Contents

List of Figures

Chapter 1

Basics of Convolutional Coding.

Contents

1.1 Relating G_∞ with the impulse responses $\left\{g_i^j\right\}_{i,j}$

The matrix G_∞ is used to encode the bit-stream (info-bits) entering the convolutional encoder. The input bits are in the format:

$$\left[u_0^0,\ u_0^1, \cdots, u_0^{k-1} | u_1^0,\ u_1^1, \cdots, u_1^{k-1} | \cdots\right] \tag{1.1}$$

and the encoded bits are in the format

$$\left[u_0^0,\ c_0^1, \cdots, c_0^{k-1} | c_1^0,\ c_1^1, \cdots, c_1^{n-1} | \cdots\right]$$

Recalling the relationship

$$c_l^j = \sum_{i=0}^{k-1} u_l^i * g_{i,l}^j \tag{1.2}$$

we make the following clarifications and assumptions:

- Remember that "the encoder has memory of order m" implies that the longest shift-register in the encoder has length m. Therefore the longest impulse response will have memory m which yields $m+1$ taps.
- If some shift registers are shorter than m, we may extend them to have length m assuming that the associated coefficients with those extra taps, are zero (zero padding). That way we ensure that all the impulse responses g_i^j will have $m+1$ taps, without changing their outputs (encoded bits).

Now considering the convolution $u_l^i * g_{i,l}^j$ yields:

$$\begin{aligned} u_l^i * g_{i,l}^j &= \sum_{t=0}^{m} u_{l-t}^i\, g_{i,t}^j \\ &= \left[u_{l-m}^i,\ u_{l-m+1}^i, \cdots, u_l^i\right] \begin{bmatrix} g_{i,m}^j \\ g_{i,m-1}^j \\ \vdots \\ g_{i,0}^j \end{bmatrix} \end{aligned}$$

and therefore the encoded bit c_l^j can be written as:

$$\left[u_{l-m}^0,\ u_{l-m+1}^0, \cdots, u_l^0 | \cdots | u_{l-m}^{k-1},\ u_{l-m+1}^{k-1}, \cdots, u_l^{k-1}\right] \begin{bmatrix} g_{0,m}^j \\ g_{0,m-1}^j \\ \vdots \\ g_{0,0}^j \\ \vdots \\ g_{k-1,m}^j \\ g_{k-1,m-1}^j \\ \vdots \\ g_{k-1,0}^j \end{bmatrix}$$

Now since we want to have the input bits in the interleaved format (1.1), we need to reorder the elements in the above expression, to get:

$$\left[u_{l-m}^0,\ \cdots,u_{l-m}^{k-1}|u_{l-m+1}^0,\cdots,u_{l-m+1}^{k-1}|\cdots|u_l^0,\cdots|u_l^{k-1}\right]\begin{bmatrix} g_{0,m}^j \\ g_{1,m}^j \\ \vdots \\ g_{k-1,m}^j \\ \vdots \\ g_{0,m-1}^j \\ g_{1,m-1}^j \\ \vdots \\ g_{k-1,m-1}^j \\ \vdots \\ g_{0,0}^j \\ g_{1,0}^j \\ \vdots \\ g_{k-1,0}^j \end{bmatrix}$$

This form makes it easy to construct all the n encoded bits $c_l^0,\cdots,c_l^{n-1}$ as:

$$\left[u_{l-m}^0,\ \cdots,u_{l-m}^{k-1}|u_{l-m+1}^0,\cdots,u_{l-m+1}^{k-1}|\cdots|u_l^0,\cdots|u_l^{k-1}\right]\begin{bmatrix} g_{0,m}^0 & g_{0,m}^1 & \cdots & g_{0,m}^{n-1} \\ g_{1,m}^0 & g_{1,m}^1 & \cdots & g_{1,m}^{n-1} \\ \vdots & \vdots & \cdots & \vdots \\ g_{k-1,m}^0 & g_{k-1,m}^1 & \cdots & g_{k-1,m}^{n-1} \\ g_{0,m-1}^0 & g_{0,m-1}^1 & \cdots & g_{0,m-1}^{n-1} \\ g_{1,m-1}^0 & g_{1,m-1}^1 & \cdots & g_{1,m-1}^{n-1} \\ \vdots & \vdots & \cdots & \vdots \\ g_{k-1,m-1}^0 & g_{k-1,m-1}^1 & \cdots & g_{k-1,m-1}^{n-1} \\ \vdots & \vdots & \vdots & \vdots \\ g_{0,0}^0 & g_{0,0}^1 & \cdots & g_{0,0}^{n-1} \\ g_{1,0}^0 & g_{1,0}^1 & \cdots & g_{1,0}^{n-1} \\ \vdots & \vdots & \cdots & \vdots \\ g_{k-1,0}^0 & g_{k-1,0}^1 & \cdots & g_{k-1,0}^{n-1} \end{bmatrix}$$

From the above expression we conclude that $M = m$ and that

$$G_l = \begin{bmatrix} g_{0,l}^0 & g_{0,l}^1 & \cdots & g_{0,l}^{n-1} \\ g_{1,l}^0 & g_{1,l}^1 & \cdots & g_{1,l}^{n-1} \\ \vdots & \vdots & \cdots & \vdots \\ g_{k-1,l}^0 & g_{k-1,l}^1 & \cdots & g_{k-1,l}^{n-1} \end{bmatrix}$$

1.2 Transform Analysis in Convolutional Codes context.

Observing equation (1.2) and since a convolution between two sequences is involved, we can use tools from discrete-time signal processing and apply them to the bit sequences, as if they were binary signals. More specifically we may define a D-transform of a sequence y_i as

$$y(D) = \sum_{i=-\infty}^{+\infty} y_i \, D^i$$

We kept the lower case y for the transformed quantity as we want to keep the upper case letters for matrices. The D-transform is the same as the z-transform substituting z with D^{-1}. All the properties of the well known z transform apply as well to D-transform. Therefore (1.2) is written as

$$\begin{aligned} c^j(D) &= \sum_{i=0}^{k-1} u^i(D) \, g_i^j(D) \\ &= \left[u^0(D), \cdots, u^{k-1}(D)\right] \begin{bmatrix} g_0^j(D) \\ \vdots \\ g_{k-1}^j(D) \end{bmatrix} \end{aligned}$$

We can now express the transformed n output (encoded) bit sequences as:

$$\left[c^0(D), c^1(D) \cdots, c^{n-1}(D)\right] = \begin{bmatrix} g_0^0(D) & g_0^1(D) & \cdots & g_0^{n-1}(D) \\ \vdots & \vdots & \cdots & \vdots \\ g_{k-1}^0(D) & g_{k-1}^1(D) & \cdots & g_{k-1}^{n-1}(D) \end{bmatrix} \Rightarrow$$

$$\mathbf{c}(D) = \mathbf{u}(D)\mathbf{G}(D)$$

In the above expression, $\mathbf{c}(D)$ is a n-dimensional vector, $\mathbf{u}(D)$ is a k-dimensional vector and $\mathbf{G}(D)$ is a $k \times n$ dimensional matrix, providing a very compact and finite dimension expression of the encoding of an infinite stream of info-bits.

