3.3 The Berlekamp-Massey Algorithm

The proof of Proposition 10 is constructive: It contains an algorithm that successively builds a linear generator. For the step from length $n$ to length $n+1$ three cases (1, 2a, 2b) are possible:

**Case 1** $d_n = 0$, hence the generator with feedback polynomial $\varphi$ next outputs $u_n$: Then $\varphi$ and $l$ remain unchanged, and so remain $\psi, t, r, d_r$.

**Case 2** $d_n \neq 0$, hence the generator with feedback polynomial $\varphi$ doesn’t output $u_n$ as next element: Then we form a new feedback polynomial $\eta$ whose corresponding generator outputs $(u_0, \ldots, u_n)$. We distinguish between:

a) $l > \frac{n}{2}$: Then $\lambda_{n+1} = \lambda_n$. We replace $\varphi$ by $\eta$ and leave $l, \psi, t, r, d_r$ unchanged.

b) $l \leq \frac{n}{2}$: Then $\lambda_{n+1} = n + 1 - \lambda_n$. We replace $\varphi$ by $\eta$, $l$ by $n + 1 - l$, $\psi$ by $\varphi$, $t$ by $l$, $r$ by $n$, $d_r$ by $d_n$.

So a semi-formal description of the Berlekamp-Massey algorithm (or BM algorithm) is:

**Input:** A sequence $u = (u_0, \ldots, u_{N-1}) \in \mathbb{K}^N$.

**Output:** The linear complexity $\lambda_N(u)$, the feedback polynomial $\varphi$ of a linear generator of length $\lambda_N(u)$ that produces $u$.

**Auxiliary variables:** $n =$ current index, initialized by $n := 0$,

$l =$ current linear complexity, initialized by $l := 0$,

$\varphi =$ current feedback polynomial $= 1 - a_1 T - \cdots - a_l T^l$, initialized by $\varphi := 1$,

invariant condition: $u_i = a_1 u_{i-1} + \cdots + a_l u_{i-l}$ for $l \leq i < n$,

$d =$ current discrepancy $= u_n - a_1 u_{n-1} - \cdots - a_l u_{n-l}$,

$r =$ previous index, initialized by $r := -1$,

$t =$ previous linear complexity,

$\psi =$ previous feedback polynomial $= 1 - b_1 T - \cdots - b_t T^t$, initialized by $\psi := 1$,

invariant condition: $u_i = b_1 u_{i-1} + \cdots + b_t u_{i-t}$ for $t \leq i < r$,

$d' =$ previous discrepancy $= u_r - b_1 u_{r-1} - \cdots - b_t u_{r-t}$, initialized by $d' := 1$,

$\eta =$ new feedback polynomial,

$m =$ new linear complexity.
Iteration steps: For $n = 0, \ldots, N - 1$:

$$d := u_n - a_1 u_{n-1} - \cdots - a_l u_{n-l}$$

If $d \neq 0$

$$\eta := \varphi - \frac{d}{\varphi} \cdot T^{n-r} \cdot \psi$$

If $l \leq \frac{n}{2}$ [linear complexity increases]

$$m := n + 1 - l$$

$$t := l$$

$$l := m$$

$$\psi := \varphi$$

$$r := n$$

$$d' := d$$

$$\varphi := \eta$$

Output: $\lambda_N(u) := l$ and $\varphi$

Of course we may output also the complete sequence $(\lambda_n)$.

As an example we apply the algorithm to the sequence 001101110. The steps where $d \neq 0$, $l \leq \frac{n}{2}$, are tagged by "[!]".

<table>
<thead>
<tr>
<th>Preconditions of the step</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n = 0$ $u_0 = 0$ $l = 0$ $\varphi = 1$</td>
<td>$d := u_0 = 0$</td>
</tr>
<tr>
<td>$r = -1$ $d' = 1$ $t = \psi = 1$</td>
<td></td>
</tr>
<tr>
<td>$n = 1$ $u_1 = 0$ $l = 0$ $\varphi = 1$</td>
<td>$d := u_1 = 0$</td>
</tr>
<tr>
<td>$r = -1$ $d' = 1$ $t = \psi = 1$</td>
<td></td>
</tr>
<tr>
<td>$n = 2$ $u_2 = 1$ $l = 0$ $\varphi = 1$</td>
<td>$d := u_2 = 1$ ![</td>
</tr>
<tr>
<td>$r = -1$ $d' = 1$ $t = \psi = 1$</td>
<td>$\eta := 1 - T^3$</td>
</tr>
<tr>
<td>$m := 3$</td>
<td></td>
</tr>
<tr>
<td>$n = 3$ $u_3 = 1$ $l = 3$ $\varphi = 1 - T^3$</td>
<td>$d := u_3 - u_0 = 1$</td>
</tr>
<tr>
<td>$r = 2$ $d' = 1$ $t = 0$ $\psi = 1$</td>
<td>$\eta := 1 - T - T^3$</td>
</tr>
<tr>
<td>$n = 4$ $u_4 = 0$ $l = 3$ $\varphi = 1 - T - T^3$</td>
<td>$d := u_4 - u_3 - u_1 = -1$</td>
</tr>
<tr>
<td>$r = 2$ $d' = 1$ $t = 0$ $\psi = 1$</td>
<td>$\eta := 1 - T + T^2 - T^3$</td>
</tr>
<tr>
<td>$n = 5$ $u_5 = 1$ $l = 3$ $\varphi = 1 - T + T^2 - T^3$</td>
<td>$d := u_5 - u_4 + u_3 - u_2 = 0$</td>
</tr>
<tr>
<td>$r = 2$ $d' = 1$ $t = 0$ $\psi = 1$</td>
<td>$\eta := 1 - T + T^2 - 2T^3$</td>
</tr>
</tbody>
</table>

