each of these treated independently from each other to form length n codewords. This is the block coding approach. If the input sequence is instead viewed as an infinite sequence and the the redundancy is added along the way, the convolutional coding approach arise. It was first presented by Peter Elias in 1955 [12]. The sequence is processed by a linear system where the number of outputs exceeds the number of inputs. In this way the resulting sequence, the code sequence, has a built in redundancy related to the difference in input and output length as well as the memory of the linear system. In the next example one of the most popular encoders is shown. In this text only binary sequences are considered, as well as mainly encoders with one input sequence and two output sequences. However, at some points a more general case is assumed.

Consider a binary (infinite) sequence

$$\boldsymbol{x} = x_0 x_1 x_2 x_3 \ldots \tag{7.67}$$

Then the sequence is fed to a linear circuit, or an encoder, with one input and two outputs. For each bit in the input sequence the output consists of two bits. That is, the output can be written as

$$\boldsymbol{y} = y_0^{(0)} y_0^{(1)}\ y_1^{(0)} y_1^{(1)}\ y_2^{(0)} y_2^{(1)}\ y_3^{(0)} y_3^{(1)} \ldots \tag{7.68}$$

In the next example an encoder for such system is shown.

EXAMPLE 7.10 In Figure 7.3 one of the most common examples of a convolutional encoder is shown. Assuming that the encoder starts in the all-zero state, the input sequence

$$\boldsymbol{x} = 10100000 \ldots \tag{7.69}$$

will give the output sequence

$$\boldsymbol{y} = 11\ 10\ 00\ 10\ 11\ 00\ 00\ 00 \ldots \tag{7.70}$$

From system theory it is well known that the output sequences can be derived as the convolution of the input sequence and the impulse responses,

$$\boldsymbol{y}^{(0)} = \boldsymbol{x} * (111) \tag{7.71}$$

$$\boldsymbol{y}^{(1)} = \boldsymbol{x} * (101) \tag{7.72}$$

$$\tag{7.73}$$

hence the name *convolutional codes*. The impulse responses (111) and (101) are often described in octal form giving 7 and 5. This specific encoder is therefore often mentioned as the $(7,5)$-encoder.

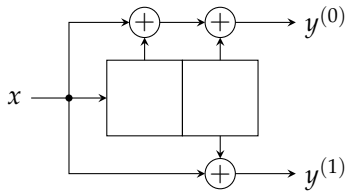

Figure 7.3: Circuit for the $(7,5)$ encoder.

By assuming a length m shift register instead of 2 as in the previous example a more general relation between the input sequence and the output sequence can be obtained, see Figure 7.4. The input sequence is fed to the shift register and at each level the symbol is multiplied with a constant g_i, $i = 1, 2, \ldots, m$. For the binary case, when having one input and one output, the multiplier means either a connection or no connection for the values 1 and 0, respectively. The output bit at time k can be derived as

$$y_k = x_k g_0 + x_{k-1} g_1 + \cdots + x_{k-m} g_m = \sum_{i=0}^{m} x_{k-i} g_i \pmod 2 \tag{7.74}$$

This relation shows the output sequence is the convolution of the input sequence and the impulse response, i.e.

$$\boldsymbol{y} = \boldsymbol{x} * (g_0 g_1 g_2 \cdots g_{m-1} g_m) \tag{7.75}$$

as was used also in the previous example. In general, the circuit of Figure 7.4 can be used to describe a code where at each time instant b information bits are encode into c code bits. At time k, x_k is a binary row vector of length b and y_k a binary row vector of length c. The coefficients in the figure are represented by $b \times c$ binary matrices. Hence, the $(7,5)$ encoder in Example 7.10 is described by the coefficient matrices

$$g_0 = \begin{pmatrix} 1 & 1 \end{pmatrix} \quad g_1 = \begin{pmatrix} 1 & 0 \end{pmatrix} \quad g_2 = \begin{pmatrix} 1 & 1 \end{pmatrix} \tag{7.76}$$

The convolution identified in (7.75) can be derived by a matrix multiplication. Assuming that the encoder starts in the all-zero state at time $k = 0$ the output from the information sequence $\boldsymbol{x} = (x_0 x_1 x_2 \ldots)$

$$\begin{aligned} \boldsymbol{y} &= \boldsymbol{x} G \\ &= (x_0 x_1 x_2 \ldots) \begin{pmatrix} g_0 & g_1 & g_2 & \cdots & g_{m-1} & g_m & & & & \\ & g_0 & g_1 & g_2 & \cdots & g_{m-1} & g_m & & & \\ & & g_0 & g_1 & g_2 & \cdots & g_{m-1} & g_m & & \\ & & & g_0 & g_1 & g_2 & \cdots & g_{m-1} & g_m & \\ & & & & \ddots & \ddots & \ddots & & \ddots & \ddots \end{pmatrix} \end{aligned} \tag{7.77}$$

