The GPS Challenge

Feedback-Shift-Registers

Generating Functions

Fibonacci FSRs

Fibonacci versus Galois *

Disclaimer: Finite Fields

There are some forward references to finite fields in this lecture.

Ignore those for the time being and think about $\mathbb{F}_2 = \{0, 1\}$.

Everything needed will be covered in the next few lectures.

Getting Around

Athena told Odysseus to “keep the Great Bear on his left”.

Global Positioning System

24 satellites, about 20,000 km above ground, moving at some 4.5 km/sec. Each has 4 highly accurate atomic clocks, everything tightly controlled by one master and four additional control stations.

The Problem:
At the other end, a GPS receiver must be cheap, small, reliable, zero maintenance.

This is a bit different from standard resource constraints on computation (time, space) but equally interesting.

Data centers now consume twice as much energy as New York City, with huge growth rates.

Fancy/Cheap System
Einstein’s Legacy

Einstein’s general theory of relativity is now over 100 years old.

A Question: Who wins?

- By general relativity, satellite clocks move faster since they are high up in Earth’s gravity field (about 45 µsec per day). BTW, nowadays one can measure the effect of lifting a super-accurate clock by 2cm.
- By special relativity, satellite clock undergo relativistic slow-down since they are moving relatively fast (about 7 µsec per day).

Final result: there is a speed-up of about 38 µsec per day.

http://physicscentral.com/explore/writers/will.cfm
And the engineers did not believe it.

Trilateration

Because of the cheapness/smallness as well as precision constraints, one cannot use radio direction finding—given the directions and knowledge of the positions of the satellites one could triangulate.

But we can use trilateration: we measure the distance to the satellites by measuring the delay of a signal. Given accurate timing and satellite location data, this suffices to determine location of the receiver.

OK, but how do we measure a delay of some 0.07 seconds? A non-solution would be to have the satellite send very short radio bursts and the receiver measure the delay of the signal.

The only thing our cheapo receiver can do in reality is to receive a nice, steady stream of bits sent from the satellite. Hence we need to somehow encode the timing information in a bit stream.

Measuring Delay, on the Cheap

Main Idea:
Send a stream of bits such that \( k \) consecutive bits suffice to determine the position in the stream.

Whole stream is periodic, but the period is quite long.

- Standard Positioning System, C/A signal: period 1023, sent once every millisecond. Accuracy: 100 m, 340 nanoseconds.
- Precision Positioning System, P and Y signals: military use, period of 267 days. Accuracy: 15 m, 200 nanoseconds.

To ascertain the delay, the GPS receiver generates the same sequence and syncs it up with the satellite signal.

The Challenge

Fix some positive integer \( k \), say, \( k = 50 \).

We want to construct a long bit sequence \( S \) such that someone reading \( k \) consecutive bits in the sequence will know where in the sequence the \( k \) bits are located.

Of course, for this to work at all, \( S \) can contain each \( k \)-bit word at most once. The most ambitious approach is to try to build \( S \) so that every \( k \)-bit word appears exactly once, leading to a length of \( 2^k + k - 1 \) bits: this is called a de Bruijn sequence of order \( k \).

But we will also settle for length approximately \( 2^k \).

Long Bit-Sequences

Well-behaved long bit-sequences have several computational applications

- global positioning system
- pseudo-random number generation
- stream ciphers

The problem in all cases is that we need a computationally cheap way to generate bit-sequences with very long periods.

In the extreme, computationally cheap here means in particular that the memory requirement should be a small constant.

The Perils of Education

Every CS major will have an immediate knee-jerk response: Just trace a Hamiltonian cycle in a de Bruijn graph . . .

This is \( B_3 \) and should look eminently familiar by now.
Recall: Euler and Hamilton

Definition
An Eulerian cycle in a digraph is a cycle that uses every edge exactly once.
A Hamiltonian cycle in a digraph is a cycle that uses every vertex exactly once.

At first glance, the two notions appear to be almost indistinguishable.
Computationally one would not expect any significant difference between the two.
Big mistake!

Hamiltonian versus Euler

Theorem
Testing whether a graph is Eulerian is linear time. Moreover, we can construct an Eulerian cycle in linear time, if it exists.

Note that the Eulerian cycle won’t be unique in general and there may be many of them, we are not talking about a complete enumeration here.

By contrast:

Theorem
Testing whether a graph is Hamiltonian is NP-complete.

A Trick

A moment’s thought shows that an Eulerian path in $B_{k-1}$ corresponds to a Hamiltonian path in $B_k$.

For the graph theorists: $B_k$ is the line graph of $B_{k-1}$.

