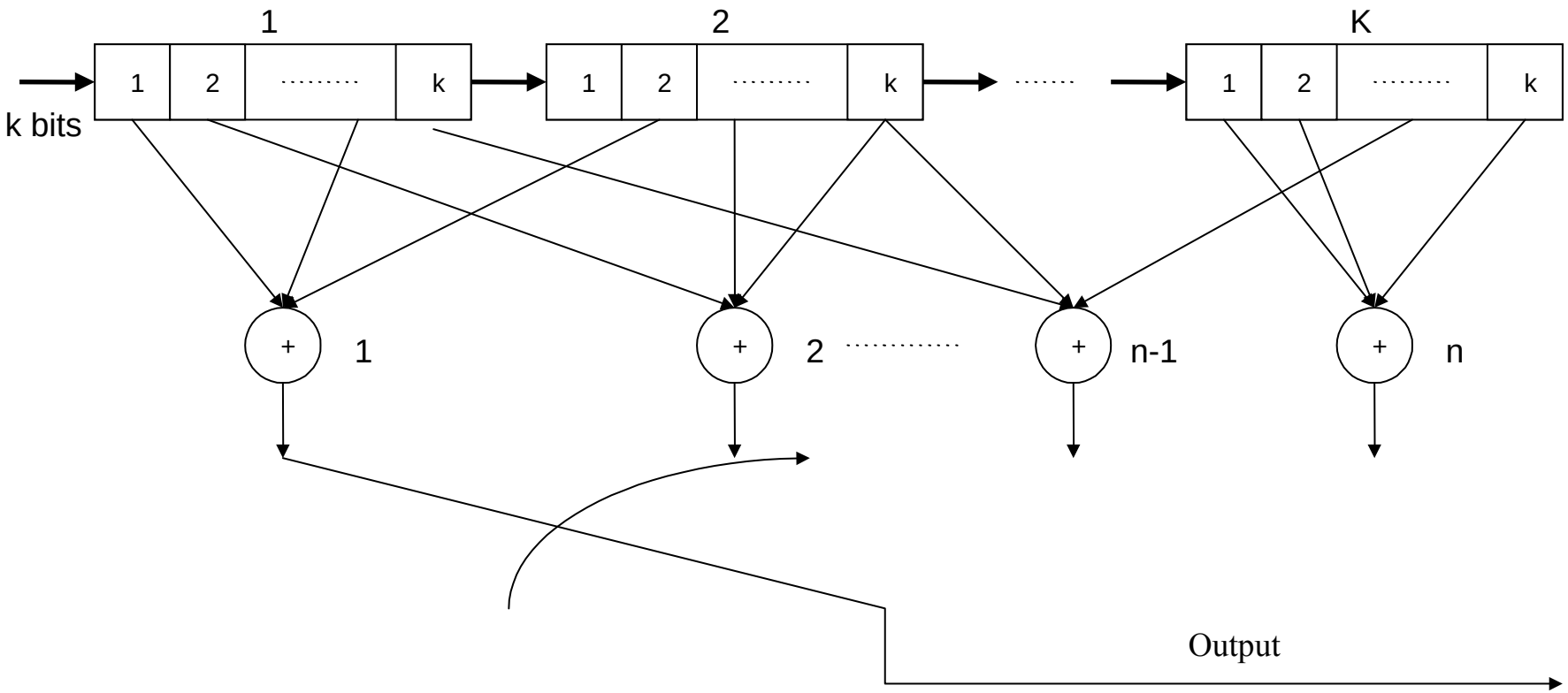
Example of Convolutional Codec

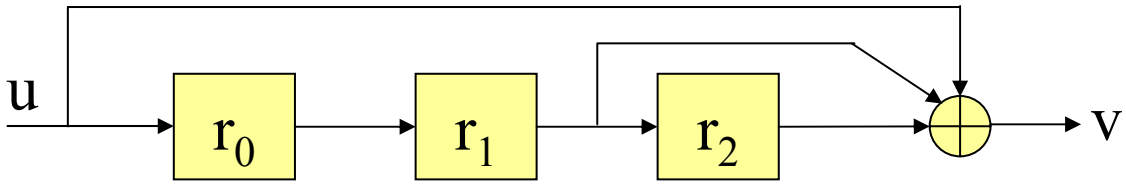
Convolutional Code Structure



Convoltuional Code

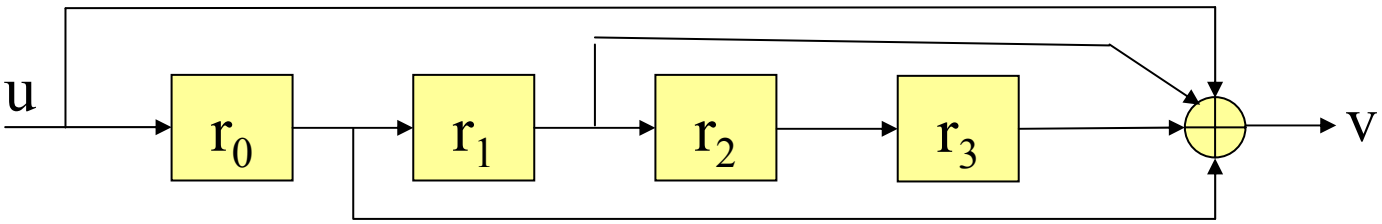
- Convolutional codes
  - k = number of bits shifted into the encoder at one time
    - k=1 is usually used!!
  - n = number of encoder output bits corresponding to the k information bits
  - $r = k/n$ = code rate
  - K = constraint length, encoder memory
- Each encoded bit is a function of the present input bits and their past ones.

Generator Sequence



$$g_0^{(1)} = 1,\ g_1^{(1)} = 0,\ g_2^{(1)} = 1,\ \text{and}\ g_3^{(1)} = 1.$$

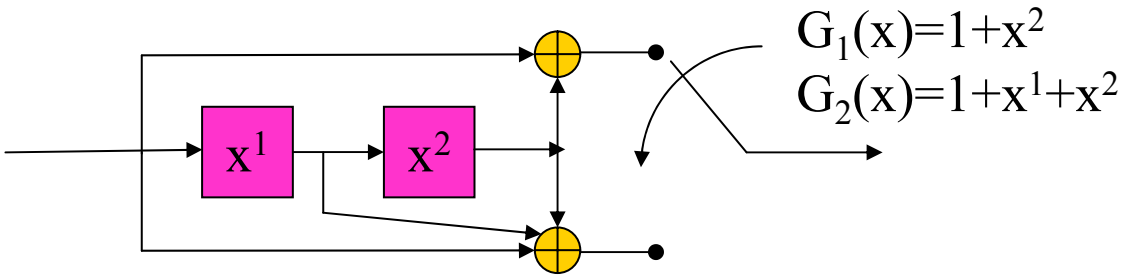
Generator Sequence: $g^{(1)}$=(1 0 1 1)



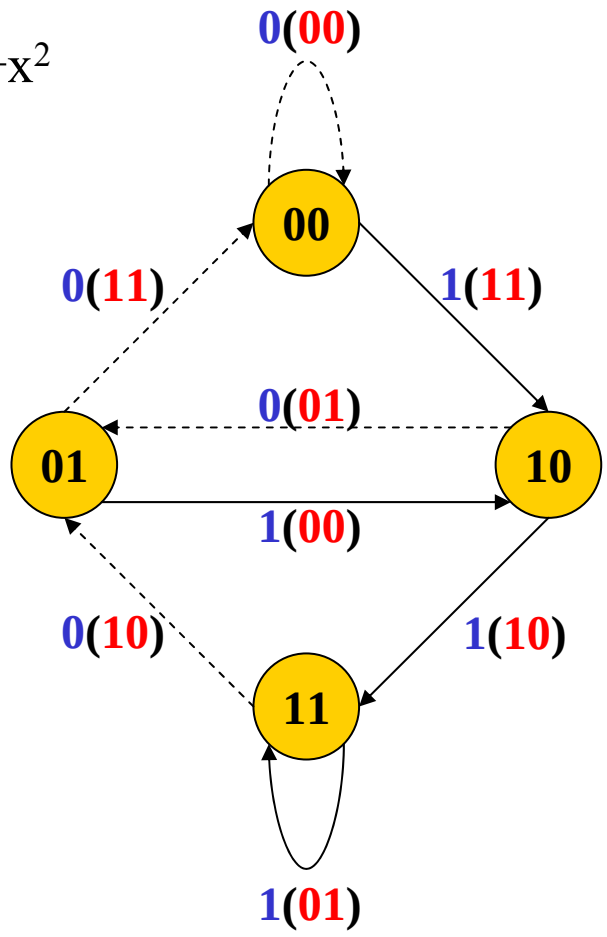
$$g_0^{(2)} = 1,\ g_1^{(2)} = 1,\ g_2^{(2)} = 1,\ g_3^{(2)} = 0,\ \text{and}\ g_4^{(2)} = 1.$$

Generator Sequence: $g^{(2)}$=(1 1 1 0 1)

Convolutional Codes
An Example – (rate=1/2 with K=2)

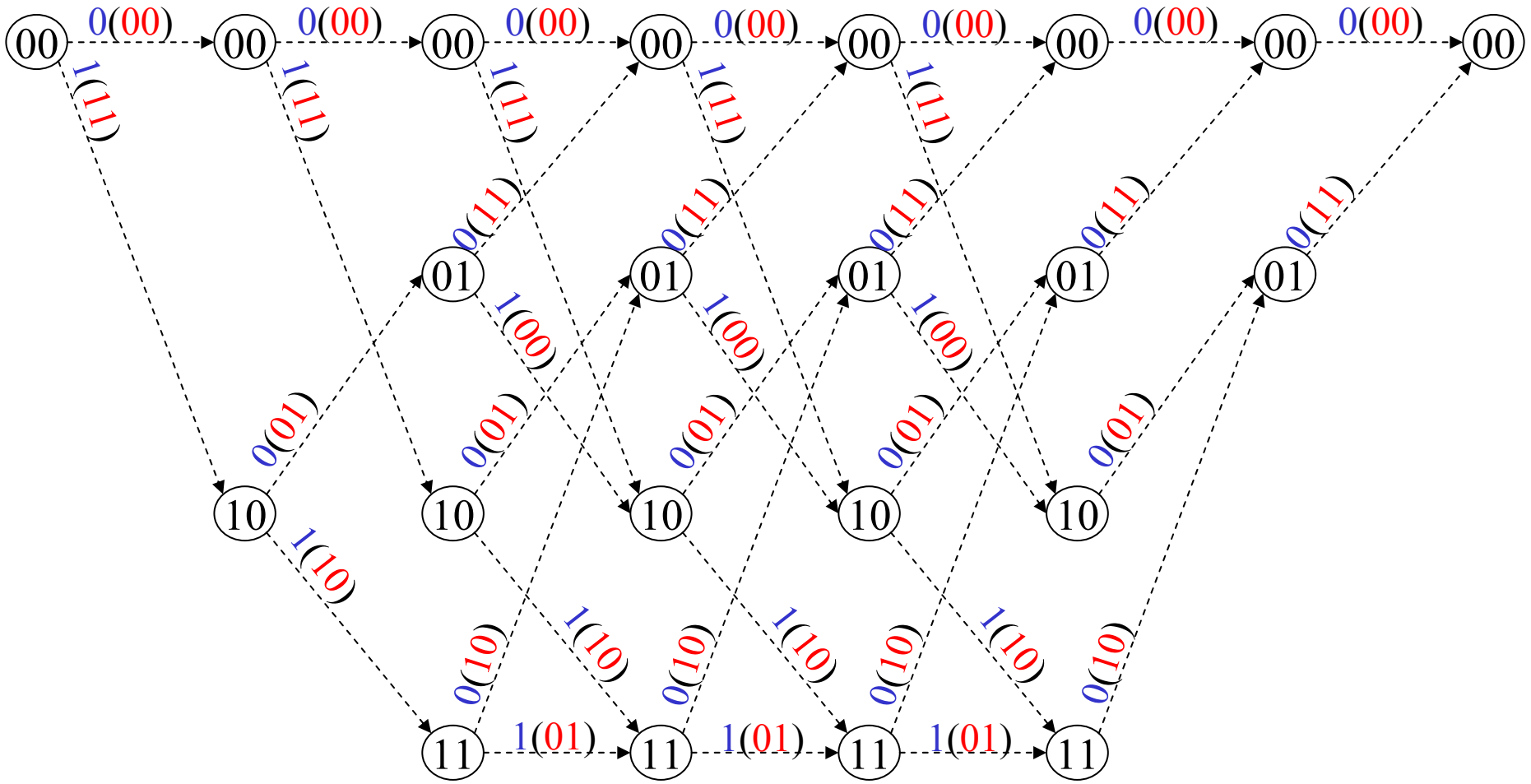


	Present	Next	Output
0	00	00	00
1	00	10	11
0	01	00	11
1	01	10	00
0	10	01	01
1	10	11	10
0	11	01	10
1	11	11	01



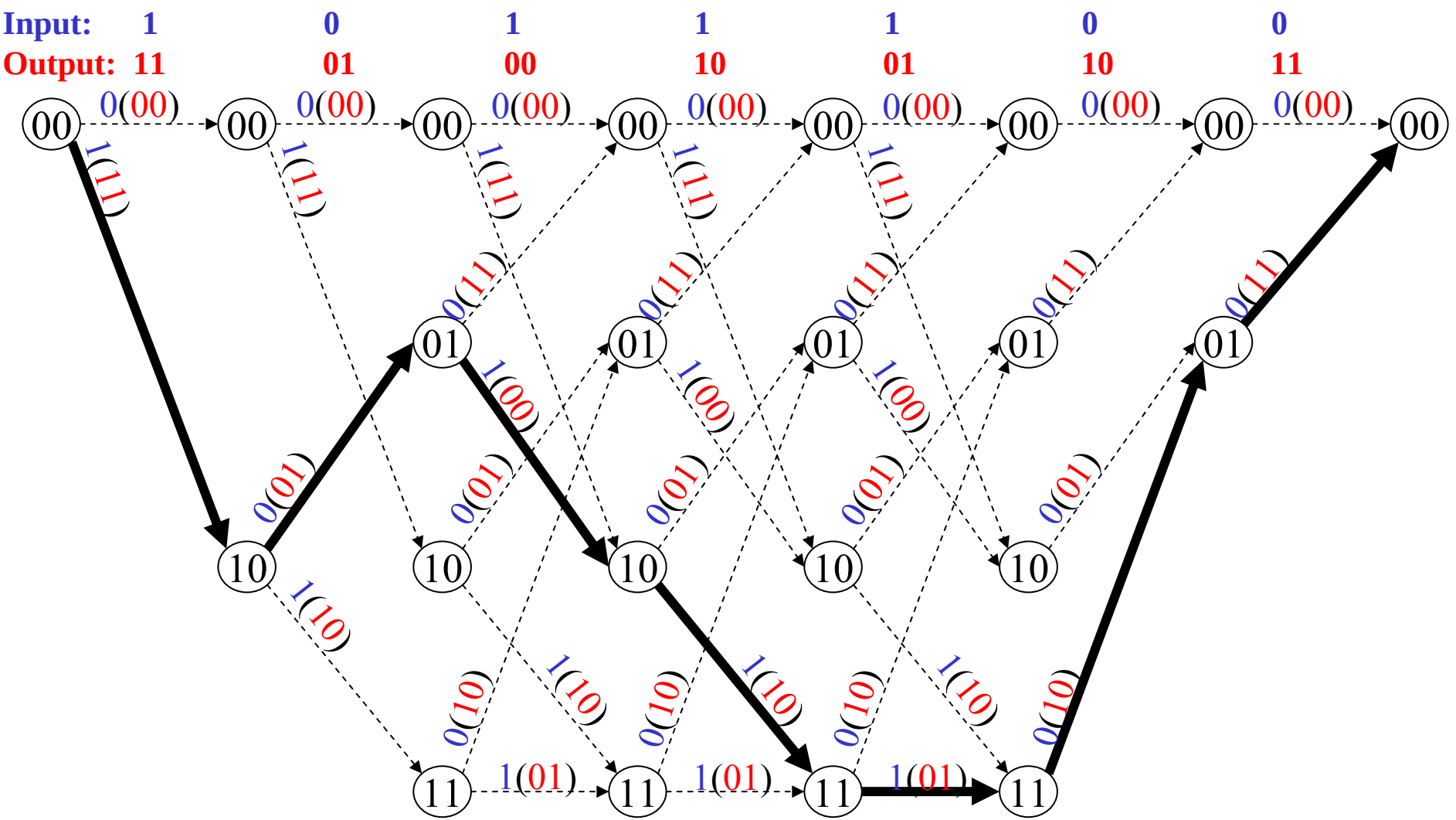
State Diagram

Trellis Diagram Representation



Trellis termination: K tail bits with value 0 are usually added to the end of the code.

Encoding Process



Viterbi Decoding Algorithm

- Maximum Likelihood (ML) decoding rule

received sequence *r* $\xrightarrow{\text{ML}}$ **detected sequence d**

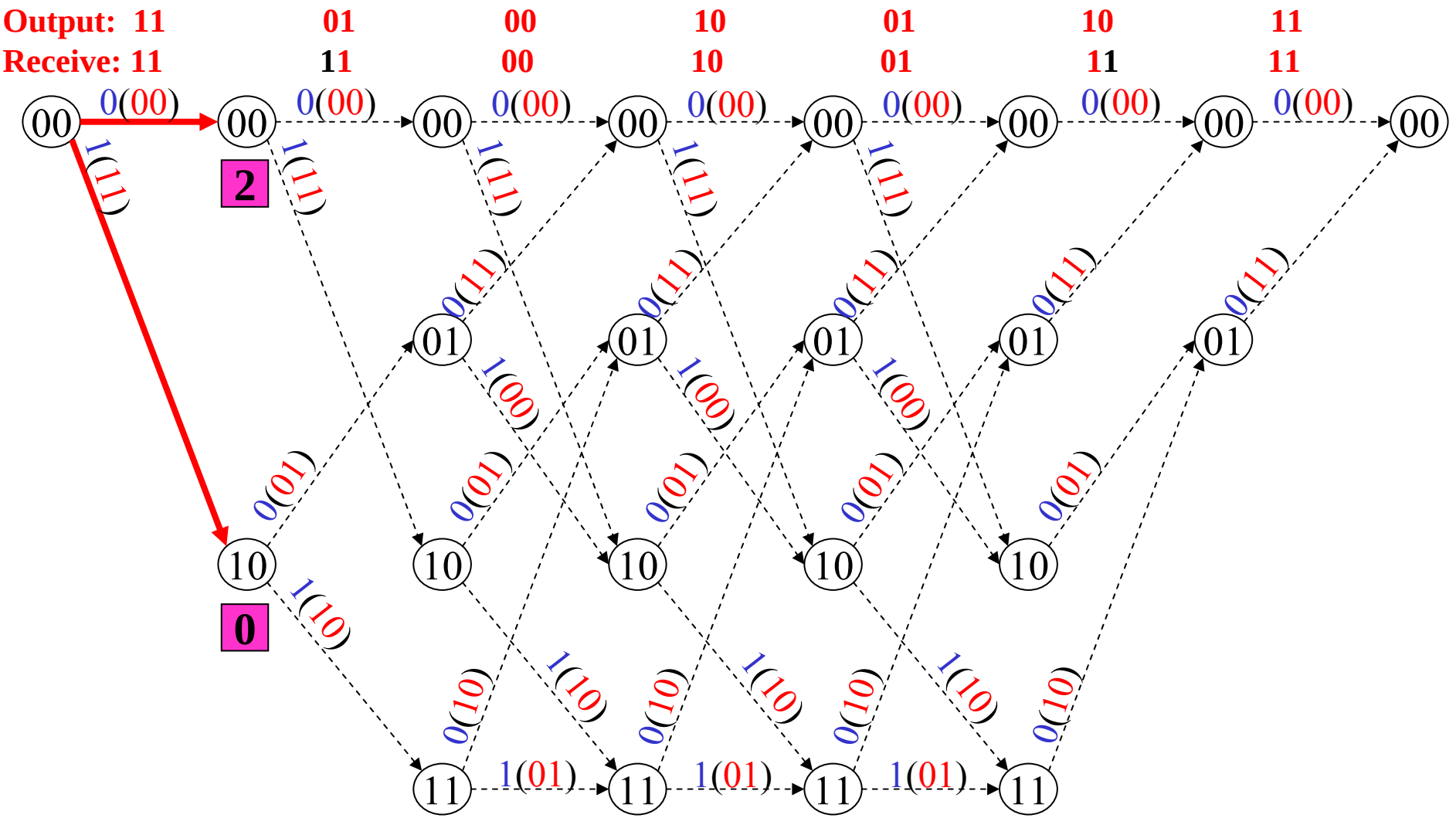
min(*d*,*r*) !!

- Viterbi Decoding Algorithm
  - An efficient search algorithm
    - Performing ML decoding rule.
    - Reducing the computational complexity.