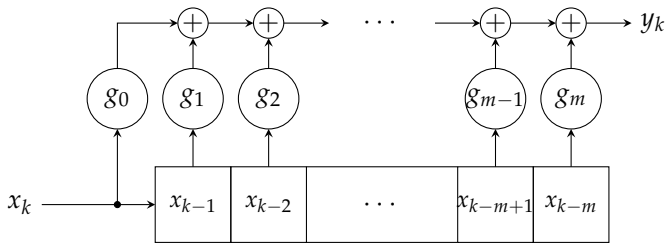

Figure 7.4: A linear circuit as encoder.

where G is the generator matrix in the time domain.

EXAMPLE 7.11 For the $(7,5)$-encoder in Example 7.10 the generator matrix is

$$G = \begin{pmatrix} 1\,1 & 1\,0 & 1\,1 & & & & \\ & 1\,1 & 1\,0 & 1\,1 & & & \\ & & 1\,1 & 1\,0 & 1\,1 & & \\ & & & 1\,1 & 1\,0 & 1\,1 & \\ & & & & 1\,1 & 1\,0 & 1\,1 \\ & & & & & \ddots & \ddots & \ddots \end{pmatrix} \tag{7.78}$$

Hence, encoding the sequence

$$\boldsymbol{x} = 101000\ldots \tag{7.79}$$

is equivalent to adding row one and three in the generator matrix to get

$$\boldsymbol{y} = 1\,1 \quad 1\,0 \quad 0\,0 \quad 1\,0 \quad 1\,1 \quad 0\,0 \quad 0\,0 \quad \ldots \tag{7.80}$$

The rate of a convolutional code is the ratio between the number of inputs and the number of outputs,

$$R = \frac{b}{c} \tag{7.81}$$

As an example, the $(7,5)$ encoder gives a rate $R = \frac{1}{2}$ code.

7.2.1 Decoding of convolutional codes

So far it is the encoder circuit that has been treated. The code is, as for block codes, the set of codewords. Since the information sequences are infinite sequences, so are the codewords. This also means that the the number of codewords is infinite. This fact might be seen as an obstacle when it comes to decoding, as for an ML decoder the received sequence should be compared with

all possible code sequences. It turns out that there is a very clever structure to compare all code sequences and with that a simple method to perform ML decoding. The decoding algorithm is the Viterbi algorithm[66] which was published in April 1967. However, at that point it was not fully understod that the algorithm was neither optimal nor practically implementable. In December the same year Forney published a technical report [15] where a structure called *Trellis* was introduced. With this it was easy to compare the different code sequences in a convolutional code. It was also clear that the Viterbi algorithm was indeed optimal. The results was later published in [16]. Since the complexity of the algorithm grows exponentially with the memory in the encoder, it was still not seen as a practical alternative. It was not until late 1968 when Heller published the first simulation results for relatively short convolutional codes [29] this view was changed. Today almost all implementations containing convolutional codes rely on the trellis structure and the Viterbi algorithm, in one way or another. Convolutional codes are also used for concatenation of codes, e.g. Turbo codes. Then, an iterative decoding procedure, based on a MAP decoding algorithm[3] is used, often called the BCJR algorithm from the inventors. This MAP algorithm also uses a the trellis structure and can, in principle, be seen as a two sweep Viterbi algorithm. In the next, the Trellis structure will be introduced first and then the Viterbi algorithm.

To start describing the trellis structure, again assume the $(7, 5)$ encoder in Example 7.10. The two memory elements in this circuit represents the memory of the code, which is called the state. This state represents everything the encoder needs to know about the past symbols in the sequence. The output and the next state at a certain time are both functions of the symbols in the memory and the current input. These two functions can be viewed in a state transition graph, as depicted in Figure 7.5.

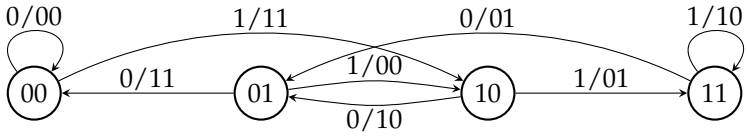

Figure 7.5: State transition graph for $(7, 5)$ encoder.

