6.3 Decoding BCH and RS Codes: The General Outline

There are many algorithms which have been developed for decoding BCH or RS codes. In this chapter we introduce a general approach. In chapter 7 we present other approaches which follow a different outline.

The algebraic decoding BCH or RS codes has the following general steps:

1. Computation of the *syndrome*.
2. Determination of an *error locator polynomial*, whose roots provide an indication of *where* the errors are. There are several different ways of finding the locator polynomial. These methods include Peterson's algorithm for BCH codes, the Berlekamp-Massey algorithm for BCH codes; the Peterson-Gorenstein-Zierler algorithm for RS codes, the Berlekamp-Massey algorithm for RS codes, and the Euclidean algorithm. In addition, there are techniques based upon Galois-field Fourier transforms.
3. Finding the roots of the error locator polynomial. This is usually done using the *Chien search*, which is an exhaustive search over all the elements in the field.
4. For RS codes or nonbinary BCH codes, the error *values* must also be determined. This is typically accomplished using *Forney's algorithm*.

Throughout this chapter (unless otherwise noted) we assume narrow-sense BCH or RS codes, that is, $b = 1$.

6.3.1 Computation of the Syndrome

Since

$$g(\alpha) = g(\alpha^2) = \cdots = g(\alpha^{2t}) = 0$$

it follows that a codeword $\mathbf{c} = (c_0, \ldots, c_{n-1})$ with polynomial $c(x) = c_0 + \cdots + c_{n-1}x^{n-1}$ has

$$c(\alpha) = \cdots = c(\alpha^{2t}) = 0.$$

For a received polynomial $r(x) = c(x) + e(x)$ we have

$$S_j = r(\alpha^j) = e(\alpha^j) = \sum_{k=0}^{n-1} e_k \alpha^{jk}, \qquad j = 1, 2, \ldots, 2t.$$

The values $S_1, S_2, \ldots, S_{2t}$ are called the syndromes of the received data.

Suppose that $\mathbf{r}$ has ν errors in it which are at locations $i_1, i_2, \ldots, i_\nu$, with corresponding error values in these locations $e_{i_j} \neq 0$. Then

$$S_j = \sum_{l=1}^{\nu} e_{i_l} (\alpha^j)^{i_l} = \sum_{l=1}^{\nu} e_{i_l} (\alpha^{i_l})^j.$$

Let

$$X_l = \alpha^{i_l}.$$

Then we can write

$$S_j = \sum_{l=1}^{\nu} e_{i_l} X_l^j \qquad j = 1, 2, \ldots, 2t. \tag{6.3}$$

For *binary codes* we have $e_{i_l} = 1$ (if there is a non-zero error, it must be to 1). For the moment we restrict our attention to binary (BCH) codes. Then we have

$$S_j = \sum_{l=1}^{\nu} X_l^j. \tag{6.4}$$

If we know X_l, then we know the location of the error. For example, suppose we know that $X_1 = \alpha^4$. This means, by the definition of X_l that $i_1 = 4$; that is, the error is in the received digit r_4. We thus call the X_l the **error locators**.

The next stage in the decoding problem is to determine the error locators X_l given the syndromes S_j.

6.3.2 The Error Locator Polynomial

From (6.4) we obtain the following equations:

$$\begin{aligned} S_1 &= X_1 + X_2 + \cdots + X_\nu \\ S_2 &= X_1^2 + X_2^2 + \cdots + X_\nu^2 \\ &\vdots \\ S_{2t} &= X_1^{2t} + X_2^{2t} + \cdots + X_\nu^{2t}. \end{aligned} \tag{6.5}$$

The equations are said to be *power-sum symmetric functions*. This gives us $2t$ equations in the ν unknown error locators. In principle this set of nonlinear equations could be solved by an exhaustive search, but this would be computationally unattractive.

Rather than attempting to solve these nonlinear equations directly, a new polynomial is introduced, the *error locator polynomial*, which casts the problem in a different, and more tractable, setting. The error locator polynomial is defined as

$$\Lambda(x) = \prod_{l=1}^{\nu}(1 - X_l x) = \Lambda_\nu x^\nu + \Lambda_{\nu-1}x^{\nu-1} + \cdots + \Lambda_1 x + \Lambda_0, \tag{6.6}$$

where $\Lambda_0 = 1$. By this definition, if $x = X_l^{-1}$ then $\Lambda(x) = 0$; that is, the roots of the error locator polynomial are at the *reciprocals* (in the field arithmetic) of the error locators.

Example 6.11 Suppose in $GF(16)$ we find that $x = \alpha^4$ is a root of an error locator polynomial $\Lambda(x)$. Then the error locator is $(\alpha^4)^{-1} = \alpha^{11}$, indicating that there is an error in r_{11}. □

6.3.3 Chien Search

Assume for the moment that we actually have the error locator polynomial. (Finding the error locator polynomial is discussed below.) The next step is to find the roots of the error locator polynomial. The field of interest is $GF(q^m)$. Being a finite field, we can examine every element of the field to determine if it is a root. There exist other ways of factoring polynomials over finite fields (see, e.g., [25, 360]), but for the fields usually used for error correction codes and the number of roots involved, the Chien search may be the most efficient.

Suppose, for example, that $\nu = 3$ and the error locator polynomial is

$$\Lambda(x) = \Lambda_0 + \Lambda_1 x + \Lambda_2 x^2 + \Lambda_3 x^3 = 1 + \Lambda_1 x + \Lambda_2 x^2 + \Lambda_3 x^3.$$

We evaluate $\Lambda(x)$ at each nonzero element in the field in succession: $x = 1$, $x = \alpha$, $x = \alpha^2, \ldots, x = \alpha^{q^m-2}$. This gives us the following:

$$\begin{aligned}\Lambda(1) &= 1 + \Lambda_1(1) + \Lambda_2(1)^2 + \Lambda_3(1)^3\\ \Lambda(\alpha) &= 1 + \Lambda_1(\alpha) + \Lambda_2(\alpha)^2 + \Lambda_3(\alpha)^3\\ \Lambda(\alpha^2) &= 1 + \Lambda_1(\alpha^2) + \Lambda_2(\alpha^2)^2 + \Lambda_3(\alpha^2)^3\\ &\vdots\\ \Lambda(\alpha^{q^m-2}) &= 1 + \Lambda_1(\alpha^{q^m-2}) + \Lambda_2(\alpha^{q^m-2})^2 + \Lambda_3(\alpha^{q^m-2})^3.\end{aligned}$$

The computations in this sequence can be efficiently embodied in the hardware depicted in Figure 6.1. A set of ν registers are loaded initially with the coefficients of the error locator polynomial, $\Lambda_1, \Lambda_2, \ldots, \Lambda_\nu$. The initial output is the sum

$$A = \sum_{j=1}^{\nu} \Lambda_j = \Lambda(x) - 1|_{x=1}\,.$$

If $A = 1$ then an error has been located (since then $\Lambda(x) = 0$). At the next stage, each register is multiplied by α^j, $j = 1, 2, \ldots, \nu$, so the register contents are $\Lambda_1\alpha, \Lambda_2\alpha^2, \ldots, \Lambda_\nu\alpha^\nu$. The output is the sum

$$A = \sum_{j=1}^{\nu} \Lambda_j\alpha^j = \Lambda(x) - 1|_{x=\alpha}\,.$$

The registers are multiplied again by successive powers of α, resulting in evaluation at α^2. This procedure continues until $\Lambda(x)$ has been evaluated at all nonzero elements of the field.

