Reed-Solomon Codes

1 Introduction

A Reed-Solomon (RS) code is an error-correcting code first described in a paper by Reed and Solomon in 1960 [8]. Encoding data using an RS code is relatively straightforward, but decoding is time-consuming. Despite major efficiency improvements made by Berlekamp and others during the 1960's [1, 4, 5, 7], only in the past few years has it become computationally possible to send high-bandwidth data using RS codes. Recent applications of RS codes include CD-ROMs, wireless communications, space communications, DSL, DVD, and digital TV.

An RS code differs from a Hamming code in that it encodes groups of bits instead of one bit at a time. We will call these groups "digits" (or "symbols" or "coefficients"). A digit is error-free if and only if all of its bits are error-free. For instance, if a digit is an 8-bit character, and three bits of the same single character are in error, we will count that as one corrupted digit. There are two corrupted digits (but more than two corrupted bits) in the following example.

Original :	10110001	11011111	01001011	01011100
Received :	10110101	11011111	01110001	01011100
Corrupted?	yes	no	yes	no

If we want to send a k-digit plaintext message, RS will send $n = k + 2s$ digits, and guarantee that the correct message can be reconstructed at the other end if there are fewer than s corrupted digits.

An example of commonly used parameters: $k = 223$, $s = 16$, $n = k+2s = 255$, giving the ability to correct 16 corrupted digits out of every 255-digit transmission. In general, the number of bits in a digit and the parameters n and s are tuned to optimize for your application. A CD-ROM can correct a burst of up to 4000 consecutive errors.

2 Outline of encoding

Now we'll describe the RS encoding. The key to it all is to view the digits of the message that is to be encoded as if they are the coefficients of a polynomial. If there are j distinct digit values (e.g., 256 distinct 8-bit digit values), we'll create a mapping between these values and the elements of a field F with j elements. Our choice of F will be described later, but for now just understand that this defines how the basic arithmetic operations work for digits. For a k-digit message whose digits are $m_0, m_1, \ldots, m_{k-1}$, we define the corresponding message polynomial to be $m(x) = m_0 + m_1 x + m_2 x^2 + \cdots + m_{k-1} x^{k-1}$, where the coefficients $m_0, m_1, \ldots, m_{k-1}$ of the polynomial $m(x)$ are elements of F. We use $F[x]$ to denote the set of polynomials with coefficients in F; so $m(x) \in F[x]$.

Recall that there is exactly one polynomial of degree $k-1$ that passes through k points $(x_0, y_0), \ldots, (x_{k-1}, y_{k-1})$ with all the x_i distinct and non-zero. If we weren't worried about errors, one alternative to transmitting the coefficients of $m(x)$ would be to evaluate $m(x)$ at k known points, send the k evaluations of $m(x)$ as our message, and then reconstruct $m(x)$ from those evaluations. This approach doesn't provide any advantages if there are guaranteed to be no errors, but it can be extended to provide the redundancy necessary to correct errors. The way we will get our redundancy is by evaluating $m(x)$ at $2s$ additional distinct non-zero points; so that what we send is $n = k + 2s$ evaluations $\{m(x_0), \ldots, m(x_{n-1})\}$, where $x_0, \ldots, x_{n-1}$ are distinct x-values agreed upon before sending the message.

We will reconstruct $m(x)$ in the following way: let $(y_0, \ldots, y_{n-1})$ be the evaluations received, with at most most s errors. Try to find a subset of $k+s$ (or more) points from $\{(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})\}$ such that a degree-(k-1) polynomial passes through the points. Such a subset must exist, since we start with $k+2s$ points and at most s points are in error. Once we have such a subset, we know that it matches evaluations of $m(x)$ for at least k distinct x-values, because at most s of the $k+s$ points are in error. Since k evaluations uniquely determine a degree-$(k-1)$ polynomial, we can reconstruct the correct $m(x)$.

The algorithm just described has the right flavor, but the step of finding a subset of consistent points is computationally difficult. Later on we'll improve this algorithm and talk about modern RS coding implementations; but first let's flesh out the construction of the field over which we take our polynomial $m(x)$.

3 Constructing the field F

3.1 Refresher on fields

We introduce here the basic algebraic definitions which will be used in this section.