1.3 Constructing a convolutional encoder structure from the generator matrices.

We assume we are given a generator matrix for a convolutional codeeither in the form G_∞ or $\mathbf{G}(D)$ and we wish to design a structure using delay elements, that will implement the appropriate encoding process.

Starting from G_∞, we identify the kn impulse responses $\{g_i^j\}$, $i = 0, \cdots, k-1$ and $j = 0, \cdots, n-1$ that are associated with the i-th input and the j-th output. Each of these impulse responses will have (or can be made to have) length $m+1$ (memory m). Let's focus on the j-th output of the encoder. This yields:

$$c^j = \sum_{i=0}^{k-1} u^i * g_i^j$$

From an implementation point of view it yields the structure:

showing that we need a bank of k shift registers, each of m taps (delay elements). The outputs of the i-th shift register are connected with the $g_{i,l}^j, \quad l = 0, \cdots, m$ gains. The outputs of those shift registers are summed together to provide c^j.

Next looking at the output r we have:

$$c^r = \sum_{i=0}^{k-1} u^i * g_i^r$$

Implementing the above, again requires a bank of k shift registers, each of length m. Only now, the outputs of the i-th shiftt register are connected with the $g_{i,l}^r, \quad l = 0, \cdots, m$ gains. The outputs of those shift registers are summed together to provide c^r. Comparing the above two structures we see that the same bank of shift registers may be used for both c^j and c^r (and therefore any other output), provided that outputs of the delay elements are connected with the appropriate set of coefficient from the appropriate impulse response. Therefore although nk impulse responses are required to specify the convolutional encoder, only k shift registers of m taps each, are necessary. This implies that the number of input symbols that are stored at every time instant in the encoder will be km and therefore the number of states of the encoder will be 2^{km}.

1.4 Systematic Convolutional Encoders.

A systematic convolutional encoder is one that reproduces the k input info-bit streams to k out of n of its outputs. For instance we may assume that the first k outputs (c^0, $c^1, \cdots, c^{k-1}$) are

identical to the k input bits $u^0,\ u^1, \cdots, u^{k-1}$. There are many ways that this may happen. For instance:

$$c^j = u^j,\ j = 0, \cdots, k-1$$

is one possibility. But so is

$$c^0 = u^{k-1},\ c^j = u^{j-1} \text{for } j = 1, \cdots, k-2$$

So even when considering the first k output bits, there are several possibilities to have the input bits mapped into the output bits (actually $k!$ possiblities).

The important thing is to have k outputs that reflect exactly what the k input bit streams are. Then it is a matter of reordering the outputs of the encoder such that $u^j = c^j,\ j = 0, \cdots, k-1$.

Since such a reordering of the outputs is always possible we usually refer to a systematic encoder as the one that ensures that $u^j = c^j,\ j = 0, \cdots, k-1$.

From the above definition of a systematic encoder, it follows that k columns of the $\mathbf{G}(D)$ matrix should (after reordering) give the identity matrix.

For example, consider the convolutional encoder with

$$\mathbf{G}(D) = \left[1 + D^2,\ \ 1 + D + D^2\right]$$

An obvious systematic encoder is

$$\mathbf{G}_1(D) = \left[1,\ \ \frac{1 + D + D^2}{1 + D^2}\right]$$

This encoder ensures that $u^0 = c^0$. However the encoder

$$\mathbf{G}_2(D) = \left[\frac{1 + D^2}{1 + D + D^2},\ \ 1\right]$$

ensures that $u^0 = c^1$ which implies that it is also a systematic encoder. If we want to put it in the more standard form where $u^0 = c^0$ we can reorder its outputs by naming the first output c^0 to be the second and the second to be the first. This is achieved by multiplying with the matrix

$$T = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

obtaining

$$\mathbf{G}_3(D) = \mathbf{G}_2(D)T = \left[1,\ \ \frac{1 + D}{1 + D + D^2}\right]$$

This indicates that there may be several systematic encoders (even in the standard form where $u^j = c^j$) for a given convolutional code. Note that although nk polynomials $g_i^j(D)$ must be

specified for a convolutional encoder, only $k(n-k)$ polynomials are required to specify a systematic encoder for the same code.

Note that the second term of this encoder is a rational function of D, implying that a feedforward implementation is impossible.

We now address the problem of physical realization of the encoder, especially when $\mathbf{G}(D)$ contains rational functions of D. To illustrate better our analysis, consider $\mathbf{G}_1(D)$ from the previous example. It implies that:

$$c^0(D) = u^0(D) = \sum_{i=0}^{\infty} u_i \, D^i$$

and

$$c^1(D) = u^0(D)\frac{1+D+D^2}{1+D^2} = u^0(D)\big(1+D+D^3+D^5+\cdots\big)$$

This implies that direct translation into a feedforward circuit (tap delay line) is impossible as an infinite number of delay elements would be required and also an infinite time must be waited before obtaining the encoded output bit.

However if we rewrite it as:

$$c^1(D)(1+D^2) = u^0(D)(1+D+D^2) \Rightarrow$$
$$c_i^1 + c_{i-2}^1 = u_i^0 + u_{i-1}^0 + u_{i-2}^0$$

then by recalling some of the canonical implementations of IIR filters we have the structure of figure 1.4

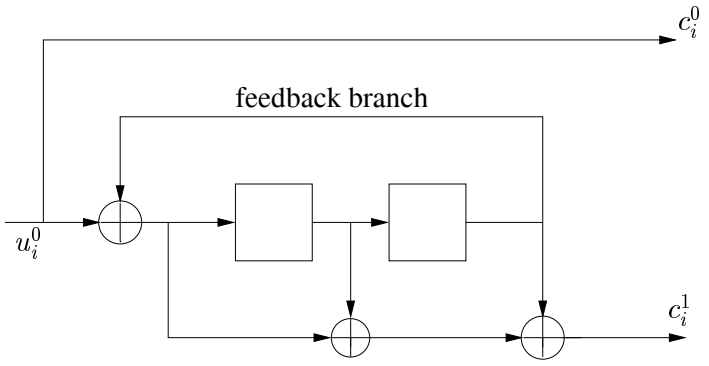

Figure 1.1: Implementation of encoder $\mathbf{G}_1(D)$ using feedback.

