alpha_0$
$\vdots$	$\vdots$	$\vdots$

We will typically use the polynomial representation.

GF(2^m) arithmetics

Elements of GF(2^m) are binary polynomials of degree $m-1$ or less.

Addition is polynomial addition mod 2. (This is equivalent to addition of binary vectors of length m.)

Multiplication in GF(2^m) will be polynomial multiplication mod $p(y)$, where $p(y)$ is a fixed irreducible polynomial of degree m.

Degree m makes sense for $p(y)$, since we want the result of a multiplication to be an element of the field, and only polynomials of degree $\leq m-1$ are elements of the field.

Why is $p(y)$ irreducible? If $p(y) = a(y)b(y)$, then $a(y)$ is a nonzero element of the field, but $a(y)b(y) \bmod p(y) = 0$, which is an issue, as $a(y)$ would have no multiplicative inverse.

$p(y)$ is called the reducing polynomial of the field.

GF(2^m) arithmetics

Theorem

GF(2^m) with the above arithmetics satisfies the field axioms.

Proof. Apart from the multiplicative inverse, every other axiom follows from the fact that they hold for binary polynomials, and polynomial addition and multiplication are consistent with mod $p(y)$.

So we only need to check that every nonzero element of the field has a multiplicative inverse. Let $a(y) \in \mathrm{GF}(2^m)$ be a nonzero element.

Consider the two sets

$$A = \{f(y) : \deg(f(y)) \leq m-1\}$$
$$B = \{a(y)f(y) \mod p(y) : \deg(f(y)) \leq m-1\}$$

Obviously $B \subseteq A$.

GF(2^m) arithmetics

The size of A is 2^m; if we can prove that all elements of B are distinct, then the size of B is also 2^m, which implies $A = B$.

Assume the opposite, that is, there exist distinct $f_1(y)$ and $f_2(y)$ such that

$$a(y)f_1(y) \mod p(y) = a(y)f_2(y) \mod p(y).$$

Then $p(y)|a(y)(f_1(y) - f_2(y))$. But $p(y)$ is irreducible, so this is only possible if either $p(y)|a(y)$ or $p(y)|f_1(y) - f_2(y)$.

But both $a(y)$ and $f_1(y) - f_2(y)$ have degree smaller than $p(y)$. $a(y)$ is nonzero, so $p(y)|a(y)$ is not possible, and we assumed $f_1(y)$ and $f_2(y)$ are distinct, so $p(y)|f_1(y) - f_2(y)$ is not possible either, leading to a contadiction.

So all elements of B are indeed distinct, so $A = B$, and since A includes the element 1, so does B, which gives the multiplicative inverse of $a(y)$.

Algebra over GF(4)

Irreducible polynomial: $p(y) = y^2 + y + 1$.

Elements of GF(4):

element	binary	polynomial
0	(00)	$0 \cdot y^1 + 0 \cdot y^0 = 0$
1	(01)	$0 \cdot y^1 + 1 \cdot y^0 = 1$
2	(10)	$1 \cdot y^1 + 0 \cdot y^0 = y$
3	(11)	$1 \cdot y^1 + 1 \cdot y^0 = y + 1$

Examples for addition:

$$y + (y + 1) = 2y + 1 = 0 \cdot y + 1 = 1,$$
$$1 + (y + 1) = y + 2 = y.$$

GF(4)

Irreducible polynomial: $p(y) = y^2 + y + 1$.

Elements of GF(4):

element	binary	polynomial
0	(00)	$0 \cdot y^1 + 0 \cdot y^0 = 0$
1	(01)	$0 \cdot y^1 + 1 \cdot y^0 = 1$
2	(10)	$1 \cdot y^1 + 0 \cdot y^0 = y$
3	(11)	$1 \cdot y^1 + 1 \cdot y^0 = y + 1$

Examples for multiplication:

$$y * y = y^2 \mod p(y) = y^2 - (y^2 + y + 1) = y + 1,$$
$$y * (y + 1) = y^2 + y \mod p(y) = y^2 + y - (y^2 + y + 1) = 1.$$

Irreducible polynomials of degree m

In order to define GF(2^m), we need an irreducible binary polynomial of degree m for the reducing polynomial $p(y)$. We have seen that this exists for $m = 2$, but what about larger values of m?

We introduce an extra property first: we call a polynomial of degree m primitive if it is irreducible and $p(y)|y^k - 1$ for $k = 2^m - 1$ and no k less than $2^m - 1$.

Theorem

For any m, there exists a primitive binary polynomial of degree m.

No proof, but primitive polynomials for low degrees:

degree	primitive polynomial	degree	primitive polynomial
2	$y^2 + y + 1$	8	$y^8 + y^4 + y^3 + y^2 + 1$
3	$y^3 + y + 1$	9	$y^9 + y^4 + 1$
4	$y^4 + y + 1$	10	$y^{10} + y^3 + 1$
5	$y^5 + y^2 + 1$	11	$y^{11} + y^2 + 1$
6	$y^6 + y + 1$	12	$y^{12} + y^6 + y^4 + 1$
7	$y^7 + y^3 + 1$	13	$y^{13} + y^4 + y^3 + y + 1$

Primitive element

Theorem

For any nonzero element $a(y) \in GF(2^m)$,

$$(a(y))^{2^m-1} = 1.$$

Proof. We had this exact same theorem for GF(q) for q prime; the same proof works.

A nonzero element $a(y)$ is primitive if $(a(y))^k = 1$ for $k = 2^m - 1$ and no k smaller than $2^m - 1$.

Theorem

There exists a primitive element in GF(2^m) for any m.

Moreover, if the reducing polynomial $p(y)$ is primitive, then y is a primitive element in GF(2^m).

No proof.

Power table

We will always assume that the reducing polynomial is primitive, and so y is always a primitive element. (Typically there are more primitive elements in GF(2^m), but we only need one.)

All nonzero elements of GF(2^m) can be obtained as the powers of a primitive element. This gives rise to the power table, where nonzero elements are matched with the proper power of y.

Example. The GF(4) power table:

1	1	y^0
2	y	y^1
3	$y+1$	y^2

($y^3 = 1$, so powers of y are 3-periodic.)

Power table

Example. The power table for GF(8) (reducing polynomial: $p(y) = y^3 + y + 1$):

1	1	y^0
2	y	y^1
3	$y + 1$	y^3
4	y^2	y^2
5	$y^2 + 1$	y^6
6	$y^2 + y$	y^4
7	$y^2 + y + 1$	y^5

For each nonzero element, the power table contains a polynomial form and a power form.

- Addition is typically easier in polynomial form, while
- multiplication is easier in power form.

The power table offers a quick way to switch between the two forms.