Here it is seen that if the current state is the all-zero state and the input is 0, the next state is the all-zero state and the output 00. If on the other hand the input is 1 the next state is 10 and the output 11. Continuing with the other three states the graph is obtained. In this way the graph describes the behaviour of the encoder circuit. Each information sequence represents a path in the graph, giving both a state sequence and a code sequence. Even though the graph gives a very good overview and a description of how the encoder works, the description of the sequences needs one more dimension, the time. To include this a two dimensional structure with the possible states listed vertically and the time

horisontal can be considered. That mean each state will exist once in every time instant, see Figure 7.6. This picture resembles the structure of a garden trellis, hence the name.

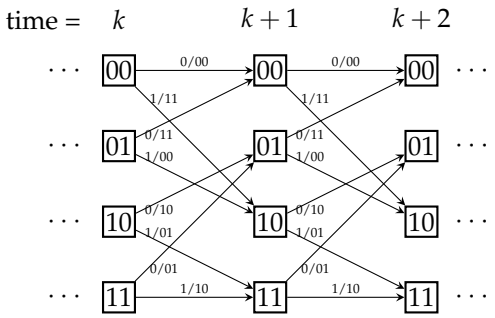

Figure 7.6: Trellis segments for $(7, 5)$ encoder.

The trellis in Figure 7.6 gives a way to describe all possible state sequences. Since there is a one to one mapping between the input sequences and the state sequences, and between the state sequences and the code sequences, this also gives a graphical view of *all possible code sequences*. Previously, the encoder was assumed to start in the all-zero state. Then at time $k = 0$ the state is known to be 00 and other states are not listed. At time $k = 1$ there are two possible states, 00 and 10. First in time $k = 2$, when the memory of the encoder has been filled, all states are possible. So to describe all codewords generated by a $(7, 5)$ encoder that starts in the zero state, use Figure 7.7. In Example 7.10 the information sequence $\boldsymbol{x} = 101000\ldots$ was used. By following this sequence in the trellis it is seen that the corresponding state sequence starts in state 00. As the first input is a 1 the next state is 10 and the output 11. The second input is 0 which gives the next state 01 and the output 10. Continuing this through the trellis the state sequence and code sequence

$$\boldsymbol{\sigma} = 00\ 10\ 01\ 10\ 01\ 00\ 00\ldots \tag{7.82}$$

$$\boldsymbol{y} = 11\ 10\ 00\ 10\ 11\ 00\ 00\ldots \tag{7.83}$$

The above sequences are marked as a path with bold edges in the trellis in Figure 7.7.

The error correcting capability of a convolutional code is determined by the minimum Hamming distance between two code sequences. In the next definition the free distance of the code is introduced. This directly translated to the minimum distance for block codes.

DEFINITION 7.5 The *free distance* for a convolutional code $\mathcal{C}$ is the minimum

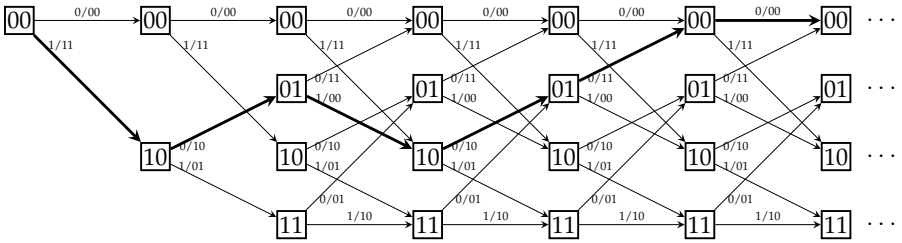

Figure 7.7: Trellis for $(7,5)$ encoder when the encoder is started at the all-zero state. The path marked with bold edges corresponds to the information sequence $\boldsymbol{x} = 101000\ldots$.