But this means we can actually construct a Hamiltonian path in $B_k$: compute an Eulerian path in $B_{k-1}$ and then lift it.

A Computational Obstruction

While the graph algorithms that construct Eulerian cycles are linear time and thus optimal there is one problem:
We need to remember whether we have already touched a node.

This requires $\Omega(n)$ bits of storage for a size $n$ graph.

In our case $n = 2^{50}$, so any linear space algorithm is out.

- Are there other ways to generate a Hamiltonian cycle in $B_k$?
- Or at least a very long cycle, something of length nearly $2^k$?
- We would like to generate the $i$th bit in time and space $O(1)$.

Succinct Representations

To hammer this home: the problem we are trying to solve is trivial in a sense: there is a simple, linear time and linear space algorithm.

Unfortunately, it’s linear in $2^{50}$, so any explicit data structure is out.

But de Bruijn graphs are highly regular and have a nice succinct representation (virtual graphs). It seems plausible that there might be an algorithm that exploits this succinct representation to find Hamiltonian cycles, or at least very long cycles.
With constant memory.

This is the end of the age of innocence for graph algorithms, huge graphs are much harder to deal with (the web, model checking).
Perhaps we could find some easily computable function of the form

\[ F: 2^k \to 2^k \]

that can be iterated to produce a (hopefully very long) cycle:

- Make sure \( F \) is injective, so all orbits are periodic.
- Pick an initial \( k \)-bit block \( x_0 \).
- Iterate to obtain a sequence \( x_{i+1} = F(x_i) \) of \( k \)-bit blocks so that \( x_i \to x_{i+1} \) is an edge in the de Bruijn graph of order \( k \).

There always is a perfect solution (based on a Hamiltonian cycle), but we have a huge constraint: \( F \) must be easily computable.

Think of this as a circuit design problem: we have \( k \) one-bit registers and want to update their contents by some simple circuitry.

In the most general scenario, we could make every new bit depend on every old bit:

\[
\begin{align*}
  x_4 & \to x_3 & x_2 & \to x_1 \\
  x_3 & \to x_2 & x_1 & \to x_0 \\
  \end{align*}
\]

We can think of generating one output bit at register \( r_0 \) during each clock cycle.

The positions \( p_i \) are the so-called taps.
Where are We? 29

By choosing taps we obtain a local function \( f : 2^k \rightarrow 2 \), which gives rise to the global function \( F : 2^k \rightarrow 2^k \), \( F(x) = (x_2, \ldots, x_k, f(x)) \).

We need to make sure the following holds:

- The global map is injective, so we get periodic orbits.
- There is a good starting point \( a \) that produces a long orbit.

Of course, this all could go completely wrong–but as we will see, things work out nicely.

Reversibility 30

Recall that we always assume a tap at register \( r_0 \) so that \( c_k = 1 \).

Proposition

The global map \( F \) based on linear feedback function \( f \) is injective.

Proof.

Recall \( a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ldots + c_k a_{n-k} \). Hence we can run the recurrence backwards:

\[ a_{n-k} = a_n - (c_1 a_{n-1} + c_2 a_{n-2} + \ldots + c_k a_{n-k+1}) \]

\[ \square \]

Note that we could have written \( + \) instead of \( - \) since we have characteristic 2. The minus is simply keeping track of the underlying arithmetic.

Generating Bit-Sequences 26

To generate a bit-sequence, we choose \( k \) initial values for the registers, say \( a = (a_{k-1}, a_{k-2}, \ldots, a_1, a_0) \). We then use the \( k \)th order linear recurrence

\[ a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ldots + c_k a_{n-k} \]

to generate a sequence \( (a_i) \in (2^k)^\omega \).

Clearly, this kind of gadget is very easy to realize in circuitry. Requires only \( k \) one-bit registers, some xor circuits, and a clock (the whole shift-register must be synchronized).

Truth in advertising: xor gates usually have two inputs, so we may need to build a little tree to get the desired feedback bit.

Again: Rain Cheque 28

We will need to use the two-element field \( \mathbb{F}_2 \) below.

A full discussion of finite fields is the next big topic, for the time being think of this simply as

Arithmetic modulo 2.

It’s just like the rationals or reals, but better.

Engineering Angle 27

Engineers love these devices: they are easy to implement and lightning fast.

Exercise

The diagram above is stolen, is it the right one?

Alternative Description 25

Another way to think about FSRs: assume there is a tap at every register, but they have weights \( c_i \in 2 \) which determine whether the tap is on or off.