Groups. A group is a tuple $(G, \times, e)$, where G is a set of elements, $\times$ is a binary operator on G, and $e \in G$ is a designated identity element. G must have the following properties:

1. $\times$ is closed. For all $a, b \in G$, $a \times b \in G$.

2. $\times$ is associative. For all $a, b, c \in G$, $(a \times b) \times c = a \times (b \times c)$.

3. For all $a \in G$, $a \times e = e \times a = a$.

4. For each $a \in G$ there is an element $a^{-1} \in G$ such that $a \times a^{-1} = a^{-1} \times a = e$ (a^{-1} is called an inverse of a).

We will often abbreviate $a \times b$ as ab when it's obvious what the operator is.

Fields. A field is a tuple $(F, +, \times, 0, 1)$, where F is the set of elements, $+$ is the addition operator, $\times$ is the multiplication operator, $0 \in F$ is the additive identity, and $1 \in F$ is the multiplicative identity. F must have the following properties:

1. $(F, +, 0)$ forms a group.

2. $\times$ is associative and distributes over $+$.

3. $(F \setminus \{0\}, \times, 1)$ forms a group.

Note we will often say "the field F" when we mean "the field whose set is F, with the standard operators for that set".

Fields that we commonly work with include the real, complex, and rational numbers. The set of integers is not a field. It satisfies the first two properties, but fails the third property because not all integers have multiplicative inverses. A set like the integers which satisfies the first two field properties is called a "ring".

3.2 Galois fields

We now turn to the question of constructing the field F from which the coefficients of $m(x)$ are drawn. A basic result from number theory is that if p is prime, then the set of integers modulo p (denoted $\mathbb{Z}_p$) is a field. So if there are a prime number p of possible digits, we can use $\mathbb{Z}_p$ as our field. However, that is true if and only if p is prime. Unfortunately, in computer applications we are likely to want digits that we can encode with $r > 1$ bits; for instance, 8-bit characters. This means we have a non-prime number 2^r of possible digits.

Fortunately, it can be proven that for any prime p and any natural number r there exists a finite field with p^r numbers. (In fact the reverse is also true—every finite field has p^r numbers, where p is prime). There is a way to generate such a field. It is called a Galois field, and it can be shown that any finite field of size p^r is isomorphic to a Galois field.

Galois fields are constructed with the help of $\mathbb{Z}_p[x]$, the set of polynomials with coefficients in $\mathbb{Z}_p$.

We're used to dealing with polynomials with real coefficients (polynomials in $\mathbb{R}[x]$), so the arithmetic of polynomials in $\mathbb{Z}_p[x]$ may seem counter-intuitive. Take $F = \mathbb{Z}_2$, for instance.

A refresher on how arithmetic modulo 2 works:

$$\begin{aligned} 0+0 &= 0 \\ 0+1 &= 1 \\ 1+0 &= 1 \\ 1+1 &= 0 \end{aligned}$$

Now for example, let $a, b \in \mathbb{Z}_2[x]$, $a(x) = x^2 + x$, $b(x) = x$. Then we can do the addition

$$\begin{aligned} a(x) &= 1x^2 + 1x \\ b(x) &= 0x^2 + 1x \\ a(x) + b(x) &= 1x^2 + 0x \quad = x^2 \end{aligned}$$

which is of course a different result than we would have gotten if $a(x), b(x) \in \mathbb{R}[x]$; in that case we would have had $(x^2 + x) + (x) = x^2 + 2x$.

Now to define modulo arithmetic for polynomials: $a(x)$ modulo $g(x)$ is another polynomial $r(x)$ with degree strictly less than $g(x)$, and satisfying $a(x) = g(x)q(x) + r(x)$ for some polynomial $q(x)$. **Exercise:** Prove that the polynomial $r(x)$ with those properties exists, is unique, and is the same as the remainder of polynomial long division when dividing $a(x)$ by $g(x)$.

Consider the set $\mathbb{Z}_p[x]$ of polynomials over the field $\mathbb{Z}_p$ and an irreducible polynomial $g(x)$ of degree r. An irreducible polynomial is one that cannot be factored into a product of lower-degree polynomials over $\mathbb{Z}_p$.

Now let F be the set of all polynomials in $\mathbb{Z}_p[x]$ with degree at most $r-1$, with the operations of polynomial additional and polynomial multiplication modulo $g(x)$. Obviously there are p^r such polynomials (each of r coefficients can take one of p possible values). We will denote the field $\mathrm{GF}(p^r)$ (recall that r is the degree of polynomial $g(x)$). Thus we now have a method of constructing finite fields with p^r elements. Note that we don't need $g(x)$

to enumerate the elements of $\mathrm{GF}(p^r)$—the elements are all the polynomials with degree at most $r-1$. However, we need $g(x)$ to define operations in this field (i.e., the multiplication of the fields' elements will be taken modulo $g(x)$)[1].

Unfortunately, there is no simple method of obtaining an irreducible polynomial (there are complicated probabilistic algorithms that can do that, but we won't describe them here). However, irreducible polynomials for most common finite fields have been found and published. Also, it can be proven that an irreducible polynomial of degree r over $\mathbb{Z}_p$ exists for every positive integer r.

Let us consider an example of the Galois field—$\mathrm{GF}(2^3)$. As stated above we can construct such a field by enumerating the possible polynomial residues modulo some irreducible polynomial of degree 3 over $\mathbb{Z}_2$. It can be shown that $1+x+x^3$ is irreducible over $\mathbb{Z}_2$.

Therefore the field we seek is $\{0, 1, x, 1+x, x^2, 1+x^2, x+x^2, 1+x+x^2\}$. That is a simple example. The most commonly used Galois field is $\mathrm{GF}(2^8)$, since we are usually interested in bytes as information units.

Finite fields (recall that any finite field of size p^r is isomorphic to a Galois field) have some nice properties. One of them is that the multiplicative group of a finite field F is cyclic. It means that a group contains an element α such that every $a \in G$ equals α^i for some integer i. α is called a generator (or a primitive element) of the group. The finite order of α is the smallest integer $k > 0$ such that $\alpha^k = 1$ (where 1 is the identity of the group). It is easy to show that the finite order of the generator of $\mathrm{GF}(p^r)$ is $p^r - 1$.

3.3 Implementing Reed-Solomon coding

Our initial intention was to view a message as a polynomial over some finite field, where message digits (e.g., bytes) are coefficients. Message units $m_0, \ldots, m_{k-1}$ are mapped to the elements of some $\mathrm{GF}(p^r)$.

[1] More on Galois fields and quotient rings: $\mathbb{Z}_p[x]$ can be mapped to F—we will divide each $f(x) \in \mathbb{Z}_p[x]$ by $g(x)$ and take the remainder as the image of $f(x)$. The mapping is not bijective, since an infinite number of different polynomials can have the same remainder modulo $g(x)$. We will call the set of polynomials with the same remainder as $f(x)$ the class of $f(x)$. When we talk about the class of $c(x)$, we'll prefer to choose $c(x)$ to be the unique polynomial in the class which is a member of F (has degree strictly less than $g(x)$). In this case, instead of saying "the class of $c(x)$" we may just say "$c(x)$". Thus we uniquely identify classes with members of F. The set of classes for an arbitrary $g(x)$ is called a quotient ring and is denoted $\frac{\mathbb{Z}_p[x]}{g(x)}$. It is easy to show that such a set really forms a ring. However, when $g(x)$ is irreducible, $\frac{\mathbb{Z}_p[x]}{g(x)}$ becomes a field. That is why irreducibility of $g(x)$ is important. A Galois field is precisely such a field—$\frac{\mathbb{Z}_p[x]}{g(x)}$.

Recall that our method of encoding was to evaluate a message polynomial $m(x) = m_0 + m_1 x + ... m_{k-1} x^{k-1}$ at $n = k + 2s$ distinct non-zero points and send those values. Again, if there are at most s errors, then there will be a set of $k+s$ error-free points, which will be consistent with the degree-$(k-1)$ polynomial $m(x)$. Any set of $k+s$ points consistent with a degree-$(k-1)$ polynomial will have at most s points in error, so there will be k error-free points, enough to guarantee that the polynomial they are consistent with is $m(x)$.

Note that n should be at most $p^r - 1$, the number of non-zero elements in the group, since otherwise we will simply not have enough different points at which to evaluate $m(x)$.

Before passing to the details of the real implementation, let us explain how the polynomial interpolation works. To avoid confusion forget for a moment that the coefficients are drawn from $\mathrm{GF}(p^r)$ and think of them as elements of any abstract field, for instance the real numbers $\mathbb{R}$ (the next paragraph is not specific to finite fields).

Suppose we are given only k polynomial evaluations,

$$m(x_0) = y_0, \ldots, m(x_{k-1}) = y_{k-1},$$

and we want to reconstruct a degree-$(k-1)$ polynomial fitting those points. Probably the simplest way to do this is to use the Lagrange method:

$$m(x) = \sum_{i=0}^{k-1} y_i \prod_{\substack{j=0 \\ j \neq i}}^{k-1} \frac{x - x_j}{x_i - x_j}.$$

Note that

$$\prod_{\substack{j=0 \\ j \neq i}}^{k-1} \frac{x - x_j}{x_i - x_j} = \begin{cases} 1, \text{if } x = x_i, \\ 0, \text{if } x = x_j, \text{ where } j \neq i. \end{cases}$$

It is easy to check that this polynomial really fits $y_0, \ldots, y_{k-1}$.

There exists another (more straightforward) approach to find the polynomial coefficients—interpolation can be viewed as a problem of solving the system of k linear equations with k unknowns $(m_0, m_1, \ldots, m_{k-1})$:

$$\begin{aligned} m_0 + m_1 \cdot x_0 + \ldots + m_{k-1} \cdot x_0^{k-1} &= y_0 \\ m_0 + m_1 \cdot x_1 + \ldots + m_{k-1} \cdot x_1^{k-1} &= y_1 \\ &\vdots \\ m_0 + m_1 \cdot x_{k-1} + \ldots + m_{k-1} \cdot x_{k-1}^{k-1} &= y_{k-1} \end{aligned}$$

This setting provides us with at least one way to deal with decoding problem: given a set of n evaluations $y_0, \ldots, y_n$ we can simply enumerate all the possible $(k+s)$-subsets of this set and find $k+s$ consistent values. Of course that is not a realistic method due to computational difficulties.

The real methods also differ in what is sent as a codeword. They use what is called systematic form encoding. Instead of n evaluations of $m(x)$ we send k coefficients of $m(x)$ (that is, the original k-digit message in plaintext) together with a parity section of $2s$ symbols. The motivation is improved efficiency in the common case that there are no errors: in that case, we would like to limit our work to a computationally easy procedure that verifies the lack of errors, and then just read off the original message without any decoding.

Let's imagine that the $2s$ parity check digits are evaluations of $m(x)$ at $2s$ distinct non-zero points $x_0, \ldots, x_{2s-1}$. How could we reconstruct the polynomial $m(x)$, given k coefficients and $2s$ evaluations, with a total of s possible errors? We can write $2s$ linear equations

$$\begin{aligned} m_0 + m_1 \cdot x_0 + \ldots + m_{k-1} \cdot x_0^{k-1} &= y_0 \\ m_0 + m_1 \cdot x_1 + \ldots + m_{k-1} \cdot x_1^{k-1} &= y_1 \\ &\vdots \\ m_0 + m_1 \cdot x_{2s-1} + \ldots + m_{k-1} \cdot x_{2s-1}^{k-1} &= y_{2s-1} \end{aligned}$$

If a set of k error-free digits (coefficients and evaluations) contains $k-l$ coefficients and l evaluations, we have l useful equations (one for each error-free evaluation) and l unknown coefficients on the left. Therefore any such set of k digits is sufficient to determine all of the unknown coefficients. As before, there must be a consistent set of $k+s$ digits, and any consistent set of $k+s$ digits will have k error-free digits, implying it must be consistent with $m(x)$.

However, real implementations of RS-coding don't send evaluations of $m(x)$ in the parity section. They use a different approach, again because we want to make the common case, in which we only need to verify that there are no errors, computationally easy.

Recall that coefficients of $m(x)$ are elements of the finite field $\mathrm{GF}(p^r)$ and that finite fields are cyclic. Take α, a primitive element of $\mathrm{GF}(p^r)$, and define a generator polynomial

$$g(x) \quad = (x-\alpha)\cdot(x-\alpha^2)\cdot\dots\cdot(x-\alpha^{2s})$$

We'll use $g(x)$ to map k-digit messages to a k-dimensional subspace of the n-dimensional space of n-digit messages. Every codeword will be a multiple of $g(x)$.

Define

$$b(x) \quad = x^{2s}\cdot m(x) \pmod{g(x)}$$

So, for some polynomial $q(x)$, $x^{2s}m(x) = q(x)g(x) + b(x)$. Now define the final codeword $c(x)$ to be

$$c(x) \quad = x^{2s}\cdot m(x) - b(x)$$

This means that our codeword will look like $\boxed{m_{k-1},\dots,m_0,-b_{2s-1},\dots,-b_0}$. We've constructed $c(x)$ so that

$$\begin{aligned} c(x) &= x^{2s}\cdot m(x) - b(x) \\ &= [q(x)g(x) + b(x)] - b(x) \\ &= q(x)g(x) \end{aligned}$$

and $c(x)$ is therefore a multiple of $g(x)$.

To check if the received codeword $r(x)$ is correct we can check its divisibility by $g(x)$, and if the answer is affirmative we can simply extract the first k elements of $r(x)$ to decode a message without any additional computation.

Before going into more detail we need to prove that the numbers we transmit really suffice to restore the original message:

Lemma 3.3.1 *For any β where $g(\beta)=0$, $\beta^{2s}\cdot m(\beta) = b(\beta)$.*

Proof. $x^{2s}\cdot m(x) = g(x)\cdot d(x) + b(x)$, for some $d(x)$. Substituting β we obtain

$$\beta^{2s}\cdot m(\beta) \quad = g(\beta)\cdot d(\beta) + b(\beta) = b(\beta)$$

The latter is true because β is a root of $g(x)$. Note also that since $g(x) = (x-\alpha)(x-\alpha^2)\cdots(x-\alpha^{2s})$, α^l are the roots of $g(x)$ and $\alpha^{l\cdot 2s}\cdot m(\alpha^l) = b(\alpha^l)$ for all $l\in\{1,\dots,2s\}$. Q.E.D.

Lemma 3.3.2 *Given any collection of k coefficients from the polynomials $m(x)$ and $b(x)$, it is possible to reconstruct the remaining coefficients of $m(x)$ and $b(x)$.*

Proof. Since $g(x)$ has degree $2s$, $b(x)$ has degree at most $2s-1$. So $b(x)$ has $2s = n-k$ coefficients. Hence $c(x)$ (i.e., $m(x)$ and $b(x)$ together) has n coefficients. We are given k of them, so $2s$ are unknown. We also have $2s$ equations following from the previous lemma:

$$\begin{aligned} \alpha^{2s} \cdot m(\alpha) &= b(\alpha) \\ \alpha^{4s} \cdot m(\alpha^2) &= b(\alpha^2) \\ \alpha^{6s} \cdot m(\alpha^3) &= b(\alpha^3) \\ &\vdots \\ \alpha^{2s(2s)} \cdot m(\alpha^{2s}) &= b(\alpha^{2s}) \end{aligned}$$

Note that we have exactly $2s$ unknowns and $2s$ equations. The linear independence of the equations can be derived from the fact that α is a generator of $\mathrm{GF}(p^r)$ with a finite order $p^r - 1$. Q.E.D.

We should also prove that the algorithm will still be able to recover from at most s errors:

Lemma 3.3.3 *If at most s of the coefficients of $m(x)$ and $b(x)$ are incorrect, it is possible to reconstruct all of the coefficients of $m(x)$ correctly.*

Proof. As before, if there are s or fewer errors, then at least $k+s$ of the coefficients are correct. This means that there is at least one set of $k+s$ or more coefficients that are consistent with each other, and these are from the original set of n coefficients (before errors were introduced).

Furthermore, suppose we find any set of $k+s$ or more coefficients that are consistent with each other. At least k of these are correct and are from the original set of n. By the previous lemma, we can use these k to find all n of the original coefficients of $m(x)$ and $b(x)$, and the original coefficients are the only ones consistent with these k. Hence all of the $k+s$ or more coefficients in the set must be from the original n coefficients.

Combining these two statements, it must be that there is a unique maximal set of $k+s$ or more coefficients that are consistent with each other, none of these coefficients are in error, and they can be used to reconstruct $m(x)$ and $b(x)$. Q.E.D.

Checking whether the received message $r(x)$ is a multiple of the generator polynomial $g(x)$ by doing polynomial division would be time consuming. To simplify the calculations we can multiply each received message $r(x)$ by some polynomial $h(x)$ and check whether the result is a multiple of $g(x) \cdot h(x)$, hoping that $g(x) \cdot h(x)$ is simple. We will do all of our polynomial arithmetic modulo $x^{p^r-1} - 1$. Below we show that $g(x)$ divides $x^{p^r-1} - 1$. Hence we can choose $h(x)$ to be $h(x) = \frac{x^{p^r}-1}{g(x)}$, and the result $g(x) \cdot h(x)$ is simply 0 modulo $x^{p^r-1} - 1$. We call $h(x)$ the parity check polynomial. Since $g(x)$ has degree $2s$, $h(x)$ has degree $k = n - 2s$.

Lemma 3.3.4 *$g(x)$ divides $x^{p^r-1} - 1$.*

Proof. Since $p^r - 1$ is a finite order of α, $\alpha^{p^r-1} = 1$, therefore $\alpha^{p^r-1} - 1 = 0$ and α is a root of $x^{p^r} - 1$. The same is true for $\alpha^2, \ldots \alpha^{2s-1}$. Also $2s - 1 < n < p^r$ (recall, that we need $n < p^r$ to be able to evaluate $m(x)$ in n different points). Therefore all α^l, where $l \in \{1, \ldots, 2s - 1\}$, are different and $x - \alpha^l$ are independent. Since $x^{p^r} - 1$ is divisible by any $x - \alpha^l$, it is divisible by their product, which is $g(x)$. Q.E.D.

For the purposes of RS algorithms n is chosen equal to $p^r - 1$ (i.e. the popular RS-code—RS(255, 223) has $n = 2^8 - 1$, and the Galois field it uses is GF(2^8)). Below we show how to check whether a received message $r(x)$ is a multiple of $g(x)$, and hence is a codeword.

Lemma 3.3.5 *$r(x) \cdot h(x) = 0 \pmod{x^n - 1}$ if and only if $r(x)$ is a codeword.*

Proof. First, suppose that $r(x)$ is a codeword. Since all codewords are multiples of $g(x) \pmod{x^n - 1}$,

$$r(x) = d(x) \cdot g(x) \pmod{x^n - 1},$$

Also

$$g(x) \cdot h(x) = x^n - 1 = 0 \pmod{x^n - 1}.$$

Hence,

$$r(x) \cdot h(x) = d(x) \cdot g(x) \cdot h(x) = 0 \pmod{x^n - 1}.$$

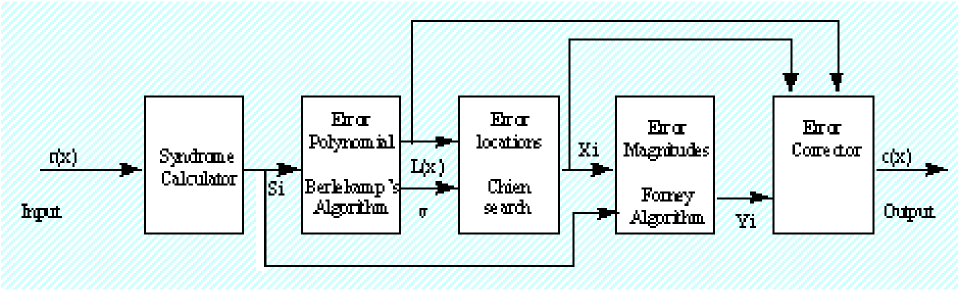

Figure 1: Decoder architecture[1]

Now suppose that $r(x) \cdot h(x) = 0 \pmod{x^n - 1}$. Let $r(x) = d(x) \cdot g(x) + f(x) \pmod{x^n - 1}$, for some $d(x)$ and $f(x)$, where $f(x)$ is the remainder of $r(x)$ after dividing by $g(x)$. Then we have

$$\begin{aligned}
(d(x) \cdot g(x) + f(x)) \cdot h(x) &= 0 \pmod{x^n - 1} \\
d(x) \cdot g(x) \cdot h(x) + f(x) \cdot h(x) &= 0 \pmod{x^n - 1} \\
f(x) \cdot h(x) &= 0 \pmod{x^n - 1} \\
f(x) &= 0
\end{aligned}$$

The third line follows from the second because $g(x) \cdot h(x) = 0 \pmod{x^n - 1}$. The fourth line follows from the third because $h(x)$ is non-zero, $f(x)$ has degree at most $2s - 1$, and $h(x)$ has degree $n - 2s$. Since $f(x)$ is 0, $r(x) = d(x) \cdot g(x) \pmod{x^n - 1}$, and hence $r(x)$ is a multiple of $g(x)$. Thus $r(x)$ is a codeword. Q.E.D.

3.4 Decoder architecture

Now, although we've proven that it's possible to reconstruct the message polynomial $m(x)$ from the message with parity (and at most s errors), and how to check whether a received message is a codeword, we haven't described an efficient algorithm for locating and correcting errors should there be any. Going into detail on the implementation is beyond the scope of this document. However, we can describe *what* the major components of the decoder do, if not *how* they do it.

Figure 1 shows the five major components of the architecture. Let $r(x) = c(x) + e(x)$ be the received data, where $c(x)$ is the transmitted codeword and $e(x)$ is the error polynomial.

[1] Figure from http://www.4i2i.com/reed_solomon_codes.htm.

Syndrome calculator. Calculates $s_i = r(\alpha^i)$, $1 \leq i \leq 2s$. These s_i values are called syndromes.

Berlekamp's algorithm. Finds the error locator polynomial $L(x)$, and the number of errors v. This involves solving s simultaneous equations with s unknowns.

Chien search. Given $L(x)$ and v, finds the roots x_i of $L(x)$.

Forney's algorithm. Given the syndromes and the roots of $L(x)$, finds the symbol error values y_i. Again, this involves solving s simultaneous equations with s unknowns.

Error corrector. Combines all of the pieces calculated above and reconstructs the original message.

4 A guide to the references

The following are to be the textbooks on error correcting codes that everyone cites: [1], [3], [6].

There are also several newer books that describe applications of Reed-Solomon codes: [9], [10].

The following research papers describe Reed-Solomon codes, including the architecture that I outlined at the end of the last lecture. The first in this list is the specific section in Berlekamp's textbook that describes the (now called) "Berlekamp-Massey" algorithm: [1, pp. 145-148], [4], [5], [7], [8].

All of these research papers, and many other important papers on coding theory are included in the following volume: [2].

References

[1] E. R. Berlekamp. *Algebraic Coding Theory.* McGraw Hill, New York, NY, 1968.

[2] E. R. Berlekamp, editor. *Key Papers in the Development of Coding Theory.* IEEE, New York, NY, 1974.

[3] R. E. Blahut. *Theory and Pratice of Error Control Codes.* Addison-Wesley, Reading, MA, 1983.

[4] R. T. Chien. Cyclic decoding procedures for Bose-Chaudhuri-Hocquenghem codes. *IEEE Transactions on Information Theory*, IT-10:357–363, October 1964.

[5] G. D. Forney, Jr. On decoding BCH codes. *IEEE Transactions on Information Theory*, IT-11:549–557, 1965.

[6] S. Liu and Jr. D. J. Costello. *Error Control Coding: Fundamentals and Applications*. Prentice-Hall, Englewood Cliffs, NJ, 1983.

[7] J. L. Massey. Shift register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, pages 122–127, January 1969.

[8] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8:300–304, 1960.

[9] S. B. Wicker and V. K. Bhargava, editors. *Reed-Solomon Codes and Their Applications*. IEEE Press, Piscataway, NJ, 1994.

[10] S.B. Wicker. *Error Control Systems for Digital Communication and Storage*. Prentice-Hall, Englewood Cliffs, NJ, 1995.