Hamming weight between two different code sequences

$$d_{\text{free}} = \min_{\substack{\boldsymbol{x}_1,\boldsymbol{x}_2\in\mathcal{C}\\ \boldsymbol{x}_1\neq\boldsymbol{x}_2}} d_H(\boldsymbol{x}_1,\boldsymbol{x}_2) \tag{7.84}$$

□

Since the mapping between the information sequences and code sequences is determined by a convolution, which is a linear mapping, the code is linear. That means, to derive the free distance it is not necessary to compare all code sequences with each other, it is enough to compare one sequence with all the other. If that fixed sequence is chosen as the all zero sequence the free distance can be derived as

$$d_{\text{free}} = \min_{\substack{\boldsymbol{x}\in\mathcal{C}\\ \boldsymbol{x}\neq\boldsymbol{0}}} w_H(\boldsymbol{x}) \tag{7.85}$$

With the free distance as the measure of separation between code sequences the same arguing as for block codes gives the following theorem on the error correction and detection capabilities for a convolutional code.

THEOREM 7.15 When using a convolutional code $\mathcal{C}$ with free distance d_{free}, it is always possible to either *detect* an error $\boldsymbol{e}$ if

$$w_H(\boldsymbol{e}) < d_{\text{free}} \tag{7.86}$$

or *correct* an error $\boldsymbol{e}$ if

$$w_H(\boldsymbol{e}) \leq \frac{d_{\text{free}}-1}{2} \tag{7.87}$$

□

EXAMPLE 7.12 By counting the number of ones along the non-zero paths in the trellis of Figure 7.7, the minimal weight is found to be 5. That gives the free distance $d_{\text{free}} = 5$, which is answered by e.g. the code sequence

$$\boldsymbol{y} = 11\ 10\ 11\ 00\ 00\ 00\ldots \tag{7.88}$$

Hence, by Theorem 7.15 it is seen that two errors can always be corrected by the code. Alternatively, any four errors can always be detected by the code.

In the description above it is assumed that the code sequences have infinite duration. Even though this exist also in practical implementations, e.g. some space applications, the information sequence is often split in finite duration vectors (or blocks). Then, each vector is encoded separately by first setting the encoder in the all-zero state and then feeding the information vector. To preserve the error correcting capability of the code, the encoder is driven back to the all-zero state after encoding the vector. With vectors of length K and an encoder with memory m the code sequences will be of length $K + m$. The trellis will then have one starting state at time $k = 0$ and one ending state at time $k = K + m$. For the $(7, 5)$ encoder such trellis is shown in Figure 7.8. To simplify the figure the labels of the branches are omitted.

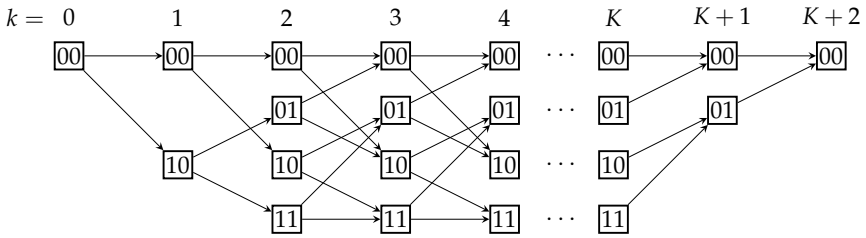

Figure 7.8: Trellis for $(7, 5)$ encoder when the encoder starts and terminates in the all-zero state. The information vector is K bits long and the code sequence $2(K + 2)$ bits.

Assuming the code vector is transmitted bitwise over a binary symmetric channel, the ML decoder can be implemented as a minimum distance decoder. Thus, the received (binary) vector should be compared to the possible transmitted vectors. The code symbols of the branches in the trellis are compared with the received symbols by using the Hamming distance. In this sense the trellis is a directed graph with the extra property that all states at a specific time has the same length from the starting state.

If both the starting state and ending state are known, e.g. the all-zero state, Viterbi's idea is as follows. Start in time $k = 0$ and state $\sigma = 00$ and let the metric for this state be $\mu_{00} = 0$. Then, for each time instance $k = \tau$ in the trellis,

label all branches to the next time instance $k = \tau + 1$ with the Hamming distance between the corresponding output and the received bits. For all the states at time $k = \tau$ there is a cumulative metric μ_σ for the lowest weight path from the starting state to this state. Then, for each state in time $k = \tau + 1$ there are two alternative paths from time $k = \tau$. The total weight for the path from the starting state is the metric for the previous state and branch weight. Keep only the brach corresponding to the lowest weight path for each state. Continuing this will at time $k = K + m$ result in one path with the least weight through the trellis. This represents the codeword with the least Hamming distance to the received vector.