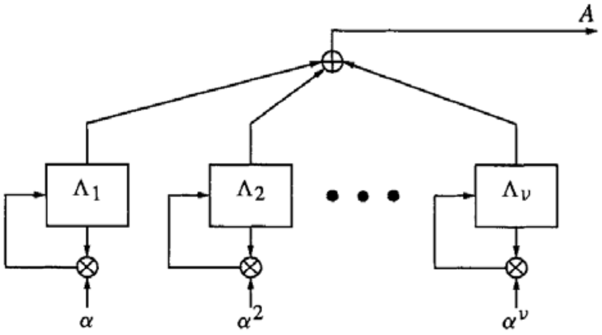


Figure 6.1: Chien search algorithm.

If the roots are distinct and all lie in the appropriate field, then we use these to determine the error locations. If they are not distinct or lie in the wrong field, then the received word is not within distance t of any codeword. (This condition can be observed if the error locator polynomial of degree ν does not have ν roots in the field that the operations take in; the remaining roots are either repeated or exist in an extension of this field.) The corresponding error pattern is said to be an *uncorrectable error pattern.* An uncorrectable error pattern results in a **decoder failure**.

6.4 Finding the Error Locator Polynomial

Let us return to the question of finding the error locator polynomial using the syndromes. Let us examine the structure of the error locator polynomial by expanding (6.6) for the case $\nu = 3$:

$$\begin{aligned}\Lambda(x) &= 1 - x(X_1 + X_2 + X_3) + x^2(X_1X_2 + X_1X_3 + X_2X_3) - x^3X_1X_2X_3 \\ &= \Lambda_0 + x\Lambda_1 + x^2\Lambda_2 + x^3\Lambda_3\end{aligned}$$

so that

$$\Lambda_0 = 1 \qquad \Lambda_1 = -(X_1 + X_2 + X_3) \qquad \Lambda_2 = X_1X_2 + X_1X_3 + X_2X_3$$

$$\Lambda_3 = -X_1X_2X_3.$$

In general, for an error locator polynomial of degree ν we find that

$$\begin{aligned}\Lambda_0 &= 1 \\ -\Lambda_1 &= \sum_{i=1}^{\nu} X_i = X_1 + X_2 + \cdots + X_\nu \\ \Lambda_2 &= \sum_{i<j} X_iX_j = X_1X_2 + X_1X_3 + \cdots + X_1X_\nu + \cdots + X_{\nu-1}X_\nu \\ -\Lambda_3 &= \sum_{i<j<k} X_iX_jX_k = X_1X_2X_3 + X_1X_2X_4 + \cdots + X_{\nu_2}X_{\nu-1}X_\nu \\ &\vdots \\ (-1)^\nu\Lambda_\nu &= X_1X_2\cdots X_\nu.\end{aligned} \tag{6.7}$$

That is, the coefficient of the error locator polynomial Λ_i is the sum of the product of all combinations of the error locators taken i at a time. Equations of the form (6.7) are referred to as the *elementary symmetric functions* of the error locators (so called because if the error locators $\{X_i\}$ are permuted, the same values are computed).

The power-sum symmetric functions of (6.5) provide a nonlinear relationship between the syndromes and the error locators. The elementary symmetric functions provide a nonlinear relationship between the coefficients of the error locator polynomial and the error locators. The key observation is that there is a *linear* relationship between the syndromes and the coefficients of the error locator polynomial. This relationship is described by the *Newton identities*, which apply over any field.

Theorem 6.11 *The syndromes (6.5) and the coefficients of the error locator polynomial are related by*

$$\begin{aligned}S_k + \Lambda_1S_{k-1} + \cdots + \Lambda_{k-1}S_1 + k\Lambda_k &= 0 \qquad 1 \le k \le \nu \\ S_k + \Lambda_1S_{k-1} + \cdots + \Lambda_{\nu-1}S_{k-\nu+1} + \Lambda_\nu S_{k-\nu} &= 0 \qquad k > \nu.\end{aligned} \tag{6.8}$$

That is,

$$\begin{aligned}
k=1:&\quad S_1+\Lambda_1=0\\
k=2:&\quad S_2+\Lambda_1 S_1+2\Lambda_2=0\\
&\quad\vdots\\
k=\nu:&\quad S_\nu+\Lambda_1 S_{\nu-1}+\Lambda_2 S_{\nu-2}+\cdots+\Lambda_{\nu-1}S_1+\nu\Lambda_\nu=0\\
k=\nu+1:&\quad S_{\nu+1}+\Lambda_1 S_\nu+\Lambda_2 S_{\nu-1}+\cdots+\Lambda_\nu S_1=0\\
k=\nu+2:&\quad S_{\nu+2}+\Lambda_1 S_{\nu+1}+\Lambda_2 S_\nu+\cdots+\Lambda_\nu S_2=0\\
&\quad\vdots\\
k=2t:&\quad S_{2t}+\Lambda_1 S_{2t-1}+\Lambda_2 S_{2t-2}+\cdots+\Lambda_\nu S_{2t-\nu}=0.
\end{aligned}\tag{6.9}$$

For k > ν, there is a linear feedback shift register relationship between the syndromes and the coefficients of the error locator polynomial,

$$S_j=-\sum_{i=1}^{\nu}\Lambda_i S_{j-i}.\tag{6.10}$$

The theorem is proved in Appendix 6.A.

Equation (6.10) can be expressed in a matrix form

$$\begin{bmatrix} S_1 & S_2 & \cdots & S_\nu\\ S_2 & S_3 & \cdots & S_{\nu+1}\\ S_3 & S_4 & \cdots & S_{\nu+2}\\ \vdots & & & \\ S_\nu & S_{\nu+1} & \cdots & S_{2\nu-1}\end{bmatrix}\begin{bmatrix}\Lambda_\nu\\ \Lambda_{\nu-1}\\ \Lambda_{\nu-2}\\ \vdots\\ \Lambda_1\end{bmatrix}=-\begin{bmatrix}S_{\nu+1}\\ S_{\nu+2}\\ \vdots\\ S_{2\nu}\end{bmatrix}.$$

The $\nu\times\nu$ matrix, which we denote M_ν, is a Toeplitz matrix, constant on the diagonals. The number of errors ν is not known in advance, so it must be determined. The Peterson-Gorenstein-Zierler decoder operates as follows.

1. Set $\nu=t$.
2. Form M_ν and compute the determinant $\det(M_\nu)$ to determine if M_ν is invertible. If it is not invertible, set $\nu\leftarrow\nu-1$ and repeat this step.
3. If M_ν is invertible, solve for the coefficients $\Lambda_1,\Lambda_2,\ldots,\Lambda_\nu$.

6.4.1 Simplifications for Binary Codes and Peterson's Algorithm

For binary codes, Newton's identities are subject to further simplifications. $nS_j=0$ if n is even and $nS_j=S_j$ if n is odd. Furthermore, we have $S_{2j}=S_j^2$, since by (6.4) and Theorem 5.15

$$S_{2j}=\sum_{l=1}^{\nu}X_l^{2j}=\left(\sum_{l=1}^{\nu}X_l^j\right)^2=S_j^2.$$

We can thus write Newton's identities (6.9) as

$$\begin{aligned} S_1 + \Lambda_1 &= 0 \\ S_3 + \Lambda_1 S_2 + \Lambda_2 S_1 + \Lambda_3 &= 0 \\ &\vdots \\ S_{2t-1} + \Lambda_1 S_{2t-2} + \cdots + \Lambda_t S_{t-1} &= 0, \end{aligned}$$

which can be expressed in the matrix equation

$$\begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ S_2 & S_1 & 1 & 0 & \cdots & 0 & 0 \\ S_4 & S_3 & S_2 & S_1 & \cdots & 0 & 0 \\ \vdots \\ S_{2t-4} & S_{2t-5} & S_{2t-6} & S_{2t-7} & \cdots & S_{t-2} & S_{t-3} \\ S_{2t-2} & S_{2t-3} & S_{2t-4} & S_{2t-5} & \cdots & S_t & S_{t-1} \end{bmatrix} \begin{bmatrix} \Lambda_1 \\ \Lambda_2 \\ \vdots \\ \Lambda_t \end{bmatrix} = \begin{bmatrix} -S_1 \\ -S_3 \\ \vdots \\ -S_{2t-1} \end{bmatrix}, \tag{6.11}$$

or $A\mathbf{\Lambda} = -\mathbf{S}$. If there are in fact t errors, the matrix is invertible, as we can determine by computing the determinant of the matrix. If it is not invertible, remove two rows and columns, then try again. Once $\mathbf{\Lambda}$ is found, we find its roots. This matrix-based approach to solving for the error locator polynomial is called *Peterson's algorithm* for decoding binary BCH codes.

For small numbers of errors, we can provide explicit formulas for the coefficients of $\Lambda(x)$, which may be more efficient than the more generalized solutions suggested below [238].

1-error correction $\Lambda_1 = S_1$.

2-error correction $\Lambda_1 = S_1, \quad \Lambda_2 = (S_3 + S_1^3)/(S_1)$.

3-error correction $\Lambda_1 = S_1, \quad \Lambda_2 = (S_1^2 S_3 + S_5)/(S_1^3 + S_3), \quad \Lambda_3 = (S_1^3 + S_3) + S_1 \Lambda_2$.

4 error correction

$$\Lambda_1 = S_1, \quad \Lambda_2 = \frac{S_1(S_7 + S_1^7) + S_3(S_1^5 + S_5)}{S_3(S_1^3 + S_3) + S_1(S_1^5 + S_5)},$$

$$\Lambda_3 = S_1^3 + S_3 + S_1\Lambda_2, \quad \Lambda_4 = \frac{(S_5 + S_1^2 S_3) + (S_1^3 + S_3)\Lambda_2}{S_1}.$$

5-error correction $\Lambda_1 = S_1$,

$$\Lambda_2 = \frac{(S_1^3 + S_3)[(S_1^9 + S_9) + S_1^4(S_5 + S_1^2 S_3) + S_3^2(S_1^3 + S_3)] + (S_1^5 + S_5)(S_7 + S_1^7) + S_1(S_3^2 + S_1 S_5)}{(S_1^3 + S_3)[(S_7 + S_1^7) + S_1 S_3(S_1^3 + S_3)] + (S_5 + S_1^2 S_3)(S_1^5 + S_5)}$$

$$\Lambda_3 = (S_1^3 + S_3) + S_1\Lambda_2$$

$$\Lambda_4 = \frac{(S_1^9 + S_9) + S_3^2(S_1^3 + S_3) + S_1^4(S_5 + S_1^2 S_3) + \Lambda_2[(s_7 + S_1^7) + S_1 S_3(S_1^3 + S_3)]}{S_1^5 + S_5}$$

$$\Lambda_5 = S_5 + S_1^2 S_3 + S_1 S_4 + \Lambda_2(S_1^3 + S_3).$$

For large numbers of errors, Peterson's algorithm is quite complex. Computing the sequence of determinants to find the number of errors is costly. So is solving the system of equations once the number of errors is determined. We therefore look for more efficient techniques.

Example 6.12 Consider the (31,21) 2-error correcting code introduced in Example 6.2, with generator $g(x) = x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1$ having roots at $\alpha, \alpha^2, \alpha^3$ and α^4. Suppose the codeword

$$c(x) = 1 + x^3 + x^4 + x^5 + x^6 + x^8 + x^{10} + x^{14} + x^{16} + x^{17} + x^{18} + x^{20} + x^{21} + x^{23} + x^{24} + x^{25}$$

is transmitted and

$$r(x) = 1 + x^3 + x^5 + x^6 + x^8 + x^{10} + x^{14} + x^{16} + x^{17} + x^{20} + x^{21} + x^{23} + x^{24} + x^{25}$$

is received. The syndromes are

$$S_1 = r(\alpha) = \alpha^{17} \qquad S_2 = r(\alpha^2) = \alpha^3 \qquad S_3 = r(\alpha^3) = 1 \qquad S_4 = r(\alpha^4) = \alpha^6.$$

Using the results above we find

$$\Lambda_1 = S_1 = \alpha^{17} \qquad \Lambda_2 = \frac{S_3 + S_1^3}{S_1} = \alpha^{22},$$

so that $\Lambda(x) = 1 + \alpha^{17}x + \alpha^{22}x^2$. The roots of this polynomial (found, e.g., using the Chien search) are at $x = \alpha^{13}$ and $x = \alpha^{27}$. Specifically, we could write

$$\Lambda(x) = \alpha^{22}(x + \alpha^{13})(x + \alpha^{27}).$$

The *reciprocals* of the roots are at α^{18} and α^4, so that the errors in transmission occurred at locations 4 and 18,

$$e(x) = x^4 + x^{18}.$$

It can be seen that $r(x) + e(x)$ is in fact equal to the transmitted codeword. □

6.4.2 Berlekamp-Massey Algorithm

While Peterson's method involves straightforward linear algebra, it is computationally complex in general. Starting with the matrix A in (6.11), it is examined to see if it is singular. This involves either attempting to solve the equations (e.g., by Gaussian elimination or equivalent), or computing the determinant to see if the solution can be found. If A is singular, then the last two rows and columns are dropped to form a new A matrix. Then the attempted solution must be *recomputed* starting over with the new A matrix.

The Berlekamp-Massey algorithm takes a different approach. Starting with a *small* problem, it works up to increasingly longer problems until it obtains an overall solution. However, at each stage it is able to re-use information it has already learned. Whereas as the computational complexity of the Peterson method is $O(\nu^3)$, the computational complexity of the Berlekamp-Massey algorithm is $O(\nu^2)$.

We have observed from the Newton's identity (6.10) that

$$S_j = -\sum_{i=1}^{\nu} \Lambda_i S_{j-i}, \qquad j = \nu + 1, \nu + 2, \ldots, 2t. \qquad (6.12)$$

This formula describes the output of a linear feedback shift register (LFSR) with coefficients $\Lambda_1, \Lambda_2, \ldots, \Lambda_\nu$. In order for this formula to work, we must find the Λ_j coefficients in such a way that the LFSR generates the known sequence of syndromes $S_1, S_2, \ldots, S_{2t}$. Furthermore, by the maximum likelihood principle, the number of errors ν determined must be the *smallest* that is consistent with the observed syndromes. We therefore want to determine the *shortest* such LFSR.

In the Berlekamp-Massey algorithm, we build the LFSR that produces the entire sequence $\{S_1, S_2, \ldots, S_{2t}\}$ by successively modifying an existing LFSR, if necessary, to produce increasingly longer sequences. We start with an LFSR that could produce S_1. We determine if that LFSR could also produce the sequence $\{S_1, S_2\}$; if it can, then no modifications are necessary. If the sequence cannot be produced using the current LFSR configuration, we determine a new LFSR that can produce the longer sequence. Proceeding inductively in this way, we start from an LFSR capable of producing the sequence $\{S_1, S_2, \ldots, S_{k-1}\}$ and modify it, if necessary, so that it can also produce the sequence $\{S_1, S_2, \ldots, S_k\}$. At each stage, the modifications to the LFSR are accomplished so that the LFSR is the shortest possible. By this means, after completion of the algorithm an LFSR has been found that is able to produce $\{S_1, S_2, \ldots, S_{2t}\}$ and its coefficients correspond to the error locator polynomial $\Lambda(x)$ of *smallest* degree.

Since we build up the LFSR using information from prior computations, we need a notation to represent the $\Lambda(x)$ used at different stages of the algorithm. Let L_k denote the length of the LFSR produced at stage k of the algorithm. Let

$$\Lambda^{[k]}(x) = 1 + \Lambda_1^{[k]}x + \cdots + \Lambda_{L_k}^{[k]}x^{L_k}$$

be the *connection polynomial* at stage k, indicating the connections for the LFSR capable of producing the output sequence $\{S_1, S_2, \ldots, S_k\}$. That is,

$$S_j = -\sum_{i=1}^{L_k} \Lambda_i^{[k]} S_{j-i} \qquad j = L_k + 1, \ldots, k. \tag{6.13}$$

Note: It is important to realize that some of the coefficients in $\Lambda^{[k]}(x)$ may be zero, so that L_k may be different from the degree of $\Lambda^{[k]}(x)$. In realizations which use polynomial arithmetic, it is important to keep in mind what the length is as well as the degree.

At some intermediate step, suppose we have a connection polynomial $\Lambda^{[k-1]}(x)$ of length L_{k-1} that produces $\{S_1, S_2, \ldots, S_{k-1}\}$ for some $k - 1 < 2t$. We check if this connection polynomial also produces S_k by computing the output

$$\hat{S}_k = -\sum_{i=1}^{L_{k-1}} \Lambda_i^{[k-1]} S_{k-i}.$$

If $\hat{S}_k$ is equal to S_k, then there is no need to update the LFSR, so $\Lambda^{[k]}(x) = \Lambda^{[k-1]}(x)$ and $L_k = L_{k-1}$. Otherwise, there is some nonzero *discrepancy* associated with $\Lambda^{[k-1]}(x)$,

$$d_k = S_k - \hat{S}_k = S_k + \sum_{i=1}^{L_{k-1}} \Lambda_i^{[k-1]} S_{k-i} = \sum_{i=0}^{L_{k-1}} \Lambda_i^{[k-1]} S_{k-i}. \tag{6.14}$$

In this case, we update the connection polynomial using the formula

$$\Lambda^{[k]}(x) = \Lambda^{[k-1]}(x) + Ax^l \Lambda^{[m-1]}(x), \tag{6.15}$$

where A is some element in the field, l is an integer, and $\Lambda^{[m-1]}(x)$ is one of the prior connection polynomials produced by our process associated with nonzero discrepancy d_m. (Initialization of this inductive process is discussed in the proof of Theorem 6.13.) Using this new connection polynomial, we compute the new discrepancy, denoted by d'_k, as

$$\begin{aligned} d'_k &= \sum_{i=0}^{L_k} \Lambda_i^{[k]} S_{k-i} \\ &= \sum_{i=0}^{L_{k-1}} \Lambda_i^{[k-1]} S_{k-i} + A \sum_{i=0}^{L_{m-1}} \Lambda_i^{[m-1]} S_{k-i-l}. \end{aligned} \tag{6.16}$$

Now, let $l = k - m$. Then, by comparison with the definition of the discrepancy in (6.14), the second summation gives

$$A \sum_{i=0}^{L_{m-1}} \Lambda_i^{[m-1]} S_{m-i} = A d_m.$$

Thus, if we choose $A = -d_m^{-1} d_k$, then the summation in (6.16) gives

$$d'_k = d_k - d_m^{-1} d_k d_m = 0.$$

So the new connection polynomial produces the sequence $\{S_1, S_2, \ldots, S_k\}$ with no discrepancy.

6.4.3 Characterization of LFSR Length in Massey's Algorithm

The update in (6.15) is, in fact, the heart of Massey's algorithm. If all we need is an algorithm to find a connection polynomial, no further analysis is necessary. However, the problem was to find the *shortest* LFSR producing a given sequence. We have produced a means of finding an LFSR, but have no indication yet that it is the shortest. Establishing this requires some additional effort in the form of two theorems.

Theorem 6.12 *Suppose that an LFSR with connection polynomial $\Lambda^{[k-1]}(x)$ of length L_{k-1} produces the sequence $\{S_1, S_2, \ldots, S_{k-1}\}$, but not the sequence $\{S_1, S_2, \ldots, S_k\}$. Then any connection polynomial that produces the latter sequence must have a length L_k satisfying*

$$L_k \geq k - L_{k-1}.$$

Proof The theorem is only of practical interest if $L_{k-1} < k - 1$; otherwise it is trivial to produce the sequence. Let us take, then, $L_{k-1} < k - 1$. Let

$$\Lambda^{[k-1]}(x) = 1 + \Lambda_1^{[k-1]} x + \cdots + \Lambda_{L_{k-1}}^{[k-1]} x^{L_{k-1}}$$

represent the connection polynomial which produces $\{S_1, \ldots, S_{k-1}\}$ and let

$$\Lambda^{[k]}(x) = 1 + \Lambda_1^{[k]} x + \cdots + \Lambda_{L_k}^{[k]} x^{L_k}$$

denote the connection polynomial which produces $\{S_1, S_2, \ldots, S_k\}$. Now we do a proof by contradiction.

Assume (contrary to the theorem) that

$$L_k \leq k - 1 - L_{k-1}. \tag{6.17}$$

From the definitions of the connection polynomials, we observe that

$$-\sum_{i=1}^{L_{k-1}} \Lambda_i^{[k-1]} S_{j-i} \quad \begin{cases} = S_j & j = L_{k-1}+1, L_{k-1}+2, \ldots, k-1 \\ \neq S_k & j = k \end{cases} \tag{6.18}$$

and

$$-\sum_{i=1}^{L_k} \Lambda_i^{[k]} S_{j-i} = S_j \qquad j = L_k+1, L_k+2, \ldots, k. \tag{6.19}$$

In particular, from (6.19), we have

$$S_k = -\sum_{i=1}^{L_k} \Lambda_i^{[k]} S_{k-i}. \tag{6.20}$$

The values of S_i involved in this summation range from S_{k-1} to S_{k-L_k}. The indices of these values form a set $\{k-L_k, k-L_k+1, \ldots, k-1\}$. By the (contrary) assumption made in (6.17), we have $k - L_k \geq L_{k-1}+1$, so that the set of indices $\{k-L_k, k-L_k+1, \ldots, k-1\}$ are a subset of the set of indices $\{L_{k-1}+1, L_{k-1}+2, \ldots, k-1\}$ appearing in (6.18). Thus each S_{k-i} appearing on the right-hand side of (6.20) can be replaced by the summation expression from (6.18) and we can write

$$S_k = -\sum_{i=1}^{L_k} \Lambda_i^{[k]} S_{k-i} = \sum_{i=1}^{L_k} \Lambda_i^{[k]} \sum_{j=1}^{L_{k-1}} \Lambda_j^{[k-1]} S_{k-i-j}.$$

Interchanging the order of summation we have

$$S_k = \sum_{j=1}^{L_{k-1}} \Lambda_j^{[k-1]} \sum_{i=1}^{L_k} \Lambda_i^{[k]} S_{k-i-j}. \tag{6.21}$$

Now setting $j = k$ in (6.18), we obtain

$$S_k \neq -\sum_{i=1}^{L_{k-1}} \Lambda_i^{[k-1]} S_{k-i}. \tag{6.22}$$

In this summation the indices of S form the set $\{k - L_{k-1}, \ldots, k-1\}$. By the (contrary) assumption (6.17), $L_k + 1 \leq k - L_{k-1}$, so the sequence of indices $\{k - L_{k-1}, \ldots, k-1\}$ is a subset of the range $L_k+1, \ldots, k$ of (6.19). Thus we can replace each S_{k-i} in the summation of (6.22) with the expression from (6.19) to obtain

$$S_k \neq \sum_{i=1}^{L_{k-1}} \Lambda_i^{[k-1]} \sum_{j=1}^{L_k} \Lambda_j^{[k]} S_{k-i-j}. \tag{6.23}$$

Comparing (6.21) with (6.23), the double summations are the same, but the equality in the first case and the inequality in the second case indicate a contradiction. Hence, the assumption on the length of the LFSRs must have been incorrect. By this contradiction, we must have

$$L_k \geq k - L_{k-1}.$$

If we take this to be the case, the index ranges which gave rise to the substitutions leading to the contradiction do not occur. □

Since the shortest LFSR that produces the sequence $\{S_1, S_2, \ldots, S_k\}$ must also produce the first part of that sequence, we must have $L_k \geq L_{k-1}$. Combining this with the result of the theorem, we obtain

$$L_k \geq \max(L_{k-1}, k - L_{k-1}). \tag{6.24}$$

We observe that the shift register cannot become shorter as more outputs are produced.

We have seen how to update the LFSR to produce a longer sequence using (6.15) and have also seen that there is a lower bound on the length of the LFSR. We now show that this lower bound can be achieved with equality, thus providing the *shortest* LFSR which produces the desired sequence.

Theorem 6.13 *In the update procedure, if $\Lambda^{[k]}(x) \neq \Lambda^{[k-1]}(x)$, then a new LFSR can be found whose length satisfies*

$$L_k = \max(L_{k-1}, k - L_{k-1}). \tag{6.25}$$

Proof We do a proof by induction. To check when $k = 1$ (which also indicates how to get the algorithm started), take $L_0 = 0$ and $\Lambda^{[0]}(x) = 1$. We find that

$$d_1 = S_1.$$

If $S_1 = 0$, then no update is necessary. If $S_1 \neq 0$, then we take $\Lambda^{[m]}(x) = \Lambda^{[0]}(x) = 1$, so that $l = 1 - 0 = 1$. Also, take $d_m = 1$. The updated polynomial is

$$\Lambda^{[1]}(x) = 1 + S_1 x,$$

which has degree L_1 satisfying

$$L_1 = \max(L_0, 1 - L_0) = 1.$$

In this case, (6.13) is vacuously true for the sequence consisting of the single point $\{S_1\}$.

Now let $\Lambda^{[m-1]}(x)$, $m < k-1$, denote the *last* connection polynomial before $\Lambda^{[k-1]}(x)$ with $L_{m-1} < L_{k-1}$ that can produce the sequence $\{S_1, S_2, \ldots, S_{m-1}\}$ but not the sequence $\{S_1, S_2, \ldots, S_m\}$. Then

$$L_m = L_{k-1};$$

hence, in light of the inductive hypothesis (6.25),

$$L_m = m - L_{m-1} = L_{k-1}, \qquad \text{or} \qquad L_{m-1} - m = -L_{k-1}. \tag{6.26}$$

By the update formula (6.15) with $l = k - m$, we note that

$$L_k = \max(L_{k-1}, k - m + L_{m-1}).$$

Using $L_{m-1} - m$ from (6.26) we find that

$$L_k = \max(L_{k-1}, k - L_{k-1}).$$

□

In the update step, we observe that the new length is the same as the old length if $L_{k-1} \geq k - L_{k-1}$, that is, if

$$2L_{k-1} \geq k.$$

In this case, the connection polynomial is updated, but there is no change in length.

The shift-register synthesis algorithm, known as Massey's algorithm, is presented first in pseudocode as Algorithm 6.1, where we use the notations

$$c(x) = \Lambda^{[k]}(x)$$

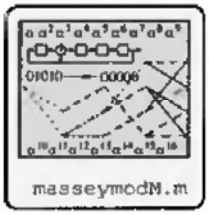
to indicate the "current" connection polynomial and

$$p(x) = \Lambda^{[m-1]}(x)$$

to indicate a "previous" connection polynomial. Also, N is the number of input symbols ($N = 2t$ for many decoding problems).

Algorithm 6.1 Massey's Algorithm

```
Input: $S_1, S_2, \ldots, S_N$
Initialize:
$L = 0$ (the current length of the LFSR)
$c(x) = 1$ (the current connection polynomial)
$p(x) = 1$ (the connection polynomial before last length change)
$l = 1$ ($l$ is $k - m$, the amount of shift in update)
$d_m = 1$ (previous discrepancy)
for $k = 1$ to $N$
  $d = S_k + \sum_{i=1}^{L} c_i S_{k-i}$ (compute discrepancy)
  if ($d = 0$) (no change in polynomial)
    $l = l + 1$
  else
    if ($2L \geq k$) then (no-length change in update)
      $c(x) = c(x) - d d_m^{-1} x^l p(x)$
      $l = l + 1$
    else (update $c$ with length change)
      $t(x) = c(x)$ (temporary storage)
      $c(x) = c(x) - d d_m^{-1} x^l p(x)$
      $L = k - L$
      $p(x) = t(x)$
      $d_m = d$
      $l = 1$
    end
  end
end
```



masseymodM.m

Example 6.13 For the sequence $S = \{1, 1, 1, 0, 1, 0, 0\}$ the feedback connection polynomial obtained by a call to `massey` is $\{1, 1, 0, 1\}$, which corresponds to the polynomial

$$C(x) = 1 + x + x^3.$$

Thus the elements of S are related by

$$S_j = S_{j-1} + S_{j-3},$$

for $j \geq 3$. Details of the operation of the algorithm are presented in Table 6.5. □

Table 6.5: Evolution of the Berlekamp-Massey Algorithm for the Input Sequence $\{1, 1, 1, 0, 1, 0, 0\}$.

k	S_k	d_k	$c(x)$	L	$p(x)$	l	d_m
1	1	1	$1+x$	1	1	1	1
2	1	0	$1+x$	1	1	2	1
3	1	0	$1+x$	1	1	3	1
4	0	1	$1+x+x^3$	3	$1+x$	1	1
5	1	0	$1+x+x^3$	3	$1+x$	2	1
6	0	0	$1+x+x^3$	3	$1+x$	3	1
7	0	0	$1+x+x^3$	3	$1+x$	4	1

Example 6.14 For the (31,21) binary double-error correcting code with decoding in Example 6.12, let us employ the Berlekamp-Massey algorithm to find the error locating polynomial. Recall from that example that the syndromes are $S_1 = \alpha^{17}$, $S_2 = \alpha^3$, $S_3 = 1$, and $S_4 = \alpha^6$. Running the Berlekamp-Massey algorithm over $GF(32)$ results in the computations shown in Table 6.6. The final connection polynomial $c(x) = 1 + \alpha^{17}x + \alpha^{22}x^2$ is the error location polynomial previously found using Peterson's algorithm. (In the current case, there are more computations using the Berlekamp-Massey algorithm, but for longer codes with more errors, the latter would be more efficient.)

Table 6.6: Berlekamp-Massey Algorithm for a Double-Error Correcting Code

k	S_k	d_k	$c(x)$	L	$p(x)$	l	d_m
1	α^{17}	α^{17}	$1+\alpha^{17}x$	1	1	1	α^{17}
2	α^3	0	$1+\alpha^{17}x$	1	1	2	α^{17}
3	1	α^8	$1+\alpha^{17}x+\alpha^{22}x^2$	2	$1+\alpha^{17}x$	1	α^8
4	α^6	0	$1+\alpha^{17}x+\alpha^{22}x^2$	2	$1+\alpha^{17}x$	2	α^8

□

6.4.4 Simplifications for Binary Codes

Consider again the Berlekamp-Massey algorithm computations for decoding a BCH code, as presented in Table 6.6. Note that d_k is 0 for every even k. This result holds in all cases for BCH codes:

Lemma 6.14 *When the sequence of input symbols to the Berlekamp-Massey algorithm are syndromes from a binary BCH code, then the discrepancy d_k is equal to 0 for all even k (when 1-based indexing is used).*

As a result, there is never an update for these steps of the algorithm, so they can be merged into the next step. This cuts the complexity of the algorithm approximately in half. A restatement of the algorithm for BCH decoding is presented below.

Algorithm 6.2 Massey's Algorithm for Binary BCH Decoding

```
Input: $S_1, S_2, \ldots, S_N$, where $N = 2t$
Initialize:
$L = 0$ (the current length of the LFSR)
```

```
$c(x) = 1$ (the current connection polynomial)
$p(x) = 1$ (the connection polynomial before last length change)
$l = 1$ ($l$ is $k - m$, the amount of shift in update)
$d_m = 1$ (previous discrepancy)
for $k = 1$ to $N$ in steps of 2
  $d = S_k + \sum_{i=1}^{L} c_i S_{k-i}$ (compute discrepancy)
  if ($d = 0$) (no change in polynomial)
    $l = l + 1$
  else
    if ($2L \geq k$) then (no-length change in update)
      $c(x) = c(x) - d d_m^{-1} x^l p(x)$
      $l = l + 1$
    else (update $c$ with length change)
      $t(x) = c(x)$ (temporary storage)
      $c(x) = c(x) - d d_m^{-1} x^l p(x)$
      $L = k - L$
      $p(x) = t(x)$
      $d_m = d$
      $l = 1$
    end
  end
  $l = l + 1$; (accounts for the values of $k$ skipped)
end
```

Example 6.15 Returning to the (31,21) code from the previous example, if we call the BCH-modified Berlekamp-Massey algorithm with the syndrome sequence $S_1 = \alpha^{17}$, $S_2 = \alpha^3$, $S_3 = 1$, and $S_4 = \alpha^6$, we obtain the results in Table 6.7. Only two steps of the algorithm are necessary and the same error locator polynomial is obtained as before. □

Table 6.7: Berlekamp-Massey Algorithm for a Double-Error Correcting code: Simplifications for the Binary Code

k	S_k	d_k	$c(x)$	L	$p(x)$	l	d_m
0	α^{17}	α^{17}	$1 + \alpha^{17}x$	1	1	2	α^{17}
2	1	α^8	$1 + \alpha^{17}x + \alpha^{22}x^2$	2	$1 + \alpha^{17}x$	2	α^8

The odd-indexed discrepancies are zero due to the fact that for binary codes, the syndromes S_j have the property that

$$(S_j)^2 = S_{2j}. \tag{6.27}$$

We call this condition the syndrome conjugacy condition. Equation (6.27) follows from (6.4) and freshman exponentiation.

For the example we have been following,

$$S_1^2 = (\alpha^{17})^2 = \alpha^3 = S_2 \qquad S_2^2 = (\alpha^3)^2 = \alpha^6 = S_4.$$

Example 6.16 We now present an entire decoding process for the three-error correcting $(15, 5)$ binary code generated by

$$g(x) = 1 + x + x^2 + x^4 + x^5 + x^8 + x^{10}.$$

Suppose the all-zero vector is transmitted and the received vector is

$$\mathbf{r} = (0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0).$$

Then $r(x) = x + x^3 + x^8$.

Step 1 Compute the syndromes. Evaluating $r(x)$ at $x = \alpha, \alpha^2, \ldots, \alpha^6$ we find the syndromes

$$S_1 = \alpha^{12} \quad S_2 = \alpha^9 \quad S_3 = \alpha^3 \quad S_4 = \alpha^3 \quad S_5 = 0 \quad S_6 = \alpha^6.$$

Step 2 Compute the error locator polynomial.

A call to the binary Berlekamp-Massey algorithm yields the following computations.

k	S_k	d_k	$c(x)$	L	$p(x)$	l	d_m
1	α^{12}	α^{12}	$1+\alpha^{12}x$	1	1	2	α^{12}
3	α^3	α^2	$1+\alpha^{12}x+\alpha^5x^2$	2	$1+\alpha^{12}x$	2	α^2
5	0	α^2	$1+\alpha^{12}x+\alpha^{10}x^2+\alpha^{12}x^3$	3	$1+\alpha^{12}x+\alpha^5x^2$	2	α^2

The error locator polynomial is thus

$$\Lambda(x) = 1 + \alpha^{12}x + \alpha^{10}x^2 + \alpha^{12}x^3.$$

Step 3 Find the roots of the error locator polynomial. Using the Chien search function, we find roots at α^7, α^{12} and α^{14}. Inverting these, the error locators are

$$X_1 = \alpha^8 \quad X_2 = \alpha^3 \quad X_3 = \alpha,$$

indicating that errors at positions 8, 3, and 1.

Step 4 Determine the error values: for a binary BCH code, any errors have value 1.

Step 5 Correct the errors: Add the error values (1) at the error locations, to obtain the decoded vector of all zeros.

□

6.5 Non-Binary BCH and RS Decoding

For nonbinary BCH or RS decoding, some additional work is necessary. Some extra care is needed to find the error locators, then the error values must be determined.

From (6.3) we can write

$$\begin{aligned}
S_1 &= e_{i_1}X_1 + e_{i_2}X_2 + \cdots + e_{i_\nu}X_\nu \\
S_2 &= e_{i_1}X_1^2 + e_{i_2}X_2^2 + \cdots + e_{i_\nu}X_\nu^2 \\
S_3 &= e_{i_1}X_1^3 + e_{i_2}X_2^3 + \cdots + e_{i_\nu}X_\nu^3 \\
&\vdots \\
S_{2t} &= e_{i_1}X_1^{2t} + e_{i_2}X_2^{2t} + \cdots + e_{i_\nu}X_\nu^{2t}.
\end{aligned}$$

Because of the e_{i_l} coefficients, these are not power-sum symmetric functions as was the case for binary codes. Nevertheless, in a similar manner it is possible to make use of an error locator polynomial.

Lemma 6.15 *The syndromes and the coefficients of the error locator polynomial* $\Lambda(x) = \Lambda_0 + \Lambda_1 x + \cdots + \Lambda_\nu x^\nu$ *are related by*

$$\Lambda_\nu S_{j-\nu} + \Lambda_{\nu-1}S_{j-\nu+1} + \cdots + \Lambda_1 S_{j-1} + S_j = 0. \qquad (6.28)$$

Proof Evaluating the error locator polynomial $\Lambda(x) = \prod_{i=1}^{\nu}(1 - X_i x)$ at an error locator X_l,

$$\Lambda(X_l^{-1}) = 0 = \Lambda_\nu X_l^{-\nu} + \Lambda_{\nu-1} X_l^{1-\nu} + \cdots + \Lambda_1 X_l^{-1} + \Lambda_0.$$

Multiplying this equation by $e_{i_l} X_l^j$ we obtain

$$e_{i_l} X_l^j \Lambda(X_l^{-1}) = e_{i_l}(\Lambda_\nu X_l^{j-\nu} + \Lambda_{\nu-1} X_l^{j+1-\nu} + \cdots + \Lambda_1 X_l^{j-1} + \Lambda_0 X_l^j) = 0 \quad (6.29)$$

Summing (6.29) over l we obtain

$$\begin{aligned} 0 &= \sum_{l=1}^{\nu} e_{i_l}(\Lambda_\nu X_l^{j-\nu} + \Lambda_{\nu-1} X_l^{j+1-\nu} + \cdots + \Lambda_1 X_l^{j-1} + \Lambda_0 X_l^j) \\ &= \Lambda_\nu \sum_{l=1}^{\nu} e_{i_l} X_l^{j-\nu} + \Lambda_{\nu-1} \sum_{i=1}^{\nu} e_{i_l} X_l^{j+1-\nu} + \cdots + \Lambda_1 \sum_{l=1}^{\nu} e_{i_l} X_l^{j-1} + \Lambda_0 \sum_{l=1}^{\nu} e_{i_l} X_l^j. \end{aligned}$$

In light of (6.3), the latter equation can be written as

$$\Lambda_\nu S_{j-\nu} + \Lambda_{\nu-1} S_{j-\nu+1} + \cdots + \Lambda_1 S_{j-1} + \Lambda_0 S_j = 0.$$

□

Because (6.28) holds, the Berlekamp-Massey algorithm (in its non-binary formulation) can be used to find the coefficients of the error locator polynomial, just as for binary codes.

6.5.1 Forney's Algorithm

Having found the error-locator polynomial and its roots, there is still one more step for the non-binary BCH or RS codes: we have to find the error values. Let us return to the syndrome,

$$S_j = \sum_{l=1}^{\nu} e_{i_l} X_l^j, \qquad j = 1, 2, \ldots, 2t.$$

Knowing the error locators (obtained from the roots of the error locator polynomial) it is straightforward to set up and solve a set of linear equations:

$$\begin{bmatrix} X_1 & X_2 & X_3 & \cdots & X_\nu \\ X_1^2 & X_2^2 & X_3^2 & \cdots & X_\nu^2 \\ \vdots & & & & \\ X_1^{2t} & X_2^{2t} & X_3^{2t} & \cdots & X_\nu^{2t} \end{bmatrix} \begin{bmatrix} e_{i_1} \\ e_{i_2} \\ \vdots \\ e_{i_\nu} \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_{2t} \end{bmatrix}. \quad (6.30)$$

However, there is a method which is computationally easier and in addition provides us a key insight for another way of doing the decoding. It may be observed that the matrix in (6.30) is essentially a Vandermonde matrix. There exist fast algorithms for solving Vandermonde systems (see, e.g., [121]). One of these which applies specifically to this problem is known as *Forney's algorithm.*

Before presenting the formula, a few necessary definitions must be established. A *syndrome polynomial* is defined as

$$S(x) = S_1 + S_2 x + S_3 x^2 \cdots + S_{2t} x^{2t-1} = \sum_{j=0}^{2t-1} S_{j+1} x^j. \quad (6.31)$$

Also an *error-evaluator polynomial* $\Omega(x)$ is defined[1] by

$$\boxed{\Omega(x) = S(x)\Lambda(x) \pmod{x^{2t}}.} \tag{6.32}$$

This equation is called the **key equation**. Note that the effect of computing modulo x^{2t} is to discard all terms of degree $2t$ or higher.

Definition 6.5 Let $f(x) = f_0 + f_1 x + f_2 x^2 + \cdots + f_t x^t$ be a polynomial with coefficients in some field $\mathbb{F}$. The **formal derivative** $f'(x)$ of $f(x)$ is computed using the conventional rules of polynomial differentiation:

$$f'(x) = f_1 + 2f_2 x + 3f_3 x^2 + \cdots + t f_t x^{t-1}, \tag{6.33}$$

where, as usual, mf_i for $m \in \mathbb{Z}$ and $f_i \in \mathbb{F}$ denotes repeated addition:

$$mf_i = \underbrace{f_i + f_i + \cdots + f_i}_{m \text{ summands}}.$$

□

There is no implication of any kind of limiting process in formal differentiation: it simply corresponds to formal manipulation of symbols. Based on this definition, it can be shown that many of the conventional rules of differentiation apply. For example, the product rule holds:

$$[f(x)g(x)]' = f'(x)g(x) + f(x)g'(x).$$

If $f(x) \in \mathbb{F}[x]$, where $\mathbb{F}$ is a field of characteristic 2, then $f'(x)$ has no odd-powered terms.

Theorem 6.16 *(Forney's algorithm) The error values for a Reed-Solomon code are computed by*

$$e_{i_k} = -\frac{\Omega(X_k^{-1})}{\Lambda'(X_k^{-1})}, \tag{6.34}$$

where $\Lambda'(x)$ is the formal derivative of $\Lambda(x)$.

Proof First note that over any ring,

$$(1 - x^{2t}) = (1 - x)(1 + x + x^2 + \cdots + x^{2t-1}) = (1 - x)\sum_{j=0}^{2t-1} x^j. \tag{6.35}$$

Observe:

$$\begin{aligned}
\Omega(x) &= S(x)\Lambda(x) \pmod{x^{2t}} \\
&= \left(\sum_{j=0}^{2t-1}\sum_{l=1}^{\nu} e_{i_l} X_l^{j+1} x^j\right)\left(\prod_{i=1}^{\nu}(1 - X_i x)\right) \pmod{x^{2t}} \\
&= \sum_{l=1}^{\nu} e_{i_l} X_l \sum_{j=0}^{2t-1}(X_l x)^j \prod_{i=1}^{\nu}(1 - X_i x) \pmod{x^{2t}} \\
&= \sum_{l=1}^{\nu} e_{i_l} X_l \left[(1 - X_l x)\sum_{j=0}^{2t-1}(X_l x)^j\right] \prod_{i \neq l}^{\nu}(1 - X_i x) \pmod{x^{2t}}.
\end{aligned}$$

[1] Some authors define $S(x) = S_1 x + S_2 x^2 + \cdots + S_{2t} x^{2t}$, in which case they define $\Omega(x) = (1 + S(x))\Lambda(x) \pmod{x^{2t+1}}$ and obtain $e_{i_k} = -X_k \Omega(X_k^{-1})/\Lambda'(X_k^{-1})$.

From (6.35),

$$(1 - X_l x) \sum_{j=0}^{2t-1} (X_l x)^j = 1 - (X_l x)^{2t}.$$

Since $(X_l x)^{2t} \pmod{x^{2t}} = 0$ we have

$$S(x)\Lambda(x) \pmod{x^{2t}} = \sum_{l=1}^{\nu} e_{i_l} X_l \prod_{i \neq l} (1 - X_i x).$$

Thus

$$\Omega(x) = \sum_{l=1}^{\nu} e_{i_l} X_l \prod_{i \neq l} (1 - X_i x).$$

The trick now is to isolate a particular e_{i_k} on the right-hand side of this expression.

Evaluate $\Omega(x)$ at $x = X_k^{-1}$:

$$\Omega(X_k^{-1}) = \sum_{l=1}^{\nu} e_{i_l} X_l \prod_{i \neq l} (1 - X_i X_k^{-1}).$$

Every term in the sum results in a product that has a zero in it, except the term when $l = k$, since that term is skipped. We thus obtain

$$\Omega(X_k^{-1}) = e_{i_k} X_k \prod_{i \neq k} (1 - X_i X_k^{-1}).$$

We can thus write

$$e_{i_k} = \frac{\Omega(X_k^{-1})}{X_k \prod_{i \neq k} (1 - X_i X_k^{-1})}. \tag{6.36}$$

Once $\Omega(x)$ is known, the error values can thus be computed. However, there are some computational simplifications.

The formal derivative of $\Lambda(x)$ is

$$\Lambda'(x) = \frac{d}{dx} \prod_{i=1}^{\nu} (1 - X_i x) = -\sum_{l=1}^{\nu} X_l \prod_{i \neq l} (1 - X_i x).$$

Then

$$\Lambda'(X_k^{-1}) = -X_k \prod_{i \neq k} (1 - X_i X_k^{-1}).$$

Substitution of this result into (6.36) yields (6.34). □

Example 6.17 Working over $GF(8)$ in a code where $t = 2$, suppose $S(x) = \alpha^6 + \alpha^3 x + \alpha^4 x^2 + \alpha^3 x^3$. We find (say using the B-M algorithm and the Chien search) that the error locator polynomial is

$$\Lambda(x) = 1 + \alpha^2 x + \alpha x^2 = (1 + \alpha^3 x)(1 + \alpha^5 x).$$

That is, the error locators (reciprocals of the roots of $\Lambda(x)$) are $X_1 = \alpha^3$ and $X_2 = \alpha^5$. We have

$$\Omega(x) = (\alpha^6 + \alpha^3 x + \alpha^4 x^2 + \alpha^3 x^3)(1 + \alpha^2 x + \alpha x^2) \pmod{x^4} = (\alpha^6 + x + \alpha^4 x^5) \pmod{x^4} = \alpha^6 + x$$

and

$$\Lambda'(x) = \alpha^2 + 2\alpha x = \alpha^2.$$

So

$$e_{i_k} = -\frac{\alpha^6 + x}{\alpha^2}\bigg|_{x=X_k^{-1}} = \alpha^4 + \alpha^5 X_k^{-1}.$$

Using the error locator $X_1 = \alpha^3$ we find

$$e_3 = \alpha^4 + \alpha^5(\alpha^3)^{-1} = \alpha$$

and for the error locator $X_2 = \alpha^5$,

$$e_5 = \alpha^4 + \alpha^5(\alpha^5)^{-1} = \alpha^5.$$

The error polynomial is $e(x) = \alpha x^3 + \alpha^5 x^5$. □

Example 6.18 We consider the entire decoding process for (15,9) code of Example 6.8, using the message and code polynomials in Example 6.10. Suppose the received polynomial is

$$r(x) = \alpha^8 + \alpha^2 x + \underline{\alpha^{13}x^2} + \alpha^3 x^3 + \alpha^5 x^4 + \alpha x^5 + \alpha^8 x^6 + \alpha x^7 + \underline{\alpha x^8} + \alpha^5 x^9 + \alpha^3 x^{10}$$
$$+ \alpha^4 x^{11} + \alpha^9 x^{12} + \alpha^{12} x^{13} + \underline{\alpha^5 x^{14}}.$$

(Errors are in the underlined positions.)

The syndromes are

$$S_1 = r(\alpha) = \alpha^{13} \quad S_2 = r(\alpha^2) = \alpha^4 \quad S_3 = r(\alpha^3) = \alpha^8$$
$$S_4 = r(\alpha^4) = \alpha^2 \quad S_5 = r(\alpha^5) = \alpha^3 \quad S_6 = r(\alpha^6) = \alpha^8$$

so

$$S(x) = \alpha^{13} + \alpha^4 x + \alpha^8 x^2 + \alpha^2 x^3 + \alpha^3 x^4 + \alpha^8 x^5$$

and the error locator polynomial determined by the Berlekamp-Massey algorithm is

$$\Lambda(x) = 1 + \alpha^3 x + \alpha^{11} x^2 + \alpha^9 x^3.$$

The details of the Berlekamp-Massey computations are shown in Table 6.8.

Table 6.8: Berlekamp-Massey Algorithm for a Triple-Error Correcting Code

k	S_k	d_k	$c(x)$	L	$p(x)$	l	d_m
1	α^{13}	α^{13}	$1+\alpha^{13}x$	1	1	1	α^{13}
2	α^4	α^{13}	$1+\alpha^6 x$	1	1	2	α^{13}
3	α^8	α	$1+\alpha^6 x+\alpha^3 x^2$	2	$1+\alpha^6 x$	1	α
4	α^2	α^5	$1+\alpha^{12}x+\alpha^{12}x^2$	2	$1+\alpha^6 x$	2	α
5	α^3	α^{10}	$1+\alpha^{12}x+\alpha^8 x^2+x^3$	3	$1+\alpha^{12}x+\alpha^{12}x^2$	1	α^{10}
6	α^8	α^5	$1+\alpha^3 x+\alpha^{11}x^2+\alpha^9 x^3$	3	$1+\alpha^{12}x+\alpha^{12}x^2$	2	α^{10}

The roots of $\Lambda(x)$ are at α, α^7 and α^{13}, so the error locators (the reciprocal of the roots) are

$$X_1 = \alpha^{14} \quad X_2 = \alpha^8 \quad X_3 = \alpha^2,$$

corresponding to errors at positions 14, 8, and 2. The error evaluator polynomial is

$$\Omega(x) = \alpha^{13} + x + \alpha^2 x^2.$$

Then the computations to find the error values are:

$$\begin{array}{llll}
X_1 = \alpha^{14}: & \Omega(X_1^{-1}) = \alpha^6 & \Lambda'(X_1^{-1}) = \alpha^5 & e_{14} = \alpha \\
X_2 = \alpha^8: & \Omega(X_2^{-1}) = \alpha^2 & \Lambda'(X_2^{-1}) = \alpha^{13} & e_8 = \alpha^4 \\
X_3 = \alpha^2: & \Omega(X_3^{-1}) = \alpha^{13} & \Lambda'(X_3^{-1}) = \alpha^{11} & e_2 = \alpha^2
\end{array}$$

The error polynomial is thus

$$e(x) = \alpha^2 x^2 + \alpha^4 x^8 + \alpha x^{14}$$

and the decoded polynomial is

$$\alpha^8 + \alpha^2 x + \alpha^{14} x^2 + \alpha^3 x^3 + \alpha^5 x^4 + \alpha x^5 + \alpha^8 x^6 + \alpha x^7 + x^8 + \alpha^5 x^9 + \alpha^3 x^{10} \\ + \alpha^4 x^{11} + \alpha^9 x^{12} + \alpha^{12} x^{13} + \alpha^2 x^{14}.$$

which is the same as the original codeword $c(x)$. □

6.6 Euclidean Algorithm for the Error Locator Polynomial

We have seen that the Berlekamp-Massey algorithm can be used to construct the error locator polynomial. In this section, we show that the Euclidean algorithm can also be used to construct error locator polynomials. This approach to decoding is often called the Sugiyama algorithm [324].

We return to the key equation:

$$\Omega(x) = S(x)\Lambda(x) \pmod{x^{2t}}. \tag{6.37}$$

Given only $S(x)$ and t, we desire to determine the error locator polynomial $\Lambda(x)$ and the error evaluator polynomial $\Omega(x)$. As stated, this problem seems hopelessly underconstrained. However, recall that (6.37) means that

$$\Theta(x)(x^{2t}) + \Lambda(x)S(x) = \Omega(x)$$

for some polynomial $\Theta(x)$. (See (5.16).) Also recall that the extended Euclidean algorithm returns, for a pair of elements (a, b) from a Euclidean domain, a pair of elements (s, t) such that

$$as + bt = c,$$

where c is the GCD of a and b. In our case, we run the extended Euclidean algorithm to obtain a sequence of polynomials $\Theta^{[k]}(x)$, $\Lambda^{[k]}(x)$ and $\Omega^{[k]}(x)$ satisfying

$$\Theta^{[k]}(x)x^{2t} + \Lambda^{[k]}(x)S(x) = \Omega^{[k]}(x).$$

This is exactly the circumstance described in Section 5.2.3. Recall that the stopping criterion there is based on the observation that the polynomial we are here calling $\Omega(x)$ must have degree $< t$.

The steps to decode using the Euclidean algorithm are summarized as follows:

1. Compute the syndromes and the syndrome polynomial $S(x) = S_1 + S_2 x + \cdots + S_{2t} x^{2t-1}$.
2. Run the Euclidean algorithm with $a(x) = x^{2t}$ and $b(x) = S(x)$, until $\deg(r_i(x)) < t$. Then $\Omega(x) = r_i(x)$ and $\Lambda(x) = t_i(x)$.
3. Find the roots of $\Lambda(x)$ and the error locators X_i.