So the feedback value placed into the last register is the convolution

\[ c_1 x_{k-1} + c_2 x_{k-2} + \ldots + c_k x_0 \]

Note that we may safely assume that there is a tap at \( r_0 \); otherwise we are just inflating \( k \) and shifting the output bit a number of times before releasing it into the light.
Additivity

Proposition
The global map $F$ is additive:
$$F(x + y) = F(x) + F(y)$$
where all addition is mod 2.

In other words, if we think of $2^k$ as an $F_2$-vector space, then the map $F$ is linear.

It follows that we can determine the value of $F(x)$ by just adding the images of the canonical basis vectors:
$$F(e_i) = F(0, \ldots, 0, 1, 0, \ldots, 0)$$

Exercise
Prove that the global map is indeed additive.

Wishful Thinking

Note that 0 is always a fixed point, so it’s useless for long orbits.
The best we could hope for is a long orbit of size $2^k - 1$. Note that the starting point $[x] \neq 0$ would not matter in this case.

In general we will see that unit vector $e_1$ is always a good place to start.

Time for some experimentation.

An Orbit

Here is the orbit of $e_1$ under the FSR with taps $c_1 = c_k = 1$, $c_i = 0$ otherwise (which we will abbreviate as $(1, k)$) for $k = 6$.

Each column corresponds to the 6 registers, $r_0$ at the bottom; time flows left to right.

This orbit has optimal length 63. Of course, this is too good to be true in general . . .

Small Spans

<table>
<thead>
<tr>
<th>$k$</th>
<th>$p$</th>
<th>$p/2^k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td>0.875</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>0.9375</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>0.65625</td>
</tr>
<tr>
<td>6</td>
<td>63</td>
<td>0.984375</td>
</tr>
<tr>
<td>7</td>
<td>127</td>
<td>0.992188</td>
</tr>
<tr>
<td>8</td>
<td>63</td>
<td>0.246994</td>
</tr>
<tr>
<td>9</td>
<td>73</td>
<td>0.142578</td>
</tr>
<tr>
<td>10</td>
<td>889</td>
<td>0.868164</td>
</tr>
<tr>
<td>11</td>
<td>1533</td>
<td>0.748535</td>
</tr>
</tbody>
</table>

Taps $(1, k)$ work on occasion, but also fail badly. But some easy conjectures pop up:

Conjecture
The period is $2^k - 1$ for $k = 2^\ell - 1$ bits.

A Surprise

If we compute the same table for taps $(k-1, k)$, we get exactly the same orbit lengths. But the orbits are different! Here is the case $k = 6$ on top, the old $(1, 6)$ orbit below.

Why?

Phasespace

Here is why things go wrong for $k = 9$:

There is one fixed point, plus 7 cycles of length 73.
And Another

\[ k = 8 \text{ with taps } 01101011. \]

Again, too many cycles to get a long orbit.

Impulse-Response Sequences

Additivity has an important side-effect when it comes to periods: we can obtain maximal period by selecting as the starting configuration the unit vector \( e_1 \).

Definition

An impulse-response sequence for \( F \) is an orbit obtained from a basis vector \( e_1 \).

Lemma (Period Lemma)

The period of any configuration \( a \) divides the period of \( e_1 \).

These cycles won't be Hamiltonian, but at least for some choices of \( k \) and the feedback function they are quite long. Moreover, the next bit can indeed be computed in constant time and space.

We need a little more machinery (which is also independently useful) for the proof of the lemma.

The Companion Matrix

Definition

Let \( a_n = c_1a_{n-1} + c_2a_{n-2} + \ldots + c_k a_{n-k} \) be linear recurrence of order \( k \) over some ring \( R \). The (Frobenius) companion matrix \( C \) of the recurrence is a \( k \times k \) matrix over \( R \) defined by

\[
C(i,j) = \begin{cases} 
  c_j & \text{if } i = 1, \\
  1 & \text{if } i = j + 1, \\
  0 & \text{otherwise.}
\end{cases}
\]

For example, for \( k = 5 \) the companion matrix looks like so:

\[
C = \begin{pmatrix} 
  c_1 & c_2 & c_3 & c_4 & c_5 \\
  1 & 0 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 & 0 \\
  0 & 0 & 1 & 0 & 0 \\
  0 & 0 & 0 & 1 & 0
\end{pmatrix}
\]

Note that there are several versions of the companion matrix in the literature.

Matrix Multiplication and Iteration

By multiplying the companion matrix with a vector representing the current bit-pattern in the registers we can compute the next bit-pattern.