ALGORITHM 7.2 (VITERBI) Let $\boldsymbol{y}$ be a code vector of length $K + m$, starting and terminating in the all-zero state, and let $\mathcal{S}_k$ be the possible states in time k. The code vector is transmitted over a BSC and the received vector denoted $\boldsymbol{r}$. Let $\mathcal{X}$ be the possible input vectors for the encoder and $S^+(\sigma, x)$ and $Y^+(\sigma, x)$ be the next state function and the output function when current state is σ and input is x. Then the Viterbi algorithm can be performed according to:

1) Initialisation:
   Let $k = 0$ and $\mu_0(\sigma = 0) = 0$
2) Expand:
   $k = k + 1$
   FOR EACH $\sigma \in \mathcal{S}_{k-1}$ and $x \in \mathcal{X}$
      $\mu = \mu_\sigma + d_H(Y(\sigma, x), r_k)$
      IF $\min\{\mu, \mu_{S^+(\sigma,x)}\} == \mu$
         $\mu_{S^+(\sigma,x)} = \mu$
         $BT_{S^+(\sigma,x)} = \sigma$
   IF $k = K + m$
      Back track from end state to starting state using BT-path to get $\hat{\boldsymbol{y}}$.
   ELSE GOTO 2)

In the case when there are two equally likely paths entering a state the surviving path should be chosen randomly.

The procedure of the Viterbi algorithm is best shown through an example. In the next example it is assumed a length four information vector is encoded by a $(7, 5)$ encoder. This is suitable for a text bok example, but in a real implementation the length of the vector should be much longer than the memory of the encoder. Otherwise the effective code rate will be considerably lowered. In the example the rate $R = 1/2$ encoder is used to encode four information bits to twelve code bits, giving an effective rate of $R_{\text{Eff}} = 4/12 \approx 0.67$. If

instead the length of the information vector is 500, the effective rate becomes $R_{\text{Eff}} = 500/1004 \approx 0.5$.

EXAMPLE 7.13 Assume the information vector $\boldsymbol{x} = 1011$ should be transmitted. To drive the encoder back to the zero state at the end two dummy-zeros are appended to form $\tilde{\boldsymbol{x}} = 1011\ 00$. Continuing with the $(7, 5)$ encoder, the code vector is

$$\boldsymbol{y} = \tilde{\boldsymbol{x}}G = 11\ 10\ 00\ 01\ 01\ 11 \tag{7.89}$$

Assume that two errors occur during the transmission; in the third and eight bit, so the received vector is

$$\boldsymbol{r} = 11\ 00\ 00\ 00\ 01\ 11 \tag{7.90}$$

According to (7.87) an MD decoder should be able to correct these two errors. The trellis in Figure 7.9 is used for decoding. For the first and second step in the algorithm there are no competing paths entering the states, and the metric of the paths are derived as the Hamming distance between the received path and the considered path. In the third step there are two branches entering each state. Starting with state 00, one path is coming from state 00 at the previous level and one from 01. The path from 00 has a cumulative metric of $\mu_{00} = 2$ and an additional branch metric of $d_H(00, 00) = 0$, which gives in total 2. The second path has a cumulative metric of $\mu_{01} = 1$ at time 2 and an addition of $d_H(11, 00) = 2$ which gives a total metric of 3. Hence, the first path has the least metric and therefore the second should be discarded. This is marked in the figure with a dashed line. Similarly, the state 01 at time 3 has two entering paths with total metric of $4 + 1 = 5$ and $1 + 1 = 2$, where the second is the surviving path and the first should be discarded. Continuing, there will be exactly one path entering each state, yielding the minimum cumulative metric. Since the Hamming distance is used as metric the paths entering each state represents the paths with minimum Hamming distance compared to the received vector up to this time. The fourth step in the trellis follows similarly. Then, at step five the input sequence is known to be a zero, appended to drive the encoder back to the all-zero state. Then there are only the possible states 00 and 01. For state 00 the two entering paths both have the metric 3, and then one of them should be chosen randomly, using e.g. coin flipping. In this example the lower path from 01 is chosen as the surviving path. In the last step in the trellis there is only one state left, 00. The surviving path entering this state corresponds to the path through the trellis with least metric. Following it back to the starting state gives the closest code vector as

$$\hat{\boldsymbol{y}} = 11\ 10\ 00\ 01\ 01\ 11 \tag{7.91}$$

which gives the most likely information vector

$$\hat{\boldsymbol{x}} = 1011\ (00) \tag{7.92}$$

The two inserted errors have been corrected by the algorithm, as was anticipated from the free distance of 5. If there are more errors the outcome depends on their distribution. If they are far apart in a long vector the code will probably be able to correct them, but if they are closely together in a burst there is a higher risk that the decoder will give an erroneous answer.

