Chapter 9

Description and Properties of Convolutional Codes

Together with block codes, convolutional codes form the second large class of codes used for error-control coding. Convolutional and block codes could be formally seen as identical from a certain point of view, but their descriptions, their properties, their decoding methods and their applications are very different. The important differences between the two classes of codes are as follows:

- Convolutional encoders do not transform information words into codewords block by block, but transform a whole sequence of information bits into a sequence of encoded bits by convolving the information bits with a set of generator coefficients.
- Convolutional codes are not constructed by analytical methods but by trial and error methods (i.e., computer search).
- Primarily, only few very simple convolutional codes are of practical interest. These are much more intelligible and easier to describe than block codes.
- Convolutional decoders can easily process soft-decision input (i.e., information on the reliability of the demodulator output) and compute soft-decision output (i.e., information on the reliability of the estimated information bits).
- In contrast to block codes, convolutional codes do not require a block synchronization.

Since soft-decision information can be processed without further expense and leads to the gains described in Section 1.7, convolutional codes should always be used in connection with soft-decision demodulators. So convolutional codes should not really be called error-correcting codes but *transmission codes*. Thus the determination of bounds for the correction or detection of single errors or burst errors is unnecessary – a strong contrast to block codes.

In this chapter, we will establish various algebraic and non-algebraic descriptions for convolutional codes and, in particular, a distance structure by an analytical approach. The main emphasis of the next chapter will be on maximum-likelihood (ML) decoding with the Viterbi algorithm, the calculation of the error probability and the performance of convolutional codes. In Chapter 11, convolutional codes are used as the basis for trellis coded modulation.

9.1 Linear Encoders and Shift Registers

For convolutional codes, the information symbols u and the encoded symbols a are usually not $q = p^m$-ary, but simply binary: $u, a \in \mathbb{F}_2 = \{0, 1\}$. All arithmetic operations are performed as modulo 2. Similar as for the fundamental principle of block encoding in Figure 1.7, the sequences of information bits and encoded bits are divided into small blocks of lengths k and n, respectively, which are now indexed by r:

$$\begin{aligned} \boldsymbol{u}_r &= (u_{r,1}, \ldots, u_{r,k}) \\ \boldsymbol{a}_r &= (a_{r,1}, \ldots, a_{r,n}). \end{aligned}$$

The assignment of the encoded n-tuples to the information k-tuples is unique and reversible as well as time-invariant but in contrast to block codes not memoryless:

Definition 9.1. *A convolutional code of rate $R = k/n$ is given by an encoder with memory as follows: the current code block depends on the current information block and the m previous information blocks*

$$\boldsymbol{a}_r = \mathit{encoder}(\boldsymbol{u}_r, \boldsymbol{u}_{r-1}, \ldots, \boldsymbol{u}_{r-m}). \tag{9.1.1}$$

The encoder is a linear shift register, i.e., the encoded bits are linear combinations of the information bits. For the formal description we introduce generator coefficients $g_{\kappa,\nu,\mu} \in \mathbb{F}_2$ with $1 \le \kappa \le k$, $1 \le \nu \le n$ and $0 \le \mu \le m$, so that the n sequences of encoded bits are convolutions of the k information bit sequences with the generator coefficients

$$a_{r,\nu} = \sum_{\mu=0}^{m} \sum_{\kappa=1}^{k} g_{\kappa,\nu,\mu} u_{r-\mu,\kappa}. \tag{9.1.2}$$

The variable m is called memory length *and $m+1$ is called* constraint length.

The parameter m not only influences the code rate but also the performance of the code and the complexity of the decoding. The constraint length is often denoted k or K, however, this might cause confusion with the information block length. Formally, block codes are special convolutional codes with $m = 0$.

For block codes, k and n are usually large but most of the convolutional codes used in practice have $k = 1$ (i.e., one information block consists of one information bit) and $n = 2$ or $n = 3$. Therefore the block structure is uninteresting and non-existent for the information bits. Instead, sequences are mapped to sequences. Nowadays, the memory length for ML decoding is technically limited to $m = 6$ or $m = 8$ (also called *short-memory codes*), except when using extremely complex decoders for special applications (see Section 12.1). However, with these relatively simple convolutional codes we can achieve coding gains comparable to those of complicated block codes.

Usually, a convolutional code is determined by the encoder which is implemented as a linear shift register. The shift register contains $\boldsymbol{u}_r, \boldsymbol{u}_{r-1}, \ldots, \boldsymbol{u}_{r-m}$. Of these $k(m+1)$ bits n linear combinations are computed which form the components of $\boldsymbol{a}_r$. Then $\boldsymbol{u}_{r+1}$ is inserted and $\boldsymbol{u}_{r-m}$ is pushed out. To begin with at $r = 0$ the shift register is loaded with zeros, i.e., $\boldsymbol{u}_{-1} = \cdots = \boldsymbol{u}_{-m} = \boldsymbol{0}$.

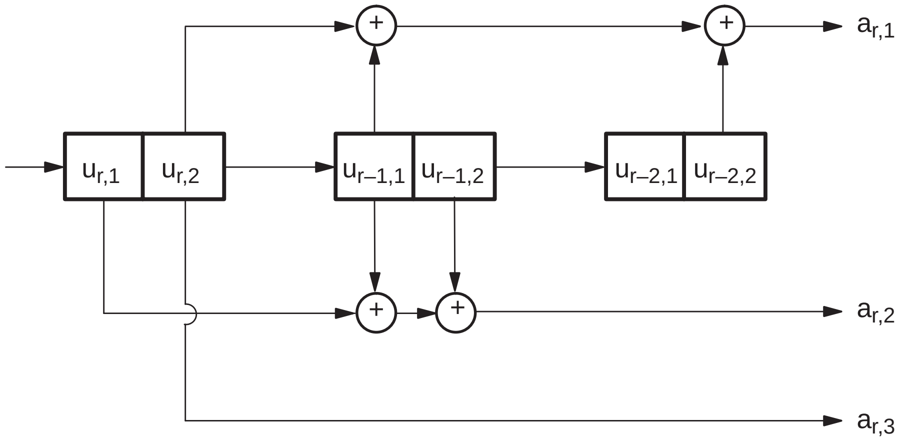


Figure 9.1. A rate-2/3 convolutional encoder with $m = 2$

Example 9.1. Consider a rate-2/3 code with $m = 2$, where the encoder is defined by the shift register in Figure 9.1. In step r, the $k = 2$ information bits $\boldsymbol{u}_r = (u_{r,1}, u_{r,2})$ are inserted and from the content $(\boldsymbol{u}_r, \boldsymbol{u}_{r-1}, \boldsymbol{u}_{r-2})$ of the shift register, $\boldsymbol{a}_r = (a_{r,1}, a_{r,2}, a_{r,3})$ is computed according to

$$\begin{aligned} a_{r,1} &= u_{r,2} + u_{r-1,1} + u_{r-2,2} \\ a_{r,2} &= u_{r,1} + u_{r-1,1} + u_{r-1,2} \\ a_{r,3} &= u_{r,2}. \end{aligned}$$

Two information bits are always mapped to 3 encoded bits and therefore $R = 2/3$ for the code rate. ■

Example 9.2 (standard example). In the following and very often throughout Chapters 9 and 10, the rate-1/2 code with $m = 2$ and the encoder shown in Figure 9.2 will be used and referred to as the *standard example*. In step r the information bit u_r is inserted into the shift register and with this $\boldsymbol{a}_r = (a_{r,1}, a_{r,2}) = (u_r + u_{r-1} + u_{r-2}, u_r + u_{r-2})$ is computed. For the information bit sequence

$$(u_0, u_1, u_2, u_3, u_4, u_5) = (1, 1, 0, 1, 0, 0)$$

the corresponding encoded bit sequence is

$$(a_0, a_1, a_2, a_3, a_4, a_5) = (11, 01, 01, 00, 10, 11).$$

This simple code is actually useful in practice. Later, we will see that the asymptotic coding gain $G_{\text{a,soft}}$ is nearly 4 dB. ■

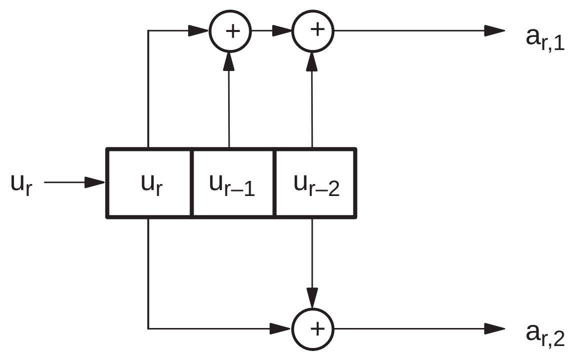


Figure 9.2. A rate-1/2 convolutional encoder with $m = 2$ (standard example)

Convolutional codes are defined by their encoding schemes which are easy to understand. However, the code created (i.e., the set of all encoded bit sequences) is not as easy to analyze. In comparison, for the block codes first the code is determined by using algebraic methods and then a suitable systematic encoder is constructed.

The convolution operation (9.1.2) can also be described by half-sided infinite matrices (i.e., the matrix continues indefinitely down and to the right). However, instead, we will use the much easier description by polynomials in the next section. A further generalization are non-linear convolutional codes, discussed in Section 10.8, where non-linear operations can also be performed on the contents of the shift register.

The definition of rate-k/n convolutional encoders can also be extended by using feedback shift registers. This leads to an extremely complicated theory of the algebraic structure of convolutional codes based on the properties of matrices with polynomial quotients, i.e., vector spaces over rings and not over fields [168]. For example, all convolutional codes can be systematically encoded with feedback, whereas in Section 9.5 a systematic encoding without feedback will turn out to perform not as well. The trellis coded modulation in Chapter 11 is based on special convolutional codes of rate $R = k/(k + 1)$, which will also be discussed with systematic feedback encoders. However, for most applications it is not really necessary to consider the algebraic theory of convolutional codes in detail. Thus under this aspect the convolutional codes seem to be much easier to understand than RS and BCH codes.

Restriction: In Chapters 8 and 9 only convolutional codes of rate $R = 1/n$ (i.e., $k = 1$) and linear encoders without feedback are considered simplifying the description and the decoding a great deal. However, the method of puncturing convolutional codes as discussed in Section 8.3 still provides nearly all code rates between 0 and 1.

9.2 Polynomial Description

The generator coefficients $g_{\nu,\mu}$ in Definition 9.1 (κ is inapplicable for $k = 1$) are associated to the *generator polynomials*

$$g_\nu(x) = \sum_{\mu=0}^{m} g_{\nu,\mu} x^\mu. \tag{9.2.1}$$

The information bit sequence is characterized by a power series and the encoded block sequence is characterized by a vector of power series:

$$u(x) = \sum_{r=0}^{\infty} u_r x^r \, \boldsymbol{a}(x) = (a_1(x), \ldots, a_n(x)), \quad a_\nu(x) = \sum_{r=0}^{\infty} a_{r,\nu} x^r. \tag{9.2.2}$$

The convolutional encoder corresponds to the polynomial multiplication

$$\underbrace{(a_1(x), \ldots, a_n(x))}_{= \boldsymbol{a}(x)} \; = \; u(x) \cdot \underbrace{(g_1(x), \ldots, g_n(x))}_{= \boldsymbol{G}(x)} \tag{9.2.3}$$

which is equivalent to

$$a_\nu(x) = u(x) g_\nu(x) \quad \text{for} \quad \nu = 1, \ldots, n. \tag{9.2.4}$$

The matrix $\boldsymbol{G}(x) = (g_1(x), \ldots, g_n(x))$ is called a *generator matrix* although for $R = 1/n$ it is only a generator vector. The set of all code sequences (i.e., the code) can be expressed by

$$\mathcal{C} \; = \; \left\{ u(x) \boldsymbol{G}(x) \;\middle|\; u(x) = \sum_{r=0}^{\infty} u_r x^r, \;\; u_r \in \{0, 1\} \right\}. \tag{9.2.5}$$

The convolutional code is linear and $\mathcal{C}$ is a vector space as it is for the block codes. For a given generator matrix, the memory length is

$$m = \max_{1 \le \nu \le n} \deg g_\nu(x). \tag{9.2.6}$$

Example 9.3. For the standard example, $\boldsymbol{G}(x) = (1 + x + x^2, 1 + x^2)$. The following encoded bit sequence is associated with the information bit sequence

$(1, 1, 0, 1, 0, 0) \leftrightarrow u(x) = 1 + x + x^3$:

$$\begin{aligned}\boldsymbol{a}(x) &= (a_1(x), a_2(x)) = u(x)\boldsymbol{G}(x)\\ &= \Big((1 + x + x^3)(1 + x + x^2), (1 + x + x^3)(1 + x^2)\Big)\\ &= (1 + x^4 + x^5, 1 + x + x^2 + x^5)\\ &\leftrightarrow (11, 01, 01, 00, 10, 11).\end{aligned}$$

A finite information bit sequence of length L leads to a finite sequence of encoded blocks of length $L + m$, see also Subsection 9.3.1. ■

In the subsequent sections, we will discuss some special classes of convolutional codes including truncated codes to transform convolutional codes to block codes (Section 9.3), punctured codes to generate convolutional codes with higher code rates than $1/n$ (Section 9.4) and some other classes such as systematic codes and transparent codes (Section 9.5). These modifications and special classes are of great practical importance in certain applications.