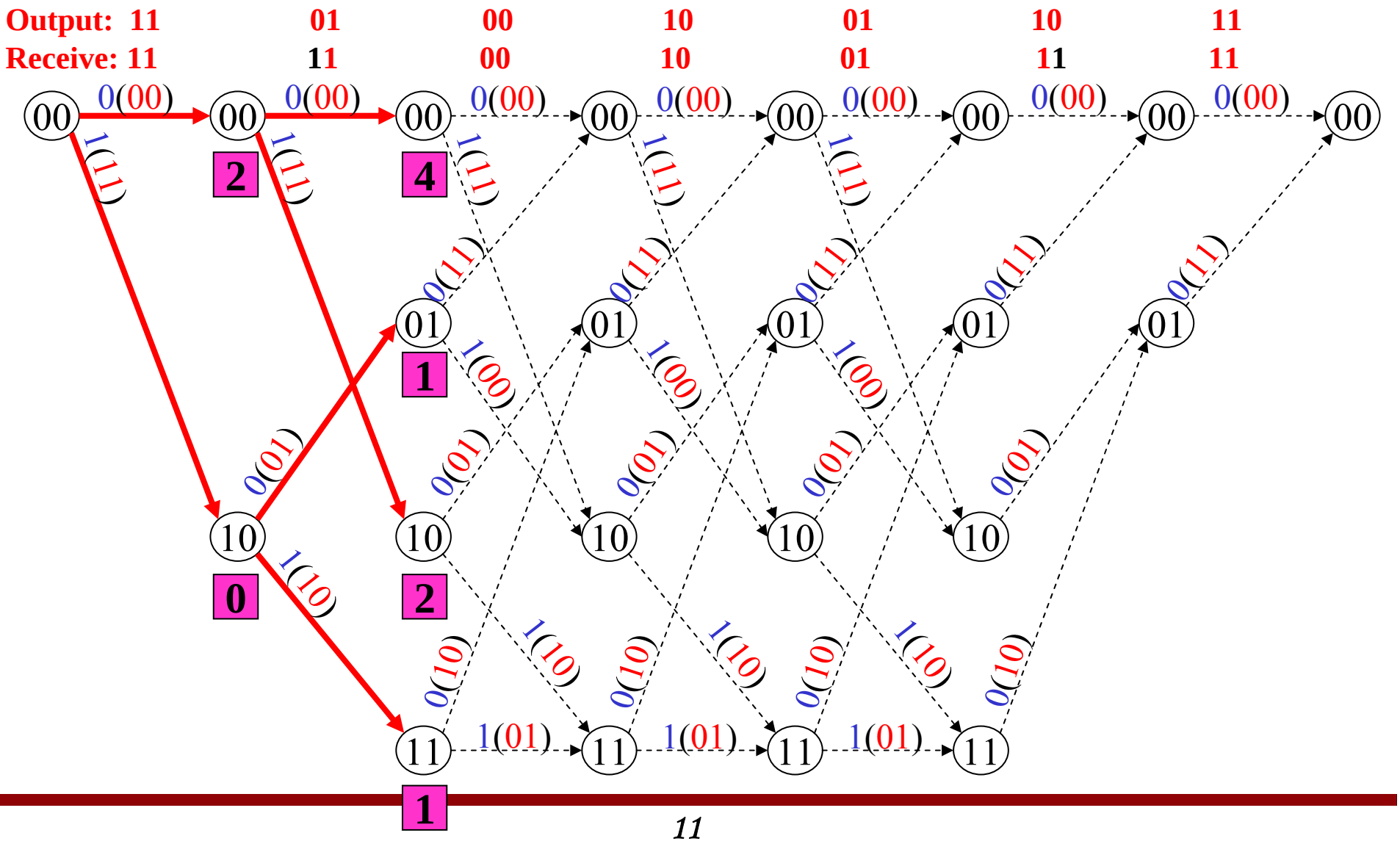
Viterbi Decoding Algorithm

- Basic concept
  - Generate the code trellis at the decoder
  - The decoder penetrates through the code trellis *level by level* in search for the transmitted code sequence
  - At each level of the trellis, the decoder computes and compares the metrics of all the partial paths entering a node
  - The decoder *stores* the partial path with the larger metric and *eliminates* all the other partial paths. The stored partial path is called the *survivor*.

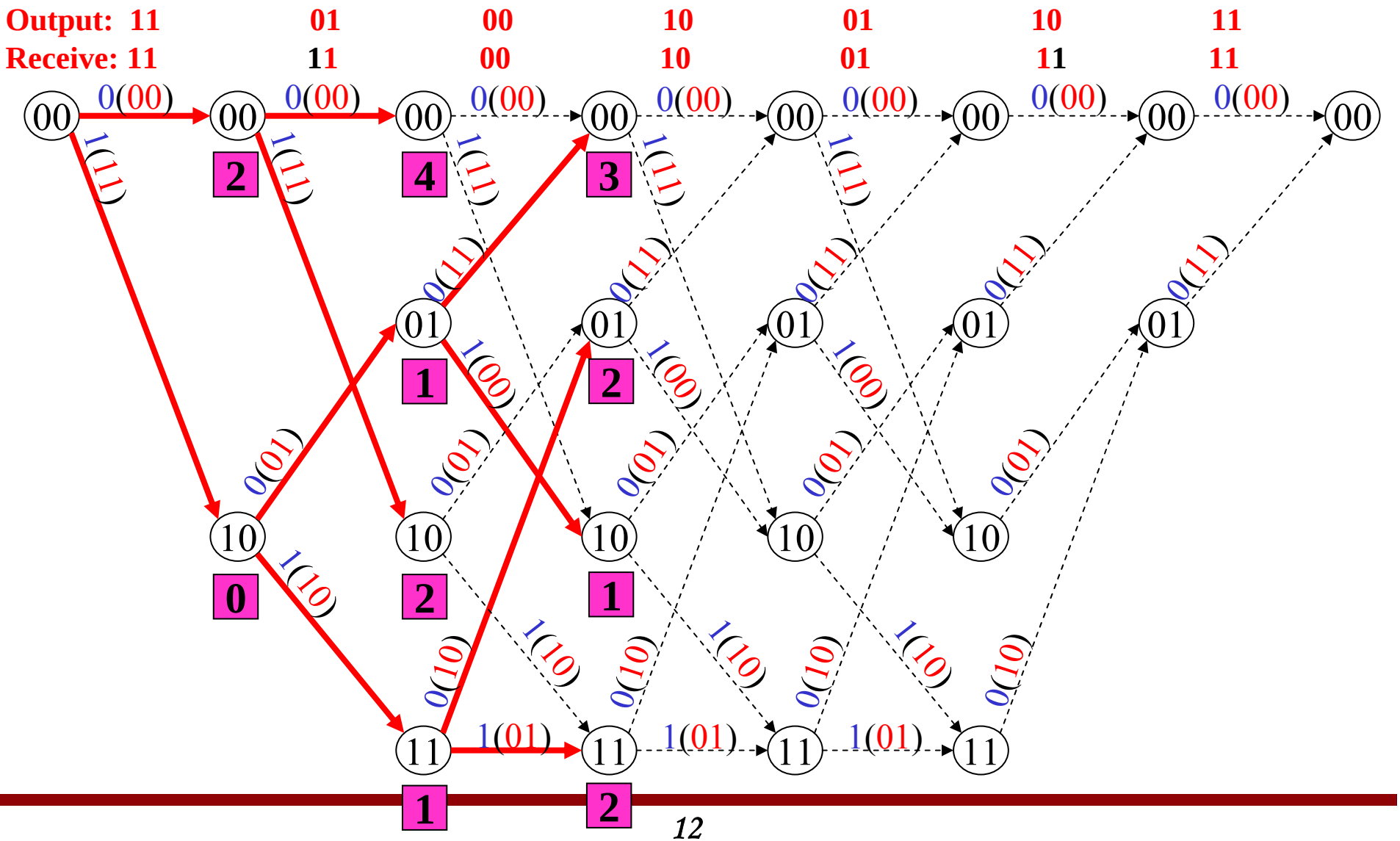
Viterbi Decoding Process



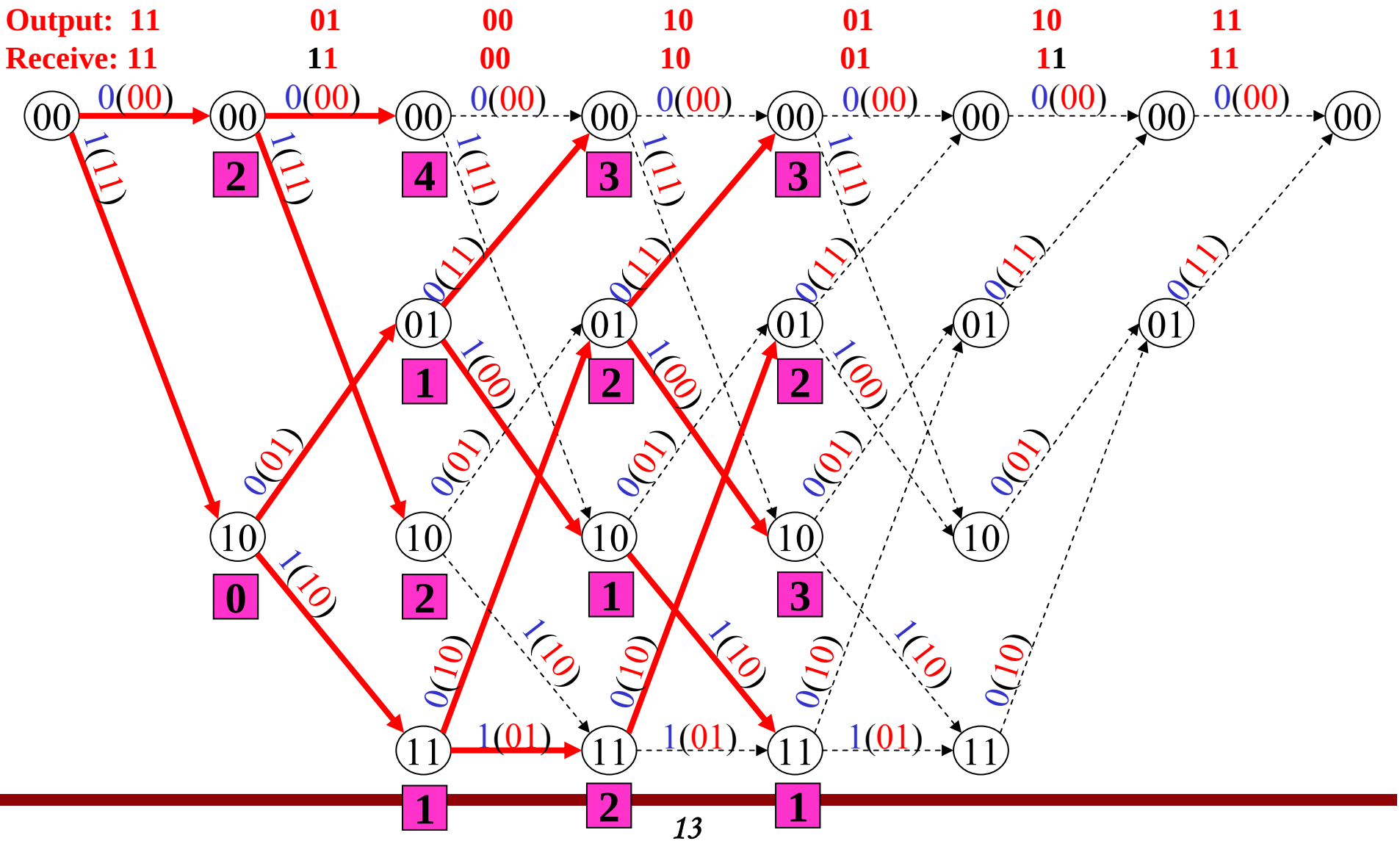
Viterbi Decoding Process



Viterbi Decoding Process

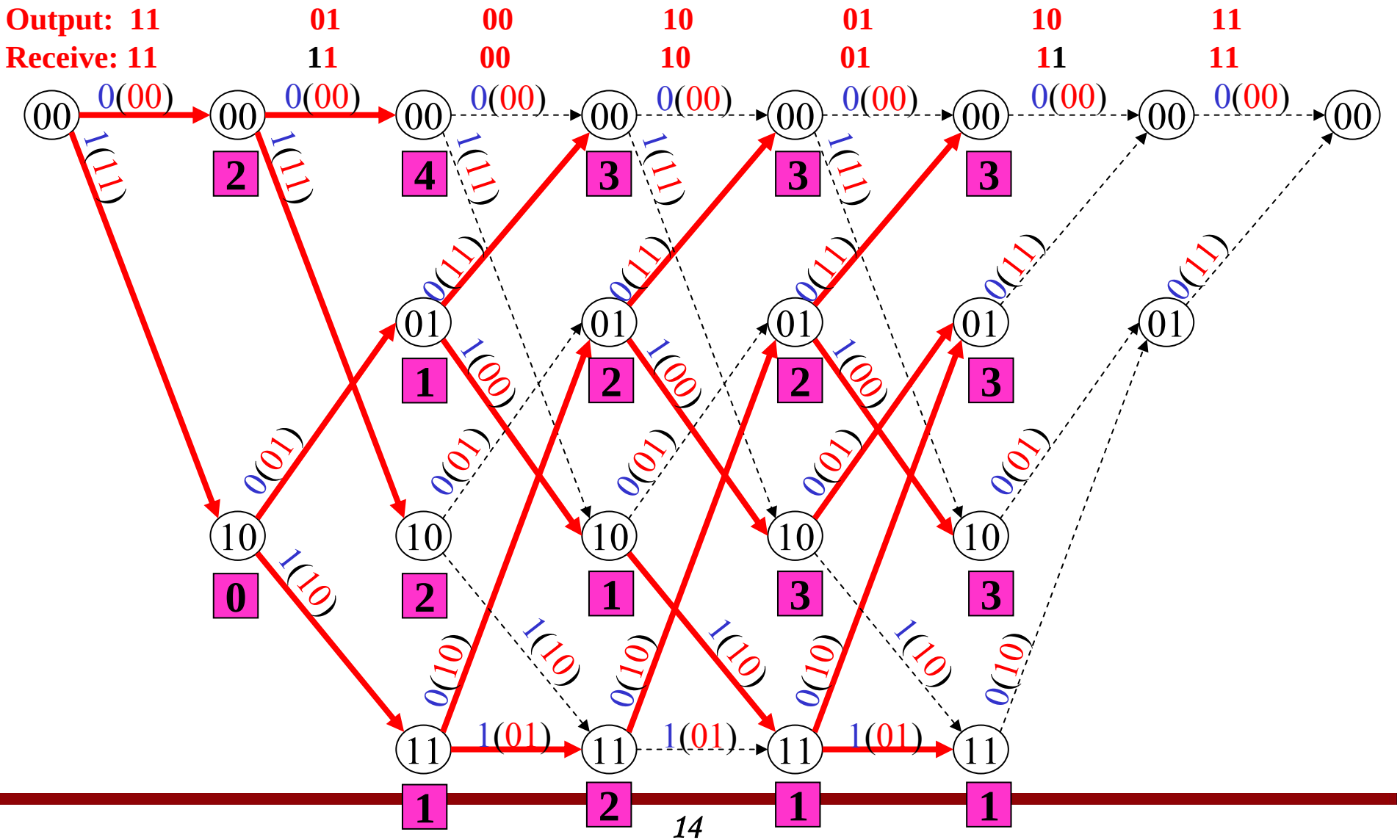


Viterbi Decoding Process

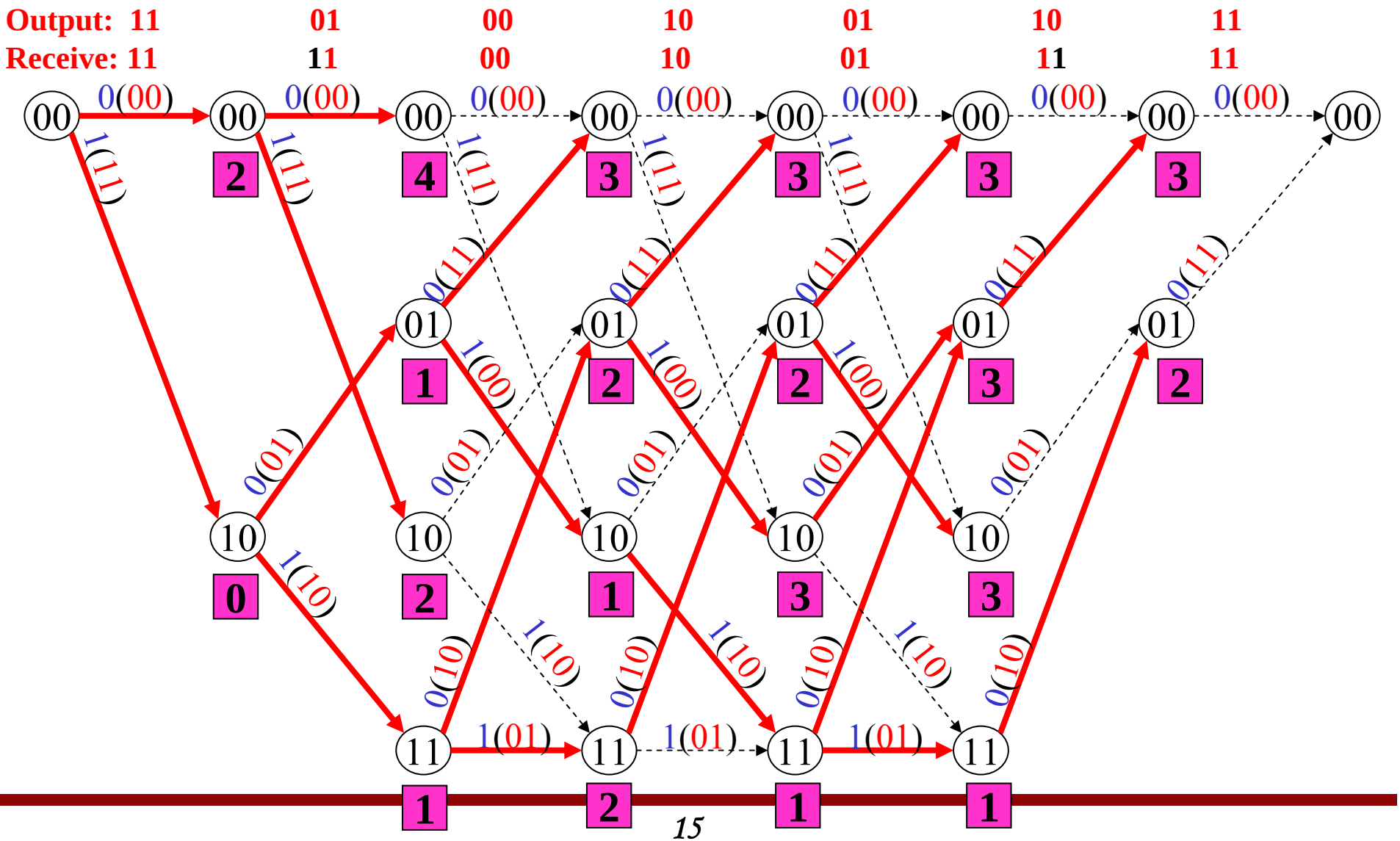


WiTS Lab
Inst. of Comm. Engr.

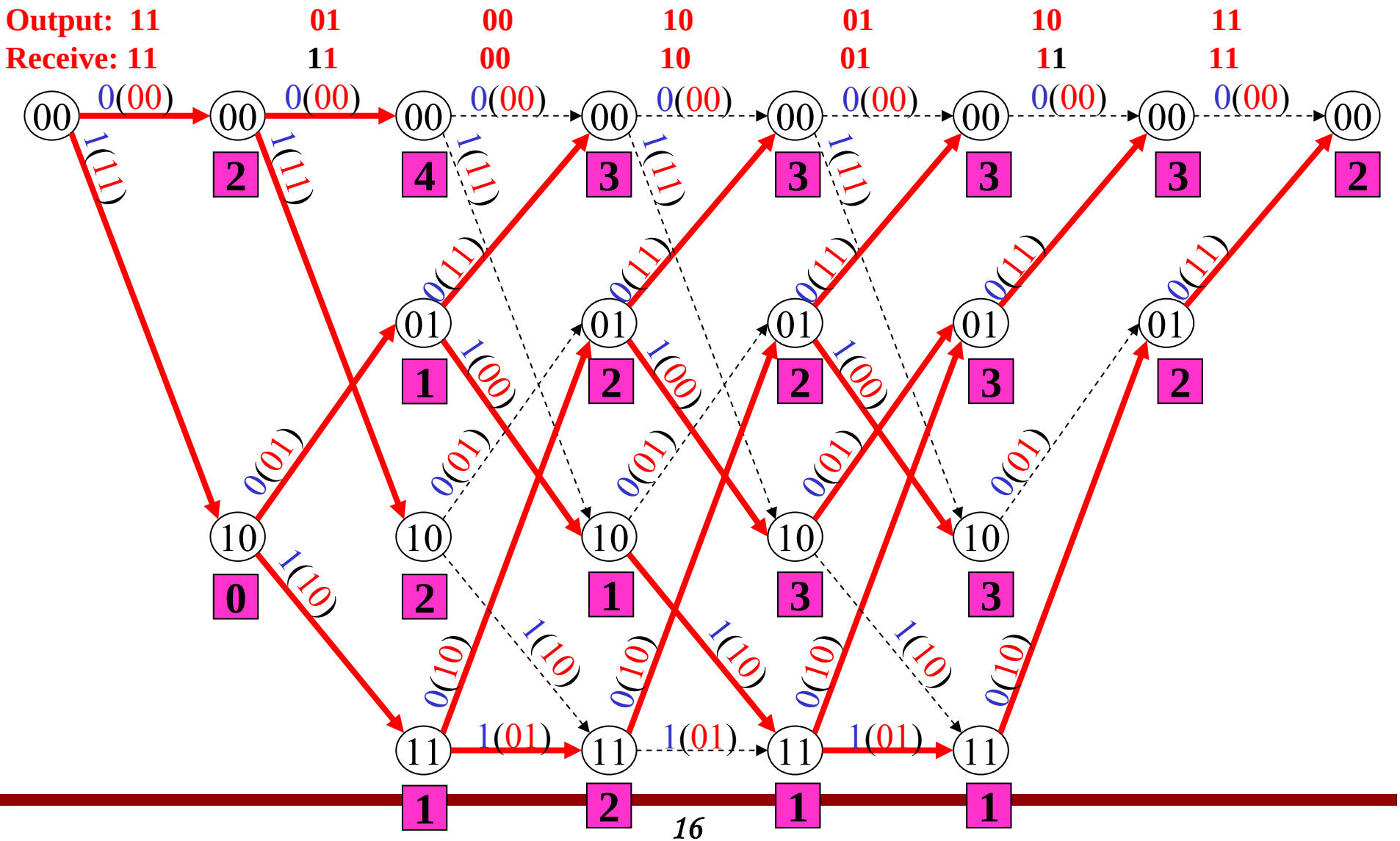
Viterbi Decoding Process



Viterbi Decoding Process

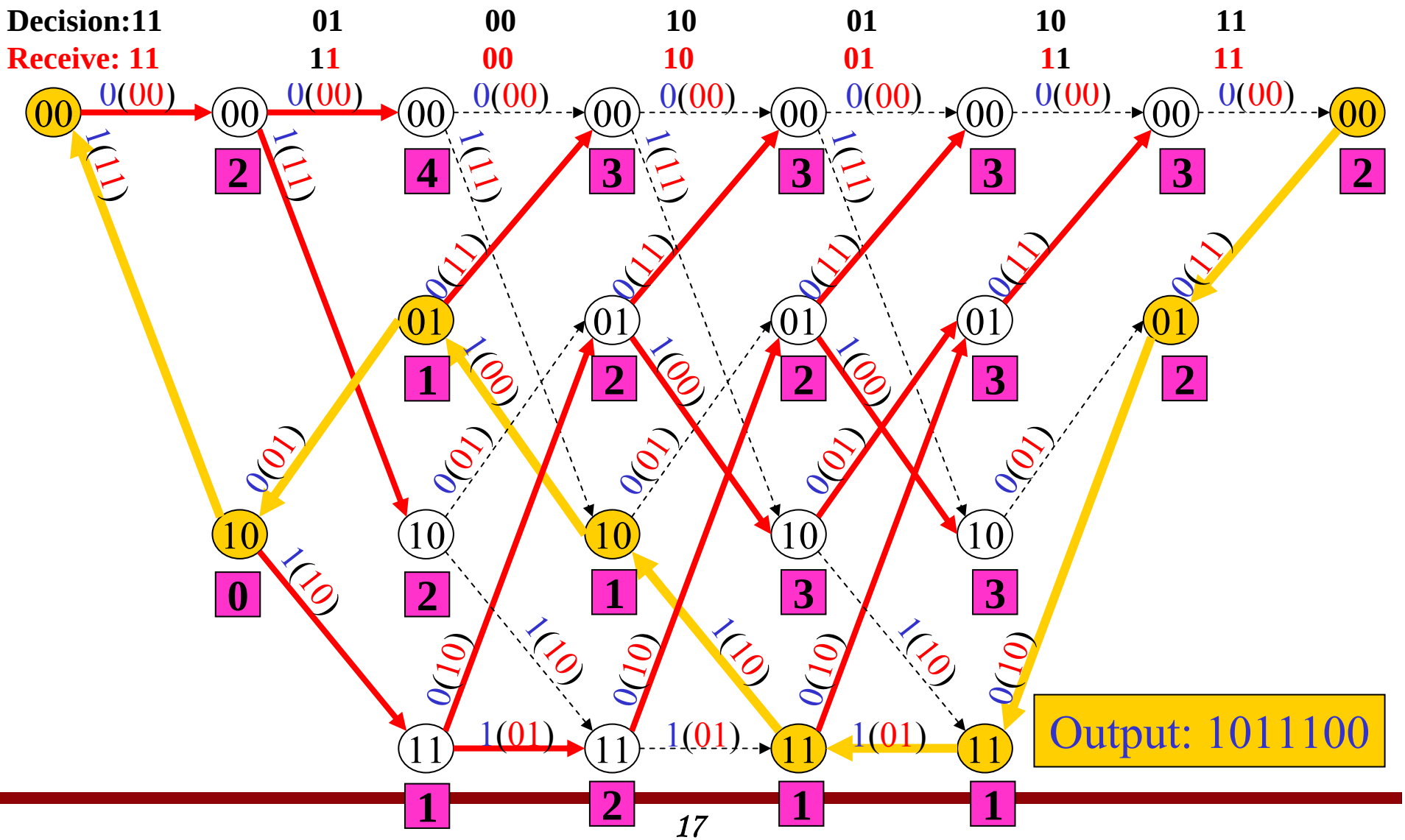


Viterbi Decoding Process



WiTS Lab
Inst. of Comm. Engr.

Viterbi Decoding Process



Convolutional Codes

Convolutional Codes

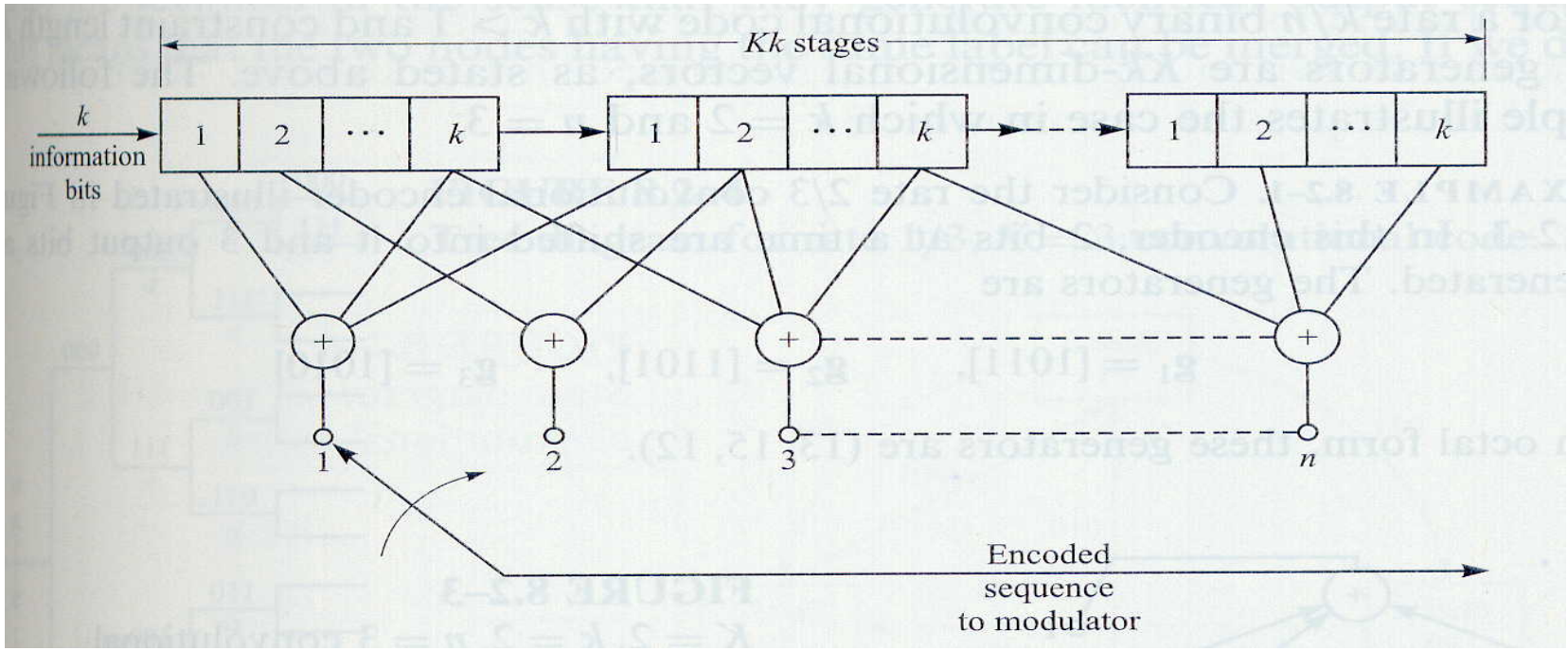
- Convolutional codes differ from block codes in that the encoder contains memory and the *n* encoder outputs at any time unit depend not only on the *k* inputs but also on *m* previous input blocks.
- An (*n, k, m*) convolutional code can be implemented with a *k*-input, *n*-output linear sequential circuit with input memory *m*.
  - Typically, *n* and *k* are small integers with $k<n$, but the memory order *m* must be made large to achieve low error probabilities.
- In the important special case when $k=1$, the information sequence is not divided into blocks and can be processed continuously.
- Convolutional codes were first introduced by Elias in 1955 as an alternative to block codes.

Convolutional Codes

- Shortly thereafter, Wozencraft proposed sequential decoding as an efficient decoding scheme for convolutional codes, and experimental studies soon began to appear.
- In 1963, Massey proposed a less efficient but simpler-to-implement decoding method called threshold decoding.
- Then in 1967, Viterbi proposed a maximum likelihood decoding scheme that was relatively easy to implement for cods with small memory orders.
- This scheme, called *Viterbi decoding*, together with improved versions of sequential decoding, led to the application of convolutional codes to deep-space and satellite communication in early 1970s.

Convolutional Code

- A convolutional code is generated by passing the information sequence to be transmitted through a linear finite-state shift register.
- In general, the shift register consists of K (k-bit) stages and n linear algebraic function generators.

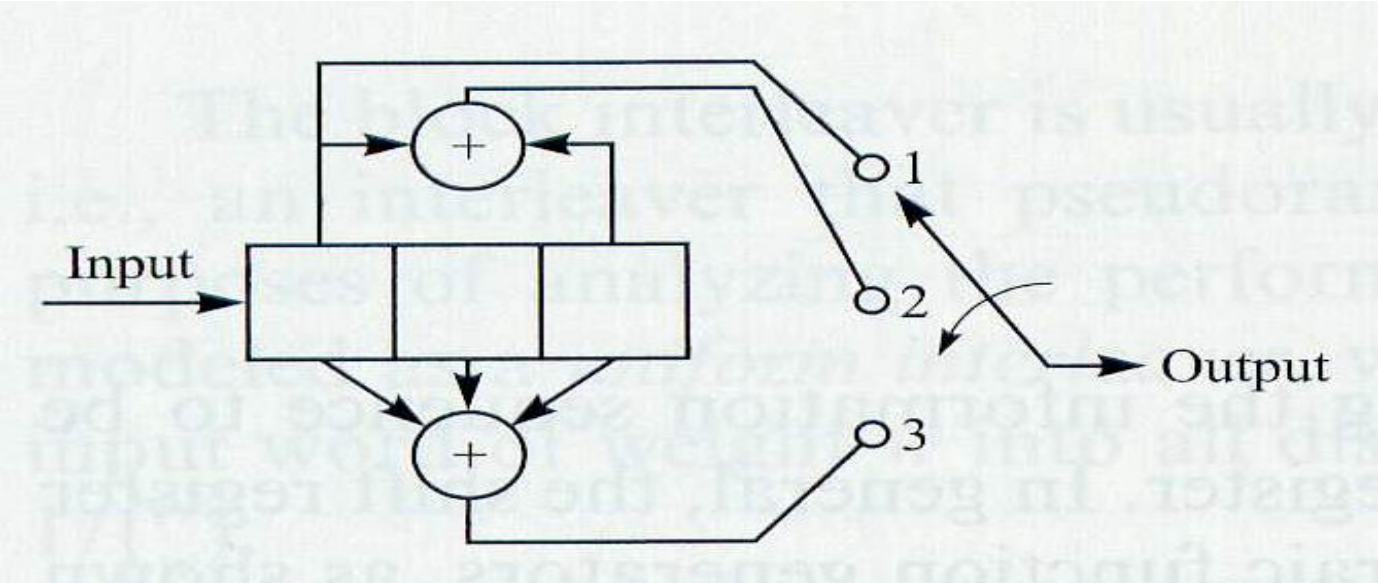


Convoltuional Code

- Convolutional codes
  - k = number of bits shifted into the encoder at one time
    - k=1 is usually used!!
  - n = number of encoder output bits corresponding to the k information bits
  - $R_c = k/n$ = code rate
  - K = constraint length, encoder memory.
- Each encoded bit is a function of the present input bits and their past ones.

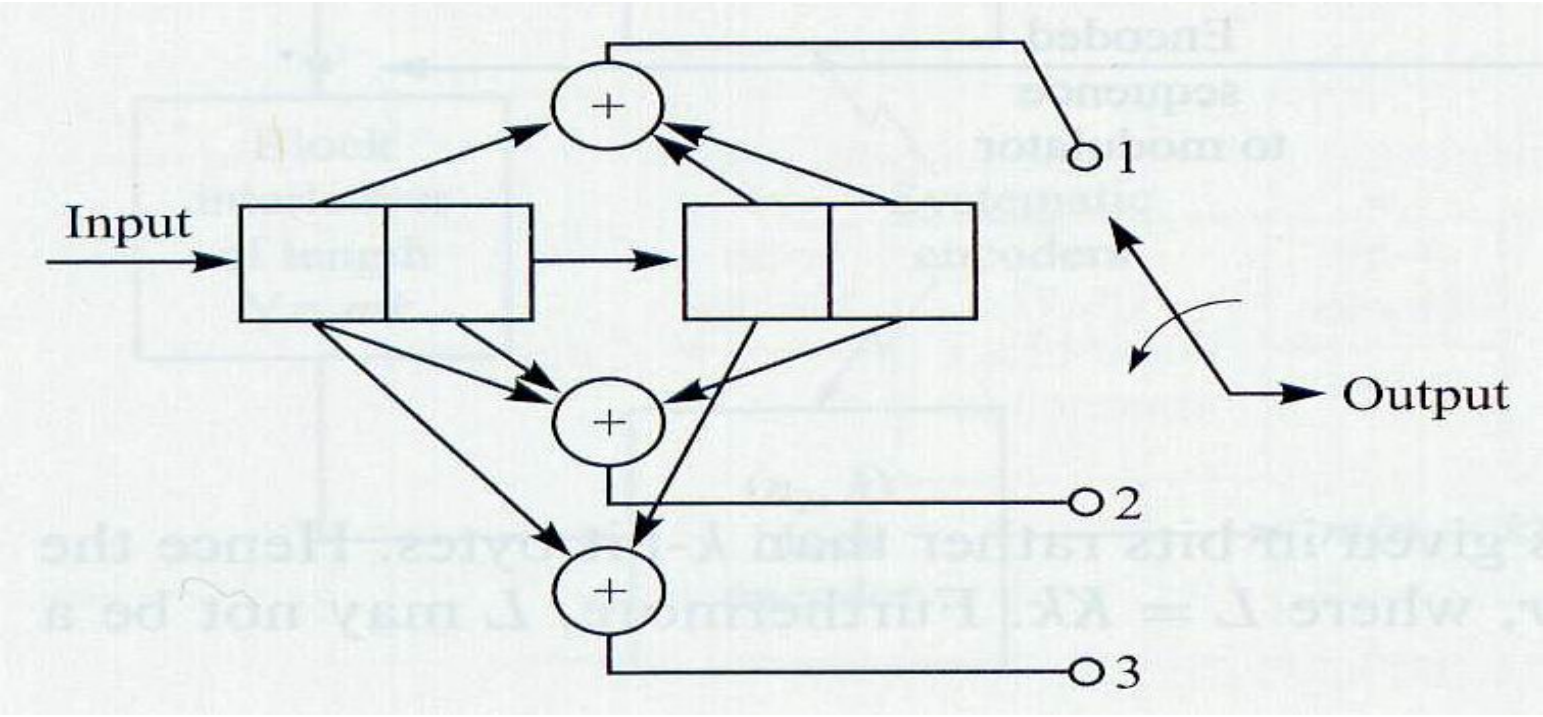
Encoding of Convolutional Code

- Example 1:
  - Consider the binary convolutional encoder with constraint length K=3, k=1, and n=3.
  - The generators are: $\mathbf{g}_1$=[100], $\mathbf{g}_2$=[101], and $\mathbf{g}_3$=[111].
  - The generators are more conveniently given in octal form as (4,5,7).



Encoding of Convolutional Code

- Example 2:
  - Consider a rate 2/3 convolutional encoder.
  - The generators are: $\mathbf{g}_1$=[1011], $\mathbf{g}_2$=[1101], and $\mathbf{g}_3$=[1010].
  - In octal form, these generator are (13, 15, 12).

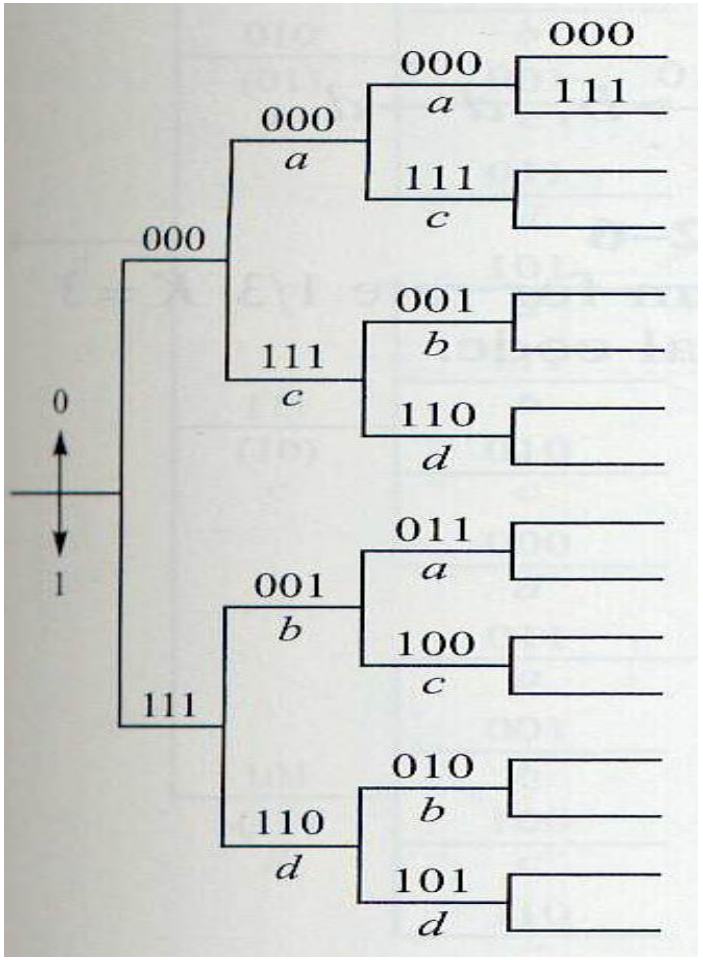


Representations of Convolutional Code

- There are three alternative methods that are often used to describe a convolutional code:
  - Tree diagram
  - Trellis diagram
  - State disgram

Representations of Convolutional Code

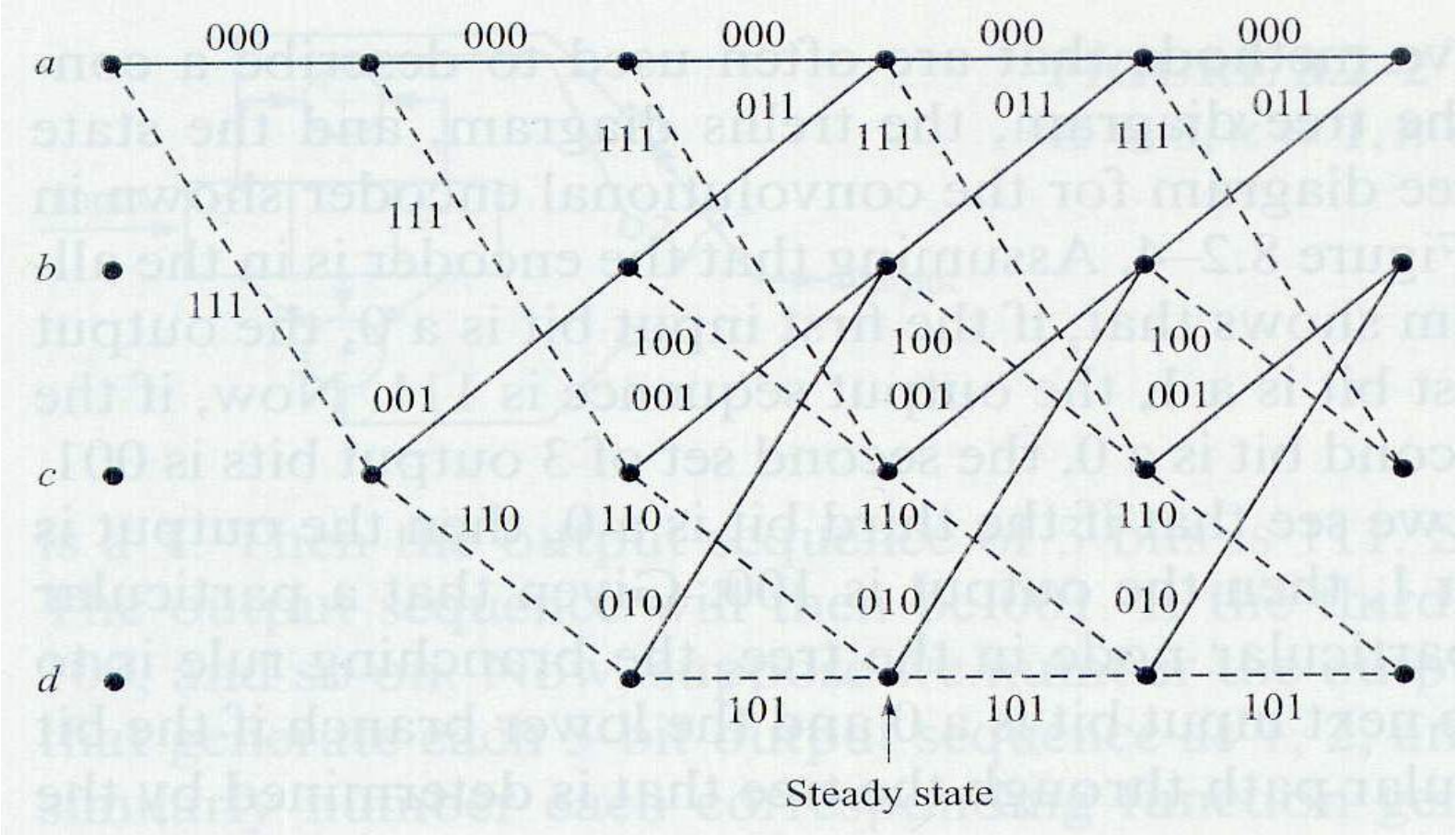
- Tree diagram
  - Note that the tree diagram in the right repeats itself after the third stage.
  - This is consistent with the fact that the constraint length K=3.
  - The output sequence at each stage is determined by the input bit and the two previous input bits.
  - In other words, we may sat that the 3-bit output sequence for each input bit is determined by the input bit and the four possible states of the shift register, denoted as a=00, b=01, c=10, and d=11.



Tree diagram for rate 1/3, K=3 convolutional code.

Representations of Convolutional Code

- Trellis diagram



Tree diagram for rate 1/3, K=3 convolutional code.

Representations of Convolutional Code

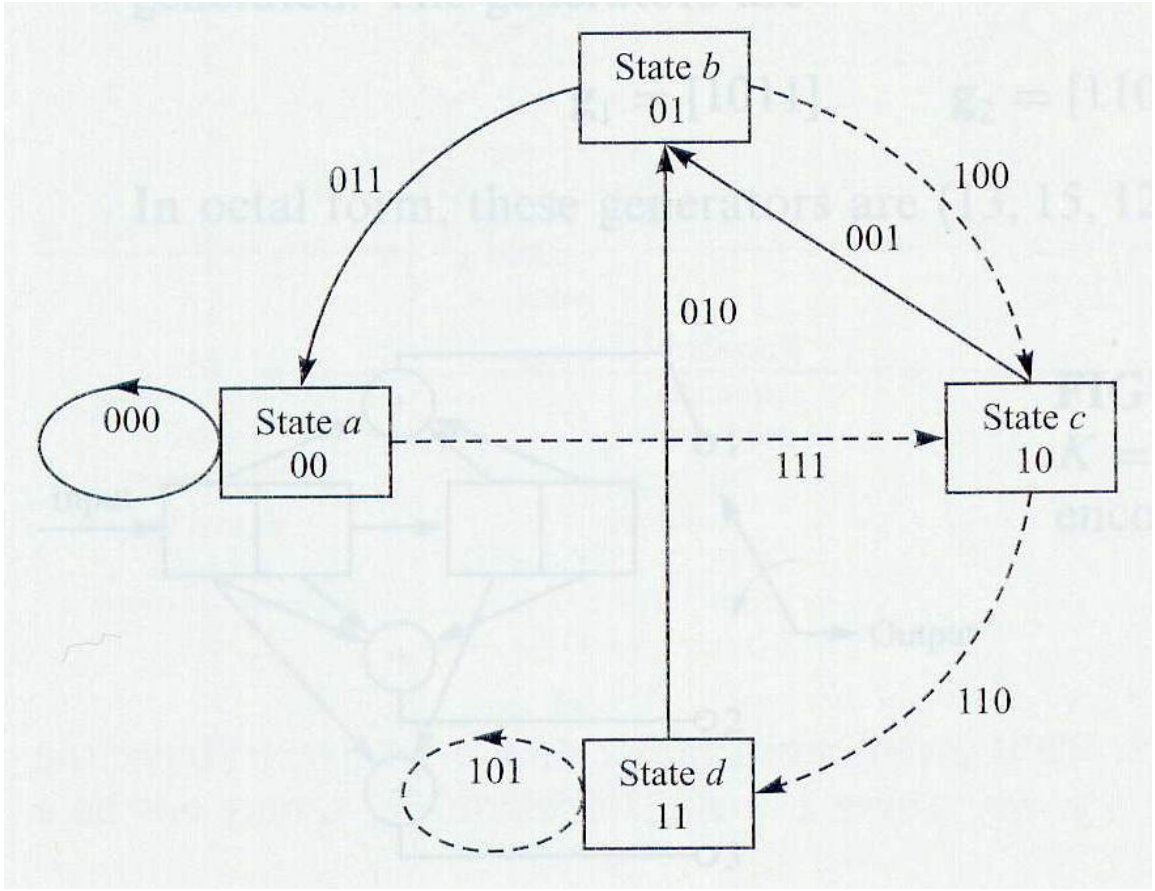
- State diagram

$$a \xrightarrow{0} a \quad a \xrightarrow{1} c$$

$$b \xrightarrow{0} a \quad b \xrightarrow{1} c$$

$$c \xrightarrow{0} b \quad c \xrightarrow{1} d$$

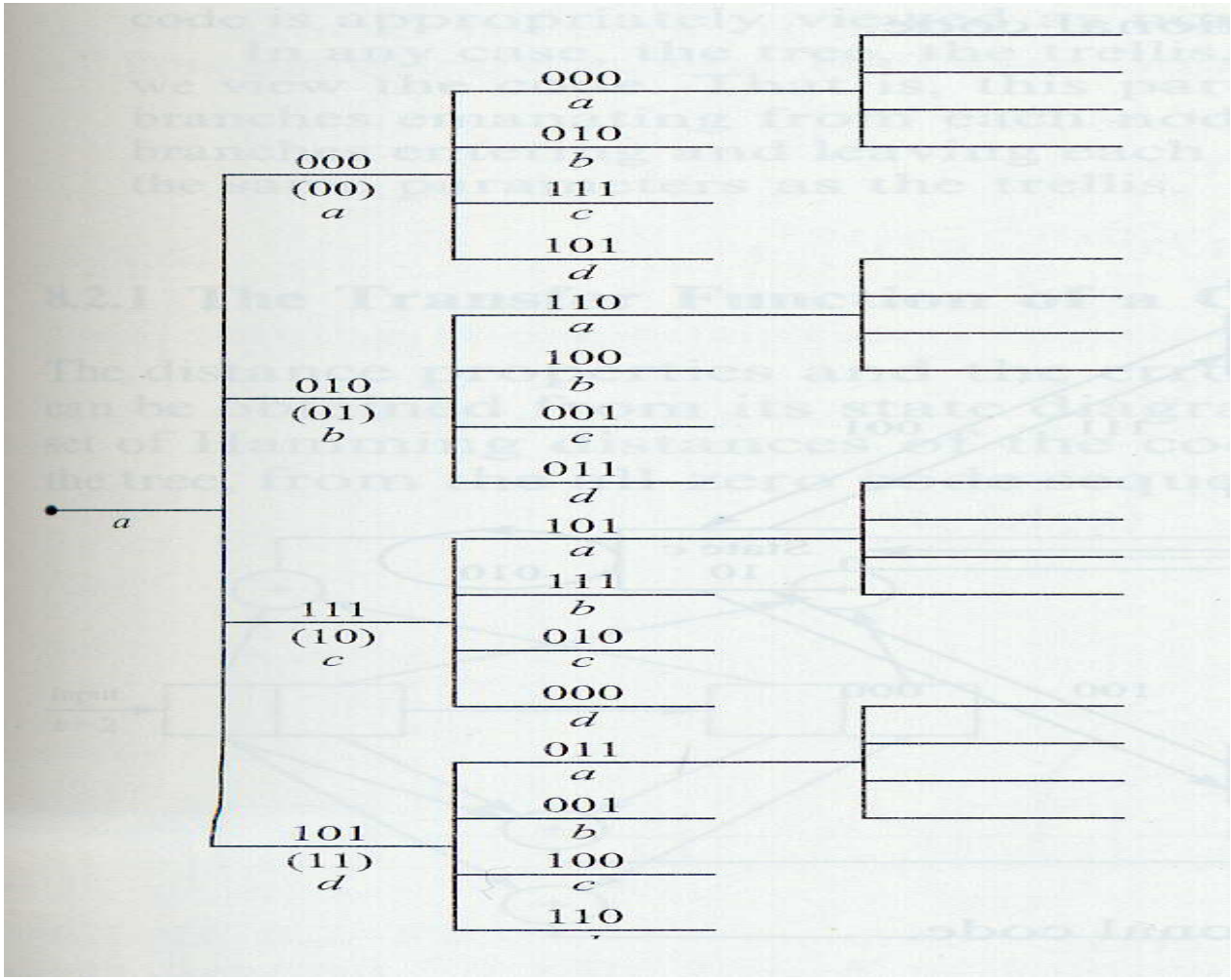
$$d \xrightarrow{0} b \quad d \xrightarrow{1} d$$



State diagram for rate 1/3, K=3 convolutional code.

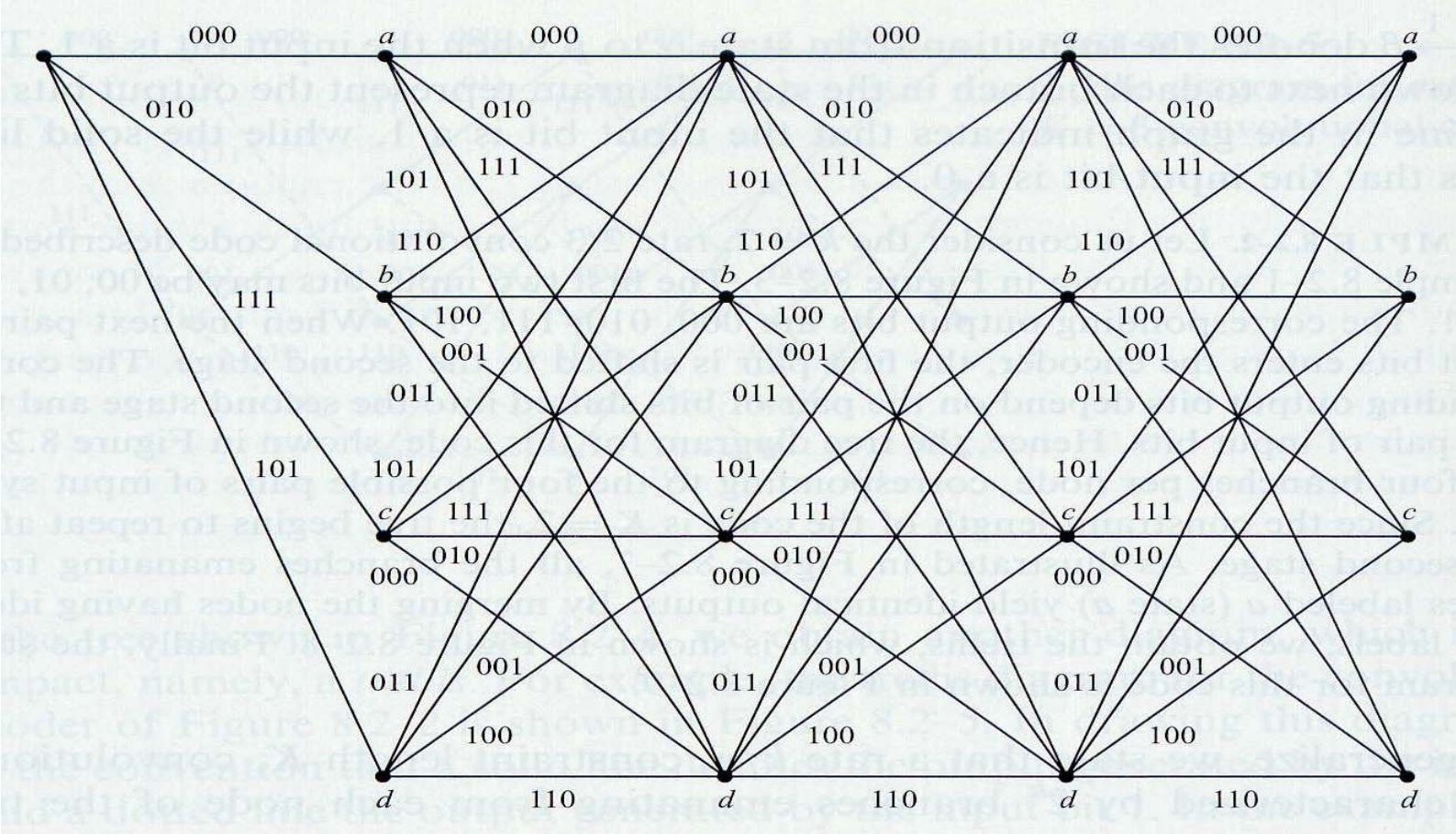
Representations of Convolutional Code

- Example: K=2, k=2, n=3 convolutional code
  - Tree diagram



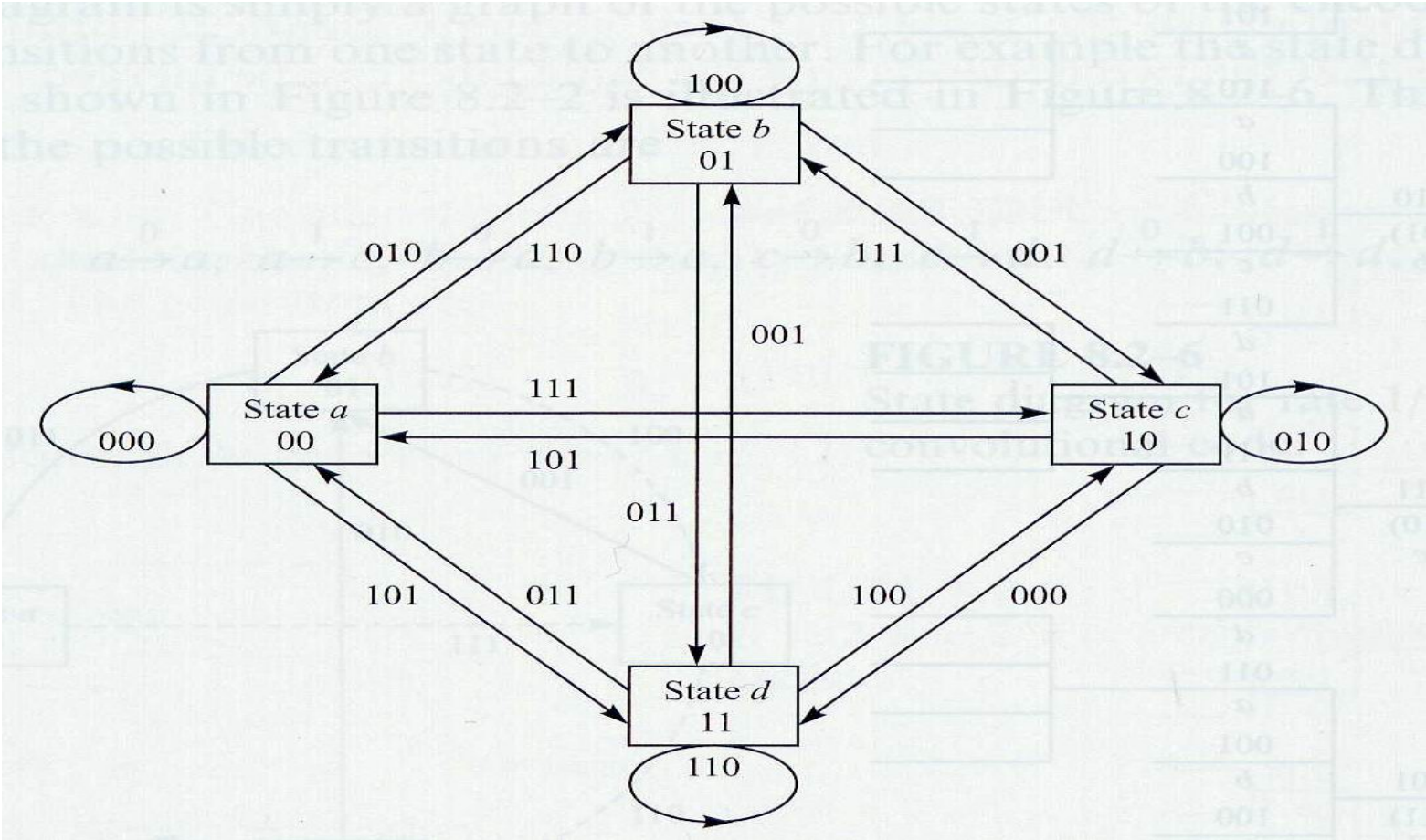
Representations of Convolutional Code

- Example: K=2, k=2, n=3 convolutional code
  - Trellis diagram



Representations of Convolutional Code

- Example: K=2, k=2, n=3 convolutional code
  - State diagram

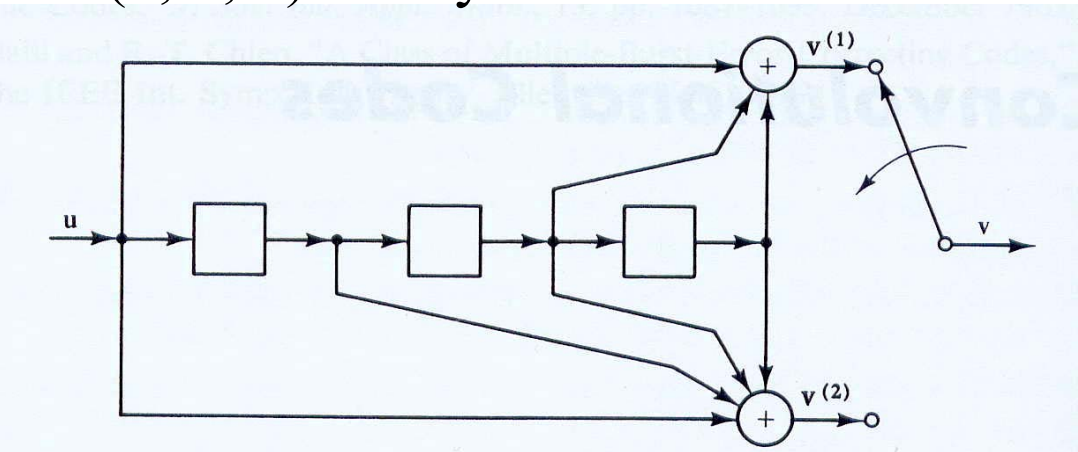


Representations of Convolutional Code

- In general, we state that a rate k/n, constraint length K, convolutional code is characterized by 2^k branches emanating from each node of the tree diagram.
- The trellis and the state diagrams each have $2^{k(K-1)}$ possible states.
- There are 2^k branches entering each state and 2^k branches leaving each state.

Encoding of Convolutional Codes

- Example: A (2, 1, 3) binary convolutional codes:



  - the encoder consists of an m= 3-stage shift register together with n=2 modulo-2 adders and a multiplexer for serializing the encoder outputs.
  - The mod-2 adders can be implemented as EXCLUSIVE-OR gates.
- Since mod-2 addition is a linear operation, the encoder is a linear feedforward shift register.
- All convolutional encoders can be implemented using a linear feedforward shift register of this type.

Encoding of Convolutional Codes

- The *information sequence* $\mathbf{u} = (u_0, u_1, u_2, \ldots)$ enters the encoder one bit at a time.
- Since the encoder is a linear system, the two encoder *output sequence* $\mathbf{v}^{(1)} = (\upsilon_0^{(1)}, \upsilon_1^{(1)}, \upsilon_2^{(1)}, \cdots)$ and $\mathbf{v}^{(2)} = (\upsilon_0^{(2)}, \upsilon_1^{(2)}, \upsilon_2^{(2)}, \cdots)$ can be obtained as the convolution of the input sequence $\mathbf{u}$ with the two encoder "impulse response."
- The impulse responses are obtained by letting $\mathbf{u} = (1\ 0\ 0\ \ldots)$ and observing the two output sequence.
- Since the encoder has an m-time unit memory, the impulse responses can last at most $m+1$ *time units*, and are written as :

$$\mathbf{g}^{(1)} = (g_0^{(1)}, g_1^{(1)}, \cdots, g_m^{(1)})$$
$$\mathbf{g}^{(2)} = (g_0^{(2)}, g_1^{(2)}, \cdots, g_m^{(2)})$$

Encoding of Convolutional Codes

- The encoder of the binary (2, 1, 3) code is

$$\mathbf{g}^{(1)} = (1\ 0\ 1\ 1)$$
$$\mathbf{g}^{(2)} = (1\ 1\ 1\ 1)$$

The impulse response $g^{(1)}$ and $g^{(2)}$ are called the *generator sequences* of the code.

- The encoding equations can now be written as

$$\mathbf{v}^{(1)} = \mathbf{u} * \mathbf{g}^{(1)}$$
$$\mathbf{v}^{(2)} = \mathbf{u} * \mathbf{g}^{(2)}$$

where * denotes discrete convolution and all operations are mod-2.

- The convolution operation implies that for all $l \geq 0$,

$$v_l^{(j)} = \sum_{i=0}^{m} u_{l-i} g_i^{(j)} = u_l g_0^{(j)} + u_{l-1} g_1^{(j)} + \cdots + u_{l-m} g_m^{(j)}, \quad j = 1, 2, .$$

where $u_{l-i} = 0$ for all $l < i$.

Encoding of Convolutional Codes

- Hence, for the encoder of the binary (2,1,3) code,

$$\upsilon_l^{(1)} = u_l + u_{l-2} + u_{l-3}$$
$$\upsilon_l^{(2)} = u_l + u_{l-1} + u_{l-2} + u_{l-3}$$

as can easily be verified by direct inspection of the encoding circuit.

- After encoding, the two output sequences are multiplexed into a signal sequence, called the *code word*, for transmission over the channel.
- The code word is given by

$$\mathbf{v} = (\upsilon_0^{(1)}\upsilon_0^{(2)}, \upsilon_1^{(1)}\upsilon_1^{(2)}, \upsilon_2^{(1)}\upsilon_2^{(2)}, \cdots).$$

Encoding of Convolutional Codes

- Example 10.1
  - Let the information sequence $\mathbf{u} = (1\ 0\ 1\ 1\ 1)$. Then the output sequences are

$$\mathbf{v}^{(1)} = (1\ 0\ 1\ 1\ 1) * (1\ 0\ 1\ 1) = (1\ 0\ 0\ 0\ 0\ 0\ 0\ 1)$$
$$\mathbf{v}^{(2)} = (1\ 0\ 1\ 1\ 1) * (1\ 1\ 1\ 1) = (1\ 1\ 0\ 1\ 1\ 1\ 0\ 1)$$

  and the code word is

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

Encoding of Convolutional Codes

- If the generator sequence $\mathbf{g}^{(1)}$ and $\mathbf{g}^{(2)}$ are interlaced and then arranged in the matrix

$$\mathbf{G} = \begin{bmatrix} g_0^{(1)} g_0^{(2)} & g_1^{(1)} g_1^{(2)} & g_2^{(1)} g_2^{(2)} & \cdots & g_m^{(1)} g_m^{(2)} & & \\ & g_0^{(1)} g_0^{(2)} & g_1^{(1)} g_1^{(2)} & \cdots & g_{m-1}^{(1)} g_{m-1}^{(2)} & g_m^{(1)} g_m^{(2)} & \\ & & g_0^{(1)} g_0^{(2)} & \cdots & g_{m-2}^{(1)} g_{m-2}^{(2)} & g_{m-1}^{(1)} g_{m-1}^{(2)} & g_m^{(1)} g_m^{(2)} \\ & & & \ddots & & & \ddots \end{bmatrix}$$

  where the blank areas are all zeros, the encoding equations can be rewritten in matrix form as $\mathbf{v} = \mathbf{uG}$.

- $\mathbf{G}$ is called the *generator matrix* of the code. Note that each row of $\mathbf{G}$ is identical to the preceding row but shifted $n = 2$ places to right, and the $\mathbf{G}$ is a semi-infinite matrix, corresponding to the fact that the information sequence $\mathbf{u}$ is of arbitrary length.

Encoding of Convolutional Codes

- If **u** has finite length L, then **G** has L rows and $2(m+L)$ columns, and **v** has length $2(m+L)$.
- Example 10.2
  - If u=(1 0 1 1 1), then

$$\mathbf{v} = \mathbf{uG}$$

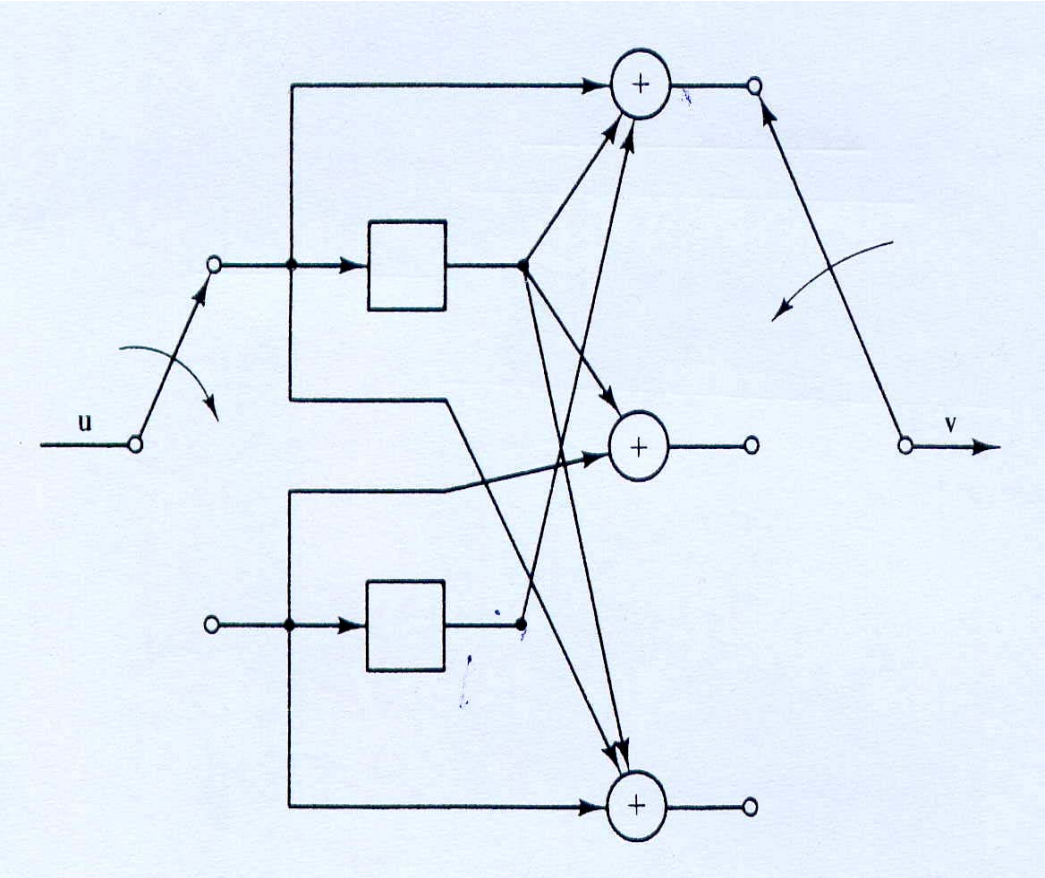
$$= (1\ 0\ 1\ 1\ 1)\begin{bmatrix} 11 & 01 & 11 & 11 & & & & & & \\ & & 11 & 01 & 11 & 11 & & & & \\ & & & & 11 & 01 & 11 & 11 & & \\ & & & & & & 11 & 01 & 11 & 11 \end{bmatrix}$$

$$= (1\ 1,\ 0\ 1,\ 0\ 0,\ 0\ 1,\ 0\ 1,\ 0\ 1,\ 0\ 0,\ 1\ 1),$$

agree with our previous calculation using discrete convolution.

Encoding of Convolutional Codes

- Consider a (3, 2, 1) convolutional codes



- Since $k = 2$, the encoder consists of two $m = 1$-stage shift registers together with $n = 3$ mode-2 adders and two multiplexers.

Encoding of Convolutional Codes

- The information sequence enters the encoder $k = 2$ bits at a time, and can be written as

$$\mathbf{u} = (u_0^{(1)}u_0^{(2)}, u_1^{(1)}u_1^{(2)}, u_2^{(1)}u_2^{(2)}, \cdots)$$

or as the two input sequences $\mathbf{u}^{(1)} = (u_0^{(1)}, u_1^{(1)}, u_2^{(1)}, \cdots)$

$\mathbf{u}^{(2)} = (u_0^{(2)}, u_1^{(2)}, u_2^{(2)}, \cdots)$

- There are three generator sequences corresponding to each input sequence.
- Let $g_i^{(j)} = (g_{i,0}^{(j)}, g_{i,1}^{(j)}, \cdots, g_{i,m}^{(j)})$ represent the generator sequence corresponding to input i and output j, the generator sequence of the (3, 2, 1) convolutional codes are

$$\mathbf{g}_1^{(1)} = (1\ \ 1), \quad \mathbf{g}_1^{(2)} = (0\ \ 1), \quad \mathbf{g}_1^{(3)} = (1\ \ 1),$$
$$\mathbf{g}_2^{(1)} = (0\ \ 1), \quad \mathbf{g}_2^{(2)} = (1\ \ 0), \quad \mathbf{g}_2^{(3)} = (1\ \ 0),$$

Encoding of Convolutional Codes

- And the encoding equations can be written as

$$\mathbf{v}^{(1)} = \mathbf{u}^{(1)} * \mathbf{g}_1^{(1)} + \mathbf{u}^{(2)} * \mathbf{g}_2^{(1)}$$
$$\mathbf{v}^{(2)} = \mathbf{u}^{(1)} * \mathbf{g}_1^{(2)} + \mathbf{u}^{(2)} * \mathbf{g}_2^{(2)}$$
$$\mathbf{v}^{(3)} = \mathbf{u}^{(1)} * \mathbf{g}_1^{(3)} + \mathbf{u}^{(2)} * \mathbf{g}_2^{(3)}$$

- The convolution operation implies that

$$\upsilon_l^{(1)} = u_l^{(1)} + u_{l-1}^{(1)} + u_{l-1}^{(2)}$$
$$\upsilon_l^{(2)} = u_l^{(2)} + u_{l-1}^{(1)}$$
$$\upsilon_l^{(3)} = u_l^{(1)} + u_l^{(2)} + u_{l-1}^{(1)},$$

- After multiplexing, the code word is given by

$$\mathbf{v} = (\upsilon_0^{(1)}\upsilon_0^{(2)}\upsilon_0^{(3)}, \upsilon_1^{(1)}\upsilon_1^{(2)}\upsilon_1^{(3)}, \upsilon_2^{(1)}\upsilon_2^{(2)}\upsilon_2^{(3)}, \cdots).$$

Encoding of Convolutional Codes

- Example 10.3
  - If u(1) = (1 0 1) and u(2) = (1 1 0), then

$$\mathbf{v}^{(1)} = (1\ \ 0\ \ 1) * (1\ \ 1) + (1\ \ 1\ \ 0) * (0\ \ 1) = (1\ \ 0\ \ 0\ \ 1)$$
$$\mathbf{v}^{(2)} = (1\ \ 0\ \ 1) * (0\ \ 1) + (1\ \ 1\ \ 0) * (1\ \ 0) = (1\ \ 0\ \ 0\ \ 1)$$
$$\mathbf{v}^{(3)} = (1\ \ 0\ \ 1) * (1\ \ 1) + (1\ \ 1\ \ 0) * (1\ \ 0) = (0\ \ 0\ \ 1\ \ 1)$$

and

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

Encoding of Convolutional Codes

- The generator matrix of a (3, 2, m) code is

$$\mathbf{G} = \begin{bmatrix} g_{1,0}^{(1)} g_{1,0}^{(2)} g_{1,0}^{(3)} & g_{1,1}^{(1)} g_{1,1}^{(2)} g_{1,1}^{(3)} & \cdots & g_{1,m}^{(1)} g_{1,m}^{(2)} g_{1,m}^{(3)} & \\ g_{2,0}^{(1)} g_{2,0}^{(2)} g_{2,0}^{(3)} & g_{2,1}^{(1)} g_{2,1}^{(2)} g_{2,1}^{(3)} & \cdots & g_{2,m}^{(1)} g_{2,m}^{(2)} g_{2,m}^{(3)} & \\ & g_{1,0}^{(1)} g_{1,0}^{(2)} g_{1,0}^{(3)} & \cdots & g_{1,m-1}^{(1)} g_{1,m-1}^{(2)} g_{1,m-1}^{(3)} & g_{1,m}^{(1)} g_{1,m}^{(2)} g_{1,m}^{(3)} \\ & g_{2,0}^{(1)} g_{2,0}^{(2)} g_{2,0}^{(3)} & \cdots & g_{2,m-1}^{(1)} g_{2,m-1}^{(2)} g_{2,m-1}^{(3)} & g_{2,m}^{(1)} g_{2,m}^{(2)} g_{2,m}^{(3)} \\ & \ddots & & & \ddots \end{bmatrix}$$

  and the encoding equation in matrix are again given by $\mathbf{v} = \mathbf{uG}$.

- Note that each set of $k = 2$ rows of G is identical to the preceding set of rows but shifted $n = 3$ places to right.

Encoding of Convolutional Codes

- Example 10.4
  - If $\mathbf{u}^{(1)} = (1\ 0\ 1)$ and $\mathbf{u}^{(2)} = (1\ 1\ 0)$, then $\mathbf{u} = (1\ 1, 0\ 1, 1\ 0)$ and $\mathbf{v} = \mathbf{uG}$

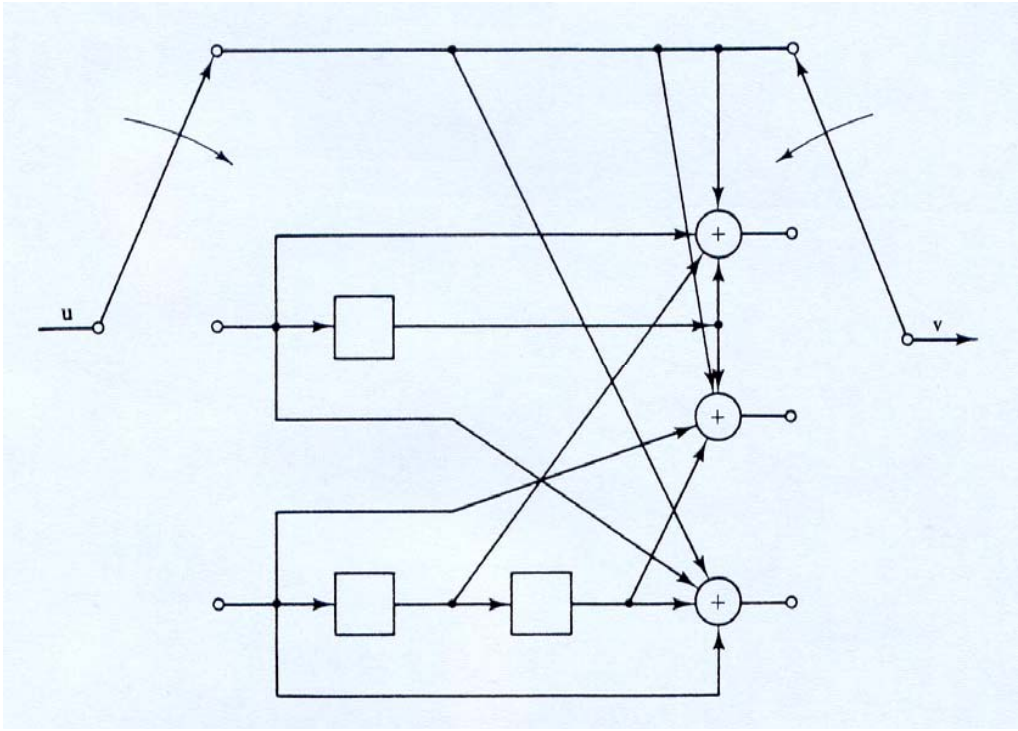
$$= (1\ 1, 0\ 1, 1\ 0)\begin{bmatrix} 1\ 0\ 1 & 1\ 1\ 1 & & \\ 0\ 1\ 1 & 1\ 0\ 0 & & \\ & 1\ 0\ 1 & 1\ 1\ 1 & \\ & 0\ 1\ 1 & 1\ 0\ 0 & \\ & & 1\ 0\ 1 & 1\ 1\ 1 \\ & & 0\ 1\ 1 & 1\ 0\ 0 \end{bmatrix}$$

$$= (1\ 1\ 0,\ \ 0\ 0\ 0,\ \ 0\ 0\ 1,\ \ 1\ 1\ 1),$$

it agree with our previous calculation using discrete convolution.

Encoding of Convolutional Codes

- In particular, the encoder now contains k shift registers, not all of which must have the same length.
- If K_i is the length of the ith shift register, then the encoder *memory order* m is defined as $m \triangleq \max_{1 \le i \le k} K_i$
- An example of a (4, 3, 2) convolutional encoder in which the shift register length are 0, 1, and 2.



Encoding of Convolutional Codes

- The *constraint length* is defined as $n_A \equiv n(m+1)$.
- Since each information bit remains in the encoder for up to $m+1$ time units, and during each time unit can affect any of the n encoder outputs, n_A can be interpreted as the maximum number of encoder outputs that can be affected by a signal information bit.
- For example, the constraint length of the (2,1,3), (3,2,1), and (4,3,2) convolutional codes are 8, 6, and 12, respectively.

Encoding of Convolutional Codes

- If the general case of an (n, k, m) code, the generator matrix is

$$\mathbf{G} = \begin{bmatrix} \mathbf{G}_0 & \mathbf{G}_1 & \mathbf{G}_2 & \cdots & \mathbf{G}_m & & \\ & \mathbf{G}_0 & \mathbf{G}_1 & \cdots & \mathbf{G}_{m-1} & \mathbf{G}_m & \\ & & \mathbf{G}_0 & \cdots & \mathbf{G}_{m-2} & \mathbf{G}_{m-1} & \mathbf{G}_m \\ & & \ddots & & & & \ddots \end{bmatrix}$$

where each $\mathbf{G}_l$ is a $k \times n$ submatrix whose entries are

$$\mathbf{G}_l = \begin{bmatrix} g_{1,l}^{(1)} & g_{1,l}^{(2)} & \cdots & g_{1,l}^{(n)} \\ g_{2,l}^{(1)} & g_{2,l}^{(2)} & \cdots & g_{2,l}^{(n)} \\ \vdots & \vdots & & \vdots \\ g_{k,l}^{(1)} & g_{k,l}^{(2)} & \cdots & g_{k,l}^{(n)} \end{bmatrix}$$

- Note that each set of k rows of $\mathbf{G}$ is identical to the previous set of rows but shifted n places to the right.

Encoding of Convolutional Codes

- For an information sequence

$$\mathbf{u} = (\mathbf{u}_0, \mathbf{u}_1, \cdots) = (u_0^{(1)} u_0^{(2)} \cdots u_0^{(k)}, u_1^{(1)} u_1^{(2)} \cdots u_1^{(k)}, \cdots)$$

and the code word $\mathbf{v} = (\mathbf{v}_0, \mathbf{v}_1, \cdots) = (v_0^{(1)} v_0^{(2)} \cdots v_0^{(n)}, v_1^{(1)} v_1^{(2)} \cdots v_1^{(n)}, \cdots)$ is given by $\mathbf{v} = \mathbf{uG}$.

- Since the code word **v** is a linear combination of rows of the generator matrix **G**, an (n, k, m) convolutional code is a linear code.

Encoding of Convolutional Codes

- A convolutional encoder generates n encoded bits for each k information bits, and $R = k/n$ is called the *code rate*.
- For an $k \cdot L$ finite length information sequence, the corresponding code word has length $n(L + m)$, where the final $n \cdot m$ outputs are generated after the last nonzero information block has entered the encoder.
- Viewing a convolutional code as a linear block code with generator matrix **G**, the block code rate is given by $kL/n(L + m)$, the ratio of the number of information bits to the length of the code word.
- If $L \gg m$, then $L/(L + m) \approx 1$, and the block code rate and convolutional code are approximately equal .

Encoding of Convolutional Codes

- If L were small, however, the ratio $kL/n(L+m)$, which is the effective rate of information transmission, would be reduced below the code rate by a fractional amount

$$\frac{k/n - kL/n(L+m)}{k/n} = \frac{m}{L+m}$$

called the *fractional rate loss*.

- To keep the fractional rate loss small, L is always assumed to be much larger than m.
- Example 10.5
  - For a (2,1,3) convolutional codes, L=5 and the fractional rate loss is 3/8=37.5%. However, if the length of the information sequence is L=1000, the fractional rate loss is only 3/1003=0.3%.

Encoding of Convolutional Codes

- In a linear system, time-domain operations involving convolution can be replaced by more convenient transform-domain operations involving polynomial multiplication.
- Since a convolutional encoder is a linear system, each sequence in the encoding equations can be replaced by corresponding polynomial, and the convolution operation replaced by polynomial multiplication.
- In the polynomial representation of a binary sequence, the sequence itself is represent by the coefficients of the polynomial.
- For example, for a (2, 1, m) code, the encoding equations become

$$\mathbf{v}^{(1)}(D) = \mathbf{u}(D)\mathbf{g}^{(1)}(D)$$
$$\mathbf{v}^{(2)}(D) = \mathbf{u}(D)\mathbf{g}^{(2)}(D),$$

where $\mathbf{u}(D) = u_0 + u_1 D + u_2 D^2 + \cdots$ is the *information sequence*.

Encoding of Convolutional Codes

- The *encoded sequences* are

$$\mathbf{v}^{(1)}(D) = \upsilon_0^{(1)} + \upsilon_1^{(1)} D + \upsilon_2^{(1)} D^2 + \cdots$$
$$\mathbf{v}^{(2)}(D) = \upsilon_0^{(2)} + \upsilon_1^{(2)} D + \upsilon_2^{(2)} D^2 + \cdots$$

- The *generator polynomials* of the code are

$$\mathbf{g}^{(1)}(D) = g_0^{(1)} + g_1^{(1)} D + \cdots + g_m^{(1)} D^m$$
$$\mathbf{g}^{(2)}(D) = g_0^{(2)} + g_1^{(2)} D + \cdots + g_m^{(2)} D^m$$

and all operations are modulo-2.

- After multiplexing, the code word become

$$\mathbf{v}(D) = \mathbf{v}^{(1)}(D^2) + D\mathbf{v}^{(2)}(D^2)$$

the indeterminate D can be interpreted as a *delay operator*, and the power of D denoting the number of time units a bit is delayed with respect to the initial bit.

Encoding of Convolutional Codes

- Example 10.6
  - For the previous (2, 1, 3) convolutional code, the generator polynomials are $\mathbf{g}^{(1)}(D) = 1+D^2+D^3$ and $\mathbf{g}^{(2)}(D) = 1+D+D^2+D^3$.
  - For the information sequence $\mathbf{u}(D) = 1+D^2+D^3+D^4$, the encoding equation are

$$\mathbf{v}^{(1)}(D) = (1+D^2+D^3+D^4)(1+D^2+D^3) = 1+D^7$$
$$\mathbf{v}^{(2)}(D) = (1+D^2+D^3+D^4)(1+D+D^2+D^3)$$
$$= 1+D+D^3+D^4+D^5+D^7,$$

  and the code word is

$$\mathbf{v}(D) = \mathbf{v}^{(1)}(D^2) + D\mathbf{v}^{(2)}(D^2) = 1+D+D^3+D^7+D^9+D^{11}+D^{14}+D^{15}.$$

  - Note that the result is the same as previously computed using convolution and matrix multiplication.

Encoding of Convolutional Codes

- The generator polynomials of an encoder can be determined directly from its circuit diagram.
- Since the shift register stage represents a one-time-unit delay, the sequence of connection (a 1 representing a connection and a 0 no connection) from a shift register to an output is the sequence of coefficients in the corresponding generator polynomial.
- Since the last stage of the shift register in an (n, 1) code must be connected to at least one output, at least one of the generator polynomials must have degree equal to the shift register length m, that is

$$m = \max_{1 \le j \le n} \left[\deg \mathbf{g}^{(j)}(D) \right]$$

Encoding of Convolutional Codes

- In an (n, k) code where $k>1$, there are n generator polynomials for each of the k inputs.
- Each set of n generators represents the connections from one of the shift registers to the n outputs.
- The length K_i of the ith shift register is given by

$$K_i = \max_{1\le j\le n}\left[\deg \mathbf{g}_i^{(j)}(D)\right], \qquad 1 \le i \le k,$$

where $\mathbf{g}_i^{(j)}(D)$ is the generator polynomial relating the ith input to the jth output, and the *encoder memory order* m is

$$m = \max_{1\le i\le k} K_i = \max_{\substack{1\le j\le n\\ 1\le i\le k}}\left[\deg \mathbf{g}_i^{(j)}\right].$$

Encoding of Convolutional Codes

- Since the encoder is a linear system, and $\mathbf{u}^{(i)}(D)$ is the ith input sequence and $\mathbf{v}^{(j)}(D)$is the jth output sequence, the generator polynomial $\mathbf{g}_i^{(j)}(D)$can be interpreted as the *encoder transfer function* relating input i to output j.
- As with k-input, n-output linear system, there are a total of $k \cdot n$ transfer functions.
- These can be represented by the $k \times n$ *transfer function matrix*

$$\mathbf{G}(\mathrm{D}) = \begin{bmatrix} \mathbf{g}_1^{(1)}(D) & \mathbf{g}_1^{(2)}(D) & \cdots & \mathbf{g}_1^{(n)}(D) \\ \mathbf{g}_2^{(1)}(D) & \mathbf{g}_2^{(2)}(D) & \cdots & \mathbf{g}_2^{(n)}(D) \\ \vdots & \vdots & & \vdots \\ \mathbf{g}_k^{(1)}(D) & \mathbf{g}_k^{(2)}(D) & \cdots & \mathbf{g}_k^{(n)}(D) \end{bmatrix}$$

Encoding of Convolutional Codes

- Using the transfer function matrix, the encoding equation for an (n, k, m) code can be expressed as

$$\mathbf{V}(D) = \mathbf{U}(D)\mathbf{G}(D)$$

where $\mathbf{U}(D) \triangleq \left[\mathbf{u}^{(1)}(D), \mathbf{u}^{(2)}(D), \cdots, \mathbf{u}^{(k)}(D)\right]$ is the k-tuple of input sequences and $\mathbf{V}(D) \triangleq \left[\mathbf{v}^{(1)}(D), \mathbf{v}^{(2)}(D), \cdots, \mathbf{v}^{(n)}(D)\right]$ is the n-tuple of output sequences.

- After multiplexing, the code word becomes

$$\mathbf{v}(D) = \mathbf{v}^{(1)}(D^n) + D\mathbf{v}^{(2)}(D^n) + \cdots + D^{n-1}\mathbf{v}^{(n)}(D^n).$$

Encoding of Convolutional Codes

- Example 10.7
  - For the previous (3, 2, 1) convolutional code

$$\mathbf{G}(D)=\begin{bmatrix}1+D & D & 1+D\\ D & 1 & 1\end{bmatrix}$$

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

  - For the input sequences $\mathbf{u}^{(1)}(D) = 1+D^2$ and $\mathbf{u}^{(2)}(D)=1+D$, the encoding equations are

$$\mathbf{V}(D)=\left[\mathbf{v}^{(1)}(D),\mathbf{v}^{(2)}(D),\mathbf{v}^{(3)}(D)\right]=\left[1+D^2,1+D\right]\begin{bmatrix}1+D & D & 1+D\\ D & 1 & 1\end{bmatrix}$$

$$=\left[1+\mathrm{D}^3,1+\mathrm{D}^3,D^2+D^3\right]$$

and the code word is

$$\mathbf{v}(\mathrm{D})=\left(1+D^9\right)+\left(1+D^9\right)D+\left(D^6+D^9\right)D^2$$

$$=1+D+D^8+D^9+D^{10}+D^{11}$$

Encoding of Convolutional Codes

- Then, we can find a means of representing the code word $\mathbf{v}(D)$ directly in terms of the input sequences.
- A little algebraic manipulation yields

$$\mathbf{v}(\mathrm{D}) = \sum_{i=1}^{k} \mathbf{u}^{(i)}\left(D^n\right)\mathbf{g}_i(D)$$

where

$$g_i(D) \triangleq \mathbf{g}_i^{(1)}\left(D^n\right) + D\mathbf{g}_i^{(2)}\left(D^n\right) + \cdots D^{n-1}\mathbf{g}_i^{(n-1)}\left(D^n\right), \quad 1 \le i \le k,$$

is a *composite generator polynomial* relating the ith input sequence to $\mathbf{v}(D)$.

Encoding of Convolutional Codes

- Example 10.8
  - For the previous (2, 1, 3) convolutional codes, the composite generator polynomial is

$$\mathbf{g}(D)=\mathbf{g}^{(1)}\left(D^2\right)+D\mathbf{g}^{(2)}\left(D^2\right)=1+D+D^3+D^4+D^5+D^6+D^7$$

and for $\mathbf{u}(D)=1+D^2+D^3+D^4$, the code word is

$$\mathbf{v}(D)=\mathbf{u}\left(D^2\right)\,\mathbf{g}(D)$$

$$=\left(1+D^4+D^6+D^8\right)\cdot\left(1+D+D^3+D^4+D^5+D^6+D^7\right)$$

$$=1+D+D^3+D^7+D^9+D^{11}+D^{14}+D^{15}$$

again agreeing with previous calculations.

Structural Properties of Convolutional Codes

- Since a convolutional encoder is a sequential circuit, its operation can be described by a state diagram.
- The state of the encoder is defined as its shift register contents.
- For an (n, k, m) code with $k > 1$, the ith shift register contains K_i previous information bits.
- Defined $K \triangleq \sum_{i=1}^{k} K_i$ as the *total encoder memory*, the encoder state at time unit l is the binary K-tuple of inputs

$$\left(u_{l-1}^{(1)} u_{l-2}^{(1)} \cdots u_{l-K_1}^{(1)} \quad u_{l-1}^{(2)} u_{l-2}^{(2)} \cdots u_{l-K_2}^{(2)} \quad \cdots \quad u_{l-1}^{(k)} u_{l-2}^{(k)} \cdots u_{l-K_k}^{(k)}\right)$$

and there are a total 2^K different possible states.

Structural Properties of Convolutional Codes

- For a $(n, 1, m)$ code, $K = K_1 = m$ and the encoder state at time unit l is simply $\left(u_{l-1}u_{l-2}\cdots u_{l-m}\right)$.
- Each new block of k inputs causes a transition to a new state.
- There are 2^k branches leaving each state, one corresponding to each different input block. Note that for an $(n, 1, m)$ code, there are only two branches leaving each state.
- Each branch is labeled with the k inputs causing the transition $\left(u_l^{(1)}u_l^{(2)}\cdots u_l^{(k)}\right)$ and n corresponding outputs $\left(\upsilon_l^{(1)}\upsilon_l^{(2)}\cdots\upsilon_l^{(n)}\right)$.
- The states are labeled $S_0, S_1, \ldots, S_{2K-1}$, where by convention S_i represents the state whose binary K-tuple representation $b_0, b_1, \ldots, b_{K-1}$ is equivalent to the integer

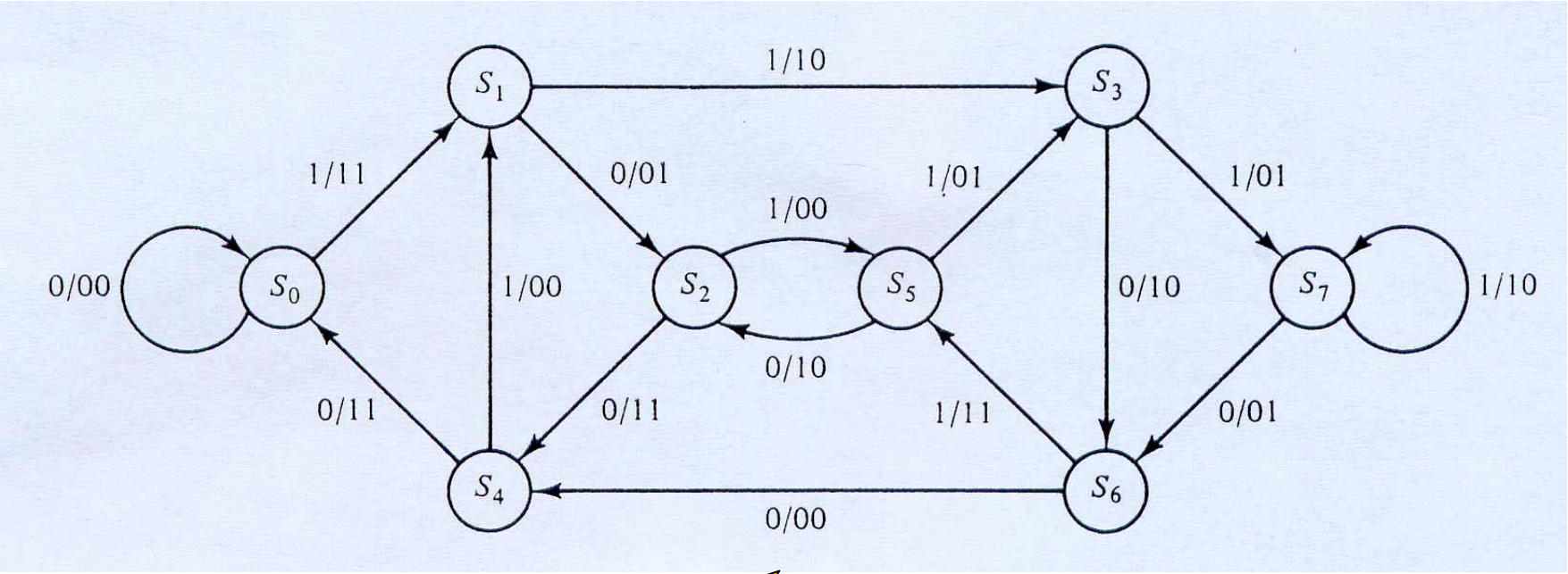
$$i = b_0 2^0 + b_1 2^1 + \cdots + b_{K-1} 2^{K-1}$$

Structural Properties of Convolutional Codes

- Assuming that the encoder is initially in state S_0 (all-zero state), the code word corresponding to any given information sequence can be obtained by following the path through the state diagram and noting the corresponding outputs on the branch labels.
- Following the last nonzero information block, the encoder is return to state S_0 by a sequence of m all-zero blocks appended to the information sequence.

Structural Properties of Convolutional Codes

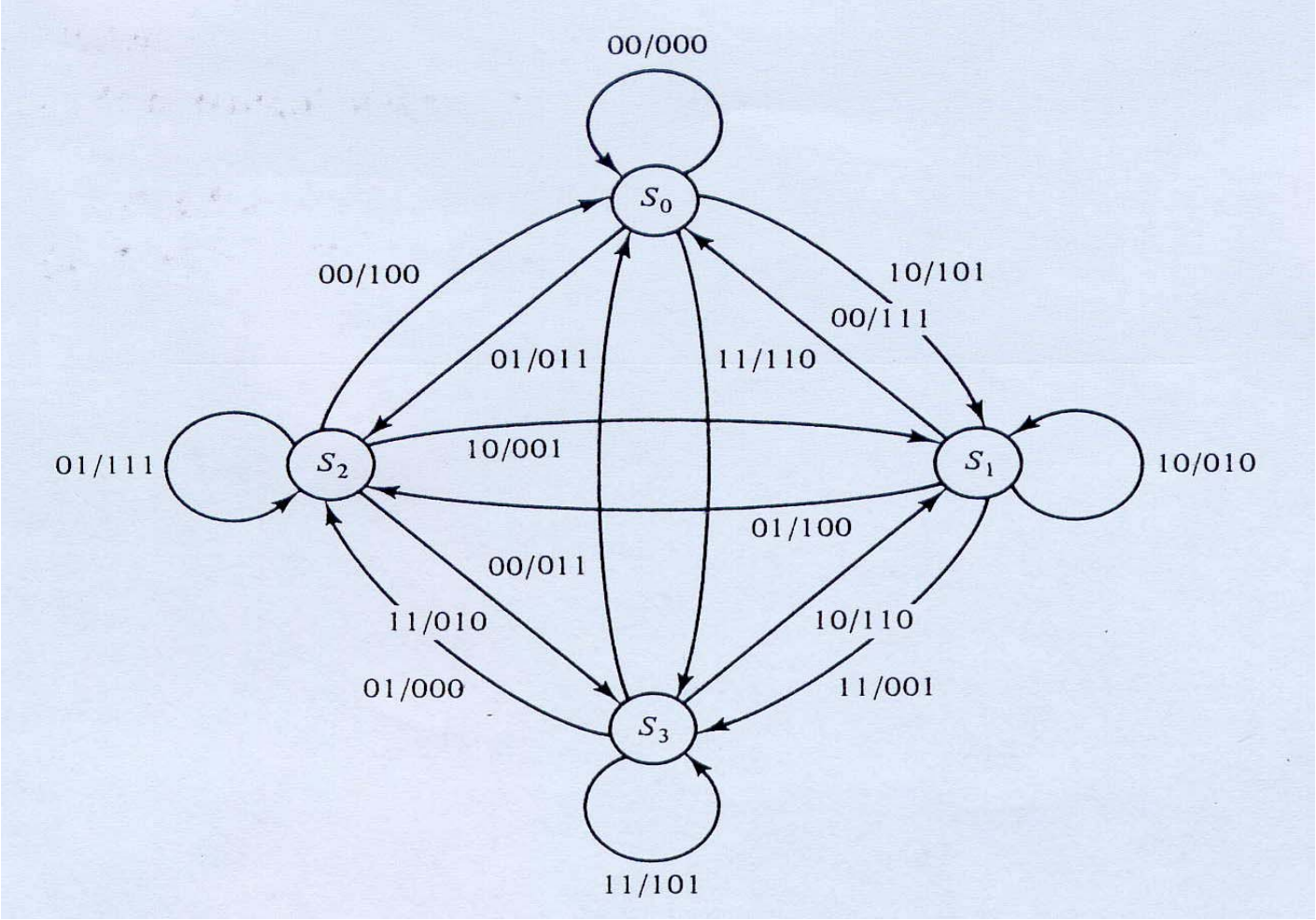
- Encoder state diagram of a (2, 1, 3) code



If **u** = (1 1 1 0 1), the code word
v = (1 1, 1 0, 0 1, 0 1, 1 1, 1 0, 1 1, 1 1)

Structural Properties of Convolutional Codes

- Encoder state diagram of a (3, 2, 1) code

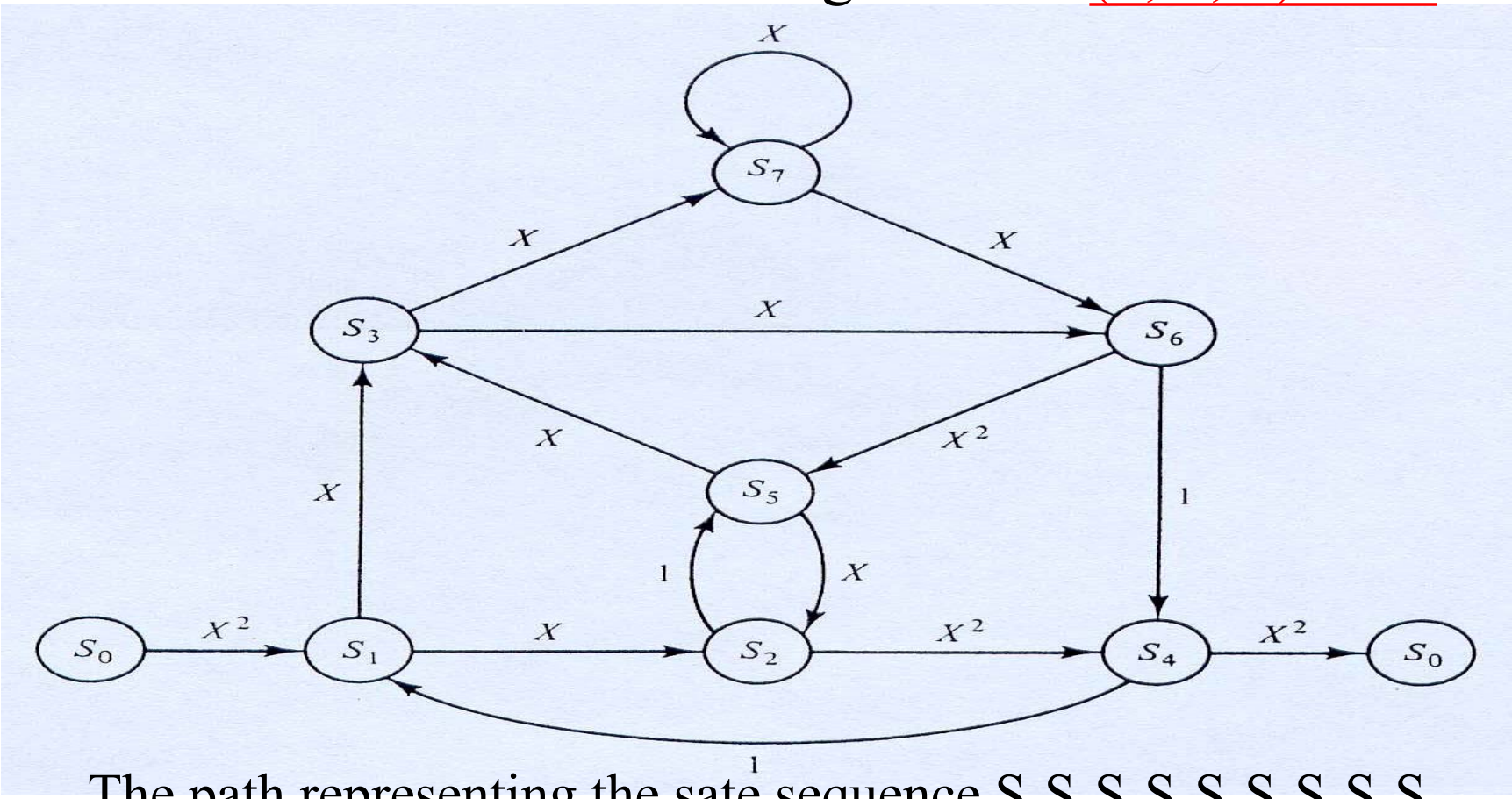


Structural Properties of Convolutional Codes

- The state diagram can be modified to provide a complete description of the *Hamming weights* of all nonzero code words (i.e. a weight distribution function for the code).
- State S_0 is split into an *initial state* and a *final state*, the self-loop around state S_0 is deleted, and each branch is labeled with a *branch gain* X^i ,where i is the weight of the n encoded bits on that branch.
- Each *path* connecting the initial state to the final state represents a nonzero code word that diverge from and remerge with state S_0 exactly once.
- The *path gain* is the product of the branch gains along a path, and the weight of the associated code word is the power of X in the path gain.

Structural Properties of Convolutional Codes

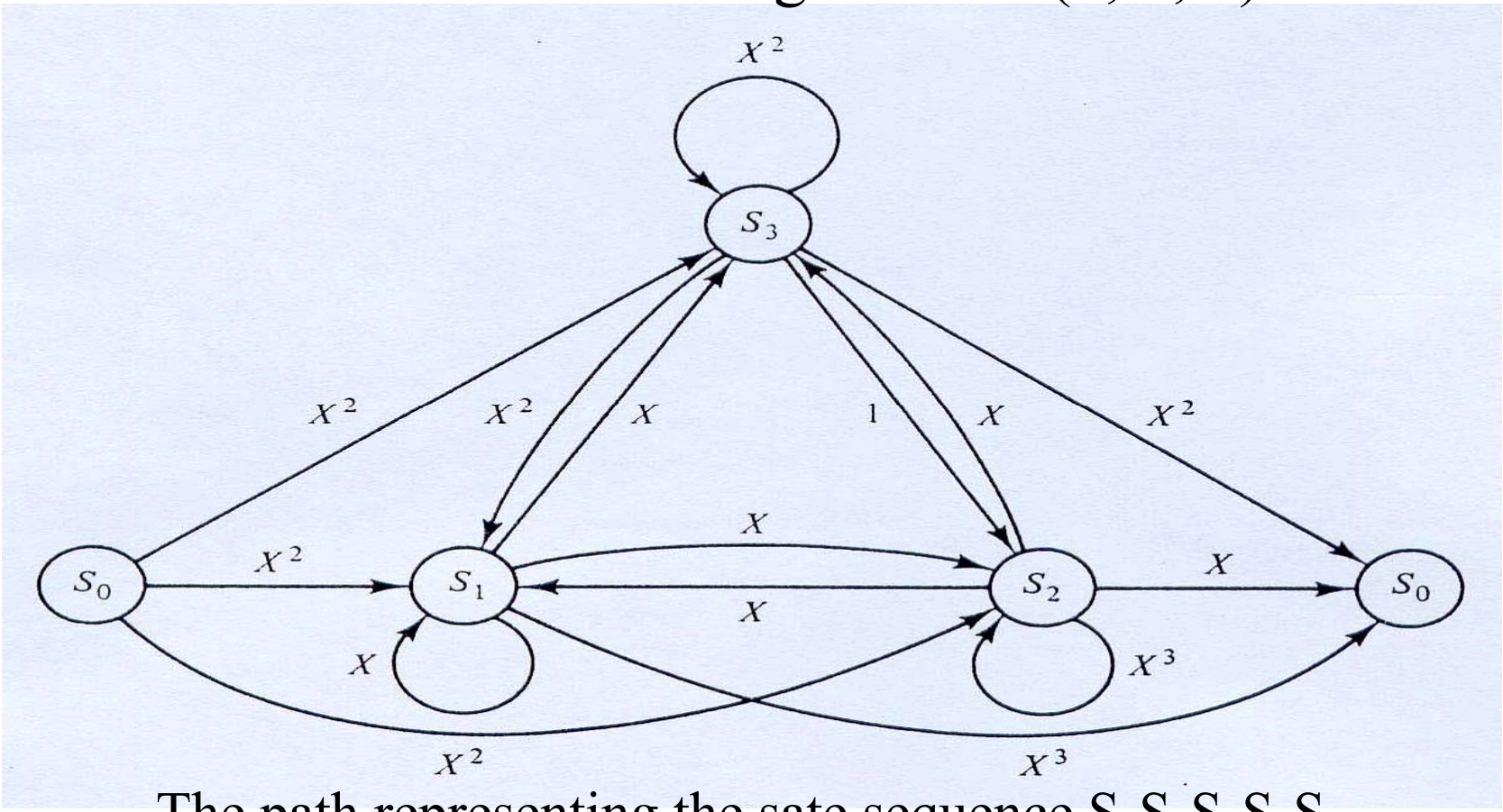
Modified encoder state diagram of a (2, 1, 3) code.



The path representing the sate sequence $S_0S_1S_3S_7S_6S_5S_2S_4S_0$ has path gain $X^2 \cdot X^1 \cdot X^1 \cdot X^1 \cdot X^2 \cdot X^1 \cdot X^2 \cdot X^2 = X^{12}$.

Structural Properties of Convolutional Codes

Modified encoder state diagram of a (3, 2, 1) code.



The path representing the sate sequence $S_0S_1S_3S_2S_0$ has path gain $X^2 \cdot X^1 \cdot X^0 \cdot X^1 = X^{12}$.

Structural Properties of Convolutional Codes

- The weight distribution function of a code can be determined by considering the modified state diagram as a signal flow graph and applying *Mason's gain formula* to compute its "generating function"

$$T(X) = \sum_i A_i X^i,$$

where A_i is the number of code words of weight i.

- In a signal flow graph, a path connecting the initial state to the final state which does not go through any state twice is called a *forward path*.
- A closed path starting at any state and returning to that state without going through any other state twice is called a *loop*.

Structural Properties of Convolutional Codes

- Let C_i be the gain of the ith loop.
- A set of loops is *nontouching* if no state belongs to more than one loop in the set.
- Let $\{i\}$ be the set of all loops, $\{i', j'\}$ be the set of all pairs of nontouching loops, $\{i'', j'', l''\}$ be the set of all triples of nontouching loops, and so on.
- Define $$\Delta = 1 - \sum_{i} C_i + \sum_{i',j'} C_{i'} C_{j'} - \sum_{i'',j'',l''} C_{i''} C_{j''} C_{l''} + \cdots,$$

  where $\sum_{i} C_i$ is the sum of the loop gains, $\sum_{i',j'} C_{i'} C_{j'}$ is the product of the loop gains of two nontouching loops summed over all pairs of nontouching loops, $\sum_{i'',j'',l''} C_{i''} C_{j''} C_{l''}$ is the product of the loop gains of three nontouching loops summed over all nontouching loops.

Structural Properties of Convolutional Codes

- And Δ_i is defined exactly like Δ, but only for that portion of the graph not touching the ith forward path; that is, all states along the ith forward path, together with all branches connected to these states, are removed from the graph when computing Δ_i.
- Mason's formula for computing the generating function $T(X)$ of a graph can now be states as

$$T(X) = \frac{\sum_i F_i \Delta_i}{\Delta},$$

where the sum in the numerator is over all forward paths and F_i is the gain of the ith forward path.

Structural Properties of Convolutional Codes

- Example (2,1,3) Code: There are 11 loops in the modified encoder state diagram.

$$
\begin{aligned}
&Loop\ 1: S_1S_3S_7S_6S_5S_2S_4S_1 && (C_1 = X^8)\\
&Loop\ 2: S_1S_3S_7S_6S_4S_1 && (C_2 = X^3)\\
&Loop\ 3: S_1S_3S_6S_5S_2S_4S_1 && (C_3 = X^7)\\
&Loop\ 4: S_1S_3S_6S_4S_1 && (C_4 = X^2)\\
&Loop\ 5: S_1S_2S_5S_3S_7S_6S_4S_1 && (C_5 = X^9)\\
&Loop\ 6: S_1S_2S_5S_3S_6S_4S_1 && (C_6 = X^8)\\
&Loop\ 7: S_1S_2S_4S_1 && (C_7 = X^3)\\
&Loop\ 8: S_2S_5S_2 && (C_8 = X)\\
&Loop\ 9: S_3S_7S_6S_5S_3 && (C_9 = X^5)\\
&Loop\ 10: S_3S_6S_5S_3 && (C_{10} = X^4)\\
&Loop\ 11: S_7S_7 && (C_{11} = X)
\end{aligned}
$$

Structural Properties of Convolutional Codes

- Example (2,1,3) Code: (cont.)
  - There are 10 pairs of nontouching loops :

Loop pair 1: $(\text{loop } 2, \text{loop } 8)$	$\left(C_2 C_8 = X^4\right)$
Loop pair 2: $(\text{loop } 3, \text{loop } 11)$	$\left(C_3 C_{11} = X^8\right)$
Loop pair 3: $(\text{loop } 4, \text{ loop } 8)$	$\left(C_4 C_8 = X^3\right)$
Loop pair 4: $(\text{loop } 4, \text{loop } 11)$	$\left(C_4 C_{11} = X^3\right)$
Loop pair 5: $(\text{loop } 6, \text{loop } 11)$	$\left(C_6 C_{11} = X^9\right)$
Loop pair 6: $(\text{loop } 7, \text{loop } 9)$	$\left(C_7 C_9 = X^8\right)$
Loop pair 7: $(\text{loop } 7, \text{loop } 10)$	$\left(C_7 C_{10} = X^7\right)$
Loop pair 8: $(\text{loop } 7, \text{loop } 11)$	$\left(C_7 C_{11} = X^4\right)$
Loop pair 9: $(\text{loop } 8, \text{loop } 11)$	$\left(C_8 C_{11} = X^2\right)$
Loop pair 10: $(\text{loop } 10, \text{loop } 11)$	$\left(C_{10} C_{11} = X^5\right)$

Structural Properties of Convolutional Codes

- Example (2,1,3) Code : (cont.)
  - There are two triples of nontouching loops :

$$\text{Loop triple } 1: \left(\text{loop } 4, \text{loop } 8, \text{loop } 11\right) \quad \left(C_4 C_8 C_{11} = X^4\right)$$

$$\text{Loop triple } 2: \left(\text{loop } 7, \text{loop } 10, \text{loop } 11\right) \quad \left(C_7 C_{10} C_{11} = X^8\right)$$

  - There are no other sets of nontouching loops. Therefore,

$$\begin{aligned}\Delta = 1 &- \left(X^8 + X^3 + X^7 + X^2 + X^4 + X^3 + X^3 + X + X^5 + X^4 + X\right) \\ &+ \left(X^4 + X^8 + X^3 + X^3 + X^4 + X^8 + X^7 + X^4 + X^2 + X^5\right) \\ &- \left(X^4 + X^8\right) = 1 - 2X + X^3\end{aligned}$$

Structural Properties of Convolutional Codes

- Example (2,1,3) Code : (cont.)
  - There are seven forward paths in this state diagram :

$$\text{Foward path 1}: \quad S_0S_1S_3S_7S_6S_5S_2S_4S_0 \quad \left(F_1 = X^{12}\right)$$

$$\text{Foward path 2}: \quad S_0S_1S_3S_7S_6S_4S_0 \quad \left(F_2 = X^{7}\right)$$

$$\text{Foward path 3}: \quad S_0S_1S_3S_6S_5S_2S_4S_0 \quad \left(F_3 = X^{11}\right)$$

$$\text{Foward path 4}: \quad S_0S_1S_3S_6S_4S_0 \quad \left(F_4 = X^{6}\right)$$

$$\text{Foward path 5}: \quad S_0S_1S_2S_5S_3S_7S_6S_4S_0 \quad \left(F_5 = X^{8}\right)$$

$$\text{Foward path 6}: \quad S_0S_1S_2S_5S_3S_6S_4S_0 \quad \left(F_6 = X^{7}\right)$$

$$\text{Foward path 7}: \quad S_0S_1S_2S_4S_0 \quad \left(F_7 = X^{7}\right).$$

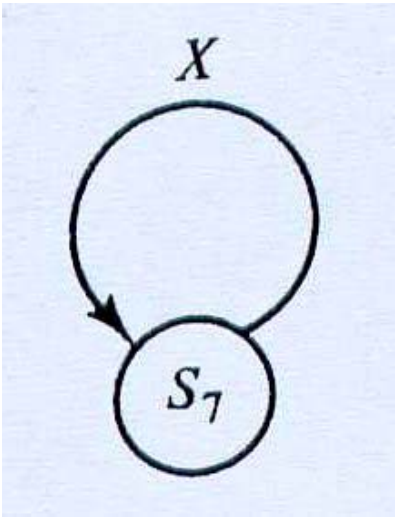
Structural Properties of Convolutional Codes

- Example (2,1,3) Code : (cont.)
  - Forward paths 1 and 5 touch all states in the graph, and hence the sub graph not touching these paths contains no states. Therefore,

$$\Delta_1 = \Delta_5 = 1.$$

  - The subgraph not touching forward paths 3 and 6:

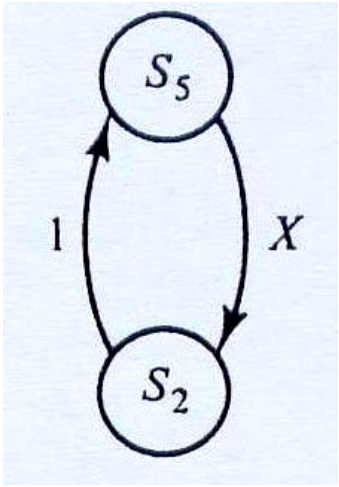
$$\Delta_3 = \Delta_6 = 1 - X$$



Structural Properties of Convolutional Codes

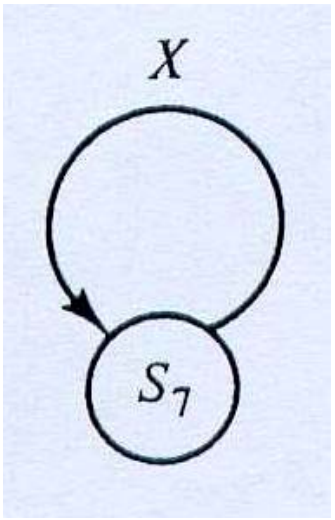
- Example (2,1,3) Code : (cont.)
  - The subgraph not touching forward path 2:

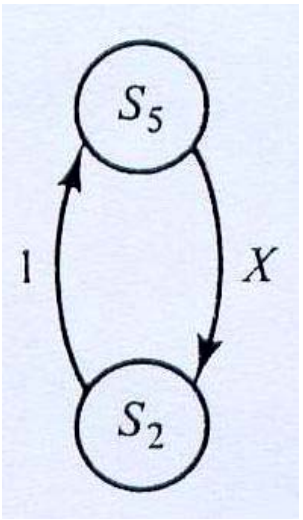
$$\Delta_2 = 1 - X$$



  - The subgraph not touching forward path 4:

$$\Delta_4 = 1 - (X + X) + (X^2)$$
$$= 1 - 2X + X^2$$

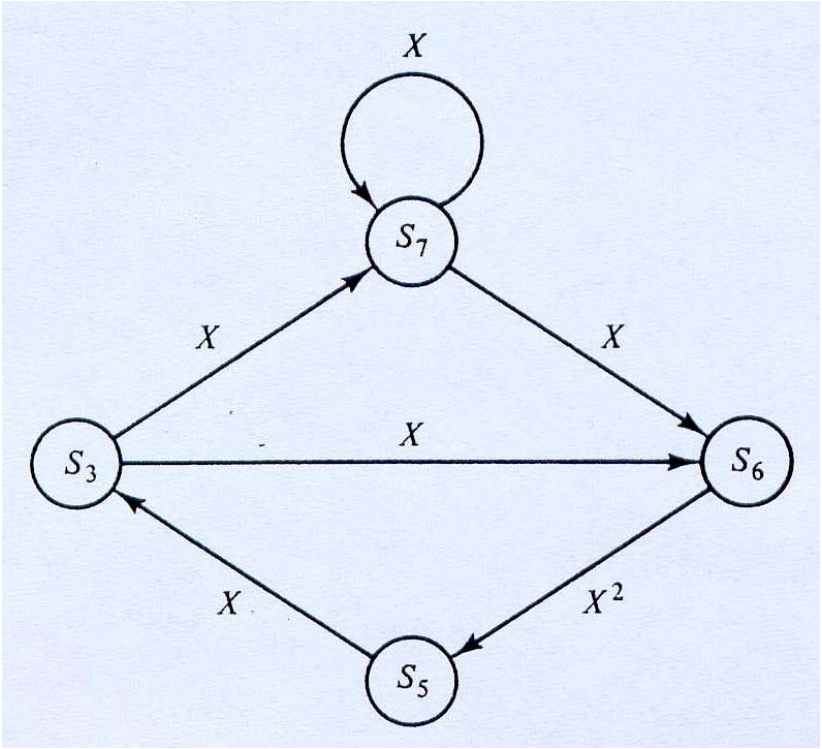




Structural Properties of Convolutional Codes

- Example (2,1,3) Code : (cont.)
  - The subgraph not touching forward path 7:

$$\Delta_7 = 1 - (X + X^4 + X^5) + (X^5)$$
$$= 1 - X - X^4$$



Structural Properties of Convolutional Codes

- Example (2,1,3) Code : (cont.)
  - The *generating function* for this graph is then given by

$$T(X) = \frac{X^{12}\cdot 1 + X^7(1-X) + X^{11}(1-X) + X^6(1-2X+X^2) + X^8\cdot 1 + X^7(1-X) + X^7(1-X-X^4)}{1-2X-X^3}$$

$$= \frac{X^6 + X^7 - X^8}{1-2X-X^3} = X^6 + 3X^7 + 5X^8 + 11X^9 + 25X^{10} + \cdots$$

  - $T(X)$ provides a complete description of the weight distribution of all nonzero code words that diverge from and remerge with state S_0 exactly once.
  - In this case, there is one such code word of weight 6, three of weight 7, five of weight 8, and so on.

Structural Properties of Convolutional Codes

- Example (3,2,1) Code : (cont.)
  - There are eight loops, six pairs of nontouching loops, and one triple of nontouching loops in the graph of previous modified encoder state diagrams : (3, 2, 1) code, and

$$\begin{aligned}\Delta &= 1-\left(X^2+X^4+X^3+X+X^2+X^1+X^2+X^3\right)\\ &\quad+\left(X^6+X^2+X^4+X^3+X^4+X^5\right)-\left(X^6\right)\\ &= 1-2X-2X^2-X^3+X^4+X^5.\end{aligned}$$

Structural Properties of Convolutional Codes

- Example (3,2,1) Code : (cont.)
  - There are 15 forward path in this graph, and

$$\sum_i F_i \Delta_i = X^5\left(1-X-X^2-X^3+X^5\right)+X^4\left(1-X^2\right)+X^6\cdot 1+X^5\left(1-X^3\right)$$
$$+X^4\cdot 1+X^3\left(1-X-X^2\right)+X^6\left(1-X^2\right)+X^6\cdot 1+X^5\left(1-X\right)$$
$$+X^8\cdot 1+X^4\left(1-X-X^2-X^3+X^4\right)+X^7\left(1-X^3\right)$$
$$+X^6\cdot 1+X^3\left(1-X\right)+X^6\cdot 1$$
$$=2X^3+X^4+X^5+X^6-X^7-X^8.$$

  - Hence, the generating function is

$$T(X)=\frac{2X^3+X^4+X^5+X^6-X^7-X^8}{1-2X-2X^2-X^3+X^4+X^5}=2X^3+5X^4+15X^5+\cdots$$

  - This code contains two nonzero code word of weight 3, five of weight 4, fifteen of weight 5, and so on.

Structural Properties of Convolutional Codes

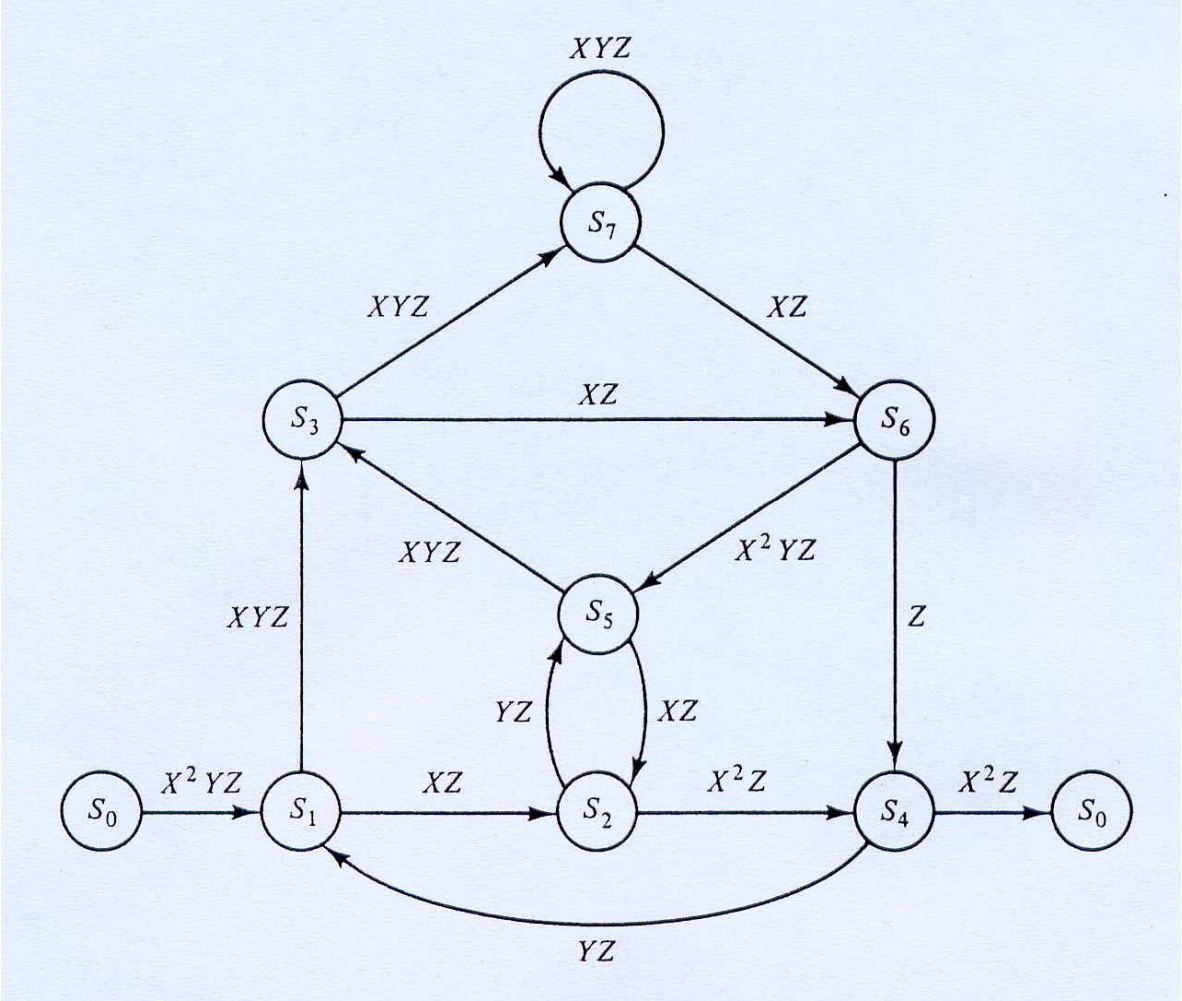
- Additional information about the structure of a code can be obtained using the same procedure.
- If the modified state diagram is augmented by labeling each branch corresponding to a nonzero information block with Y^j, where j is the weight of the k information bits on the branch, and labeling every branch with **Z**, the generating function is given by

$$T(X,Y,Z)=\sum_{i,j,l} A_{i,j,l} X^i Y^j Z^l .$$

- The coefficient $A_{i,j,l}$ denotes the number of code words with weight i, whose associated information sequence has weight j, and whose length is l branches.

Structural Properties of Convolutional Codes

The augment state diagram for the (2, 1, 3) codes.