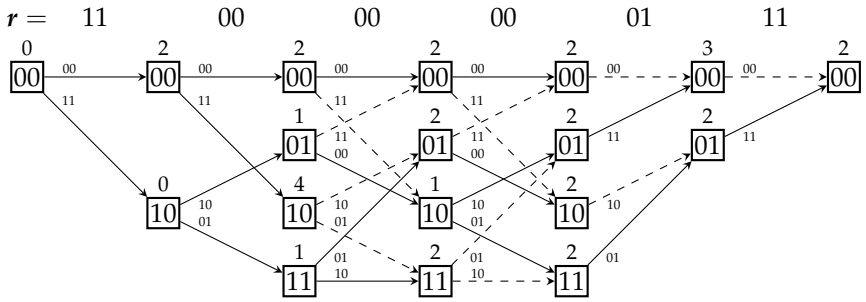

Figure 7.9: The trellis used to decode $\boldsymbol{r} = 11\ 00\ 00\ 00\ 01\ 11$ for the $(7,5)$ code.

7.2.2 D-transform representation

In the beginning of this section it was seen that the code sequence equals the information sequences convolved with the impulse response of the circuit. This convolution can be expressed as the multiplication by an (infinite) matrix. As in many applications, a convolution in the time domain is easier viewed as a multiplication in a transform domain. Since there is a finite number of amplitude levels, without order, the normal discrete time transforms for real or complex sequences, such as the discrete Fourier transform or the $\mathcal{Z}$-transform, cannot be used. Instead it is common to define a new transform, often named the $\mathcal{D}$-transform.[3]

DEFINITION 7.6 Consider a sequence

$$\boldsymbol{x} = x_0 x_1 x_2 x_3 \ldots \tag{7.93}$$

with or without starting and/or ending time. Then the $\mathcal{D}$-transform of the se-

[3] In a strict mathematical meaning it is doubtful that it should be called a transform. The variable D does not have a mathematical meaning as frequency in the Fourier transform or a complex number as in the $\mathcal{Z}$-transform. But for our purpose, considering sequences of elements from a finite field, the usage is very similar.

quence is

$$x(D) = x_0 + x_1 D + x_2 D^2 + x_3 D^3 + \cdots = \sum_{i=-\infty}^{\infty} x_i D^i \tag{7.94}$$

□

In the definition the sum is taken from $i = -\infty$ but it is often assumed that the sequences are causal, i.e. starting at time 0. This can be solved by arguing that the sequence is zero up to time 0. The variable D works as a position marker in the sense that the coefficient before D^k describes what is happening at time instant k. Next two important properties of the $\mathcal{D}$-transform will be derived. Firstly, a convolution in the time domain equals a multiplication in the transform domain, and secondly, the transform representation of periodic sequences.

Considering a sequence $\boldsymbol{x}$ that is fed to a linear circuit with impulse response $\boldsymbol{g}$, then the output sequence is given by the convolution $\boldsymbol{y} = \boldsymbol{x} * \boldsymbol{g}$. The symbol at time i in $\boldsymbol{y}$ can then be expressed as

$$y_i = \sum_j x_j g_{i-j} \tag{7.95}$$

Hence, the $\mathcal{D}$-transform of $\boldsymbol{y}$ becomes

$$\begin{aligned} y(D) &= \sum_i y_i D^i = \sum_i \sum_j x_j g_{i-j} D^i \\ &= \sum_j \sum_m x_j g_m D^{j+m} \\ &= \Big(\sum_j x_j D^j\Big)\Big(\sum_j g_m D^m\Big) = x(D)g(D) \end{aligned} \tag{7.96}$$

where in the third equality the summation order is interchanged and the variable change $m = i - j$ applied. This shows that a convolution in the time domain equals a multiplication in the D-domain.

A periodic sequence can be written as

$$[x_0 x_1 \ldots x_{n-1}]^\infty = x_0 x_1 \ldots x_{n-1} x_0 x_1 \ldots x_{n-1} \ldots \tag{7.97}$$

where $x_0 x_1 \ldots x_{n-1}$ is the periodically repeated sequence and n the period. To derive the $\mathcal{D}$-transform first consider a sequence with period n and only one 1,

$$\begin{aligned} [10\ldots0]^\infty &\to 1 + D^n + D^{2n} + \cdots \\ &= \frac{(1+D^n)(1+D^n+D^{2n}+\cdots)}{1+D^n} = \frac{1}{1+D^n} \end{aligned} \tag{7.98}$$

In the last equality there is also term D^M in the numerator, where M tends to infinity. Since M denotes the time instant, the term is vanishing in infinite time and will not affects the derivations.