9.3 Truncated Convolutional Codes

Three methods for converting a convolutional code to a block code are described in the following subsections.

9.3.1 Direct Truncation

The first method known as *direct truncation* generates a (Ln, L) block code as follows: nach jeweils L information bits und somit Ln encoded bits wird das linear shift register auf all-zeros zurückgesetzt. The resultant block code has the same code rate $1/n$ as the original convolutional code. However, the code has the disadvantage that there is little or none error protection afforded to the last bits in each block. The information sequence $u(x) = x^{L-1}$ of Hamming weight 1 is mapped to the encoded sequence $a(x) = (a_{L-1,0}, \ldots, a_{L-1,n-1})x^{L-1}$ of weight less than or equal to n, so the minimum Hamming distance of the resultant (Ln, L) block code is limited to the very small value of n even if L is large. Deshalb hat diese Methode of direct truncation keine große Bedeutung und wird hier nur der Vollständigkeit halber erwähnt.

Example 9.4. For the standard example $\boldsymbol{G}(x) = (1 + x + x^2, 1 + x^2)$, the information bit sequence $\boldsymbol{u} = (1, 1, 0, 1)$ is mapped to the encoded bit sequence $\boldsymbol{a} = (11, 01, 01, 00)$ in case of direct truncation. ■

9.3.2 Usual Method of Truncation (Zero Tail)

The second method to generate a block code from a convolutional code is known as *truncation* (other widely used names are *zero tail*, *block convolutional codes*, *terminated codes* or *segmented codes*). After L information bits, m known bits (called *tail bits*) are inserted into the data stream of the information bits, for this zeros are usually used. These $L+m$ information bits only influence the $L+m$ code blocks $\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{L-1+m}$ and no further blocks. This also follows from (9.2.4) and (9.2.6):

$$\deg a_\nu(x) \leq \deg u(x) + m.$$

The result is a $((L+m)n, L)$ block code which maps L information bits to $(L+m)n$ encoded bits. So, the code rate is reduced from $R = 1/n$ to

$$R_{\text{trunc}} = \frac{L}{L+m} \cdot \frac{1}{n} \qquad \Big(< \frac{1}{n} = R \Big). \tag{9.3.1}$$

For a large L this loss of rate is negligible. The advantage is a block structure which, for example, avoids error propagation during decoding. Especially for data with a frame structure, almost always truncated convolutional codes are used in practice.

Example 9.5. We assume $n = 2$. Simplifying (9.1.2) by omitting κ leads to

$$a_{r,0} = \sum_{\mu=0}^{m} g_{0,\mu} u_{r-\mu}, \qquad a_{r,1} = \sum_{\mu=0}^{m} g_{1,\mu} u_{r-\mu}.$$

For the example of $m = 2$ and $L = 4$, the truncated convolutional code can be written as a block encoder

$$(a_{0,0}\ a_{0,1} | a_{1,0}\ a_{1,1} | a_{2,0}\ a_{2,1} | a_{3,0}\ a_{3,1} | a_{4,0}\ a_{4,1} | a_{5,0}\ a_{5,1})$$
$$= (u_0, u_1, u_2, u_3) \cdot \left(\begin{array}{cc|cc|cc|cc|cc|cc} g_{0,0} & g_{1,0} & g_{0,1} & g_{1,1} & g_{0,2} & g_{1,2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & g_{0,0} & g_{1,0} & g_{0,1} & g_{1,1} & g_{0,2} & g_{1,2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & g_{0,0} & g_{1,0} & g_{0,1} & g_{1,1} & g_{0,2} & g_{1,2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & g_{0,0} & g_{1,0} & g_{0,1} & g_{1,1} & g_{0,2} & g_{1,2} \end{array}\right)$$

The generator matrix of the $(12,4)$ block code attains the following form for the standard example with $\boldsymbol{G}(x) = (1+x+x^2, 1+x^2)$

$$\boldsymbol{G} = \left(\begin{array}{cc|cc|cc|cc|cc|cc} 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \end{array}\right)$$

and so the numerical example from Example 9.3 kann per matrix multiplication erneut nachvollzogen werden. Durch simple Berechnung aller 16 codewords erweist sich die minimum distance of this block code als $d_{\min} = 5$. ■

Die Verwendung von matrices geschieht hier nur zum besseren Verständnis und wegen der Analogie zu block codes, tatsächlich werden matrices zum praktischen Umgang mit convolutional codes meistens nicht benötigt.

9.3.3 Tail Biting Codes

Die vorangehende method of truncation by insertion of zeros hat sehr große praktische Bedeutung und bei großen Werten von L fällt die Verringerung der code rate durch den factor of $L/(L+m)$ nicht ins Gewicht. Wenn jedoch sehr kurze information words mit einem block convolutional code übertragen werden sollen, dann kann die nachfolgend dargestellte Technik als eine attraktive Alternative angewendet werden.

The third method to generate a (Ln, L) block code from a convolutional code is known as *tail biting* [202, 203]. Jeweils L information bits are mapped to Ln encoded bits, im Gegensatz zu direct truncation wird das linear shift register jetzt aber mit den last m information bits initialisiert. Also enthält das linear shift register vor der Berechnung des ersten encoded blocks exakt die gleichen information bits wie nach einem zusätzlichen Shift nach der Berechnung des letzten blocks:

$$\begin{array}{rclllllll}
\boldsymbol{a}_0 & = & \text{encoder}(& u_0, & u_{L-1}, & u_{L-2}, & \ldots & u_{L-m+1}, & u_{L-m} \quad) \\
\boldsymbol{a}_1 & = & \text{encoder}(& u_1, & u_0, & u_{L-1}, & \ldots & u_{L-m+2}, & u_{L-m+1} \quad) \\
 & \vdots & & & & & & & \\
\boldsymbol{a}_{L-1} & = & \text{encoder}(& u_{L-1}, & u_{L-2}, & u_{L-3}, & \ldots & u_{L-m}, & u_{L-m-1} \quad)
\end{array}$$

The advantage of this method over the second method of truncation is that the code rate remains unchanged. The advantage over the first method of direct truncation is that all information bits are afforded the same amount of error protection, however, as a disadvantage the decoder is more complex for tail biting than for the standard convolutional code as we will see in Subsection 10.2.3.

Example 9.6. We assume again the standard example with $n = 2$ and $m = 2$. For $L = 4$, the truncated convolutional code can be written as a block encoder operation in the usual form of Definition 5.1

$$\begin{aligned}
&(a_{0,0}\ a_{0,1} | a_{1,0}\ a_{1,1} | a_{2,0}\ a_{2,1} | a_{3,0}\ a_{3,1}) \\
&\qquad = (u_0, u_1, u_2, u_3) \cdot \left(\begin{array}{cc|cc|cc|cc}
g_{0,0} & g_{1,0} & g_{0,1} & g_{1,1} & g_{0,2} & g_{1,2} & 0 & 0 \\
0 & 0 & g_{0,0} & g_{1,0} & g_{0,1} & g_{1,1} & g_{0,2} & g_{1,2} \\
g_{0,2} & g_{1,2} & 0 & 0 & g_{0,0} & g_{1,0} & g_{0,1} & g_{1,1} \\
g_{0,1} & g_{1,1} & g_{0,2} & g_{1,2} & 0 & 0 & g_{0,0} & g_{1,0}
\end{array} \right)
\end{aligned}$$

The generator matrix of this $(8, 4)$ block code attains the following form for the standard example

$$\boldsymbol{G} = \left(\begin{array}{cc|cc|cc|cc}
1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 & 0 & 0 & 1 & 1
\end{array} \right)$$

Für die information bit sequence $u = (1, 1, 0, 1)$ ergibt sich die encoded bit sequence $\boldsymbol{a} = (01, 10, 01, 00)$, für das Beispiel $u = (1, 0, 1, 0)$ ergibt sich jedoch $\boldsymbol{a} = (00, 10, 00, 10)$ und somit beträgt die minimum distance of the resultant $(8, 4)$ block code with $L = 4$ only $d_{\min} = 2$. Für den $(12, 6)$ block code with $L = 6$ gilt $d_{\min} = 3$ und für den $(16, 8)$ block code with $L = 8$ gilt $d_{\min} = 4$. Der maximale Wert $d_{\min} = 5$ wird für den $(20, 10)$ block code with $L = 10$ erreicht und bei größerem L erhöht sich $d_{\min}$ nicht weiter. ■

Viele weitere Details zu tail biting of convolutional codes finden sich in [66] und ein survey of new bounds and search results in [156]. For the truncation method, the information bit stream is manipulated. Manipulations in the encoded bit stream lead to the following class of codes.

9.4 Punctured Convolutional Codes

9.4.1 The Basic Principle

Each P consecutive code blocks are combined to a group. Of the nP encoded bits in this group, l encoded bits are deleted (punctured) and not transmitted. Therefore, P information bits are mapped to $nP - l$ code bits. So, the code rate is increased from $R = 1/n$ to

$$R_{\text{punc}} = \frac{P}{nP - l} \qquad \Big(> \frac{1}{n} = R \Big). \tag{9.4.1}$$

Of course, $l < (n - 1)P$ must be valid to guarantee a code rate smaller than 1 to be able to reconstruct the information bits from the encoded bits.

The value P is called *puncturing period* and the rate-$1/n$ code is called *mother code*. The positions of those encoded bits to be punctured are fixed in a *puncturing pattern*. With increasing P nearly all code rates between $1/n$ and 1 can be achieved. Examples for a mother code with $R = 1/2$ are shown in Table 9.2. In particular for $l = P - 1$, $R_{\text{punc}} = P/(P + 1)$.

Table 9.1. Code rates R_{punc} for punctured convolutional codes (from a mother code of rate $R = 1/2$)

	$l=0$	$l=1$	$l=2$	$l=3$	$l=4$	$l=5$	$l=6$	$l=7$
$P=2$	2/4	2/3						
$P=3$	3/6	3/5	3/4					
$P=4$	4/8	4/7	4/6	4/5				
$P=8$	8/16	8/15	8/14	8/13	8/12	8/11	8/10	8/9

In practice, instead of using rate-k/n codes as in Definition 9.1 punctured codes are mostly used. A punctured rate-2/3 code with a rate-1/2 mother code is not as good as a non-punctured rate-2/3 code, however, with the Viterbi algorithm the puncturing method causes no extra effort for the decoding procedure. The complexity of the decoding algorithm is only determined by the mother code und not by the puncturing pattern, this will be shown in Section 10.?.

Usually a code with large P is better, hence, $R = 4/6$ is not quite as good as $R = 8/12$. There are extensive tables of the best punctured codes, e.g., in [183, 199]. However, the results hardly vary with the puncturing length and the other constraints: on the one hand the selection of the mother code and the puncturing scheme can be optimized for a specific code rate or on the other hand one can optimize to a whole code family with various rates and an unchanged mother code.

9.4.2 Rate Compatible Punctured Convolutional (RCPC) Codes

Of great practical importance are the *rate compatible punctured convolutional (RCPC) codes*, introduced by J.Hagenauer [184], where a code family is derived with various rates from a mother code by embedding the high-rated codes in the low-rated codes. Codes with a higher rate are only created by further puncturings of codes with a lower rate, hence, all bits used for higher rates are also used for lower rates. One can switch between various code rates without considerable additional effort which is obvious for the encoder but will be proven for the decoder in Section 10.?. The significant advantage of RCPC codes is that the code rate can be switched at any time without having any negative effects on the distance properties around the point of switching.