Proposition

\( C^t a \) is the content of the registers at time \( t \) where \( a = (a_{k-1}, \ldots, a_0) \) is the initial configuration.

Computationally this produces the following speed-up: using a standard matrix multiplication algorithm and fast exponentiation it would take us \( O(k^3 \log t) \) steps to compute the state of the system at time \( t \).

This is an example of predictability or computational compressibility: it does not take \( \Omega(t) \) steps to find the configuration at time \( t \).

Proof of Period Lemma

Let \( q \) be the period of the impulse-response sequence generated by \( e_1 \), \( C \) the companion matrix.

Then \( C^q e_1 = e_1 \) and hence for any \( i \geq 0 \) we have

\[
C^{q+i} e_1 = C^q e_1
\]

and therefore

\[
(C^q - I) \cdot (C^q e_1) = 0.
\]

Since \( c_k \neq 0 \), the vectors \( C^q e_1 \) must span the whole space; hence we must have \( C^q = I \).

It follows that \( q \) is a period of any configuration, and the least period must divide \( q \).

- The GPS Challenge
- Feedback-Shift-Registers
- Generating Functions
- Fibonacci FSRs
- Fibonacci versus Galois *
The key question now is: Where should the taps go?

Note that we actually have a problem if we really succeed: If the cycles obtained from an impulse-response sequence are indeed very long then we cannot really find out by simulation, at least not for large $k$.

Hence, we need a computational shortcut, some way of computing periods from the tap positions without brute-force simulation.

Exercise

For a reasonably small value of $k$, say, $k = 10$, determine the complete cycle structure of $2^k$ for all possible feedback shift-registers of order $k$.

There is nothing more practical than a good theory.
Kurt Lewin

Consider the sequence $(a_n)$ of bits in the rightmost register $r_0$. From the initial configuration $a = (a_0, a_1, \ldots, a_{k-1})$ we get the recurrence

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ldots + c_k a_{n-k}$$

where the coefficients $c_i$ are determined by the taps as above, $n \geq k$.

Is there any chance that we can find a good description for the generating function

$$G(x) = \sum_n a_n x^n$$

so that

$$G(x) = \frac{1 - x}{1 - x - x^4}$$

Consider initial conditions $a = (1, 0, 0, 0)$ and taps $(1, 4)$, so we are dealing with a Fibonacci-type recurrence $a_n = a_{n-1} + a_{n-4}$ for $n \geq 4$.

Then for $G(x) = \sum a_n x^n$

$$G(x) = 1 + \sum_{n \geq 4} a_n x^n$$

$$= 1 + x \sum_{n \geq 3} a_n x^n + x^4 \sum_{n \geq 0} a_n x^n$$

$$= 1 - x + x \sum_{n \geq 3} a_n x^n + x^4 \sum_{n \geq 0} a_n x^n$$

$$= 1 - x + (x + x^4) G(x)$$

$$G(x) = \frac{1 - x}{1 - x - x^4}$$

The generating function for our feedback shift-registers is

$$G(x) = \frac{1 - \sum_{i=0}^{k-1} \left( \sum_{j=0}^{i} c_j a_{n-j} \right)x^i}{1 - \sum_{i} c_i x^i}$$

where $c_0 = -1$.

Sketch of proof.

$$G(x) = \sum_{i} c_i x^i \left( a_i x^{-1} + \ldots + a_{-1} x^1 + \sum_n a_n x^n \right)$$
Feedback Polynomial

Definition
The denominator of this rational function is called the feedback polynomial or the connection polynomial of the sequence \((a_n)\).

Note that one can solve the equation

\[
- \sum_{i=0}^{k-1} \sum_{j=0}^{i} a_{i-j} \) \(x^i = 1
\]

for \(a_i\). The solution yields the initial conditions for a sequence whose generating function simplifies to the reciprocal of the feedback polynomial:

\[
G(x) = \frac{1}{1 - \sum_i c_i x^i}
\]

In general, however, the numerator is some polynomial of degree less than \(k\).

Example

For taps \((1,5)\) and the impulse-response sequence we get

\[
G(x) = \frac{1-x}{1-x-x^5}
\]

By Taylor expansion, we can compute a few terms of this series:

\[
1 + x^2 + x^6 + x^7 + x^8 + x^9 + x^{11} + x^{16} + x^{17} + \ldots
\]

Right in spirit, but actually wrong: we need coefficients in \(F_2\), not integers.

Wrong Field

We are computing in characteristic 0, but we should be using characteristic 2. Then the expansion looks like

\[
1 + x^2 + x^6 + x^7 + x^8 + x^{11} + x^{16} + x^{17} + \ldots
\]

and that is the right answer.