It is to be noted that the non-systematic encoder $\mathbf{G}(D)$ can be implemented with the same number of delay elements (2) but without feedback branch, as shown in figure 1.4

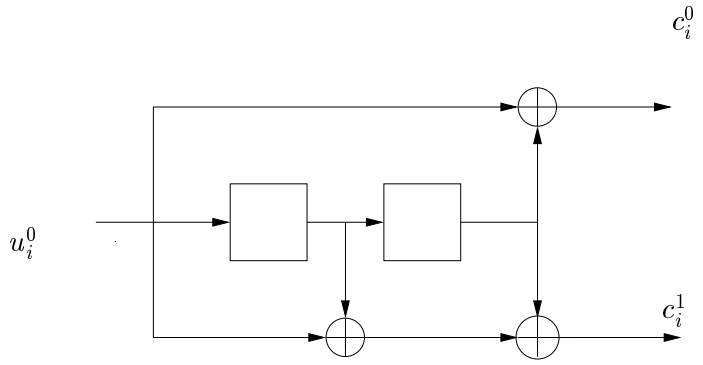

Figure 1.2: Implementation of encoder $\mathbf{G}(D)$.

1.5 Graphical representation of encoders.

The operation of a convolutional encoder can be represented using state diagrams and trellis diagrams. State diagrams give a compact representation of the encoding process, but hide the evolution of the encoding process over time. Trellis diagrams give that extra dimension.

The state of the convolutional encoder is defined as the content of the delay elements of the encoder at a particular time instant. Since a (n, k, m) code has km delay elements, they can store anyone out of 2^{mk} bits. Therefore there will be 2^{mk} possible states. Transition from one state to the other is due to the k incoming info-bits. At time i, k info-bits are entering the encoder, causing it to transition to a new state and at the same time the output of the encoder is n coded bits. Transition from one state to the other is through a branch that is labeled in the form $u_i^0 u_i^1 \cdots u_i^{k-1}/c_i^0 c_i^1 \cdots c_i^{n-1}$.

Since there are 2^k combinations of incoming info-bits, then from any given state we may go to 2^k possible outgoing states. Therefore there will be 2^k branches outgoing from any state. Those branches are entering 2^k states at time $i+1$. Next, we focus on a particular state, and are interested in how many incoming branches exist. We give an explanation which will also provide us with an additional result. Let's imagine the delay elements of the encoder as lying on a $k \times m$ array. We rearrange them (conceptually) such as to form a row of km delay elements. An appropriate way to arrange them is to consider first the k delay elements of the first column of the array. Then consider the next k elements of the second column of the array, e.t.c. until we consider the last k elements of the m-th column of the array, as shown in figure 1.5. At time i the content of those delay elements are $u_{i-1}^0, \cdots, u_{i-1}^{k-1}, u_{i-2}^0, \cdots, u_{i-2}^{k-1}, \cdots, u_{i-m}^0, \cdots, u_{i-m}^{k-1}$. Then at time $i+1$ the encoder will transit to the state $u_i^0, \cdots, u_i^{k-1}, u_{i-1}^0, \cdots, u_{i-1}^{k-1}, \cdots, u_{i-m+1}^0, \cdots, u_{i-m+1}^{k-1}$. Now assume that the encoder at time i was at the state $u_{i-1}^0, \cdots, u_{i-1}^{k-1}, u_{i-2}^0, \cdots, u_{i-2}^{k-1}, \cdots, \tilde{u}_{i-m}^0, \cdots, \tilde{u}_{i-m}^{k-1}$. Then

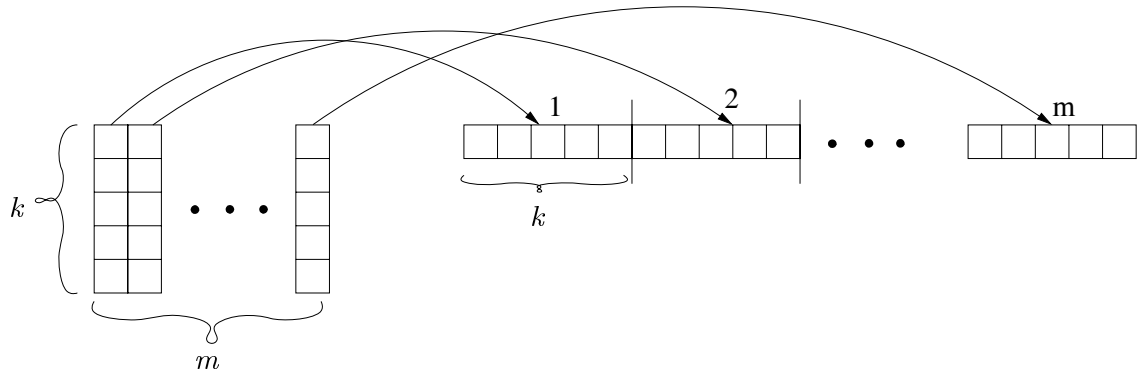

Figure 1.3:

if at time i the info-bits $u_i^0, \cdots, u_i^{k-1}$ are entering the encoder, the state at which it will transit to will be $u_i^0, \cdots, u_i^{k-1}, u_{i-1}^0, \cdots, u_{i-1}^{k-1}, \cdots, u_{i-m+1}^0, \cdots, u_{i-m+1}^{k-1}$, i.e. the same state as before. Therefore as long as only the last k bits of the state at time i are different, the encoder will always transit to the same state at time $i+1$, upon encoding of the info bits $u_i^0, \cdots, u_i^{k-1}$. Since there are 2^k states differing only the last k bits, it follows that these states will all transit to the same state upon encoding of the same k info bits. This shows that there are 2^k incoming branches to any state and that all these branches are labeled with the same k info-bits. At the same time, the ordering of the delay elements of the convolutional encoder that is suggested in the right part of figure 1.5 provides a good convention for naming the states of the system. We will use the convention σ_ζ to denote the state when the content of the delay elements in the form $u_{i-1}^0, \cdots, u_{i-1}^{k-1}, u_{i-2}^0, \cdots, u_{i-2}^{k-1}, \cdots, \tilde{u}_{i-m}^0, \cdots, \tilde{u}_{i-m}^{k-1}$ equals the integer ζ. For example for the $(3,2,2)$ encoder the state σ_3 means that the content of the delay elements is (0011) or in the array ordering of the delay elements it corresponds to $\begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}$. With this in mind, the general form of the labeling we will use for both the state transition diagram and the trellis diagram will have the format of figure 1.5.