Table 9.2. Complete puncturing table for an RCPC code (rate-1/4 mother code with $m = 4$ and $P = 8$)

Puncturing table				R_{punc}
1111 0111	1000 1000	0000 0000	0000 0000	8/9
1111 1111	1000 1000	0000 0000	0000 0000	8/10
1111 1111	1010 1010	0000 0000	0000 0000	8/12
1111 1111	1110 1110	0000 0000	0000 0000	8/14
1111 1111	1111 1111	0000 0000	0000 0000	8/16
1111 1111	1111 1111	1000 1000	0000 0000	8/18
1111 1111	1111 1111	1100 1100	0000 0000	8/20
1111 1111	1111 1111	1110 1110	0000 0000	8/22
1111 1111	1111 1111	1111 1111	0000 0000	8/24
1111 1111	1111 1111	1111 1111	1000 1000	8/26
1111 1111	1111 1111	1111 1111	1010 1010	8/28
1111 1111	1111 1111	1111 1111	1110 1110	8/30
1111 1111	1111 1111	1111 1111	1111 1111	8/32

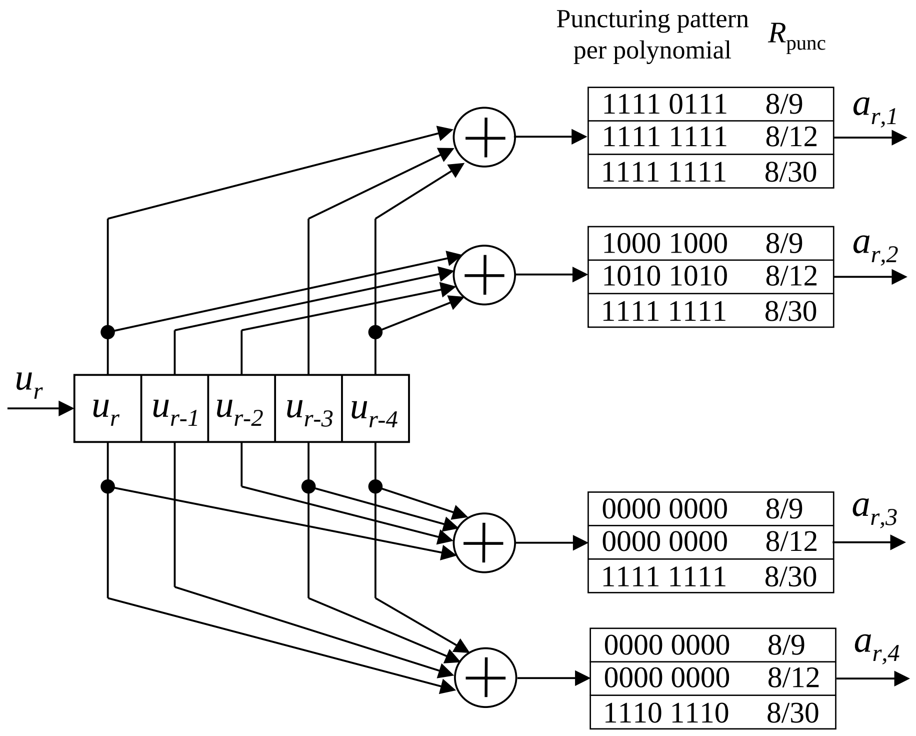


Figure 9.3. RCPC encoder with three exemplary code rates (rate-1/4 mother code with $m = 4$ and $P = 8$)

An example for RCPC codes is shown in Table 9.2 [144, 184], where rate compatible punctured codes with rates from 8/9 to 1/4 are derived from a rate-1/4 mother code with the generator matrix

$$\boldsymbol{G}(x) = (1 + x^3 + x^4, 1 + x + x^2 + x^4, 1 + x^2 + x^3 + x^4, 1 + x + x^3 + x^4).$$

A zero in the puncturing pattern indicates puncturing. In Table 9.2 the first group of 8 bits in the puncturing table refers to first generator polynomial, the second group to the second polynomial, and so on. The RCPC encoder for three exemplary code rates is shown in detail in Figure 9.3, although all necessary information is already contained in Table 9.2. For code rates greater than or equal to 1/2 only the two first polynomials from $\boldsymbol{G}(x)$ are used.

Two important applications for RCPC codes are discussed below.

- Transmission channels with varying quality where the code rate is adapted by an ARQ method (if a feedback channel is available) according to the channel properties. Typically, for a good channel only the punctured code is used and only if the channel quality is deteriorating the punctured positions are transmitted as well to raise the correctability.

- Data sources where the different bits within a frame have varying importance and therefore require a varying error protection. This is also called *unequal* *error protection (UEP) coding.* An interesting application is the source encoding of speech in digital mobile radio systems (see Sections 12.3 and 12.4 for more details).

Figure 9.4 shows the bit-error rate of an RCPC code with four code rates 8/10, 8/12, 8/14 and 8/16 with puncturing pattern according to Table 9.2. For these specific code rates, the puncturing period reduziert sich effektiv von $P = 8$ auf $P = 4$, da die four entsprechenden puncturing patterns in Table 9.2 jeweils consist of two repetitions of a pattern of length four.

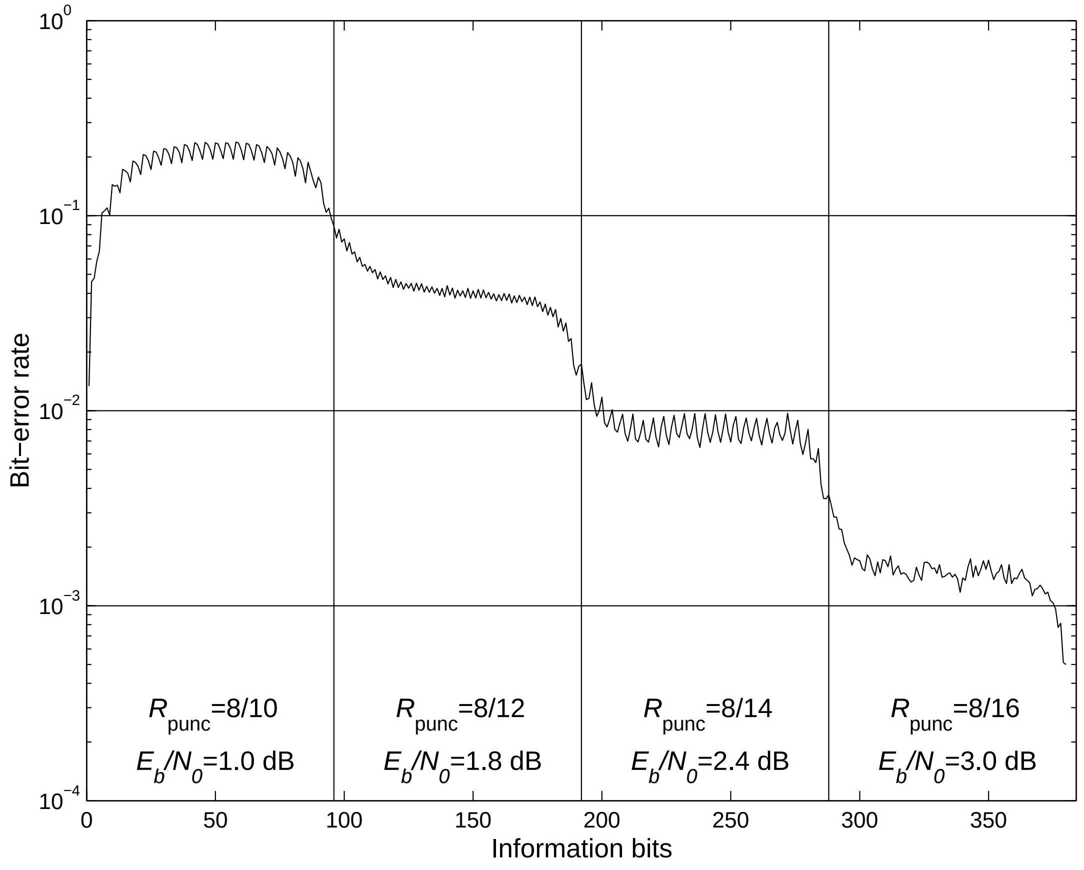


Figure 9.4. Performance of an RCPC oder with four code rates (at $E_c/N_0 = 0$ dB)

In Figure 9.4 wird jede der vier code rates auf eine Gruppe von 96 information bits angewendet (wobei in der letzten Gruppe 4 terminating zeros enthalten sind), so daß die insgesamt 384 information bits of a frame are mapped to $120 + 144 + 168 + 384 = 624$ encoded bits. Die bit-error rate in Figure 9.4 wird durch Viterbi decoding (siehe next chapter) with soft decisions gewonnen. Dabei wird $E_c/N_0 = 0$ dB vorausgesetzt so daß die in Figure 9.4 angegeben E_b/N_0 Werte für die einzelnen code rates resultieren. Die bit-error rate für jedes einzelne der 384 information bits wurde durch Mittelung über 80000 frames geschätzt, insgesamt wurden also $384 \cdot 80000 = 30.72$ Mbit simuliert. Die deutlich erkennbaren Oszillationen in der bit-error rate bei der ersten und dritten code rate entsprechen exakt der puncturing period of 4 und bei der zweiten code rate der puncturing period of 2. Dagegen sind bei der vierten code rate ohne puncturing einerseits keine Oszillationen mehr zu erwarten, andererseits wären sie hier aber auch nicht mehr erkennbar da die quasi-random inaccuracies als Folge einer zu kurzen Simulationsdauer (..run) überwiegen. Nebenbei bemerkt findet sich die averaged bit-error rate bei $R = 1/2$ auch in Table 10.8.

Der besondere Vorteil des RCPC Prinzips ist daß die bit-error rate einen ziemlich glatten Verlauf around the point of rate switching aufweist. Wenn dagegen the puncturing pattern von high-rated codes nicht in diejenigen der low-rated codes embedded wären, dann würde die bit-error rate im Bereich der rate transitions kurzzeitig erheblich ansteigen.

9.5 Further Specific Classes of Convolutional Codes

9.5.1 Systematic Codes

In a systematic encoder at least one component in the code block corresponds to the information bit.

Where block codes are nearly exclusively systematically encoded this is not possible for convolutional codes if the linear combinations in Definition 9.1 are not extended by feedbacks. Some of the best convolutional codes can not be systematically encoded without feedback which will be shown in Section 9.5.

9.5.2 Transparent Codes

For every encoded bit sequence in transparent codes the binary complement must be an encoded bit sequence as well (this is of course not necessary for the first m code blocks).

Since the all-zero sequence is a code sequence then so is the all-one sequence. The code is transparent if and only if every linear combination consists of an odd number of information bits or every generator polynomial has an odd weight. With this definition an all-one information sequence leads to an all-one code sequence. Most of the good convolutional codes are not transparent, however, the so-called *industry standard code* with $m = 6$ and $R = 1/2$ as in Table 9.3a and $R = 1/3$ as in (9.5.2) is transparent.

The advantage of transparent codes is that if there is a change of polarity or a 180 degree phase shift of the physical channel, the code remains invariant. The inversion of the encoded bits corresponds to the inversion of the information bits. In combination with the differential precoding [25, 112, 144] the result is a rotational invariance. This subject will be discussed in detail in Section 11.8.

9.6 Catastrophic Codes and Encoder Inverse

Example 9.7. Consider the encoder shown in Figure 9.3 of a rate-1/2 code with $\boldsymbol{G}(x) = (1+x, 1+x^2)$ which is similar to the standard example. The all-one information sequence results in the encoded sequence $\boldsymbol{a}(x) \leftrightarrow (11, 01, 00, 00, 00, \ldots)$. This can also be derived by using $u(x) = 1 + x + x^2 + x^3 + \cdots = 1/(1-x)$ in $u(x)\boldsymbol{G}(x) = \dfrac{1}{1-x}\Big(1+x, (1+x)^2\Big) = (1, 1+x)$.

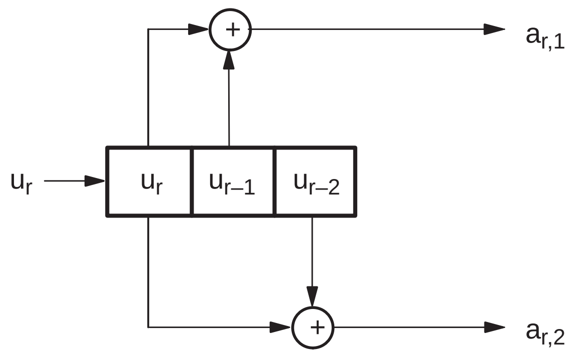


Figure 9.5. A catastrophic rate-1/2 convolutional encoder

Thus the all-one and the all-zero sequences there are two information sequences of infinite Hamming distance with corresponding code sequences which only differ in three positions. Therefore three errors during transmission are enough to produce an infinite number of errors during decoding (snowball effect on error propagation). Such an encoder is called *catastrophic*. Due to the linearity of the code, an equivalent criterium for catastrophic codes can be expressed by the Hamming weights instead of the distances. ■

There is a very simple easily provable criterion to decide whether a convolutional code is catastrophic or not:

Theorem 9.1. *The definition of a* non-catastrophic encoder *is that every information sequence of infinite weight implies an encoded sequence of infinite weight or equivalently every finite encoded sequence implies a finite information sequence. This is equivalent to the generator polynomials not having a common divisor:*

$$\mathrm{GCD}\Big(g_1(x), \ldots, g_n(x)\Big) = 1. \tag{9.6.1}$$

So if the factorization of the generator polynomials into irreducible factors produces a common divisor, we have a catastrophic encoder (as in Example 9.4).

If the greatest common divisor consists only of powers of x, we only have a superfluous delay in the encoder.