By similar derivations, if the 1 is in position i, the $\mathcal{D}$-transform is

$$[0 \dots 010 \dots 0]^\infty \to \frac{D^i}{1+D^n} \tag{7.99}$$

Altogether, the $\mathcal{D}$-transform of a general periodic sequence is

$$\begin{aligned}[x_0 x_1 x_2 \dots x_{n-1}]^\infty &\to x_0 \frac{1}{1+D^n} + x_1 \frac{D}{1+D^n} + \dots + x_{n-1} \frac{D^{n-1}}{1+D^n} \\ &= \frac{x_0 + x_1 D + x_2 D^2 + \dots + x_{n-1} D^{n-1}}{1+D^n}\end{aligned} \tag{7.100}$$

That is, a periodical sequence is represented by a rational function in the D-transform, and vice versa. The next theorem summarises the properties of the D-transform.

THEOREM 7.16 For the $\mathcal{D}$-transform the following properties hold,

$$\boldsymbol{x} * \boldsymbol{g} \quad \xrightarrow{D} \quad x(D)g(D) \tag{7.101}$$

$$[x_0 x_1 \dots x_{n-1}]^\infty \quad \xrightarrow{D} \quad \frac{x_0 + x_1 D + \dots + x_{n-1} D^{n-1}}{1+D^n} \tag{7.102}$$

□

According to (7.75) the code sequence $\boldsymbol{y}$ is formed by a convolution with $\boldsymbol{g} = g_0 g_1 \dots g_m$, which can be written by the $\mathcal{D}$-transform as

$$G(D) = g_0 + G_1 D + \dots + g_m D^m \tag{7.103}$$

which is the *generator matrix*. The code sequence is formed as

$$y(D) = x(D)G(D) \tag{7.104}$$

The $(7,5)$ encoder is characterised by $g_0 = (1\ 1)$, $g_1 = (1\ 0)$ and $g_2 = (1\ 1)$ and the generator matrix becomes

$$G(D) = (1\ 1) + (1\ 0)D + (1\ 1)D^2 = (1 + D + D^2 \quad 1 + D^2) \tag{7.105}$$

There are several other encoders than the here described $(7,5)$ encoder. The generator matrix

$$G(D) = (1 + D + D^2 + D^3 \quad 1 + D + D^3) \tag{7.106}$$

describes an encoder with memory $m = 3$ and free distance $d_{\text{free}} = 6$. The vector for the coefficients of the polynomials are (1111) and (1101), which in octal representations are 17_8 and 15_8. The encoder is therefore mentioned as the $(17, 15)$ encoder. The circuit can also have more than one input sequence as for the encoder circuit in Figure 7.10. The number of inputs for the encoder equals the number of rows in the generator matrix. That is, each input corresponds to a row in the matrix and each output a column. In this case generator matrix becomes

$$G(D) = \begin{pmatrix} 1+D & D & 1 \\ D^2 & 1 & 1+D+D^2 \end{pmatrix} \tag{7.107}$$

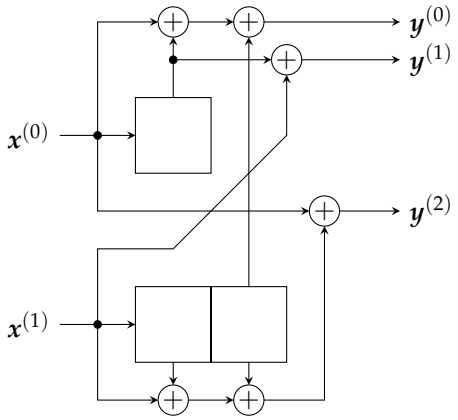

Figure 7.10: Encoder for the generator matrix $G(D)$ in (7.107).

Furthermore, if the circuit contains feedback the entries in the generator matrix becomes rational functions such as

$$G(D) = \begin{pmatrix} \frac{1+D+D^2}{1+D^2} & 1 \end{pmatrix} \tag{7.108}$$

and

$$G(D) = \begin{pmatrix} \frac{1}{1+D+D^2} & \frac{D}{1+D^3} & \frac{1}{1+D^3} \\ \frac{D^2}{1+D^3} & \frac{1}{1+D^3} & \frac{1}{1+D} \end{pmatrix} \tag{7.109}$$

From system theory the first generator generator matrix can be realised as in Figure 7.11.