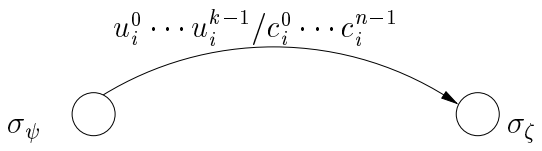

Figure 1.4: Labeling of branches and states

Example: Using the encoder $\mathbf{G}(D)$ yields: We now construct the trellis diagram for the same encoder. We can do it based on the state transition diagram of figure 1.5, as follows:

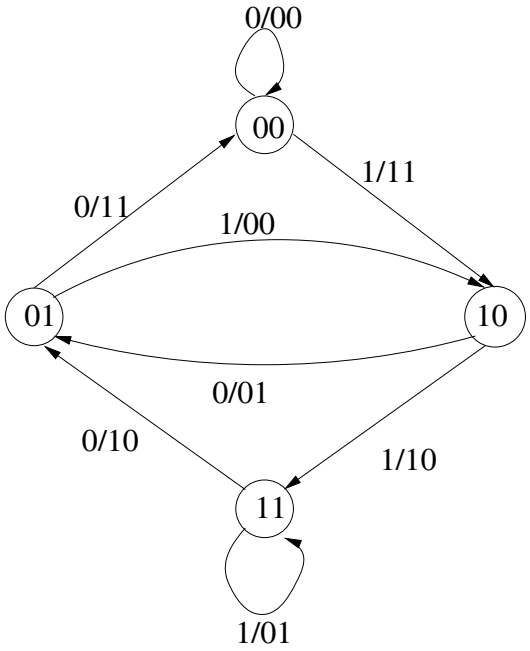

Figure 1.5: State transition diagram for encoder $\mathbf{G}(D)$.

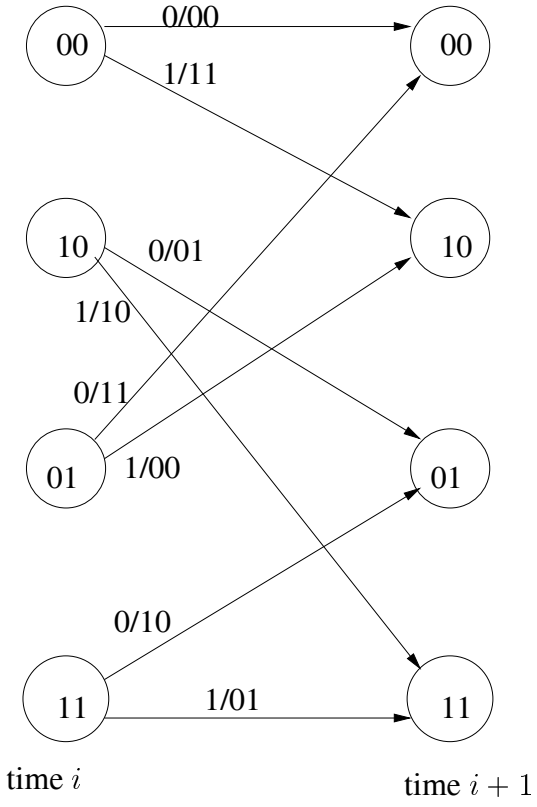

Figure 1.6: Trellis diagram for encoder $\mathbf{G}(D)$.

1.6 Free Distance of convolutional codes.

The free distance of a convolutional code (not an encoder), is defined as the minimum Hamming weigth of any non-zero codeword of that code. In order to find it, we can use the trellis diagram and start from the all zero state. We will consider paths that diverge from the zero state and merge back to the zero state, with out intermediate passes through the zero state. For the previous code defined by $\mathbf{G}(D)$ we can show that it's minimum distance is 5. Later when we will study the Viterbi algorithm, we will see an algorithmic way to determine the free distance of a convolutional code.

1.7 Catastrophic Encoders.

First of all we want to clarify that there is no such thing as catastrophic convolutional code. However there are catastrophic convolutional encoders and here is what we mean:

Consider the case where the encoder $\mathbf{G}(D)$ is used to encode an info bit stream $\mathbf{u}^0(D) = 1$ (corresponds to the transmission of a Dirack, $\mathbf{u}^0 = [1, 0, 0, \cdots]$). The codeword at the output of the encoder is $\mathbf{c}(D) = [1+D^2,\ 1+D+D^2]$ (corresponds to the sequence $[1, 1, 0, 1, 1, 1, 0, 0, 0, \cdots]$. Now assume that we use the equivalent encoder $\tilde{\mathbf{G}}(D) = (1 + D)\mathbf{G}(D) = [1 + D + D^2 + D^3,\ 1 + D^3]$ and that the input bit stream is $\mathbf{u}^0(D) = \frac{1}{1+D}$, (corresponds to the sequence $\tilde{\mathbf{u}}^0 = [1, 1, 1, 1, 1 \cdots]$). However the output of the encoder is the same as before, since:

$$\begin{aligned}
\mathbf{c}(\tilde{D}) &= \tilde{\mathbf{u}}^0 \frac{1}{1+D} \mathbf{G}(\tilde{D}) \\
&= \frac{1}{1+D}(1+D)\mathbf{G}(D) \\
&= \mathbf{u}^0(D)\mathbf{G}(D) \\
&= \mathbf{c}(D)
\end{aligned}$$

So far we don't see anything strange. However, coding is used to protect information from errors, and assume that through the transmission of the encoded bit stream through a noisy channel, we receive the sequence

$$\mathbf{r} = [1\underline{0}|0\underline{0}|\underline{0}1|00|00|\cdots]$$

where the underbar denotes a bit that has been affected by the noisy channel (flipped). The above received sequence suggests that 3 errors were performed during reception.

A MAP detector, upon reception of $\mathbf{r}$ would produce as closest match, the sequence $\hat{\mathbf{c}} = [000000\cdots]$ that is associated with the info bit stream $\hat{\mathbf{u}}^0(D) = 0$ (or $\mathbf{u}^0 = [0000\cdots]$). The

convolutional decoder knows that $\mathbf{r}$ cannot be a codeword, because $d_{\text{free}} = 5$. Since $\mathbf{r}$ has a Hamming weigth of 2, the all zero code word will be selected as the most likely transmitted codeword. If $\mathbf{G}(D)$ was used as the encoder, we see that the detected info word ($[0000\cdots]$) and the transmitted info word ($[1,0,0,0,\cdots]$) differ in only 1 position. However if the encoder $\tilde{\mathbf{G}}(D)$ was used, the transmitted info word was $[1111\cdots]$ and therefore it will differ from the detected info word in an infinite number of positions, althoug only a finite number of channel errors (3) occured. This is highly undesirable in transmission systems and the encoder $\tilde{\mathbf{G}}(D)$ is said to be catastrophic.