Proof. "Non-catastrophic $\Longrightarrow$ GCD=1": we will use the method of contradiction, i.e., we assume that there exists a common divisor $p(x) \neq 1$,

$$\boldsymbol{G}(x) = \Big(p(x)\tilde{g}_1(x), \ldots, p(x)\tilde{g}_n(x)\Big).$$

The constant term in $p(x)$ is not zero, otherwise all generator polynomials would contain the factor x and the encoding would be pointlessly delayed. With a little calculation,

$$u(x) = \frac{1}{p(x)} = \sum_{r=0}^{\infty} u_r x^r$$

turns out to be a power series without negative powers. Then again $u(x)$ cannot be a polynomial because otherwise the product of two polynomials would be 1, which is impossible according to the degree formula. Thus $u(x)$ is a sequence of infinite weight. If interpreted as a information sequence we obtain the code sequence

$$\boldsymbol{a}(x) = u(x)\boldsymbol{G}(x) = \Big(\tilde{g}_1(x), \ldots, \tilde{g}_n(x)\Big)$$

of finite weight. This is a contradiction to the presumption of the encoder being non-catastrophic, therefore $p(x) = 1$.

"Non-catastrophic $\Longleftarrow$ GCD=1": in general, the greatest common divisor of several variables can be represented by a linear combination of these variables (see the Euclidean Algorithm, Theorem A.8). Thus there exist polynomials $f_1(x), \ldots, f_n(x)$ with

$$\sum_{\nu=1}^{n} f_\nu(x) g_\nu(x) = \mathrm{GCD}\Big(g_1(x), \ldots, g_n(x)\Big) = 1. \tag{9.6.2}$$

So the corresponding information sequence of the code sequence $\boldsymbol{a}(x) = \Big(a_1(x), \ldots, a_n(x)\Big) = u(x)\Big(g_1(x), \ldots, g_n(x)\Big)$ can be expressed by

$$\begin{aligned}\sum_{\nu=1}^{n} a_\nu(x) f_\nu(x) &= \sum_{\nu=1}^{n} u(x) g_\nu(x) \cdot f_\nu(x) \\ &= u(x) \cdot \sum_{\nu=1}^{n} g_\nu(x) f_\nu(x) = u(x).\end{aligned}$$

If $\boldsymbol{a}(x)$ is of finite weight, then so is $u(x)$, thus the encoder is non-catastrophic.

■

Theorem 9.1 can be generalized to all convolutional codes of rate $R = k/n$ as follows. For the (k, n)-dimensional generator matrix there are $\binom{n}{k}$ possibilities to choose k out of n columns. So $\binom{n}{k}$ (k, k)-dimensional polynomial matrices can be formed of which we are to calculate the determinants. In extension of (9.4.1) these determinants are not allowed to have a common divisor.

The polynomials $f_1(x), \ldots, f_n(x)$ in (9.4.2) form an encoder inverse, which produces the information sequence of an encoded sequence. This will be explained in detail by the following example. However, the encoder inverse is not a decoder since it can not process code sequences with error patterns and it can not deal with soft decisions. The Viterbi decoder contains the encoder inverse implicitly, so that the polynomials $f_1(x), \ldots, f_n(x)$ need not be known in practice.

Example 9.8. For $\boldsymbol{G}(x) = (g_1(x), g_2(x)) = (1 + x + x^2, 1 + x^2)$, $g_1(x)$ is irreducible, thus the standard example is a non-catastrophic encoder. For $(f_1(x), f_2(x)) = (x, 1 + x)$,

$$f_1(x)g_1(x) + f_2(x)g_2(x) = 1$$

is easily verifiable. Thus

$$\hat{u}(x) = \boldsymbol{a}(x)\begin{pmatrix} x \\ 1+x \end{pmatrix} = u(x)\Big(1 + x + x^2, 1 + x^2\Big)\begin{pmatrix} x \\ 1+x \end{pmatrix} = u(x).$$

This relation is demonstrated in Figure 9.4. Note the recursive description of encoder and encoder inverse

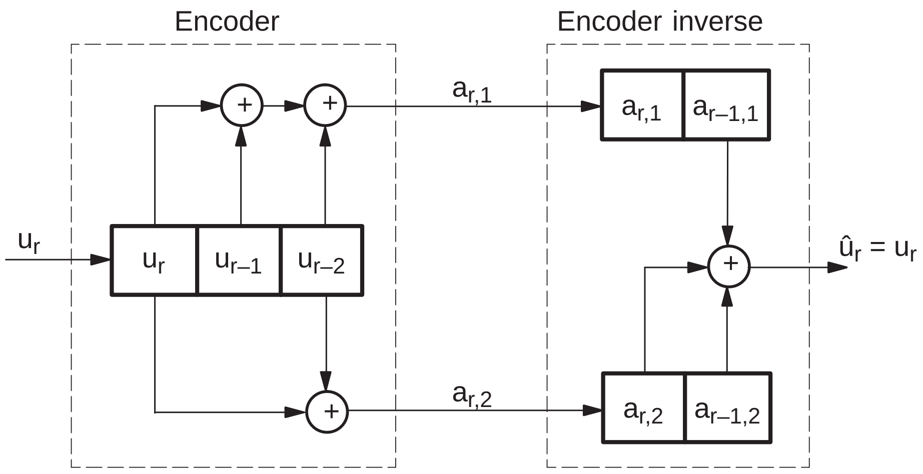


Figure 9.6. Transparent concatenation of encoder and encoder inverse (standard example)

$$\begin{aligned} \text{encoder:}\quad & a_{r,1} = u_r + u_{r-1} + u_{r-2}, \quad a_{r,2} = u_r + u_{r-2} \\ \text{encoder inverse:}\quad & \hat{u}_r = a_{r-1,1} + a_{r,2} + a_{r-1,2}, \end{aligned}$$

which can be verified by $\hat{u}_r = (u_{r-1}+u_{r-2}+u_{r-3})+(u_r+u_{r-2})+(u_{r-1}+u_{r-3}) = u_r$. ■

Example 9.9. For $\boldsymbol{G}(x) = (g_1(x), g_2(x)) = (1+x, 1+x^2) = (1+x)(1, 1+x)$, we have a catastrophic encoder. There are no two polynomials $f_1(x), f_2(x)$ with $f_1(x)g_1(x) + f_2(x)g_2(x) = 1$, since

$$f_1(x)g_1(x) + f_2(x)g_2(x) = (1+x) \cdot \underbrace{\Big(f_1(x) + f_2(x)(1+x)\Big)}_{\text{polynomial}} = 1$$

is impossible according to the degree formula. ■

9.7 Distance Properties and Optimum Convolutional Codes

9.7.1 The Free Distance and Tables of Rate-1/2 and Rate-1/3 Codes

The performance of a block code is mainly expressed by the minimum distance, i.e., by the minimum Hamming weight of all codewords. A corresponding characterization for convolutional codes is given in the following definition.

Definition 9.2. *The* minimum free distance d_f *(otherwise also denoted* d_free *or* d_∞*) of a convolutional code is defined as the minimum Hamming weight of all code sequences:*

$$d_\mathrm{f} \;=\; \min\left\{ w_H(u(x)\,\boldsymbol{G}(x)) \;\middle|\; u(x) = \sum_{r=0}^{\infty} u_r x^r \neq 0 \right\}. \tag{9.7.1}$$

Except for punctured codes we can always set $u_0 = 1$*. Two code sequences always differ in at least* d_f *positions. A convolutional code is called* optimum*, if* d_f *is the maximum over all other convolutional codes with the same code rate and the same memory length* m*.*

In Subsections 9.3.1 and 9.3.2 hatten convolutional codes betrachtet, die zugleich auch spezielle block codes sind. Für diese Fälle gilt natürlich $d_\mathrm{f} = d_\mathrm{min}$. Zu beachten ist aber daß mit größerer block length die minimum distance d_min nicht weiter anwächst, wie es eigentlich bei class of block codes with good performance der Fall wäre. Hieraus darf man aber nicht den Schluss ziehen daß convolutional codes (egal ob mit oder ohne truncation) weniger gut als block codes seien.

Table 9.3a shows some optimum convolutional codes with their generator polynomials for rates $R = 1/2$ and $R = 1/3$ and memory lengths $m = 2 \ldots 6$. In the extended Table 9.3b with $m = 2 \ldots 16$ the generator polynomials are octal encoded. The octal encoding is not unique throughout the literature, e.g., the polynomial $1 + x + x^3$ is described by $(15)_{\text{octal}} = 001\ 101$ as well as by $(64)_{\text{octal}} = (110\ 100)$, however, the octal decoding is unique. For $m \le 6$ the Tables 9.3a and 9.3b are identical.

Table 9.3a. Generator polynomials for $R = 1/2$ and $R = 1/3$ optimum convolutional codes with $m = 2 \ldots 6$ (from [79])

$g_1(x)$	$g_2(x)$	$g_3(x)$	d_f
$1+x+x^2$	$1+x^2$		5
$1+x+x^3$	$1+x+x^2+x^3$		6
$1+x^3+x^4$	$1+x+x^2+x^4$		7
$1+x+x^3+x^5$	$1+x^2+x^3+x^4+x^5$		8
$1+x^2+x^3+x^5+x^6$	$1+x+x^2+x^3+x^6$		10
$1+x+x^2$	$1+x^2$	$1+x+x^2$	8
$1+x+x^3$	$1+x+x^2+x^3$	$1+x^2+x^3$	10
$1+x^2+x^4$	$1+x+x^3+x^4$	$1+x+x^2+x^3+x^4$	12
$1+x^2+x^4+x^5$	$1+x+x^2+x^3+x^5$	$1+x^3+x^4+x^5$	13
$1+x^2+x^3+x^5+x^6$	$1+x+x^4+x^6$	$1+x+x^2+x^3+x^4+x^6$	15

Table 9.3b. Generator polynomials in octal notation for $R = 1/2$ and $R = 1/3$ optimum convolutional codes with $m = 2 \ldots 16$ (from [79])

m	$R=1/2$			$R=1/3$			
	$g_1(x)$	$g_2(x)$	d_f	$g_1(x)$	$g_2(x)$	$g_3(x)$	d_f
2	5	7	5	5	7	7	8
3	64	74	6	54	64	74	10
4	46	72	7	52	66	76	12
5	65	57	8	47	53	75	13
6	554	744	10	554	624	764	15
7	712	476	10	452	662	756	16
8	561	753	12	557	663	711	18
9	4734	6624	12	4474	5724	7154	20
10	4672	7542	14	4726	5562	6372	22
11	4335	5723	15	4767	5723	6265	24
12	42554	77304	16	42554	43364	77304	24
13	43572	56246	16	43512	73542	76266	26
14	56721	61713	18				
15	447254	627324	19				
16	716502	514576	20				

The transition from $R = 1/2$ to $R = 1/3$ not only adds another generator polynomial $g_3(x)$ but may also change $g_1(x)$ and $g_2(x)$. The optimum codes are not always unique, e.g., for $R = 1/2$ and $m = 5$ we also have the generator matrix

$$(53, 75)_{\text{octal}} = (1 + x^2 + x^4 + x^5, 1 + x + x^2 + x^3 + x^5).$$

Most of the codes listed here were found by Odenwalder and Larson by computer search in 1970. Codes with other rates, systematic codes and also transparent codes can be found in [17, 25, 95, 114, 128, 144, 151] and more completely in [29, 66, 75, 79, 126, 147]. The distance properties of tail biting codes are considered in detail in [66].

9.7.2 The Industry Standard Code

Tables 9.3a and 9.3b contain the *Industry Standard Code* $(171, 133)_{\text{octal}}$ for $R = 1/2$ and $m = 6$ which is one of the most important convolutional codes, nowadays implemented in encoder/decoder chips by various manufacturers such as [216]. Also based on the industry standard code is the so-called pragmatic trellis coded modulation which will be discussed in Section 11.11. For the rate-1/3 version of the industry standard code, the following generator polynomials are given in [216]

$$\begin{aligned}(171, 133, 165)_{\text{octal}} = &(1 + x + x^2 + x^3 + x^6,\ \ 1 + x^2 + x^3 + x^5 + x^6,\\ &1 + x + x^2 + x^4 + x^6),\end{aligned} \tag{9.7.2}$$

where the first two generator polynomials of the rate-1/2 code have been copied without changes. The industry standard code is also used with puncturing to generate various code rates $R_{\text{punc}} = P/(P + 1)$ between 1/2 and 16/17. The best puncturing patterns for the most important code rates are shown in Table 9.4.