Of course, computing Taylor expansions is not really an answer at all: we need to calculate the period, and the first few Taylor coefficients tell us nothing about that.

Period and Polynomial Divisors

Theorem
Let \((a_n)\) be a sequence with generating function \(G(x) = 1/g(x)\). Then the period of \((a_n)\) is the least \(p > 0\) such that \(g(x)\) divides \(1 - x^p\).

Proof.
If the period is \(p\) then

\[
1/g(x) = (a_0 + a_1 x + \ldots + a_{p-1} x^{p-1}) (1 + x^p + x^{2p} + \ldots)
\]

So \(1 - x^p = g(x) \cdot (a_0 + a_1 x + \ldots + a_{p-1} x^{p-1})\) and \(g(x)\) divides \(1 - x^p\), as required.

Changing Initial Conditions

Suppose we change the initial conditions to \((1,1,0,0,1)\). Then

\[
G(x) = \frac{1 - x^2 + x^4}{1-x-x^5}
\]

and the orbit looks like:

Exercise
Match the series from above with the picture.

And Back

On the other hand, if \(1 - x^p = g(x)(b_0 + b_1 x + \ldots + b_{p-1} x^{p-1})\) then

\[
1/g(x) = (b_0 + b_1 x + \ldots + b_{p-1} x^{p-1}) / (1 - x^p)
\]

Comparing coefficients we get \(a_n = b_n \mod p\), so the period must of the sequence must divide \(p\).

Since \(p\) is minimal, they must agree.

\(\square\)
**Long Cycles and Primality**

**Definition**
The least $p > 0$ such that $g(x)$ divides $1 - x^p$ in $\mathbb{F}[x]$ is called the exponent of $g(x)$.

It is not clear how to compute exponents efficiently, but one can show the following.

**Theorem**
*If a shift-register sequence of span $k$ has maximum length $2^k - 1$, then the corresponding polynomial $g(x)$ must be irreducible.*

Unfortunately, this condition is not sufficient.
But quite a bit is known about the exponents of irreducible polynomials.

---

**Example Mersenne Prime**

31 is a Mersenne prime, $k = 5$.

There are exactly 6 irreducible polynomials of degree 5 in $\mathbb{F}_2[x]$, the orbit for the last is plotted below, and duly has length 31.

- $1 + x^2 + x^4, 1 + x^3 + x^5, 1 + x + x^2 + x^3 + x^5, 1 + x + x^2 + x^3 + x^5$, $1 + x^2 + x^3 + x^4 + x^5$
- $1 + x + x^2 + x^4 + x^5, 1 + x + x^3 + x^4 + x^5, 1 + x^2 + x^3 + x^4 + x^5$

---

**Exploiting Mersenne**

Here is a nice trick that exploits primality.

As we will see in a while, any irreducible polynomial $g(x)$ of degree $k$ divides $1 - x^{2^k - 1}$. Hence the exponent of $g(x)$ must divide $2^k - 1$.

So if $2^k - 1$ happens to be prime (a so-called Mersenne prime), then the exponent must be equal to $2^k - 1$.

It is an open problem whether infinitely many Mersenne primes exist, but for the following values of $k$ we do get a Mersenne prime $2^k - 1$:

- $2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127$

These are all appropriate $k$'s less than 256.
BTW, the largest known prime is a Mersenne prime (presumably $M_{74,207,281}$):

$$2^{74,207,281} - 1$$ (discovered January 2016).

---

**Primitive Field Elements aka WTF**

Here is what is actually going on, assuming a little field theory:

It is known that the multiplicative subgroup of every finite field is cyclic.

More precisely, if we have $K = \mathbb{F}/(f)$ where $f$ is irreducible and primitive then we can choose $\alpha = x \mod f$ as a generator.

But the order of $\alpha$ is $2^k - 1$, and likewise for $\alpha^{-1}$.

Hence if we choose the taps of our FSR according to the coefficients of $f$ we obtain maximal period.

Basta.
Here is a wild & woolly idea: maybe shift register sequences can be explained completely in terms of finite fields, something along the lines of Vandermonde matrix.

The matrix of this system has a special form: it’s (the transpose of) a Vandermonde matrix. However, there are enough initial vectors to produce different sequences as there are $2^k$ initial conditions.

Definition

The trace function of $F_{2^k}$ over $F_2$ is defined by

$$\text{Tr} : F_{2^k} \to F_2 \quad \text{Tr}(z) = \sum_{i<k} z^{2^i}$$

Proposition

$	ext{Tr}$ is linear and its range is indeed $F_2$.

Exercise

Prove the last proposition.

The question arises how many of the $2^k$ possible initial vectors are of the special form

$$\text{Tr}(\beta), \text{Tr}(\alpha^{-1}\beta), \ldots, \text{Tr}(\alpha^{2-k}\beta), \text{Tr}(\alpha^{1-k}\beta).$$

The determinant of $V$ is

$$V(\gamma_1, \ldots, \gamma_m) = \begin{pmatrix}
1 & \gamma_1 & \gamma_1^2 & \cdots & \gamma_1^{n-1} \\
1 & \gamma_2 & \gamma_2^2 & \cdots & \gamma_2^{n-1} \\
1 & \gamma_3 & \gamma_3^2 & \cdots & \gamma_3^{n-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \gamma_m & \gamma_m^2 & \cdots & \gamma_m^{n-1}
\end{pmatrix}$$

The determinant of $V$ is $\prod_{i<j}(\gamma_i - \gamma_j)$.

It follows that our matrix is invertible. But then $\beta = 0$, done. Hence all $2^k$ initial conditions can be generated by the right choice of the multiplier $\beta$.

Consider a linear homogeneous equation of order $k$ over $\mathbb{Z}$:

$$a_n = a_1n_{n-1} + \cdots + a_kn_{n-k}$$

- The sequence $(a_n)$ is rational if the generating function $G(x) = \sum a_n x^n$ is rational.
- $(a_n)$ is polynomial if $a_n = p(n)$ for some $p \in \mathbb{Q}[x]$.
- $(a_n)$ has polynomial growth if $a_n = O(n^d)$ for some $d \geq 0$.

One can similarly define algebraic and transcendental sequences.
Lemma
Every polynomial sequence is rational.

Proof.
\[
\frac{1}{1 - x} = \sum x^n = \sum \binom{n}{k} x^n
\]
Since \((\binom{n}{k})\) forms a basis for polynomials in \(n\) we are done. \(\square\)

Eventually Quasi-Polynomial Sequences

(aₙ) is eventually quasi-polynomial if there is a period \(N\) and a cutoff \(n₀\) such that
\[aₙ = pₙ \text{ mod } N(n)\]
for all \(n ≥ n₀\) and \(p₀, \ldots, p_N - 1 ∈ \mathbb{Q}[x]\).

Example: \(aₙ = \lfloor (n + 1)/2 \rfloor\) with period 2, \(n₀ = 0\), \(g₀(n) = n/2\) and \(g₁(n) = (n + 1)/2\).

It is easy to see that eventually quasi-polynomial sequences also are rational: sum the \(x^i G_i(x^N)\) and correct for the first few terms.

Polynomial Growth

Lemma
Suppose \((aₙ)\) is linear homogeneous and has polynomial growth. Then \((aₙ)\) is eventually quasi-polynomial.

Why? The poles of the generating functions are roots of unity and make periodic contributions.
If there are roots inside the unit circle, we get exponential growth (as in \(2^n\) with generating function \(1/(1 - 2x))\).

■ The GPS Challenge
■ Feedback-Shift-Registers
■ Generating Functions
■ Fibonacci FSRs
■ Fibonacci versus Galois *

Fibonacci Feedback Shift-Registers

The feedback shift-registers we have considered so far generate linear recurrent sequences according to
\[aₙ = c₁aₙ₋₁ + c₂aₙ₋₂ + \ldots + cₖaₙ₋ₖ \]
Since this generalizes the classical Fibonacci sequence, these FSR are also called Fibonacci feedback shift-registers (FFSR): we can think of the classical Fibonacci sequence as the special case taps \((1, 2)\), executed over the ring \(\mathbb{Z}\) rather than the field \(\mathbb{Z}_2\).

We already have an excellent description of these sequences in terms of finite fields. Here are just a few more comments.

Characteristic Polynomial

Definition
The characteristic polynomial of a square matrix \(M\) is defined by
\[\gamma(x) = |M - xI|\]
For a companion matrix \(C\) it is easy to see that the characteristic polynomial has the form
\[\gamma(x) = x^k - c₁x^{k-1} - c₂x^{k-2} - \ldots - cₖ\]
Compare this to the connection polynomial
\[g(x) = -1 + c₁x + c₂x² + \ldots + cₖx^k\]
So they are “mirror images” of each other: \(-x^k\gamma(1/x) = g(x)\).
Can we exploit this fact?
Reciprocal Polynomials 73

Definition
The reciprocal of a polynomial \( f(x) \) of degree \( k \) is the polynomial
\[
f^*(x) = x^k f(1/x).
\]

The map \( f \mapsto f^* \) is not very well behaved (it is not a homomorphism), but we have the following properties.

Proposition
Let \( f \) and \( g \) be two polynomials.
- If \( f(0) \neq 0 \) then \( (f^*)^* = f \).
- \( (f \cdot g)^* = f^* \cdot g^* \).

Exercise
Prove the last proposition.

Connection and Characteristic 74

By the last proposition, \( f \) is irreducible if, and only if, \( f^* \) is so. Irrreducible.

Moreover, in characteristic 2 we have \((1 + x^d)^* = 1 + x^d\), so the exponents of an irreducible polynomial and its reciprocal are the same.

Hence, as far as irreducibility and exponent are concerned, there is no difference between the connection polynomial \( g(x) \) of a FFSR and the characteristic polynomial \( \gamma(x) \) of its companion matrix.

Warning: Some authors use \( \gamma(x) \) instead of \( g(x) \) (but refer to it as the connection polynomial).

Example Degree 10 75

For degree 10 there are 99 irreducible polynomials in \( \mathbb{F}_2[x] \).
The frequencies for their exponents are

<table>
<thead>
<tr>
<th>exp</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>33</td>
<td>6</td>
</tr>
<tr>
<td>93</td>
<td>30</td>
</tr>
<tr>
<td>341</td>
<td>60</td>
</tr>
</tbody>
</table>

So almost 2/3 of the irreducible polynomials would produce maximum length FSR sequences.

Note that the map \( f \mapsto f^* \) moves polynomials only within each of the groups.

Counting Irreducibles 76

How many irreducible polynomials of degree \( k \) are there in \( \mathbb{F}_2[x] \)?

Let’s write \( I_k \) for this number, so trivially \( 0 \leq I_k < 2^k \).

Proposition
\[
\frac{1}{1 - 2z} = \prod_{m \geq 1} \left( \frac{1}{1 - z^m} \right)^{I_m}
\]

This comes down to the simple observation that every polynomial is a product of powers of irreducible ones, and this decomposition is essentially unique: if we order the factors in some canonical way, then the decomposition is unique.

Exercise
Prove the proposition using the hint above.

Möbius Inversion 77

A bit more fumbling shows that

Lemma
Let \( I_m \) be the number of irreducible polynomials in \( \mathbb{F}_2[x] \) of degree \( m \). Then
\[
2^m = \sum_{d|m} d I_d.
\]

We can apply Möbius inversion to this expression to find:
\[
I_m = \frac{1}{m} \sum_{d|m} \mu(m/d).
\]

where \( \mu \) is the Möbius function.

A Table 78

The last expression is fairly elegant, but it’s not clear what one should expect numerically.

<table>
<thead>
<tr>
<th>( m )</th>
<th>( I_m )</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>99</td>
</tr>
<tr>
<td>11</td>
<td>186</td>
</tr>
<tr>
<td>12</td>
<td>335</td>
</tr>
<tr>
<td>13</td>
<td>630</td>
</tr>
<tr>
<td>14</td>
<td>1161</td>
</tr>
<tr>
<td>15</td>
<td>2182</td>
</tr>
<tr>
<td>16</td>
<td>4080</td>
</tr>
<tr>
<td>17</td>
<td>7710</td>
</tr>
<tr>
<td>18</td>
<td>14532</td>
</tr>
<tr>
<td>19</td>
<td>27594</td>
</tr>
<tr>
<td>20</td>
<td>52377</td>
</tr>
</tbody>
</table>

Incidentally, \( I_{50} = 22517997465744 \).
From the point of view of the circuitry needed to implement a FSR, the Fibonacci type has one small drawback: it involves the addition of all the tapped registers. In particular when the FSR is programmable (i.e., when the taps can be set arbitrarily), this incurs a bit of overhead.

Could we modify the device slightly so that both the shift and the feedback operations can be performed in "one step" by the circuitry?

Another crazy idea: How about reversing all the arrows in an FFSR? In other words, feed the content of the last register back to all the tapped registers.

Recall: Fibonacci FSR

Our Fibonacci FSRs look like so:

![Fibonacci FSR Diagram]

The taps sample the registers and we send the sum back to the last register (after shifting everybody out of the way).

Galois Feedback Shift-Registers

These devices are called Galois feedback shift-registers (GFSR). The global function now looks like so:

$$F : 2^k \to 2^k$$

$$F(x) = (0, x_1, x_2, \ldots, x_{k-1}) + x_k c_k$$

Thus if $$x_k = 0$$ we simply shift; otherwise we shift and then flip all bits in certain positions.

This is really what we did in the discussion of finite fields to explain multiplication by $$x$$ modulo $$f(x)$$.

Galois versus Fibonacci

This looks quite similar to our original Fibonacci design, so one might expect the bit sequences obtained from a GFSR to be somehow related to the FFSR sequences.

First off, let us agree that the bit sequence $$(b_n)$$ generated by a GFSR is read off at register $$r_0$$.

Note that given initial values $$a_k, \ldots, a_0$$ it is in general not the case that $$b_i = a_i$$ for $$i < k$$ (for FFSR there is no problem).

We have $$b_0 = a_0$$, but the other bits may differ.

Second, for a GFSR it is not so easy to write down a recurrence equation: the distributed nature of the changes speeds up the circuits, but makes the math more complicated.
Experiment 85

Time to stop worrying and start computing.

Here is a picture for a two-tap GFSR with \( c = (1, 0, 0, 0, 1) \).

The first 4 steps a pure shifts, and at step 5 the masking takes effect.
At any rate, the picture looks quite similar to the Fibonacci case, so we should expect similar results.

Exploiting Periodicity 87

Again we may assume that \( c_k = 1 \); otherwise we are simply inflating \( k \).

But then the induced map \( F : 2^k \rightarrow 2^k \) is injective, just as in the Fibonacci case.

Hence the corresponding bit-sequence \( (b_n) \) obtained at register \( r_0 \) is strictly periodic and its generating function must be rational:

\[
B(x) = \frac{P(x)}{Q(x)}
\]

where \( P \) is some polynomial of degree less than the degree of \( Q \).

- What are the polynomials in question?
- How do they compare to the polynomials for a Fibonacci SR?

Generating Functions for Galois FSR 88

Theorem

Let \( B(x) \) be the generating function for the sequence obtained at register \( r_0 \) in a Galois FSR. Then

\[
B(x) = \frac{\sum_{i=1}^{k-1} a_i x^i}{1 + \sum_{i=1}^{k} c_i x^i}
\]

Proof.

For simplicity we consider only \( b_n \) for \( n \geq k \), the first few values require slightly more fumbling.

Since the registers are shifted at each step, \( b_n \) depends only on \( b_{n-k}, \ldots, b_{n-1} \) but none of the earlier values. Moreover, because of the masking whenever the output bit is 1 we have

\[
b_n = \sum_{i=1}^{k} c_i b_{n-i}.
\]

Done by comparing coefficients.

Déjà vu all over again 89

So the connection polynomials are the same – but recall that we are numbering the \( c \) coefficients backwards. In other words, you have to take the mirror image of the connection vector in the Fibonacci case.

The numerators are harmless: they comprise all polynomials of degree at most \( k - 1 \), but to get the same polynomial we need different initial conditions.

Thus, we get exactly the same bit-sequences, but we have to adjust the taps and the initial conditions.

This is really surprising, it’s intuitively far from clear that GFSRs and FFSRs are equivalent in this sense.

Example: \( \mathbb{F}_{16} \) 90

Consider the irreducible polynomial \( f(x) = 1 + x^3 + x^4 \in \mathbb{F}_2[x] \).

In this case \( \alpha = x/(f) \) is primitive and generates the whole multiplicative subgroup.

\[
\begin{align*}
ap^0 &= 1 & a^8 &= x + x^2 + x^3 \\
ap^1 &= x & a^9 &= 1 + x^2 \\
ap^2 &= x^2 & a^{10} &= x + x^3 \\
ap^3 &= x^3 & a^{11} &= 1 + x^2 + x^3 \\
ap^4 &= 1 + x^3 & a^{12} &= 1 + x \\
ap^5 &= 1 + x + x^3 & a^{13} &= x + x^2 \\
ap^6 &= 1 + x + x^2 + x^3 & a^{14} &= x^2 + x^3 \\
ap^7 &= 1 + x + x^2 & a^{15} &= 1
\end{align*}
\]
Here is FSR that multiplies elements in \( \mathbb{F}_{16} \) by \( \alpha \).

The shift corresponds to the actual multiplication by \( x \), and the potential flip to reduction according to \( x^4 = 1 + x^3 \).

- (Near) Hamiltonian cycles play an important role in GPS.
- Also very important in algebraic coding theory.
- The standard graph theory argument for their construction is inapplicable in resource bounded computation.
- Feedback shift-registers provide an excellent, cheap way to construct long cycles.
- Come in two flavors: Fibonacci and Galois.
- Analysis and design requires finite field theory; in particular primitive elements.