Catastrophic encoders are to be avoided and several criteria have been developped in order to identify them. A very popular one is by Massey and Sain and provides a sufficient and necessary condition for an encoder to be non-catastrophic.

THEOREM 1

If $\mathbf{G}(D)$ is a rational generator matrix for a convolutional code, then the following conditions are equivalent. If $\mathbf{G}(D)$ satisfies any of these (it will therefore satisfy all others), the encoder will be non-catastrophic.

1. *No infinite Hamming weight input $\mathbf{u}(D)$ produces a finite Hamming weight output $\mathbf{c}(D)$ (this is equivalent to: All infinite Hamming weight input sequences, produce infinite Hamming weight codewords).*

2. *Let $\beta(D)$ be the least common multiple of the denominators in the entries of $\mathbf{G}(D)$. Define $\tilde{\mathbf{G}}(D) = \beta(D)\mathbf{G}(D)$ (i.e. clear the denominators. Transform all entries to polynomials). Let $\alpha(D)$ be the greatest common divisor of all $k \times k$ minors of $\tilde{\mathbf{G}}(D)$. When $\frac{\alpha(D)}{\beta(D)}$ is reduced to lowest terms (no more common factors) yielding $\frac{\tilde{\alpha}(D)}{\tilde{\beta}(D)}$, then $\tilde{\alpha}(D) = D^j$ for some $j \geq 0$. (Note that if $\mathbf{G}(D)$ is a polynomial matrix, then $\beta(D) = 1$ since all denominators equal 1.*

3. *$\mathbf{G}(D)$ has a polynomial[1] right inverse $\mathbf{K}(D)$ of finite weight, i.e. $\mathbf{G}(D)\mathbf{K}(D) = I_k$ (identity $k \times k$ matrix).*

EXAMPLE 1

Consider the encoder $\tilde{\mathbf{G}}(D) = [1 + D + D^2 + D^3,\ 1 + D^3]$ which has polynomial entries. Therefore $\beta(D) = 1$. The greatest common divisor of all minors of $\tilde{\mathbf{G}}(D)$ is $\alpha(D) = \gcd\{1 + D + D^2 + D^3,\ 1 + D^3\} = 1 + D$. This $\alpha(D)$ is not in the form D^j, violating condition 2, implying that the encoder is catastrophic. $\diamondsuit$

COROLLARY 1

Every systematic generator matrix of a convolutional code, is non-catastrophic.

PROOF

Descriptive proof:

In a systematic encoder, the input sequence somehow appears in the codeword. Therefore any infinite weigth input will necessary produce an infinite weigth output, satisfying condition 1.

Formal proof:

If $\mathbf{G}(D) = [I_k,\ P(D)]$ then $K(D) = \begin{bmatrix} I \\ 0 \end{bmatrix}$ is a right inverse of $\mathbf{G}(D)$. Thus by condition 3 $\Rightarrow \mathbf{G}(D)$ is non-catastrophic. ■

1.8 Octal notation of convolutional encoders.

Polynomial generator matrices of convolutional encoders have entries that are polynomials of D. Let $p(D)$ be an entry of $\mathbf{G}(D)$, in the form $p_0 + p_1 D + p_2 D^2 + \cdots, p_m D^m$. Then we form the binary number $p_m \cdots p_2 p_1 p_0$ and zero pad it on the right such that the total number of digits is an integer multiple of 3. Then to every 3 digits we associate an octal number.

EXAMPLE 2

The generator matrix $\mathbf{G}(D) = [1 + D + D^3 + D^4 + D^6,\ 1 + D^3 + D^4 + D^5 + D^6]$ *is expressed as* $[554,\ 744]$ *since*

$$1 + D + D^3 + D^4 + D^6 \rightarrow \underbrace{101}_{5}\,\underbrace{101}_{5}\,\underbrace{1\quad 00}_{\underbrace{\text{zero-padding}}_{4}}$$

$$1 + D^3 + D^4 + D^5 + D^6 \rightarrow \underbrace{111}_{7}\,\underbrace{100}_{4}\,\underbrace{1\quad 00}_{\underbrace{\text{zero-padding}}_{4}}$$

♢

1.9 Physical realizations of convolutional encoders.

In general the entries of a convolutional encoder $\mathbf{G}(D)$ will be rational functions in the form $p(D)/q(D)$ where $p(D)$ and $q(D)$ are polynomials of the same order L (if they don't have the same order, we complete the order defficient polynomial with appropriate zero coefficients, such as to have the same order L). More specifically we have:

$$p(D) = \sum_{l=0}^{L} p_l D^l$$

and

$$q(D) = \sum_{l=0}^{L} q_l D^l$$

This rational entry corresponds to the transfer function of a linear time invariant (LTI) system and therefore can be implemented in many possible ways. Of particular interest are two implementations: the controller canonical form and the observer canonical form. If $x(D)$ is the input and $y(D)$ the output of the LTI, then

$$y(D) = x(D)\frac{p(D)}{q(D)} \Rightarrow$$

$$y_j = \frac{1}{q_0}\left[\sum_{l=0}^{L} p_l\, x_{j-l} - \sum_{l=1}^{L} q_l\, y_{j-l}\right]$$

The controller canonical form is illustrated in figure 1.9. This implementation has the adders

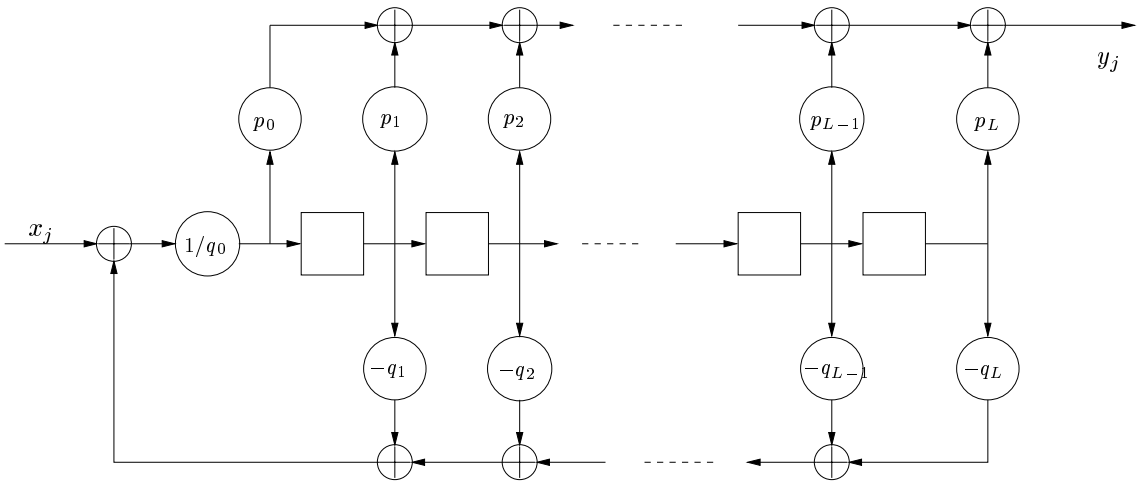

Figure 1.7: Controller canonical form for $p(D)/q(D)$.

out of the delay elements path. A dual implementation of the above that has the adders in the delay elements path is known as observer canonical form and is illustrated in figure 1.9.