Table 9.4. Best puncturing patterns for the industry standard code (from [216])

Code rate R_{punc}	Puncturing period P	Puncturing pattern for the $(171, 133)_{\text{octal}}$ rate-1/2 code (0 = punctured encoded bit)
2/3	2	1 0 1 1
3/4	3	1 0 1 1 1 0
4/5	4	1 0 0 0 1 1 1 1
5/6	5	1 0 1 0 1 1 1 0 1 0
7/8	7	1 0 0 0 1 0 1 1 1 1 1 0 1 0
15/16	15	1 0 0 1 1 0 1 0 0 1 0 1 1 0 1 1 1 1 0 0 1 0 1 1 0 1 0 0 1 0

The application of the puncturing patterns from Table 9.4 is illustrated with an example for $P = 3$. The encoder first generates the sequence $\boldsymbol{a}_r = (a_{r,1}, a_{r,2})$

which is ordered as

$$\cdots \left|\begin{array}{ccc} a_{1,1} & \boxed{a_{2,1}} & a_{3,1} \\ & & \\ a_{1,2} & a_{2,2} & \boxed{a_{3,2}} \end{array}\right|\begin{array}{ccc} a_{4,1} & \boxed{a_{5,1}} & a_{6,1} \\ & & \\ a_{4,2} & a_{5,2} & \boxed{a_{6,2}} \end{array}\right| \cdots$$

Prior to transmission all bits in the boxes are punctured or deleted and not transmitted. For this example, two out of six bits are deleted in each pattern, so only the sequence $a_{1,1}\ \ a_{1,2}\ \ a_{2,2}\ \ a_{3,1}\ \ a_{4,1}\ \ a_{4,2}\ \ a_{5,2}\ \ a_{6,1}$ is actually transmitted and the code rate is $R_{\text{punc}} = 3/4$ of course.

9.7.3 Bounds on the Free Distance

For the information sequence $u(x) = 1$ we obtain the code sequence $\boldsymbol{a}(x) = \boldsymbol{G}(x)$ providing us with a general upper bound for the free distance:

$$d_{\text{f}} \le w_H(\boldsymbol{G}(x)) = \sum_{\nu=1}^{n} w_H(g_\nu(x)) \tag{9.7.3}$$

$$\le n \cdot (m+1). \tag{9.7.4}$$

Often there is equality in the first inequality, whereas the second inequality can never be tight for non-catastrophic codes. For a systematic rate-1/2 code we have $\boldsymbol{G}(x) = (g_1(x), 1)$ and therefore

$$d_{\text{f}} \le m + 2.$$

Since Tables 9.3a and 9.3b show larger free distances for the optimum codes, optimum codes can not be systematic.

Example 9.10. (1) For the standard example with $\boldsymbol{G}(x) = (1+x+x^2, 1+x^2)$, $d_{\text{f}} \le 5$ according to (9.5.3). In Subsection 9.7.2, we will show that $d_{\text{f}} = 5$.

(2) For $\boldsymbol{G}(x) = (1+x+x^2, 1+x+x^2)$ (9.5.3) implies $d_{\text{f}} \le 6$, however, it is a catastrophic encoder and we only have $d_{\text{f}} = 4$, since $u(x) = 1+x$ generates $\boldsymbol{a}(x) = (1+x^3, 1+x^3)$. Thus there can not be a rate-1/2 code with $d_{\text{f}} = 6$.

(3) For the catastrophic code with $G(x) = (1+x, 1+x^2)$, we have $d_{\text{f}} = 3$ according to Example 9.4, however, when restricting to finite sequences we have $d_{\text{f}} = 4$ for the truncated code. ■

Theorem 9.2 (Heller Bound). *The minimum free distance d_{f} of a rate-$1/n$ convolutional code with memory length m satisfies*

$$d_{\text{f}} \le \min_{l \ge 1} \left\lfloor \frac{(l+m)n}{2(1-2^{-l})} \right\rfloor. \tag{9.7.5}$$

The asymptotic Heller bound for $m \to \infty$ satisfies

$$\lim_{m\to\infty} d_{\text{f}} \le \frac{m \cdot n}{2}. \tag{9.7.6}$$

Proof. We consider information sequences of length l with m truncating zeros. The corresponding encoded sequences constitute a $((l+m)n, l, d_{\min})_2$ block code. The application of the Plotkin bound from Theorem 4.11 implies

$$d_\mathrm{f} = d_{\min} \leq \frac{(l+m)n \cdot 2^{l-1}}{2^l - 1}$$

and the proof of (9.5.5) is complete. The asymptotic bound (9.5.6) follows immediately with $l = 1$. ■

The Heller bound proves to be very tight in comparison with the codes from Table 9.3. This is also demonstrated with Figure 9.5 (for the non-punctured rates from 1/6 to 1/2), where the difference between the Heller bound and the exact minimum free distance is very small. For $l = 1$, (9.5.5) is identical to (9.5.4).

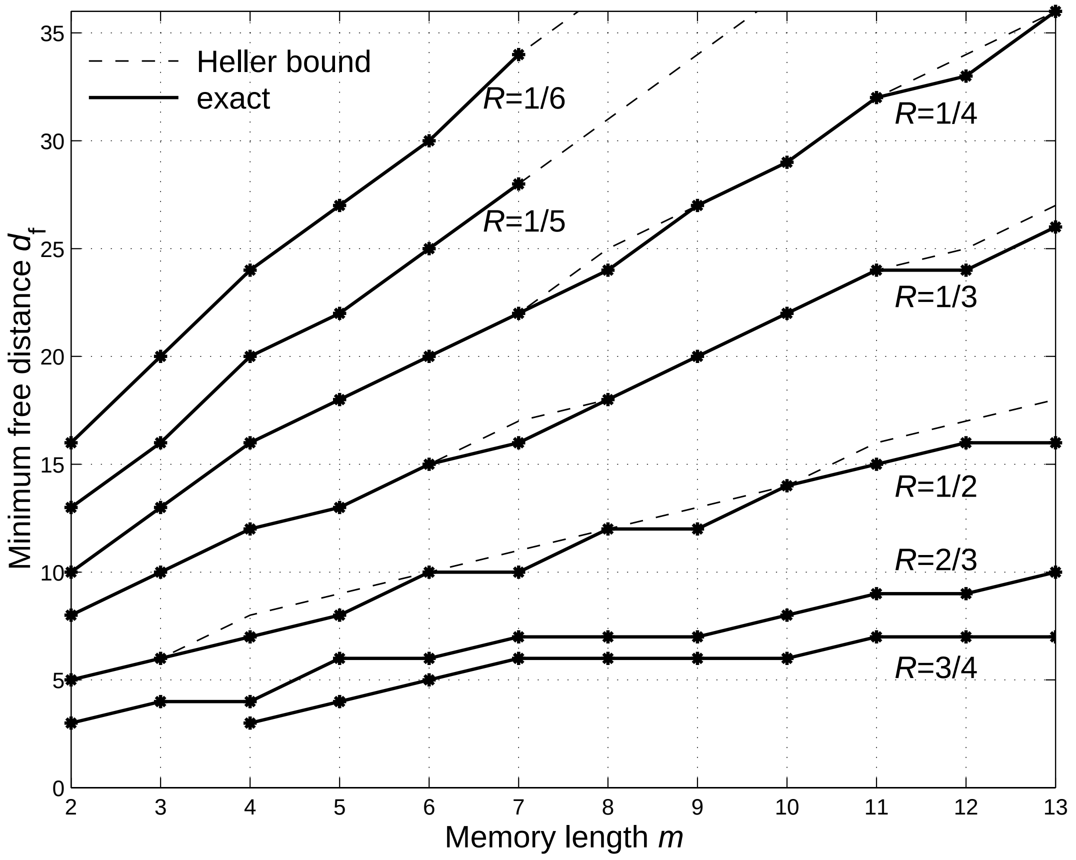


Figure 9.7. Minimum free distance versus memory length for various code rates

Figure 9.5 or the asymptotic result (9.5.6) indicate that the minimum free distance is approximately proportional to the memory length and is also strongly dependent on the code rate, of course. However, the free distance does not always continuously increase with m, e.g., for $R = 1/2$ increasing from $m = 6$ to $m = 7$ is of no advantage at all but will double the decoder complexity. The codes with rates $R = 2/3$ and $R = 3/4$ in Figure 9.5 are punctured codes which are mentioned in [183]. Other tables of punctured codes have slightly different free distances depending on the criteria chosen for punctured codes as explained in Subsection 9.3.2.

For the minimum distance of the block code emerging from a truncated convolutional code, $d_{\text{min}} = d_{\text{f}}$. However, truncated convolutional codes are asymptotically bad block codes according to Section 3.4, since for a constant code rate the minimum distance remains constant and thus the distance rate converges to zero for increasing block length. Nevertheless, the convolutional codes still prove to be very powerful. The Definition 9.2 of optimum codes was made in anticipation of ML decoding with the Viterbi algorithm (see Chapter 10). For other decoding schemes different distance properties are the criteria for the selection of the code. A detailed overview of further distance definitions is given in [29].

Until now, we have described convolutional codes and their encoders by polynomials or shift registers. To describe the code itself we will introduce the trellis diagram (important for ML decoding) in Section 9.6 and the state transition diagram (important for the calculation of the distance properties) in Section 9.7.

9.8 The Trellis Diagram

The description of a convolutional code by a trellis diagram is the basic idea for the decoding. The trellis diagram consists of repeated trellis segments, which uniquely correspond to the convolutional encoder.

9.8.1 Trellis Segments

According to (9.1.1), the current code block $\boldsymbol{a}_r$ is a function of the current information bit u_r and the previous m information bits $u_{r-1}, \ldots, u_{r-m}$. The binary m-tuple $(u_{r-1}, \ldots, u_{r-m})$ is called a *state*. The values of these 2^m states are denoted $\zeta_1, \ldots, \zeta_{2^m}$ where $\zeta_1 = (0, \ldots, 0)$ is the *zero state*. The state at time r (referring to the information bits or the code blocks) is denoted $z_r \in \{\zeta_1, \ldots, \zeta_{2^m}\}$. The current information bit and the current state deliver the current code block and the next state:

$$\boldsymbol{a}_r = \text{encoder}(u_r, z_r) \quad , \quad z_{r+1} = \text{shift function}(u_r, z_r). \tag{9.8.1}$$

Example 9.11. For the standard example with $\boldsymbol{G}(x) = (1 + x + x^2, 1 + x^2)$ the states are numbered as $\zeta_1 = 00$, $\zeta_2 = 10$, $\zeta_3 = 01$, $\zeta_4 = 11$, leading to the state tables

Present state	Output bits	
$\boldsymbol{a}_r$	$u_r = 0$	$u_r = 1$
$z_r = 00$	00	11
$z_r = 10$	10	01
$z_r = 01$	11	00
$z_r = 11$	01	10

Present state	Next state	
z_{r+1}	$u_r = 0$	$u_r = 1$
$z_r = 00$	00	10
$z_r = 10$	01	11
$z_r = 01$	00	10
$z_r = 11$	01	11

In Figure 9.6 these tables are represented by a trellis segment. The branches between the states are labeled with u_r, $\boldsymbol{a}_r$. The generator matrix only influences the $\boldsymbol{a}_r$ labeling but not the state transitions. ■

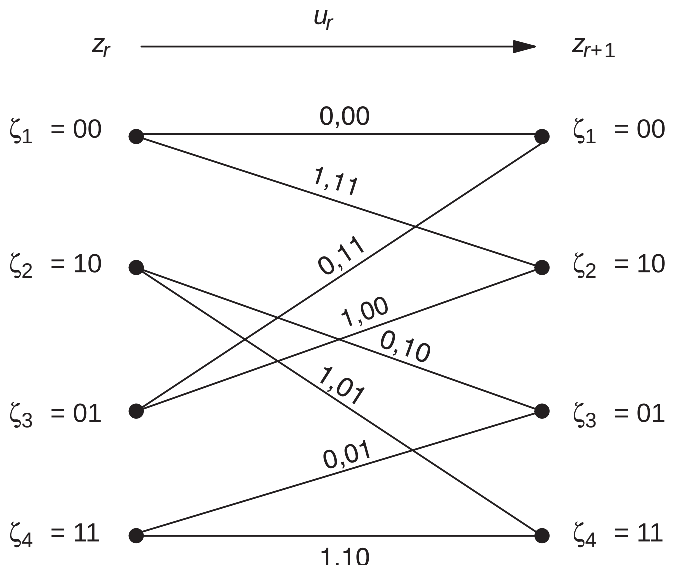


Figure 9.8. Trellis segment for the standard example

A *trellis segment* always has 2^m states. For rate-$1/n$ codes, there are always two *branches* leaving each state $z_r = \zeta_i$ and two branches arriving at each state $z_{r+1} = \zeta_j$. If the states are numbered as inverse dual numbers, we obtain the symmetry shown in Figure 9.7.

All in all there are four branches from ζ_i and $\zeta_{i+2^{m-1}}$ towards ζ_{2i-1} and ζ_{2i}. The labeling of the branches with u_r can be omitted, since the upper outgoing branches always correspond to $u_r = 0$ and the lower outgoing branches always correspond to $u_r = 1$. In ζ_i and $\zeta_{i+2^{m-1}}$ the first part $\boldsymbol{u} = (u_{r-1}, \ldots, u_{r-m+1})$ is constant. During the transition from z_r to z_{r+1}, u_{r-m} drops out and $\boldsymbol{u}$ forms the last part of ζ_{2i-1} and ζ_{2i}. This process will become even more apparent for the more complicated case of $m = 4$ displayed in Figure 9.8.

For rate-k/n convolutional codes with a memory length of m there are 2^k branches leaving and 2^k branches arriving at each of the 2^m states. Thus there

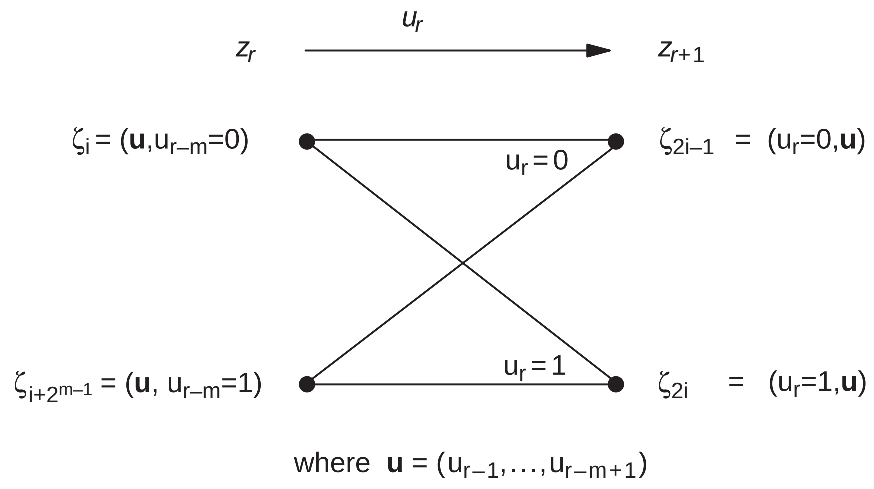

u_r
z_r
z_{r+1}
$\zeta_i = (\mathbf{u}, u_{r-m}=0)$
$\zeta_{2i-1} = (u_r=0, \mathbf{u})$
$u_r = 0$
$\zeta_{i+2^{m-1}} = (\mathbf{u}, u_{r-m}=1)$
$\zeta_{2i} = (u_r=1, \mathbf{u})$
$u_r = 1$
where $\mathbf{u} = (u_{r-1}, \ldots, u_{r-m+1})$

Figure 9.9. The elementary part of the general trellis segment

are 2^{m+k} branches per trellis segment. Corresponding examples for such more complex trellis segments will be discussed in Chapter 11.

9.8.2 Complete Trellis Diagrams

A complete *trellis diagram* (or simply called *trellis*) is constructed by repeated identical trellis segments placed behind each other. Since the shift register is initialized with zeros, the trellis diagram has m incomplete *starting segments* corresponding to the encoder's departure from the zero state. For truncated codes, there are also m incomplete *final segments* corresponding to the encoder's return to the zero state. States are also called *nodes*. The trellis diagram is a *directed graph* with a *start node* and a *final node*. A sequence of branches throughout the trellis is called a *path*. For non-truncated codes the paths can also be of infinite length.

Example 9.12. The trellis diagram for the standard example is shown in Figure 9.9 where it is terminated after 4 information bits. The branches are labeled with the code blocks. The thick lined path corresponds to

information sequence	$1, 1, 0, 1, (0, 0)$
encoded sequence	$11, 01, 01, 00, 10, 11$
state sequence	$\zeta_1, \zeta_2, \zeta_4, \zeta_3, \zeta_2, \zeta_3, \zeta_1.$

The same code sequence was already calculated by the generator matrix in Example 9.3. ■

For punctured codes the labeling of the branches with the code blocks is actually time-variant (i.e., varying from segment to segment), but in Section

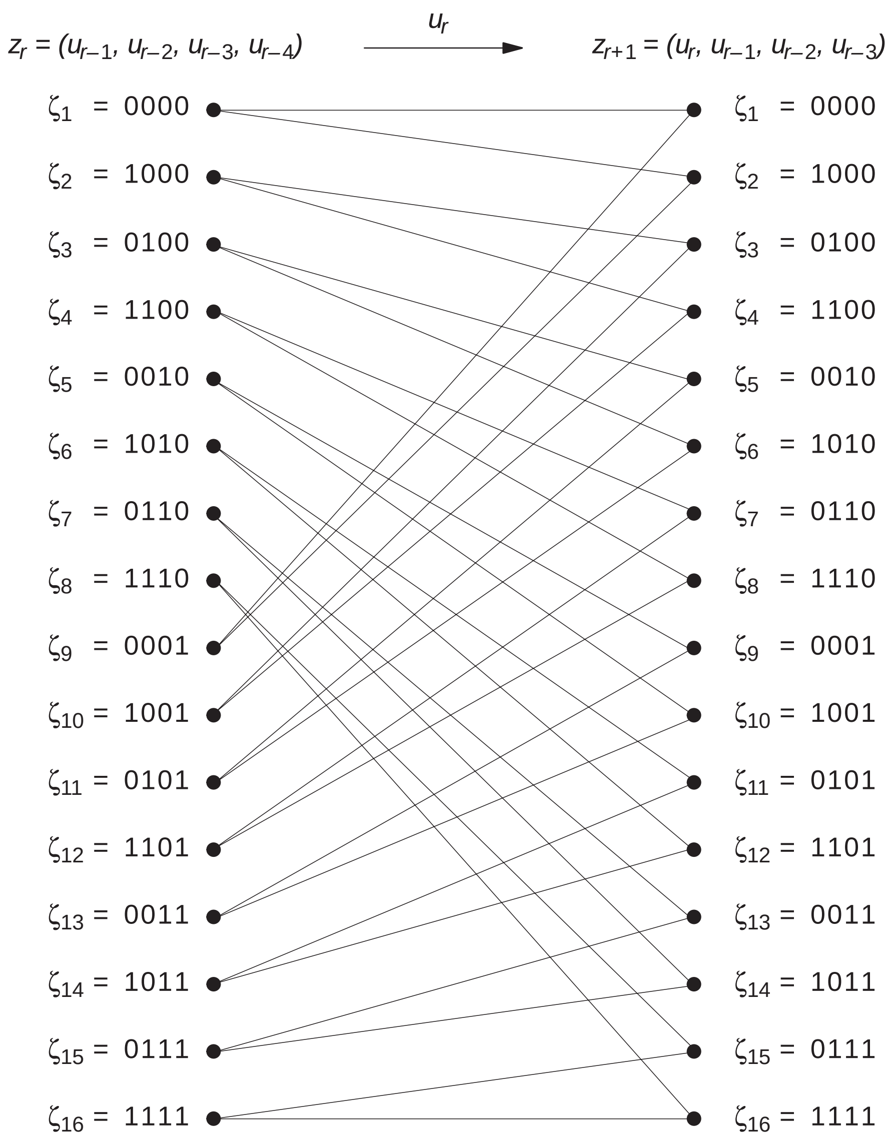
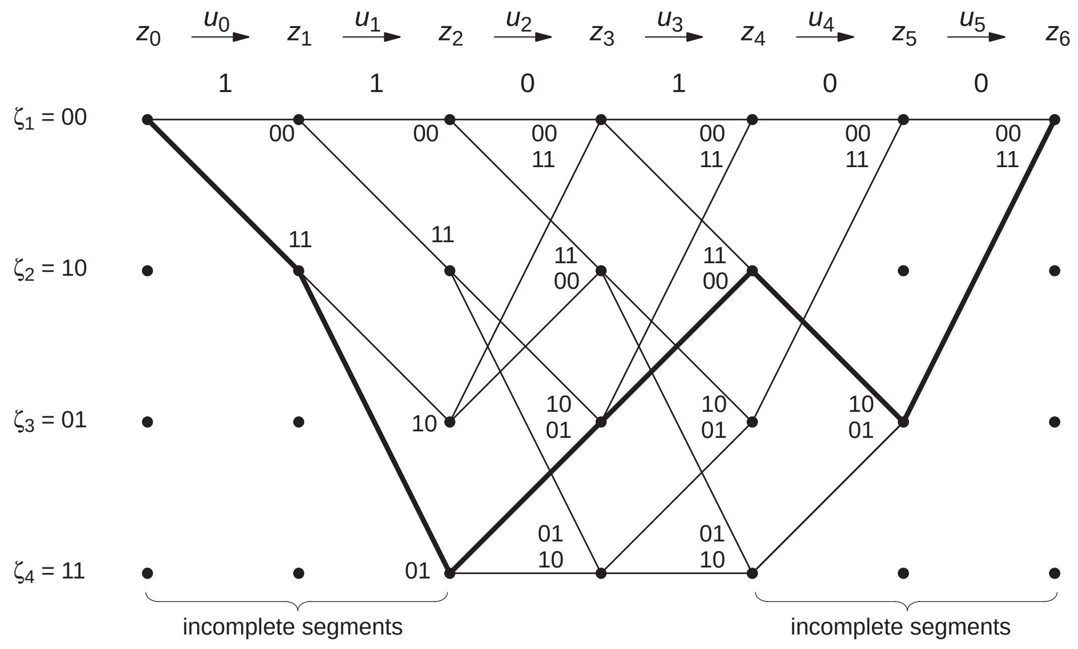


Figure 9.10. Trellis segment for a memory length of $m = 4$

10.4 we will see that the puncturing pattern can be very easily incorporated into the decoding algorithm.

The information sequence, code sequence and state sequence are uniquely related to each other. By using the method "paths through the trellis" the code is a lot easier to overlook than when using the generator matrix. A path of finite length through the trellis is called a *fundamental path* (or *detour path*) which starts and ends at the zero state but does not touch it in between. The length of the fundamental path is at least $m + 1$ segments or $(m + 1)n$ encoded bits. Every path in the terminated trellis consists of a finite sequence of fundamental paths and all-zero branches.



z_0 $\xrightarrow{u_0}$ z_1 $\xrightarrow{u_1}$ z_2 $\xrightarrow{u_2}$ z_3 $\xrightarrow{u_3}$ z_4 $\xrightarrow{u_4}$ z_5 $\xrightarrow{u_5}$ z_6
1 1 0 1 0 0
ζ_1 = 00
ζ_2 = 10
ζ_3 = 01
ζ_4 = 11
00
11
00
11
00 11
11 00
10 01
01 10
10
01
incomplete segments
incomplete segments

Figure 9.11. Trellis diagram for the standard example

The free distance d_f was defined as the minimum Hamming weight of all paths beginning at the zero state in the trellis. Of course, the Hamming weight refers to the labeling of the branches with the code blocks. A truncated code and a non-truncated code have the same free distance.

Theorem 9.3. *For non-catastrophic codes the free distance can be determined from the fundamental paths alone.*

Proof. We are to show that the minimum weight of all code sequences is only achieved for a finite information sequence. Assume the contrary: let there be an information sequence of infinite weight with a code sequence weight $\leq w_H(\boldsymbol{G}(x)) < \infty$. However, this is not possible for non-catastrophic codes. ■

This theorem connects the expressions catastrophic, free distance and fundamental path. So for the determination of d_f it is sufficient only to consider paths of finite length and it does not matter where the path begins. Hence, Figure 9.9 implies that $d_\mathrm{f} = 5$ for the standard example. For catastrophic codes there are paths of infinite length and finite weight which do not touch the zero state. For non-catastrophic codes this is impossible.

9.9 State Diagrams and Weight Enumerators

9.9.1 The State Transition Diagram

With the trellis diagram all code properties are already contained in only one segment. The *state transition diagram* (also called *signal flowchart*) is a weighted *directed graph* (also called *digraph*) which is constructed from a trellis segment by identifying the nodes in step $r+1$ with the ones in step r.

The method will become more apparent when looking at the state transition diagram for the standard example shown in Figure 9.10. A solid branch corresponds to $u_r = 0$ and a dotted branch corresponds to $u_r = 1$. The branches are labeled again with the code blocks.

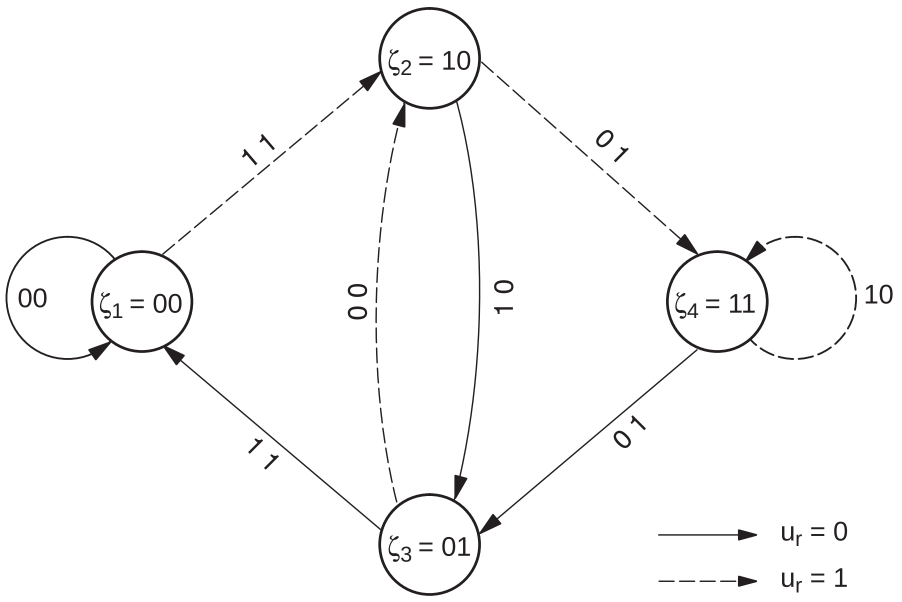


Figure 9.12. State transition diagram for the standard example

In general, there are 2^m states with 2^{m+1} branches for rate $1/n$-codes. The possible code sequences correspond to the possible state sequences in the state transition diagram. The state transition diagram is a *finite deterministic automat* (also called *Mealy automat*). Usually, there is a loop at the zero state with $u_r = 0$ and $\boldsymbol{a}_r = \boldsymbol{0}$. The diagram is always connected, since every state can be reached from every other state with an appropriate information sequence. The generator matrix only influences the labeling of the branches.

The fundamental paths are clearly visible in the state transition diagram. Observe that a fundamental path starts at the zero state, leaves the zero state and ends when it returns to the zero state. According to Theorem 9.2, if the encoder is non-catastrophic, the minimum weight of the code sequence of all fundamental paths corresponds to the free distance.

9.9.2 The Weight Enumerator Function for Convolutional Codes

To calculate the error probability not only the free distance but also further weight parameters are required, similar as for the block codes. For convolutional codes it is necessary to have an overview of the complete distance structure of the code, i.e., all fundamental paths should be described by an analytical expression.

Definition 9.3. *Let $t(d,i,j)$ be the number of fundamental paths which start at step zero with*

$$\begin{aligned} d &= \textit{Hamming weight of the sequence of encoded blocks} \\ i &= \textit{Hamming weight of the information sequence} \\ j &= \textit{length of the fundamental path,} \end{aligned}$$

where the length of the fundamental path is measured in code blocks or equivalently in information bits. The $t(d,i,j)$ are called the complete path enumerators. *The formal power series of D,I,J*

$$T(D,I,J) = \sum_{d=d_\mathrm{f}}^{\infty} \sum_{i=1}^{\infty} \sum_{j=m+1}^{\infty} t(d,i,j) D^d I^i J^j \tag{9.9.1}$$

is called the weight enumerator *of the convolutional code (there are many other names in usage, including* complete path enumerator*,* distance transfer function*,* generating function*,* transmission gain *or* code spectrum*).*

The powers and the weight coefficients are non-negative integers, i.e., $T(D,I,J)$ is without $\mathbb{F}_2$ arithmetic. Obviously every fundamental path has a code sequence weight of at least d_f, an information sequence weight of at least 1 and a length of at least $m+1$. So, the free distance is the minimum power of D in $T(D,I,J)$. Every fundamental path is uniquely mapped, but not injective, to a term $D^d I^i J^j$, since there can be various paths with the same combination of d,i,j.

For the actual calculation of the weight enumerator we will introduce another state in the *modified state transition diagram* (also called detour flowchart) by splitting the zero state into

$$\begin{aligned} \zeta_1 &= \text{zero state of the outgoing branches} \\ \zeta_0 &= \text{zero state of the incoming branches.} \end{aligned}$$

The direct branch from ζ_1 to ζ_0 (corresponding to $u_r = 0$, $\boldsymbol{a}_r = \boldsymbol{0}$) is omitted. Thus, every path from ζ_1 to ζ_0 is a fundamental path. Every branch in the modified state transition diagram is labeled with $D^d I^i J$ as given in Definition 9.3, where $0 \le d \le n$ for the code block weight and $0 \le i \le 1$ for the information bit weight.

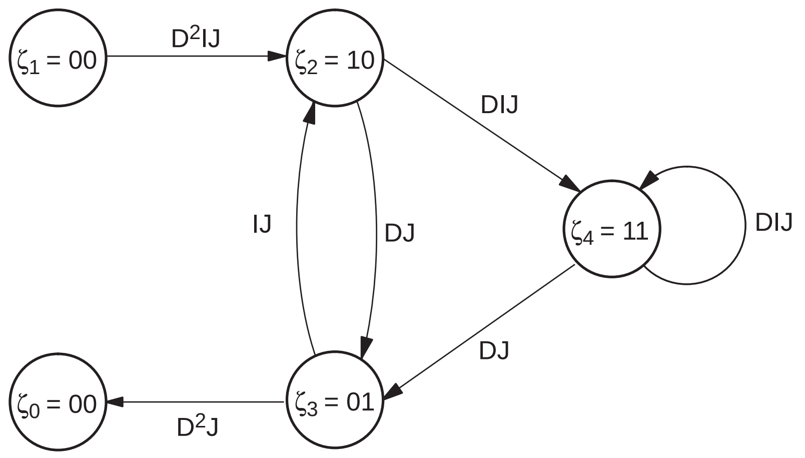


Figure 9.13. Modified state transition diagram for the standard example

Figure 9.11 illustrates the modified state transition diagram for the standard example. Some fundamental paths are listed below.

$$\begin{array}{lcll}
\zeta_1,\zeta_2,\zeta_3,\zeta_0 & : & D^2IJ\cdot DJ\cdot D^2J & = & D^5IJ^3\\
\zeta_1,\zeta_2,\zeta_3,\zeta_2,\zeta_4,\zeta_3,\zeta_0 & : & D^2IJ\cdot DJ\cdot IJ\cdot DIJ\cdot DJ\cdot D^2J & = & D^7I^3J^6\\
\zeta_1,\zeta_2,\zeta_4,\zeta_3,\zeta_2,\zeta_3,\zeta_0 & : & D^2IJ\cdot DIJ\cdot DJ\cdot IJ\cdot DJ\cdot D^2J & = & D^7I^3J^6.
\end{array}$$

Thus, there are at least two complete paths of length 6 with a code sequence weight of 7 and an information sequence weight of 3. The positions of the zeros and ones in the code sequence are not recorded, since this information is only of interest during decoding and not for the distance structure. For the calculation of the error probability it is not important where in the code sequence the zeros and ones are.

To calculate $T(D,I,J)$ we will use a standard signal flowchart technique. A formal *state variable* X_s $(0 \le s \le 2^m)$ is assigned to each state with the following linear equations $(r \neq 1)$:

$$X_r = \sum_{\substack{s \\ \text{branch from } \zeta_s \text{ to } \zeta_r}} D^d I^i J \cdot X_s. \tag{9.9.2}$$

The sum is only over those two values of s for which there is a branch from ζ_s towards ζ_r; from this branch d and i are determined. Since ζ_1 has no predecessing state, $r \neq 1$. Thus, we have a system of equations of the state variables X_s, and since $T(D,I,J)$ describes the paths from ζ_1 to ζ_0, we obtain the weight enumerator

$$T(D,I,J) = \frac{X_0}{X_1}. \tag{9.9.3}$$

Example 9.13. For the standard example in Figure 9.11 the system of the state equations is obtained as

$$X_2 = D^2IJ \cdot X_1 + IJ \cdot X_3 \tag{9.9.4}$$

$$X_3 = DJ \cdot X_2 + DJ \cdot X_4 \tag{9.9.5}$$

$$X_4 = DIJ \cdot X_2 + DIJ \cdot X_4 \tag{9.9.6}$$

$$X_0 = D^2J \cdot X_3. \tag{9.9.7}$$

(9.7.6) implies that

$$X_4 = \frac{DIJ}{1-DIJ} X_2. \tag{9.9.8}$$

(9.7.5) and (9.7.8) imply that

$$X_3 = \left(DJ + DJ\frac{DIJ}{1-DIJ} \right) X_2 = \frac{DJ}{1-DIJ} X_2. \tag{9.9.9}$$

(9.7.9) and (9.7.4) imply that

$$X_2 = \frac{1-DIJ}{DJ} X_3 = D^2IJ \cdot X_1 + IJ \cdot X_3. \tag{9.9.10}$$

Implying that

$$\left(\frac{1-DIJ}{DJ} - IJ \right) X_3 = D^2IJ \cdot X_1, \tag{9.9.11}$$

$$\frac{X_3}{X_1} = \frac{D^2IJ}{\dfrac{1-DIJ}{DJ} - IJ} = \frac{D^3IJ^2}{1-DIJ-DIJ^2}. \tag{9.9.12}$$

Finally, (9.7.7) and (9.7.12) imply that

$$T(D,I,J) = \frac{X_0}{X_1} = D^2J\frac{X_3}{X_1} = \frac{D^5IJ^3}{1-DIJ(1+J)}. \tag{9.9.13}$$

Formally $\dfrac{1}{1-\omega} = \sum_{r=0}^{\infty} \omega^r$ together with $\omega = DIJ(1+J)$ implying that

$$\begin{aligned} T(D,I,J) &= D^5IJ^3 \sum_{r=0}^{\infty} \Big(DIJ(1+J) \Big)^r \\ &= \sum_{r=0}^{\infty} D^{5+r} I^{1+r} J^{3+r} (1+J)^r \\ &= D^5IJ^3 + D^6I^2J^4(1+J) + D^7I^3J^5(1+J)^2 \\ &\quad + D^8I^4J^6(1+J)^3 + (\ldots)J^7 \\ &= D^5IJ^3 + D^6I^2J^4 + D^6I^2J^5 + D^7I^3J^5 \\ &\quad + 2D^7I^3J^6 + D^8I^4J^6 + (\ldots)J^7, \end{aligned} \tag{9.9.14}$$

where the fundamental paths are sorted by the powers of J (or their length) and by the powers of D. ■

Definition 9.4. *A simplification of the weight enumerator by suppressing J results in the so-called* distance spectra *which are sequences w_d and c_d of integers. The expansion in the powers of D leads to*

$$T(D,1,1) = \sum_{d=d_\mathrm{f}}^{\infty} w_d D^d \quad \textit{with} \quad w_d = \sum_{i,j} t(d,i,j) \tag{9.9.15}$$

and the differentiation with respect to I leads to

$$\frac{\partial T}{\partial I}(D,1,1) = \sum_{d=d_\mathrm{f}}^{\infty} c_d D^d \quad \textit{with} \quad c_d = \sum_{i,j} i \cdot t(d,i,j). \tag{9.9.16}$$

The spectral component w_d represents the number of fundamental paths with an encoded sequence Hamming weight of d, and c_d includes additionally a weighting with the information sequence Hamming weight.

Example 9.14. We continuation Example 9.10.

$$T(D,I,J) = \frac{D^5 I J^3}{1 - DIJ(1+J)} = \sum_{r=0}^{\infty} D^{5+r} I^{1+r} J^{3+r} (1+J)^r$$

implies that

$$T(D,1,1) = \frac{D^5}{1-2D} = D^5 \sum_{r=0}^{\infty} 2^r D^r = D^5 + 2D^6 + 4D^7 + \cdots \tag{9.9.17}$$

and from
$$\frac{\partial T}{\partial I} = \frac{D^5 J^3 (1 - DIJ(1+J)) + D^5 I J^3 \cdot DJ(1+J)}{(1 - DIJ(1+J))^2}$$
we conclude that

$$\begin{aligned} \frac{\partial T}{\partial I}(D,1,1) &= \frac{D^5}{1 - 4D(1-D)} = D^5 \sum_{r=0}^{\infty} 4^r D^r (1-D)^r \\ &= D^5 + 4D^6 + 12D^7 + \cdots . \end{aligned} \tag{9.9.18}$$

Thus we obtain the distance spectra

d	5	6	7	8	9	10	11	12	13	14	15	16	$\cdots$
w_d	1	2	4	8	16	32	64	128	256	512	1024	2028	$\cdots$
c_d	1	4	12	32	80	192	448	1024	2304	5120	11264	24576	$\cdots$

where $(1-D)^r = \sum_{s=0}^{r} \binom{r}{s} (-D)^s$ could be used to simplify the numerical evaluation of c_d. ■

Even for convolutional codes which are only a little more complicated, the calculation of the distance spectra can not be done without the help of computers. A table of distance spectra of the optimum convolutional codes can be found, e.g., in [25, 95].

9.9.3 Algorithms for Computing the Weight Enumerator Function

Only for very simple convolutional codes $T(D, I, J)$ can be calculated using a closed analytical expression. For practical purposes only the lower powers need to be known. For this we will develop two methods of calculation below.

The state equations (9.7.2) are put into a matrix equation. Let $\boldsymbol{X} = (X_2, \ldots, X_{2^m})^T$ be the $(2^m - 1)$-dimensional state vector, together with a $(2^m - 1, 2^m - 1)$-dimensional matrix $\boldsymbol{S}$ and two $(2^m - 1)$-dimensional column vectors $\boldsymbol{a}$ and $\boldsymbol{b}$

$$\begin{pmatrix} X_0 \\ \hline \boldsymbol{X} \end{pmatrix} = J \cdot \left(\begin{array}{c|c} 0 & \boldsymbol{a}^T \\ \hline \boldsymbol{b} & \boldsymbol{S} \end{array}\right) \cdot \begin{pmatrix} X_1 \\ \hline \boldsymbol{X} \end{pmatrix}. \tag{9.9.19}$$

Since every state can only have 2 branches from and 2 branches towards itself, the $(2^m, 2^m)$-dimensional matrix is only sparsely occupied. Written in full the equation system is:

$$\begin{aligned} X_0 &= J \cdot \boldsymbol{a}^T \boldsymbol{X} \\ \boldsymbol{X} &= J \cdot \boldsymbol{b} X_1 + J \cdot \boldsymbol{S}\boldsymbol{X}. \end{aligned} \tag{9.9.20}$$

The last equation implies that

$$(\boldsymbol{E} - J\boldsymbol{S})\boldsymbol{X} = J\boldsymbol{b}X_1 \quad \text{or} \quad \boldsymbol{X} = J(\boldsymbol{E} - J\boldsymbol{S})^{-1}\boldsymbol{b}X_1.$$

Thus we obtain a closed expression for the weight enumerator which however cannot yet be effectively numerically calculated:

$$T(D, I, J) = \frac{X_0}{X_1} = J^2 \cdot \boldsymbol{a}^T(\boldsymbol{E} - J\boldsymbol{S})^{-1}\boldsymbol{b}. \tag{9.9.21}$$

The formula for the geometric series is also valid for matrices leading to a representation of the weight enumerator as a power series:

$$T(D, I, J) = \sum_{r=0}^{\infty} J^{2+r} \cdot \boldsymbol{a}^T \boldsymbol{S}^r \boldsymbol{b}. \tag{9.9.22}$$

With $\boldsymbol{d}_r = \boldsymbol{S}^r\boldsymbol{b}$ we finally obtain a recursion which is easy to handle:

$$T(D, I, J) = \sum_{r=0}^{\infty} J^{2+r} \cdot \boldsymbol{a}^T \boldsymbol{d}_r \quad , \quad \boldsymbol{d}_r = \boldsymbol{S}\boldsymbol{d}_{r-1} \quad , \quad \boldsymbol{d}_0 = \boldsymbol{b}. \tag{9.9.23}$$

Example 9.15. Continuation of Example 9.10: the equation system as in (9.7.19) is

$$\begin{pmatrix} X_0 \\ \hline X_2 \\ X_3 \\ X_4 \end{pmatrix} = \left(\begin{array}{c|ccc} & & D^2J & \\ \hline D^2IJ & & IJ & \\ & DJ & & DJ \\ & DIJ & & DIJ \end{array}\right) \cdot \begin{pmatrix} X_1 \\ \hline X_2 \\ X_3 \\ X_4 \end{pmatrix}$$

thus

$$\boldsymbol{a} = \begin{pmatrix} 0 \\ D^2 \\ 0 \end{pmatrix} \quad \boldsymbol{b} = \begin{pmatrix} D^2 I \\ 0 \\ 0 \end{pmatrix} \quad \boldsymbol{S} = \begin{pmatrix} 0 & I & 0 \\ D & 0 & D \\ DI & 0 & DI \end{pmatrix}.$$

Recursive evaluation leads to the results

$$\boldsymbol{d}_0 = \boldsymbol{b} = \begin{pmatrix} D^2 I \\ 0 \\ 0 \end{pmatrix} \qquad \boldsymbol{a}^T \boldsymbol{d}_0 = 0$$

$$\boldsymbol{d}_1 = \boldsymbol{S}\boldsymbol{d}_0 = \begin{pmatrix} 0 \\ D^3 I \\ D^3 I^2 \end{pmatrix} \qquad \boldsymbol{a}^T \boldsymbol{d}_1 = D^5 I$$

$$\boldsymbol{d}_2 = \boldsymbol{S}\boldsymbol{d}_1 = \begin{pmatrix} D^3 I^2 \\ D^4 I^2 \\ D^4 I^3 \end{pmatrix} \qquad \boldsymbol{a}^T \boldsymbol{d}_2 = D^6 I^2$$

$$\boldsymbol{d}_3 = \boldsymbol{S}\boldsymbol{d}_2 = \begin{pmatrix} D^3 I^3 \\ D^4 I^2 + D^5 I^3 \\ D^4 I^3 + D^5 I^4 \end{pmatrix} \qquad \boldsymbol{a}^T \boldsymbol{d}_3 = D^6 I^2 + D^7 I^3.$$

By summing the $\boldsymbol{a}^T \boldsymbol{d}_r$ weighted by J^{2+r} we obtain the same result of the weight enumerator as in (9.7.14). ■

Below we will develop an alternative method for calculating the weight enumerator, where $\sigma = 2^m - 1$ will be an abbreviation for the dimension of the matrix $\boldsymbol{S}$. The characteristic polynomial is written as

$$\begin{aligned} f(\lambda) = \det(\lambda \boldsymbol{E} - \boldsymbol{S}) &= \sum_{i=0}^{\sigma} \alpha_i \lambda^i \\ &= \alpha_0 + \alpha_1 \lambda + \cdots + \alpha_{\sigma-1}\lambda^{\sigma-1} + \lambda^\sigma, \end{aligned} \tag{9.9.24}$$

where λ is a scalar placeholder and the coefficients α_i are polynomials in D, I, J and can be easily obtained by calculating the determinant. According to the Cayley-Hamilton theorem, the matrix $\boldsymbol{S}$ satisfies its characteristic polynomial, i.e.,

$$f(\boldsymbol{S}) = \alpha_0 \boldsymbol{E} + \alpha_1 \boldsymbol{S} + \cdots + \alpha_{\sigma-1}\boldsymbol{S}^{\sigma-1} + \boldsymbol{S}^\sigma = \boldsymbol{0}. \tag{9.9.25}$$

The multiplication with $\boldsymbol{S}^{r-\sigma}$ results in

$$\boldsymbol{S}^r = -\Big(\alpha_0 \boldsymbol{S}^{r-\sigma} + \alpha_1 \boldsymbol{S}^{r-\sigma+1} + \cdots + \alpha_{\sigma-1}\boldsymbol{S}^{r-1}\Big).$$

Thus, for the polynomials $h_r = \boldsymbol{a}^T \boldsymbol{S}^r \boldsymbol{b}$ we have the recursion

$$h_r = -\Big(\alpha_0 h_{r-\sigma} + \alpha_1 h_{r-\sigma+1} + \cdots + \alpha_{\sigma-1} h_{r-1}\Big) \tag{9.9.26}$$

and according to (9.7.22) the result is

$$T(D, I, J) = \sum_{r=0}^{\infty} J^{2+r} h_r. \tag{9.9.27}$$

9.10 Problems

9.1. Prove that the standard example $\boldsymbol{G}(x) = (1+x+x^2, 1+x^2)$ is the unique solution as a non-catastrophic optimum rate-1/2 code with $m = 2$.

9.2. Calculate the free distance of $\boldsymbol{G}(x) = (1, 1+x+x^2+x^3)$ directly without the weight enumerator.

9.3. Do

$$\begin{aligned} \boldsymbol{G}_1(x) &= (1+x+x^2, 1+x+x^2+x^3) \\ \boldsymbol{G}_2(x) &= (1+x+x^2+x^3+x^4, 1+x^4) \end{aligned}$$

create non-catastrophic encoders?

9.4. Determine the modified state transition diagram for the systematic encoder $\boldsymbol{G}(x) = (1, 1+x+x^2)$. Calculate d_f.

9.5. Calculate the weight enumerator and the distance spectra for the encoder $\boldsymbol{G}(x) = (1, 1+x)$.

9.6. Draw a trellis segment for the encoder $\boldsymbol{G}(x) = (1+x+x^3, 1+x+x^2+x^3)$ given in Table 9.3a.

9.7. Calculate the encoder inverse of the industry standard code

$$\boldsymbol{G}(x) = (1+x^2+x^3+x^5+x^6, 1+x+x^2+x^3+x^6)$$

by using the Euclidean algorithm (see Section A.?).

9.8. Find the polynomial description of the rate-2/3 convolutional encoder in Figure 9.1 as well as an encoder inverse (by trial and error). Determine the shift register implementation of the encoder inverse.

9.9. For the standard example $\boldsymbol{G}(x) = (1+x+x^2, 1+x^2)$ prove the existence of the so-called *expansion factor* δ, so that for all pairs of information and code sequences

$$w_H(\boldsymbol{a}) = w_H(\boldsymbol{u}) + \delta. \tag{9.10.1}$$

Calculate δ.

9.10. In extension of Theorem 9.1 prove the following equivalent characterization: a convolutional encoder is non-catastrophic if and only if there exists an encoder inverse without feedbacks.

9.11. In extension of Theorem 9.1 and Problem 9.10 prove the following equivalent characterization: a convolutional encoder is non-catastrophic if and only if there is no path loop with a code sequence weight of zero in the state transition diagram.

By using an example, show that the loop of length 1 in the 11-state does not result in an equivalent characterization, i.e., find a catastrophic encoder in which the loop of length 1 in the 11-state has a positive code block weight.

9.12. Let $p(x) \in \mathbb{F}_2[x]_s$ be a polynomial of degree s with coefficients in $\mathbb{F}_2$ and $p_0 = p(0) = 1$. Prove that

$$\frac{1}{p(x)} = u(x) = \sum_{r=0}^{\infty} u_r x^r$$

can be described by a power series without negative powers which was already used for the proof of Theorem 9.1. Create the coefficients u_r by using a feedback shift register.

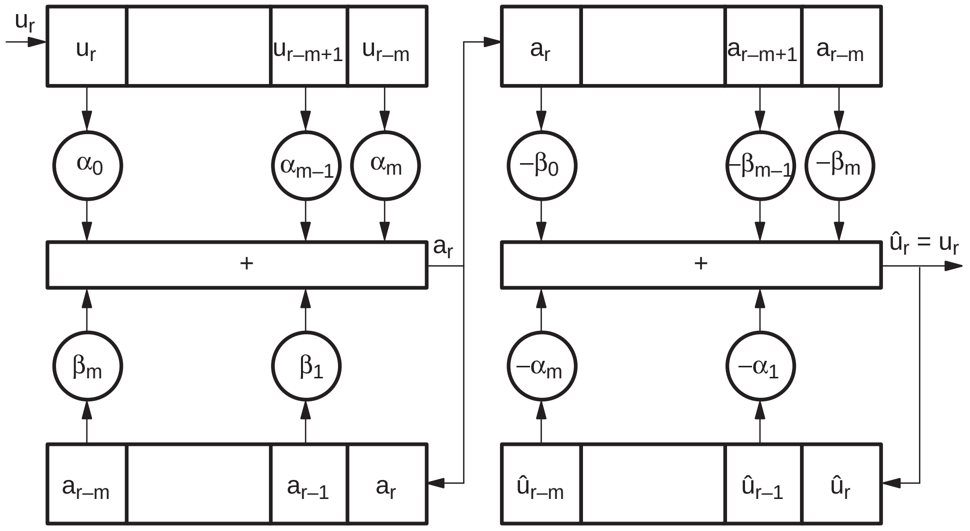


Figure 9.14. A transparent concatenation of feedback shift registers

9.13. Prove that the combination of the rate-1/1 encoder and its encoder inverse in Figure 9.12 is transparent (let $\alpha_0 = 1$ and $\beta_0 = -1$). Find the polynomial representations. Rearrange the feedback shift registers so that a central register chain of length $m + 1$ is sufficient.

9.14. Determine a systematic encoder $\boldsymbol{G}_s(x)$ with feedbacks for the standard example with $\boldsymbol{G}(x) = (1 + x + x^2, 1 + x^2)$. Is the encoder catastrophic? Prove that $\boldsymbol{G}(x)$ and $\boldsymbol{G}_s(x)$ create exactly the same code. Calculate an encoder inverse.

9.15. By using a numerical example prove that the rate-1/1 convolutional encoder $\boldsymbol{G}(x) = (1 + x)$ is catastrophic. Determine an encoder inverse with feedbacks similar to Figures 9.4 and 9.12. What happens if the encoder and the encoder inverse are swapped?

9.16. All generator polynomials of the encoder are replaced by their reciprocal polynomials, see (A.6.3). Prove that this does not change the weight distribution of the convolutional code.