From now on the results differ depending on the characteristic of the base field $K$. First assume char $K \neq 2$. Then the procedure continues as follows:
A Sage program for the char 2 case is in Sage Example 3.1. It uses the
preconditions of the step

| n = 6 | u_6 = 1 | l = 3 | \varphi = 1 - T + T^2 - 2T^3 | r = 2 | d' = 1 | t = 0 | \psi = 1 | d := u_6 - u_5 + u_4 - 2u_3 = -2 \[1\] |
| n = 7 | u_7 = 1 | l = 4 | \varphi = 1 - T + T^2 - 2T^3 + 2T^4 | r = 6 | d' = -2 | t = 3 | \psi = 1 - T + T^2 - 2T^3 | d := u_7 - u_6 + u_5 - 2u_4 + 2u_3 = 3 |
| n = 8 | u_8 = 0 | l = 4 | \varphi = 1 + \frac{1}{2} T - \frac{1}{2} T^2 - \frac{1}{2} T^3 - T^4 | r = 6 | d' = -2 | t = 3 | \psi = 1 - T + T^2 - 2T^3 | d := u_8 + \frac{1}{2} u_7 - \frac{1}{2} u_6 - \frac{1}{2} u_5 - u_4 = -2 \[1\] |

The resulting sequence of linear complexities is

\[\lambda_0 = 0, \lambda_1 = 0, \lambda_2 = 0, \lambda_3 = 3, \lambda_4 = 3, \lambda_5 = 3, \lambda_6 = 3, \lambda_7 = 4, \lambda_8 = 4, \lambda_9 = 5,\]

and the generating formula is

\[u_i = \frac{1}{2} u_{i-1} + \frac{3}{4} u_{i-2} + \frac{1}{4} u_{i-3} + \frac{5}{4} u_{i-4} - \frac{1}{2} u_{i-5} \quad \text{for } i = 5, \ldots, 8.\]

For char \(K = 2\) the last three iteration steps look differently:

| n = 6 | u_6 = 1 | l = 3 | \varphi = 1 - T - T^2 | r = 2 | d' = 1 | t = 0 | \psi = 1 | d := u_6 - u_5 - u_4 = 0 |
| n = 7 | u_7 = 1 | l = 3 | \varphi = 1 - T - T^2 | r = 2 | d' = 1 | t = 0 | \psi = 1 | d := u_7 - u_6 - u_5 = 1 \[1\] |
| n = 8 | u_8 = 0 | l = 5 | \varphi = 1 - T - T^2 - T^5 | r = 7 | d' = 1 | t = 3 | \psi = 1 - T - T^2 | d := u_8 - u_7 - u_6 - u_5 = 1 |

In this case the sequence of linear complexities is

\[\lambda_0 = 0, \lambda_1 = 0, \lambda_2 = 0, \lambda_3 = 3, \lambda_4 = 3, \lambda_5 = 3, \lambda_6 = 3, \lambda_7 = 3, \lambda_8 = 5, \lambda_9 = 5,\]

and the generating formula is

\[u_i = u_{i-3} + u_{i-5} \quad \text{for } i = 5, \ldots, 8.\]

A Sage program for the char 2 case is in Sage Example 3.1. It uses the
function _bmAlg_ from Appendix B.2.

Figure 3.2 shows the growth of the linear complexities.
Sage Example 3.1 Applying the BM-algorithm

```
sage: u = [0,0,1,1,0,1,1,1,0]
sage: res = bmAlg(u)
sage: res
[[0, 0, 0, 3, 3, 3, 3, 3, 5, 5], T^5 + T^3 + 1]
```

Figure 3.2: The sequence of linear complexities. The red line is for char \( K \neq 2 \).

The cost of the BM algorithm is \( O(N^2 \log N) \).

The sequence \( (\lambda_n)_{n \in \mathbb{N}} \) or (for finite output sequences) \( (\lambda_n)_{0 \leq n \leq N} \) is called the **linearity profile** of the sequence \( u \).

Here is the linearity profile of the first 128 bits of the sequence that we generated by an LFSR in Section 1.10:

\[
(0, 1, 1, 2, 2, 3, 3, 4, 4, 4, 4, 4, 7, 7, 7, 7, 7, 7, 8, 8, 9, 9, 10, 10, 11, 11, 12,
12, 13, 13, 13, 16, 16, 16, 16, \ldots),
\]

its graphic representation is in Figure 3.3.

In Section 1.11 we’ll generate a “perfect” pseudorandom sequence. The linearity profile of its first 128 bits is:

\[
(0, 1, 1, 1, 1, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 8, 8, 8, 8, 8, 8, 8, 12, 12, 12, 12,
12, 12, 12, 12, 12, 12, 17, 17, 17, 17, 17, 17, 18, 18, 18, 18, 18, 20, 20, 20, 20, 21, 21,
22, 22, 22, 24, 24, 24, 24, 24, 24, 24, 28, 28, 28, 28, 28, 28, 28, 29, 29, 30, 30, 31,\]

...
In the second example we see a somewhat irregular oscillation around the diagonal, as should be expected for a “good” random sequence. The first example also shows a similar behaviour, but only until the linear complexity of the sequence is reached.