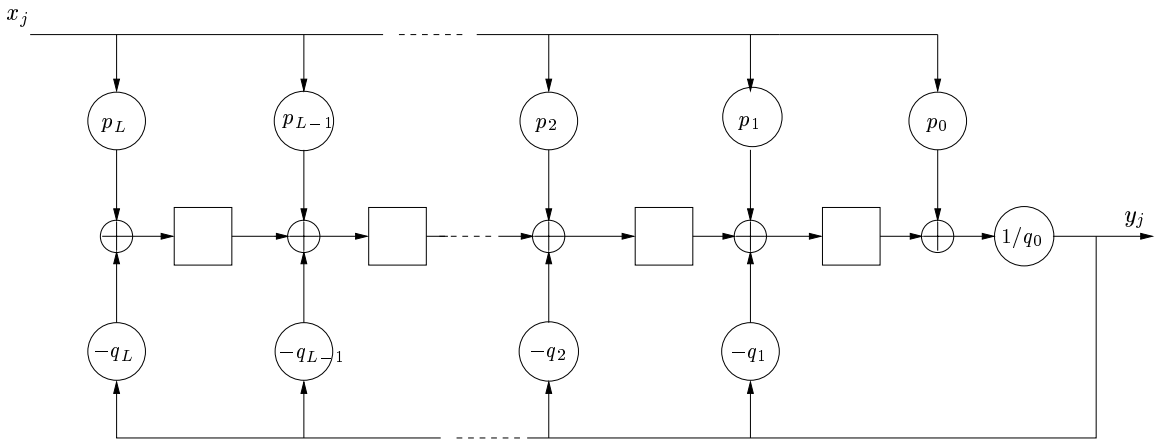

Figure 1.8: Observer canonical form for $p(D)/q(D)$.

DEFINITION 1

The McMillan degree of $\mathbf{G}(D)$ is the minimum number of delay elements that you can find in any realization of $\mathbf{G}(D)$.

DEFINITION 2

An encoder $\mathbf{G}(D)$ for a code $\mathcal{C}$ is called minimal if it's McMillan degree is the smallest that is achievable with any equivalent encoder for $\mathcal{C}$.

This is equivalent to say that if we consider all possible equivalent encoders of $\mathcal{C}$ and find the minimum of their McMillan degrees, then the encoder with that minimal McMillan degree will be a minimal encoder for $\mathcal{C}$.

DEFINITION 3

The degree of $\mathcal{C}$ is the degree of a minimal encoder of $\mathcal{C}$.

Question:

Given an encoder, how can we identify if it is minimal? If it is not, how can we find a minimal encoder?

Some answers: (many more details in pp.97-113 of Schlegel's book)

A realizable encoder $\mathbf{G}(D)$ is minimal iff $\mathbf{G}(D)$ has a polynomial right inverse $K_1(D)$ and an antipolynomial right inverse $K_2(D)$

COROLLARY 2

Any systematic encoder is minimal

PROOF

A systematic encoder has the form $\mathbf{G}(D) = [I_k,\ P(D)]$. Then $K(D) = \begin{bmatrix} I \\ 0 \end{bmatrix}$ is a right inverse of $\mathbf{G}(D)$ and is both polynomial and antipolynomial. ■

1.10 Input Output Weight enumerating function of a convolutional code.

It is often of interest to compute the distribution of the Hamming weights of all possible codewords of a convolutional code. What we actually have in mind is a polynomial (with eventually infinite number of terms) of the form

$$A(X) = \sum_d A_d X^d \tag{1.3}$$

where d is the Hamming weight of a codeword and A_d is the number of codewords of weight d. For instance since the encoder $\mathbf{G}(D) = [1 + D^2,\ 1 + D + D^2]$ has free distance 5, it follows that the first term of $A(X)$ will be $1X^5$ since $\mathbf{G}(D)$ has only one codeword of weight 5 and no codewords of smaller weight (except the all zeros sequence).

Equation (1.3) is refered as the Weight Enumerating Function (WEF) of the convolutional code. Based on the state transition diagram, there is an algorithm from control system's theory, known as Mason's rule, that provides a systematic way to compute the WEF. We describe the steps and illustrate them with an example.

1. We split the all zero state into 2 states labeled S_0, one been considered as the input and the other as the output of the signal flow graph and we neglect the zero weight branch looping around S_0.

2. We label all branches of the state diagram with the label X^w, where w is the Hamming weight of that branch.

3. We consider the set of forward paths $\mathcal{S}_0$ that initiate from state S_0 and terminate at S_0 without passing from any state more than one time. The gain of each forward path is denoted as $F_i,\ i \in \mathcal{S}_0$

4. We consider the set of cycles (loops). A cycle is a path that starts from any state and terminates to that state, without passing through any other state more than one time. We denote $\mathcal{S}_1$ the set of all cycles and $C_i^{(1)}$ the gain of the i-th cycle.

5. We consider the set of pairs of non-touching cycles. Two cycles are non touching if they don't have any states in common. We denote as $\mathcal{S}_2$ the set of all non touching pairs of cycles. Assuming that the cycles i and j are non touching, the gain of the pair $(i,j) \in \mathcal{S}_2$ is defined as $C_{i,j}^{(2)} = C_i^{(1)} C_j^{(1)}$.

6. Similarly we consider the set of triplets of non-touching cycles and define the gain of the triplet (i,j,m) as $C_{i,j,m}^{(3)} = C_i^{(1)} C_j^{(1)} C_m^{(1)}$, and so on for quadruplets e.t.c. .

7. Then

$$\boxed{A(X) = \frac{\sum_i F_i\, \Delta_i}{\Delta}} \tag{1.4}$$

where

$$\Delta = 1 - \sum_i C_i^{(1)} + \sum_{(i,j)} C_{i,j}^{(2)} - \sum_{(i,j,m)} C_{i,j,m}^{(3)} + \cdots$$

and Δ_i is defined as Δ but for the portion of the graph that remains after we erase the states associated with the i-th forward path, and the branches originating from or merging to those states.