In Figure 7.12 the resulting bit error rate after the decoder when using the Hamming code, described in the previous section, and three convolutional codes

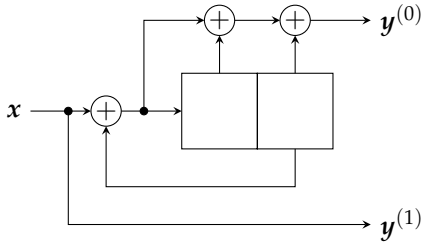

Figure 7.11: Encoder for the generator matrix $G(D)$ in (7.108).

with generator matrices

$$G(D) = \begin{pmatrix} 1+D & 1 \end{pmatrix} \tag{7.110}$$

$$G(D) = \begin{pmatrix} 1+D+D^2 & 1+D^2 \end{pmatrix} \tag{7.111}$$

$$G(D) = \begin{pmatrix} 1+D+D^2+D^3 & 1+D+D^3 \end{pmatrix} \tag{7.112}$$

In the figure they are mentioned as $(3,2)$, $(7,5)$ and $(17,15)$, respectively. The bit error rate is plotted against the signal to noise ratio E_b/N_0, where E_b is the energy per information bit and N_0 the Gaussian noise parameter (see Chapter 9). When using a binary antipodal signalling, i.e. BPSK, and hard decision at the receiver, the channel is modelled as a BSC with crossover probability

$$\epsilon = Q\left(\sqrt{2\frac{E_b}{N_0}R}\right) \tag{7.113}$$

where $Q(\cdot)$ is the error function

$$Q(x) = \int_x^\infty \frac{1}{\sqrt{2\pi}} e^{-z^2/2} dz \tag{7.114}$$

i.e. the probability that the outcome of a normalised Gaussian random variable exceeds x.

The purpose of the bit error rate plots in Figure 7.12 is to show that the encoding schemes considered in this chapter works. However, the described codes are very low complexity as well as performance. If the complexity is increased the performance will also increase. There are both block codes and convolutional codes today that perform very close to the Shannon limit. For a more thorough treatment of convolutional codes refer to e.g. [35, 43].

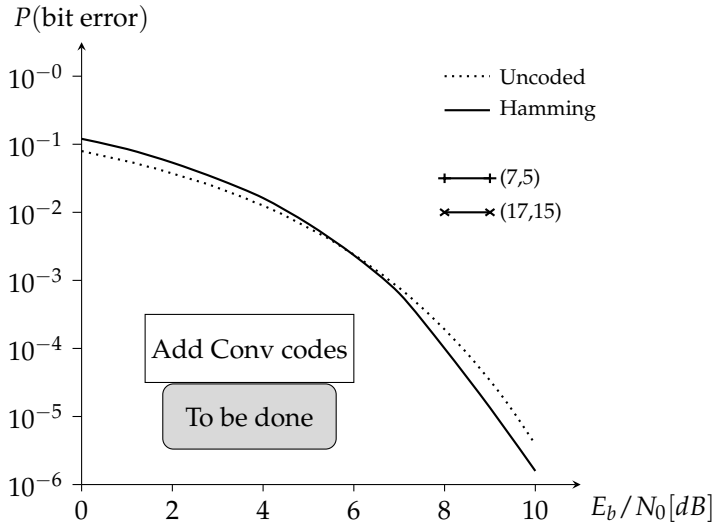

Figure 7.12: Plot of resulting bit error rate as a function of E_b/N_0 for the (16,7) Hamming code and three different convolutional codes with rate $R = 1/2$ and memory 1, 2 and 3. For comparison also the uncoded case is plotted. The channel is assumed to be a BSC.

7.2.3 Bounds on convolutional codes

To be done

Heller bound:

THEOREM 7.17 The free distance for a convolutional code with rate $R = b/c$ with encoder memory m satisfies

$$d_{\text{free}} \leq \min_{i \geq 1} \left\{ \left\lfloor \frac{(m+i)c}{2(1-2^{-bi})} \right\rfloor \right\} \tag{7.115}$$

□

$$\delta = \lim_{m \to \infty} \frac{d_{\text{free}}}{mc} \tag{7.116}$$

Heller's asymptotic bound: