1.6 Feedback Shift Registers

Feedback shift registers (FSR) are a classical and popular method of generating pseudorandom sequences. The method goes back to Golomb in 1955 [2], but is often named after Tausworthe who picked up the idea in a 1965 paper. FSRs are especially convenient for hardware implementation.

An FSR of length l is specified by a Boolean function $f\colon \mathbb{F}_2^l \longrightarrow \mathbb{F}_2$, the "feedback function". Figure 1.7 shows the mode of operation—representing f by a Boolean circuit yields an explicit construction plan. The output consists of the rightmost bit u_0, all the other bits are shifted to the right by one position, and the leftmost cell is filled by the bit $u_l = f(u_{l-1}, \ldots, u_0)$. Thus the recursive formula

$$u_n = f(u_{n-1}, \ldots, u_{n-l}) \quad \text{for } n \geq l \tag{3}$$

represents the complete sequence.

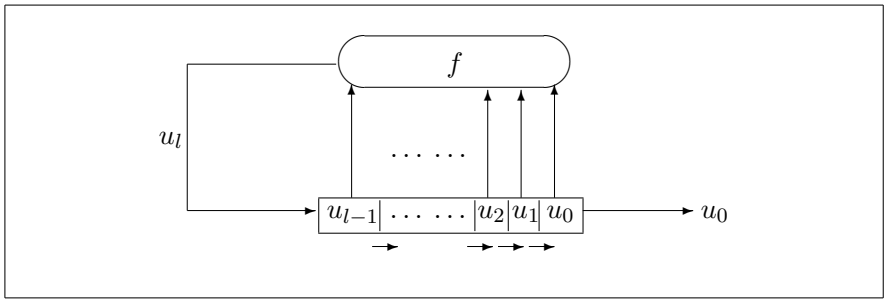

Figure 1.7: A feedback shift register (FSR)

The bits $u_0, \ldots, u_{l-1}$ form the start value. The "key expansion" transforms the short sequence $u = (u_0, \ldots, u_{l-1})$ (the effective key) of length l into a key stream $u_0, u_1, \ldots$ of arbitrary length. In a cryptographic context the bits $u_0, u_1, \ldots$ form the key stream. In other contexts it might be unnecessary to conceal the output bits, but even then hiding the initial state might make sense, starting the output sequence at u_l. Additionally in a cryptographic context treating the internal parameters, that is the feedback function f or some of its coefficients, as components of the key makes sense. Then the effective key length is larger than l.

In this respect the realization in hardware differs from a software implementation: Hardware allows using an adjustable feedback function only by complex additional circuits. Thus in the hardware case we usually assume an unchangeable feedback function, and (at least in the long run) we cannot prevent the attacker from figuring it out. In contrast a software implementation allows a comfortable change of the feedback function at any time such that it may serve as part of the key.

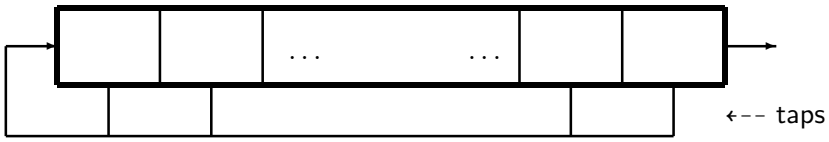

Figure 1.8: Simple graphical representation of an LFSR

For a Sage example we use the procedure `fsr` from Appendix C.1 and the construction of a Boolean function from Appendix E.3 of Part II. (Attach the modules `Bitblock.sage`, `boolF.sage`, `FSR.sage`.)

Sage Example 1.3 Generating a bitsequence by a nonlinear FSR

```
sage: f2 = BoolF([1,1,1,0,1,1,0,0,0,1,0,0,0,1,1,0])
sage: start = [0,1,1,1]
sage: seq = fsr(f2,start,20); seq
[1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0]
```

The simplest and best understood instances of FSRs are the linear feedback shift registers (LFSR). Their feedback functions f are linear. From Part II we know that a linear function

$$f\colon \mathbb{F}_2^l \longrightarrow \mathbb{F}_2$$

is simply a partial sum from an l-bit block:

$$f(u_{n-1}, \ldots, u_{n-l}) = \sum_{j=1}^{l} a_j u_{n-j}, \tag{4}$$

where the coefficients a_j are 0 or 1. If I is the subset of indices j with $a_j = 1$, then the iteration (3) takes the form

$$u_n = \sum_{j \in I} u_{n-j}. \tag{5}$$

A simple graphical representation of an LFSR is shown in Figure 1.8. Here the subset I defines the contacts ("taps") that feed the respective cell contents into the feedback sum.

In SageMath we implement a special class `LFSR`, see Appendix C.1 whose use is demonstrated in the code sample 1.4.

For a good choice of the parameters, see Section 1.9, the sequence has a period of about 2^l, the number of possible different states of the register, and statistical tests are hardly able to distinguish it from a uniformly distributed true random sequence, see Section 1.10. It is remarkable that such a simple

Sage Example 1.4 Generating a bitsequence by a linear FSR

```
sage: coeff = [0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,1]
sage: reg = LFSR(coeff)
sage: start = [0,1,1,0,1,0,1,1,0,0,0,1,0,0,1,1]
sage: reg.setState(start)
sage: bitlist = reg.nextBits(20); bitlist
[1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1]
```

approach generates pseudorandom sequences of fairly high quality! Of course the initial state $u = (0, \ldots, 0)$ is inappropriate. For an initial state $\neq 0$ the maximum possible period is $2^l - 1$, see Section 1.9.

For using an LFSR for bitstream encryption the secret inner parameters—the coefficients $a_1, \ldots, a_l$—as well as the initial state $u_0, \ldots, u_{l-1}$ together constitute the key. In contrast the length l of the register is assumed as known to the attacker. Beware of Section 2.3!