EXAMPLE 3

Consider the encoder $\mathbf{G}(D) = [1 + D^2 + D^3,\ 1 + D + D^2 + D^3]$. *The state transition diagram is shown in figure 3. Splitting state* S_0 *and neglecting the zero weight branch looping around it yields the following graph: Examining the above state diagram we found that:*

- *There are 7 forward paths:*
  - *Path 1:* $S_0 S_1 S_3 S_7 S_6 S_5 S_2 S_4 S_0$ *with gain* $F_1 = X^{12}$
  - *Path 2:* $S_0 S_1 S_3 S_7 S_6 S_4 S_0$ *with gain* $F_2 = X^7$
  - *Path 3:* $S_0 S_1 S_3 S_6 S_5 S_2 S_4 S_0$ *with gain* $F_3 = X^{11}$
  - *Path 4:* $S_0 S_1 S_3 S_6 S_4 S_0$ *with gain* $F_4 = X^6$
  - *Path 5:* $S_0 S_1 S_2 S_5 S_3 S_7 S_6 S_4 S_0$ *with gain* $F_5 = X^8$

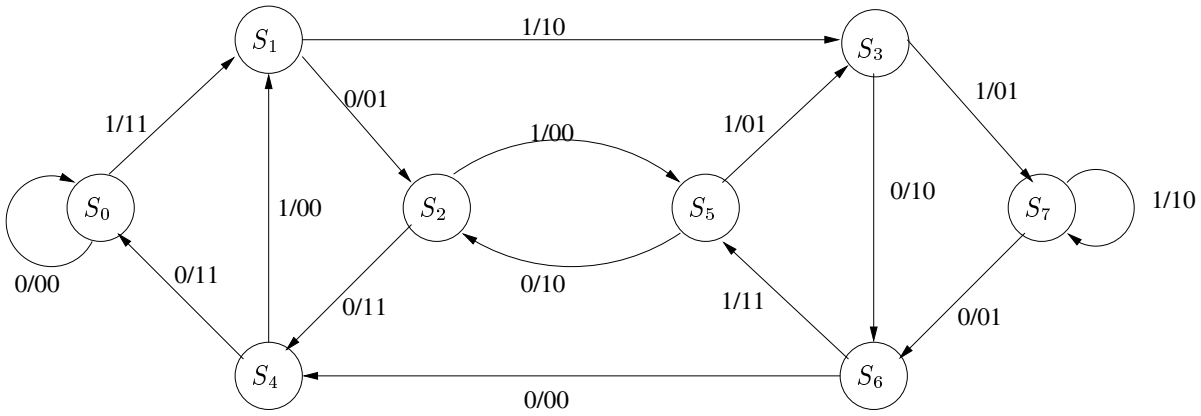

Figure 1.9: State transition diagram for $\mathbf{G}(D) = [1 + D^2 + D^3,\ 1 + D + D^2 + D^3]$.

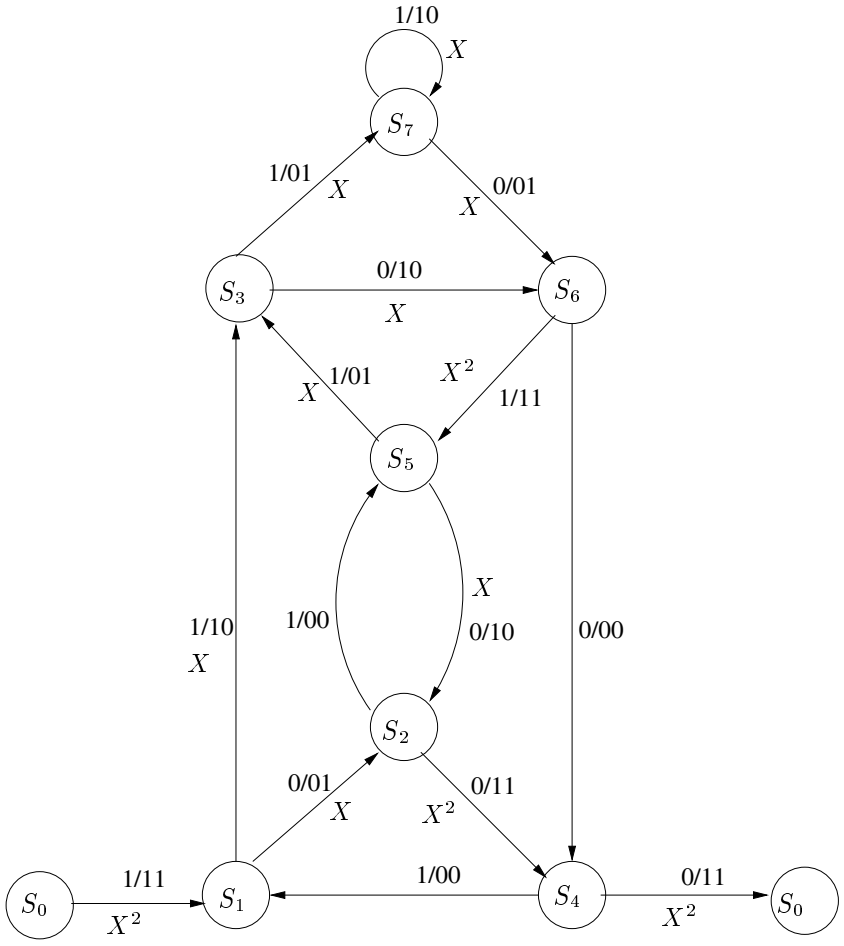

Figure 1.10: Modified state transition diagram for $\mathbf{G}(D) = [1 + D^2 + D^3,\ 1 + D + D^2 + D^3]$.

  - *Path 6:* $S_0 S_1 S_2 S_5 S_3 S_6 S_4 S_0$ *with gain* $F_6 = X^7$
  - *Path 7:* $S_0 S_1 S_2 S_4 S_0$ *with gain* $F_7 = X^7$

- *There are 11 cycles:*
  - *Cycle 1:* $S_1 S_3 S_7 S_6 S_5 S_2 S_4 S_1$ *with gain* $C_1^{(1)} = X^8$.
  - *Cycle 2:* $S_1 S_3 S_7 S_6 S_4 S_1$ *with gain* $C_2^{(1)} = X^3$.
  - *Cycle 3:* $S_1 S_3 S_6 S_5 S_2 S_4 S_1$ *with gain* $C_3^{(1)} = X^7$.
  - *Cycle 4:* $S_1 S_3 S_6 S_4 S_1$ *with gain* $C_4^{(1)} = X^2$.
  - *Cycle 5:* $S_1 S_2 S_5 S_3 S_7 S_6 S_4 S_1$ *with gain* $C_5^{(1)} = X^4$.
  - *Cycle 6:* $S_1 S_2 S_5 S_3 S_6 S_4 S_1$ *with gain* $C_6^{(1)} = X^3$.
  - *Cycle 7:* $S_1 S_2 S_4 S_1$ *with gain* $C_7^{(1)} = X^3$.
  - *Cycle 8:* $S_2 S_5 S_2$ *with gain* $C_8^{(1)} = X$.
  - *Cycle 9:* $S_3 S_7 S_6 S_5 S_3$ *with gain* $C_9^{(1)} = X^5$.
  - *Cycle 10:* $S_3 S_6 S_5 S_3$ *with gain* $C_{10}^{(1)} = X^4$.
  - *Cycle 11:* $S_7 S_7$ *with gain* $C_{11}^{(1)} = X$.

- *There are 10 pairs of nontouching cycles:*
  - *(2,8) with gain* $C_{2,8}^{(2)} = X^4$
  - *(3,11) with gain* $C_{3,11}^{(2)} = X^8$
  - *(4,8) with gain* $C_{4,8}^{(2)} = X^3$
  - *(4,11) with gain* $C_{4,11}^{(2)} = X^3$
  - *(6,11) with gain* $C_{6,11}^{(2)} = X^4$
  - *(7,9) with gain* $C_{7,9}^{(2)} = X^8$
  - *(7,10) with gain* $C_{7,10}^{(2)} = X^7$
  - *(7,11) with gain* $C_{7,11}^{(2)} = X^4$
  - *(8,11) with gain* $C_{8,11}^{(2)} = X^2$
  - *(10,11) with gain* $C_{10,11}^{(2)} = X^5$

- *Finally there are 2 triplets of nontouching cycles:*

- *(4,8,11) with gain* $C^{(3)}_{4,8,11} = X^4$
- *(7,10,11) with gain* $C^{(3)}_{7,10,11} = X^8$

Computation of Δ *yields:*

$$\begin{aligned}\Delta &= 1 - 2X - X^3\\ \Delta_1 = \Delta_5 &= 1\\ \Delta_3 = \Delta_6 &= 1 - X\\ \Delta_2 &= 1 - X\\ \Delta_4 &= 1 - 2X + X^2\\ \Delta_7 &= 1 - X - X^4\end{aligned}$$

Application to (1.4) yields:

$$\begin{aligned}A(X) &= \frac{X^6 + X^7 - X^8}{1 - 2X - X^3}\\ &= X^6 + 3X^7 + 5X^8 + 11X^9 + 25X^{10} + \cdots\end{aligned} \tag{1.5}$$

Therefore the convolutional code described by $\mathbf{G}(D)$ *has* $d_{\text{free}} = 6$ *and there is only one path achieving it. In addition there are 3 paths of weight 7, 5 paths of weight 8, 11 paths of weigth 9 e.t.c.* $\diamondsuit$

The above example focused on computing the WEF of a convolutional code. In several occasions, additional information about the structure of the convolutional code is required. For instance we may be interested on the weight of the info-bit stream that produces the codeword of weight 7 and of how many branches (successive jumps from a sequence to another) it consists of. This yields to a generalization of the previously described Mason rule as follows: We want to compute the function $A(W, X, L) = \sum_{w,d,l} A_{w,d,l} W^w X^d L^l$ called the Input Output Weigth Enumerating Function (IOWEF), where $A_{w,d,l}$ is the number of codewords with input weight w, output weight d and consist of l branches. The only thing to change in the already described Mason's rule is the labeling of the branches in the state transition diagram. We now label each branch as $W^w X^d L^l$ instead of just X^d previously. Those being the gains of each branch, we

apply the same technique to compute $\Delta,\ \Delta_i$ e.t.c. . Application to example 3 yields:

$$\begin{aligned} A(W,X,L) = & \frac{X^6W^2L^5 + X^7WL^4 - X^8W^2L^5}{1 - XW(L+L^2) - X^2W^2(L^4 - L^3) - X^3WL^3 - X^4W^2(L^3 - L^4)} \\ = & \quad X^6W^2L^5 + X^7(WL^4 + W^3L^6 + W^3L^7) + \\ & \quad X^8(W^2L^6 + W^4L^7 + W^4L^8 + 2W^4L^9) + \cdots \end{aligned} \tag{1.6}$$

Comparing (1.5) and (1.6) we see that the convolutional code of example 3 has 5 codewords of weight 8 out of which 1 has input weight 2 and consists of 6 branches, 1 has input weight 4 and consists of 7 branches, 1 has input weight 4 and consists of 8 branches and 2 have input weigth 4 and consist of 9 branches. Therefore we have a much more detailled information about the structure of that code.

1.11 Maximum Likelihood Decoding of Convolutional Codes.

We assume that a stream of info bits $u = \left\{[u_i^0,\ u_i^1, \cdots, u_i^{k-1}]\right\}_i$ are encoded and produce the codeword $c = \left\{[c_i^0,\ c_i^1, \cdots, c_i^{n-1}]\right\}_i$. The codeword is transmitted through a Discrete Memoryless Channel (DMC) and the sequence $r = \left\{[r_i^0,\ r_i^1, \cdots, r_i^{n-1}]\right\}_i$ is received at the demodulator's output.

The Maximum Aposteriori Probability (MAP) criterion for detection (decoding) of convolutional codes implies that the codeword $\hat{c} = \left\{[\hat{c}_i^0,\ \hat{c}_i^1, \cdots, \hat{c}_i^{n-1}]\right\}_i$ is detected, iff:

$$Pr(\hat{c}|r) \geq Pr(\tilde{c}|r)$$

for all possible sequences $\tilde{c} \neq \hat{c}$. In other words, $Pr(\hat{c}|r)$ is the maximum among all other probabilities $Pr(\tilde{c}|r)$. Then using Bayes rule it follows that:

$$\begin{aligned} \frac{Pr(r|\hat{c})\,Pr(\hat{c})}{Pr(r)} &\geq \frac{Pr(r|\tilde{c})\,Pr(\tilde{c})}{Pr(r)} \Rightarrow \\ Pr(r|\hat{c})\,Pr(\hat{c}) &\geq Pr(r|\tilde{c})\,Pr(\tilde{c}) \Rightarrow \end{aligned}$$

and assuming that all codewords are equally likely, then

$$Pr(r|\hat{c}) \geq Pr(r|\tilde{c})$$