Structural Properties of Convolutional Codes

- Example (2,1,3) Code:
  - For the graph of the augment state diagram for the (2, 1, 3) codes, we have:

$$\begin{aligned}\Delta = 1 &- (X^8Y^4Z^7 + X^3Y^3Z^5 + X^7Y^3Z^6 + X^2Y^2Z^4 + X^4Y^4Z^7 \\ &+ X^3Y^3Z^6 + X^3YZ^3 + XYZ^2 + X^5Y^3Z^4 + X^4Y^2Z^3 + XYZ) \\ &+ (X^4Y^4Z^7 + X^8Y^4Z^7 + X^3Y^3Z^6 + X^3Y^3Z^5 + X^4Y^4Z^7 \\ &+ X^8Y^4Z^7 + X^7Y^3Z^6 + X^4Y^2Z^4 + X^2Y^2Z^3 + X^5Y^3Z^4) \\ &- \left(X^4Y^4Z^7 + X^8Y^4Z^7\right) \\ &= 1 + XY\left(Z + Z^2\right) - X^2Y^2\left(Z^4 - Z^3\right) - X^3\left(YZ^3 - Y^3Z^6\right) \\ &- X^4Y^2\left(Z^3 - Z^4\right) - X^8\left(Y^3Z^6 - Y^4Z^7\right) - X^9Y^4Z^7\end{aligned}$$

Structural Properties of Convolutional Codes

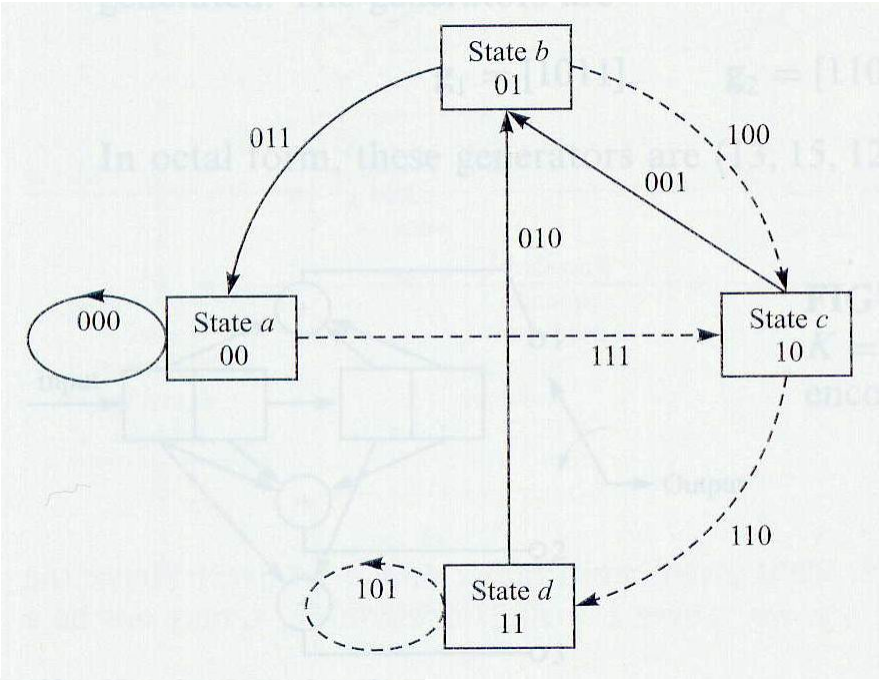
- Example (2,1,3) Code: (cont.)
  - $$\sum_i F_i \Delta_i = X^{12}Y^4Z^8 \cdot 1 + X^7Y^3Z^6\left(1 - XYZ^2\right) + X^{11}Y^3Z^7\left(1 - XYZ\right)$$
    $$+ X^6Y^2Z^5(1 - XY\left(Z + Z^2\right) + X^2Y^2Z^3) + X^8Y^4Z^8 \cdot 1$$
    $$+ X^7Y^3Z^7\left(1 - XYZ\right) + X^7YZ^4\left(1 - XYZ - X^4Y^2Z^3\right)$$
    $$= X^6Y^2Z^5 + X^7YZ^4 - X^8Y^2Z^5$$
  - Hence the generating function is
    $$T(X,Y,Z) = \frac{X^6Y^2Z^5 + X^7YZ^4 - X^8Y^2Z^5}{\Delta}$$
    $$= X^6Y^2Z^5 + X^7\left(YZ^4 + Y^3Z^6 + Y^3Z^7\right)$$
    $$+ X^8\left(Y^2Z^6 + Y^4Z^7 + Y^4Z^8 + 2Y^4Z^9\right) + \cdots$$

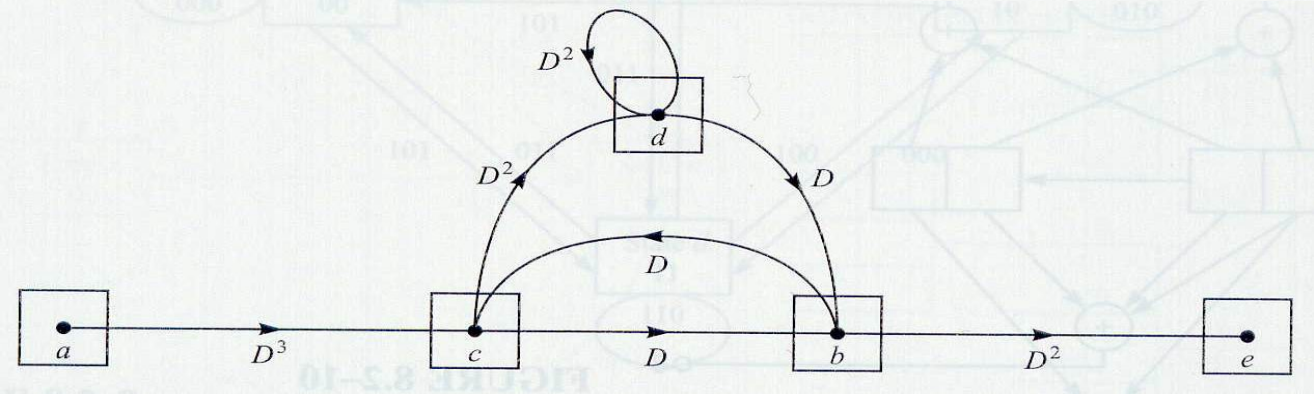
Structural Properties of Convolutional Codes

- Example (2,1,3) Code: (cont.)
  - This implies that the code word of weight 6 has length 5 branches and an information sequence of weight 2, one code word of weight 7 has length 4 branches and information sequence weight 1, another has length 6 branches and information sequence weight 3, the third has length 7 branches and information sequence weight 3, and so on.

The Transfer Function of a Convolutional Code

- The state diagram can be used to obtain the distance property of a convolutional code.
- Without loss of generality, we assume that the all-zero code sequence is the input to the encoder.





The Transfer Function of a Convolutional Code

- First, we label the branches of the state diagram as either $D^0=1$, D^1, D^2, or D^3, where the exponent of D denotes the Hamming distance between the sequence of output bits corresponding to each branch and the sequence of output bits corresponding to the all-zero branch.
- The self-loop at node a can be eliminated, since it contributes nothing to the distance properties of a code sequence relative to the all-zero code sequence.
- Furthermore, node a is split into two nodes, one of which represents the input and the other the output of the state diagram.

The Transfer Function of a Convolutional Code

- Use the modified state diagram, we can obtain four state equations:

$$X_c = D^3 X_a + D X_b$$

$$X_b = D X_c + D X_d$$

$$X_d = D^2 X_c + D^2 X_d$$

$$X_e = D^2 X_b$$

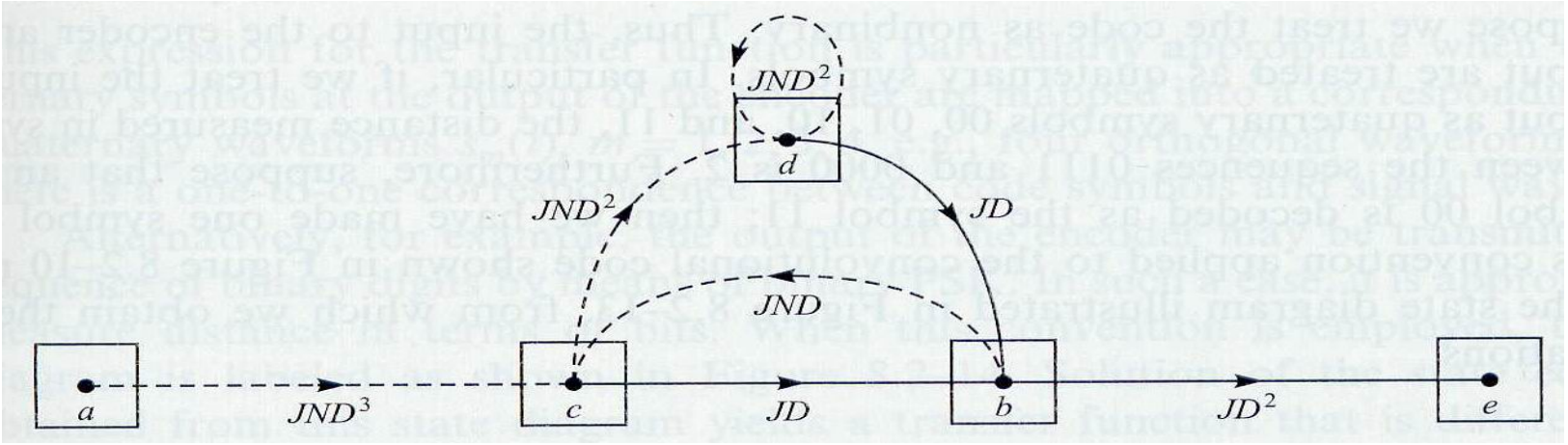
- The transfer function for the code is defined as $T(D)=X_e/X_a$. By solving the state equations, we obtain:

$$T(D) = \frac{D^6}{1-2D^2} = D^6 + 2D^8 + 4D^{10} + 8D^{12} + \cdots = \sum_{d=6}^{\infty} a_d D^d$$

$$a_d = \begin{cases} 2^{(d-6)/2} & (\text{even d}) \\ 0 & (\text{odd d}) \end{cases}$$

The Transfer Function of a Convolutional Code

- The transfer function can be used to provide more detailed information than just the distance of the various paths.
- Suppose we introduce a factor N into all branch transitions caused by the input bit 1.
- Furthermore, we introduce a factor of J into each branch of the state diagram so that the exponent of J will serve as a counting variable to indicate the number of branches in any given path from node a to node e.



The Transfer Function of a Convolutional Code

- The state equations for the state diagram are:

$$X_c = JND^3 X_a + JNDX_b$$

$$X_b = JDX_c + JDX_d$$

$$X_d = JND^2 X_c + JND^2 X_d$$

$$X_e = JD^2 X_b$$

- Upon solving these equations for the ratio X_e/X_a, we obtain the transfer function:

$$T(D,N,J) = \frac{J^3 ND^6}{1 - JND^2(1+J)}$$

$$= J^3 ND^6 + J^4 N^2 D^8 + J^5 N^2 D^8 + J^5 N^3 D^{10}$$

$$+ 2J^6 N^3 D^{10} + J^7 N^3 D^{10} + \cdots$$

The Transfer Function of a Convolutional Code

- The exponent of the factor J indicates the length of the path that merges with the all-zero path for the first time.
- The exponent of the factor N indicates the number of 1s in the information sequence for that path.
- The exponent of D indicates the distance of the sequence of encoded bits for that path from the all-zero sequence.

- Reference:

  John G. Proakis, "*Digital Communications*," Fourth Edition, pp. 477— 482, McGraw-Hill, 2001.

Structural Properties of Convolutional Codes

- An important subclass of convolutional codes is the class of *systematic codes*.
- In a systematic code, the first k output sequences are exact replicas of the k input sequences, i.e.,

$$\mathbf{v}^{(i)} = \mathbf{u}^{(i)}, \qquad i = 1, 2, \ldots, k,$$

and the generator sequences satisfy

$$\mathbf{g}_i^{(j)} = \begin{cases} 1 & \text{if } j = i \\ 0 & \text{if } j \neq i \end{cases}, \quad i = 1,2,\ldots k,$$

Structural Properties of Convolutional Codes

- The generator matrix is given by

$$\mathbf{G} = \begin{bmatrix} \mathbf{I} & \mathbf{P}_0 & \mathbf{0} & \mathbf{P}_1 & \mathbf{0} & \mathbf{P}_2 & \cdots & \mathbf{0} & \mathbf{P}_m & \cdots & & & \\ & & \mathbf{I} & \mathbf{P}_0 & \mathbf{0} & \mathbf{P}_1 & \cdots & \mathbf{0} & \mathbf{P}_{m-1} & \mathbf{0} & \mathbf{P}_m & & \\ & & & & \mathbf{I} & \mathbf{P}_0 & \cdots & \mathbf{0} & \mathbf{P}_{m-2} & \mathbf{0} & \mathbf{P}_{m-1} & \mathbf{0} & \mathbf{P}_m \\ & & & & & \ddots & & & & & & \ddots & \\ & & & & & \cdots & & & & & & & \end{bmatrix}$$

where $\mathbf{I}$ is the $k \times k$ identity matrix, $\mathbf{0}$ is the $k \times k$ all-zero matrix, and $\mathbf{P}_l$ is the $k \times (n-k)$ matrix

$$\mathbf{P}_l = \begin{bmatrix} \mathbf{g}_{1,l}^{(k+1)} & \mathbf{g}_{1,l}^{(k+2)} & \cdots & \mathbf{g}_{1,l}^{(n)} \\ \mathbf{g}_{2,l}^{(k+1)} & \mathbf{g}_{2,l}^{(k+2)} & \cdots & \mathbf{g}_{2,l}^{(n)} \\ \vdots & \vdots & & \vdots \\ \mathbf{g}_{k,l}^{(k+1)} & \mathbf{g}_{k,l}^{(k+2)} & \cdots & \mathbf{g}_{k,l}^{(n)} \end{bmatrix},$$

Structural Properties of Convolutional Codes

- And the *transfer matrix* becomes

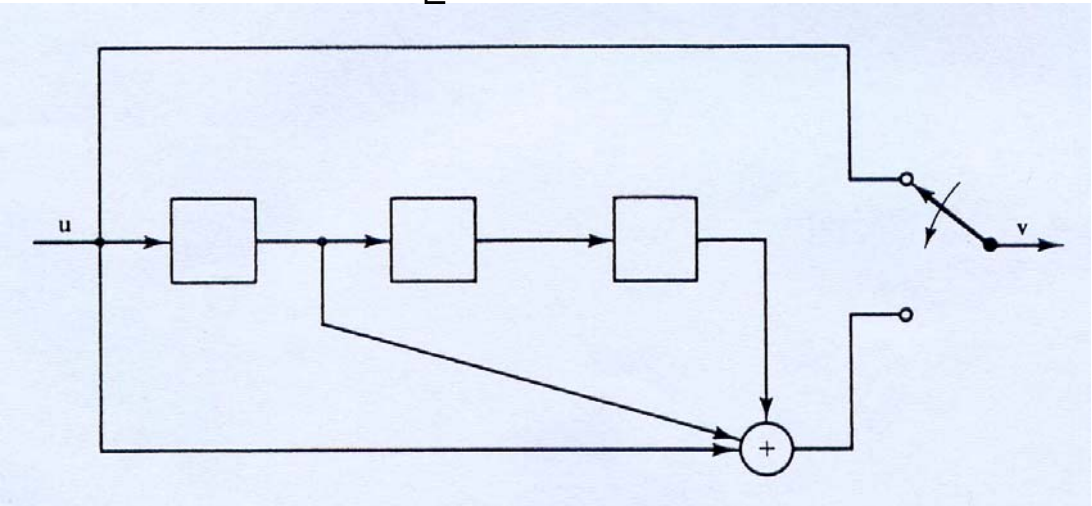
$$\mathbf{G}(D)=\begin{bmatrix} 1 & 0 & \cdots & 0 & \mathbf{g}_1^{(k+1)}(D) & \cdots & \mathbf{g}_1^{(n)}(D) \\ 0 & 1 & \vdots & 0 & \mathbf{g}_2^{(k+1)}(D) & \cdots & \mathbf{g}_2^{(n)}(D) \\ \vdots & \vdots & & \vdots & \vdots & & \\ 0 & 0 & \cdots & 1 & \mathbf{g}_k^{(k+1)}(D) & \cdots & \mathbf{g}_k^{(n)}(D) \end{bmatrix}$$

- Since the first k output sequences equal the input sequences, they are called *information sequences* and the last $n - k$ output sequences are called *parity sequences*.
- Note that whereas in general $k \cdot n$ generator sequences must be specified to define an (n, k, m) convolutional code, only $k \cdot (n - \mathrm{k})$ sequences must be specified to define a systematic code.
- Systematic codes represent a subclass of the set of all possible codes.

Structural Properties of Convolutional Codes

- Example (2,1,3) Code:
  - Consider the (2, 1, 3) systematic code. The generator sequences are $g^{(1)}=(1\ 0\ 0\ 0)$ and $g^{(2)}=(1\ 1\ 0\ 1)$. The generator matrix is

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & & & & \\ & & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & & \\ & & & & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ & & & & & \ddots & & & & & & \ddots \end{bmatrix}$$



The (2, 1, 3) systematic code.

Structural Properties of Convolutional Codes

- Example (2,1,3) Code:
  - The transfer function matrix is

g(1)=(1 0 0 0) and g(2)=(1 1 0 1).

$$\mathbf{G}(D) = [1 \quad 1 + D + D^3].$$

  - For an input sequence $\mathbf{u}(D) = 1 + D^2 + D^3$, the information sequence is

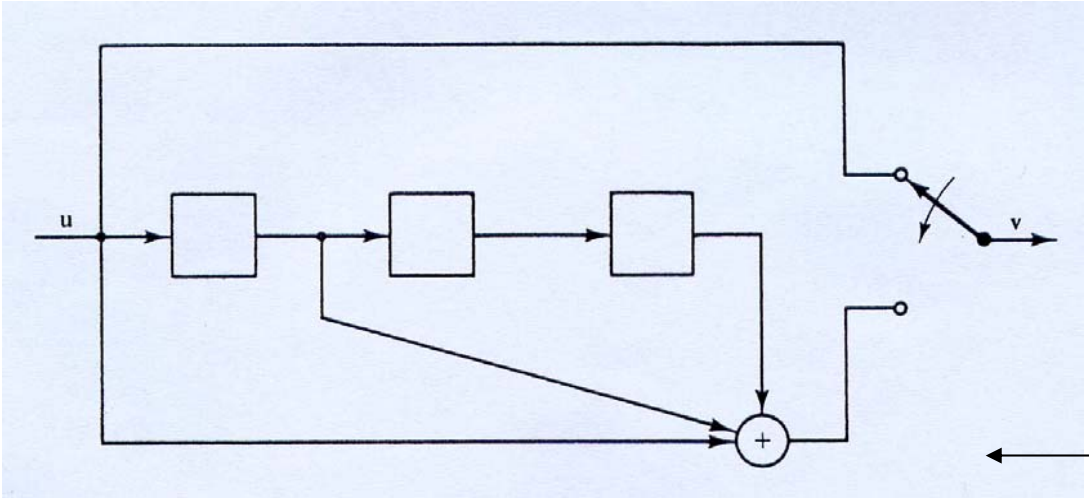
$$\mathbf{v}^{(1)}(D) = \mathbf{u}(D)\mathbf{g}^{(1)}(D) = \left(1 + D^2 + D^3\right)\cdot 1 = 1 + D^2 + D^3$$

and the parity sequence is

$$\mathbf{v}^{(2)}(D) = \mathbf{u}(D)\mathbf{g}^{(2)}(D) = \left(1 + D^2 + D^3\right)\left(1 + D + D^3\right)$$
$$= 1 + D + D^2 + D^3 + D^4 + D^5 + D^6$$

Structural Properties of Convolutional Codes

- One advantage of systematic codes is that encoding is somewhat simpler than for nonsystematic codes because less hardware is required.
- For an (n, k, m) systematic code with $k > n - k$, there exits a modified encoding circuit which normally requires fewer than K shift register states.



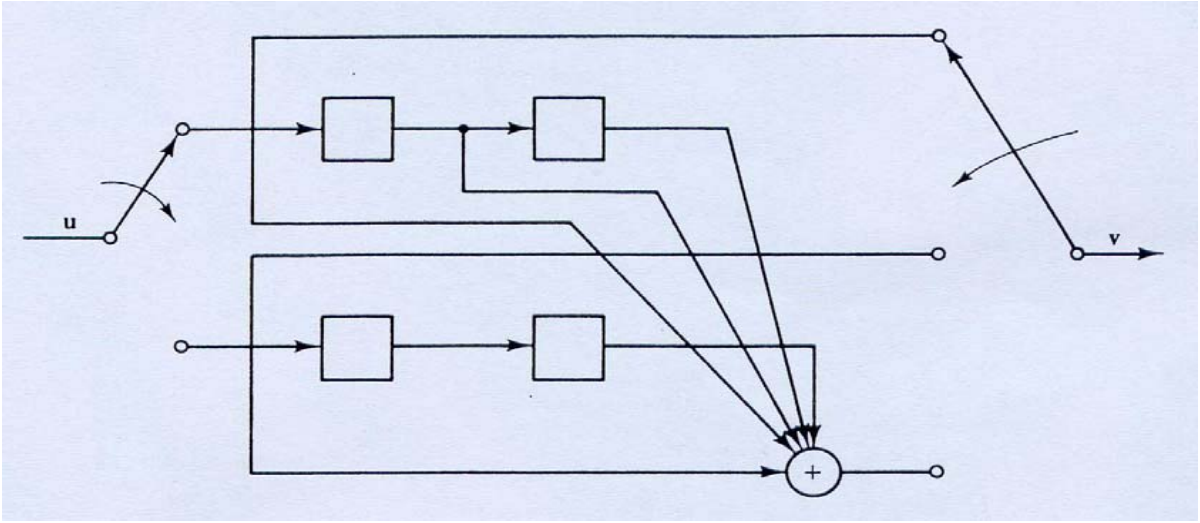
The (2, 1, 3) systematic code requires only one modulo-2 adder with three inputs.

Structural Properties of Convolutional Codes

- Example (3,2,2) Systematic Code:
  - Consider the (3, 2, 2) systematic code with transfer function matrix

$$\mathbf{G}(D)=\begin{bmatrix}1 & 0 & 1+D+D^2\\ 0 & 1 & 1+D^2\end{bmatrix}$$

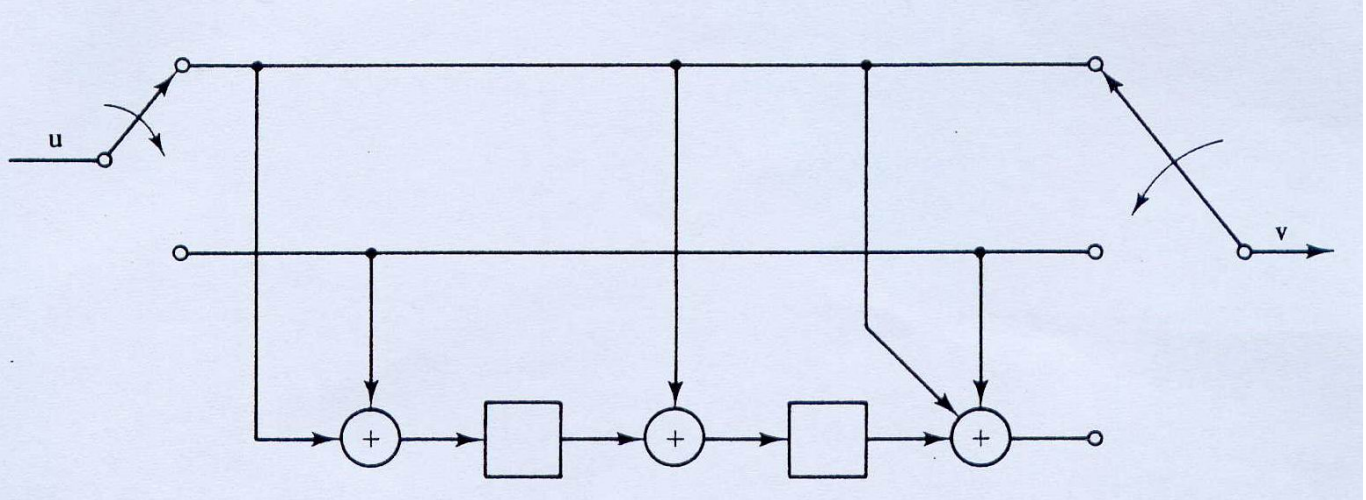
  - The straightforward realization of the encoder requires a total of $K = K_1 + K_2 = 4$ shift registers.



Structural Properties of Convolutional Codes

- Example (3,2,2) Systematic Code:
  - Since the information sequences are given by $\mathbf{v}^{(1)}(D) = \mathbf{u}^{(1)}(D)$ and $\mathbf{v}^{(2)}(D) = \mathbf{u}^{(2)}(D)$, and the parity sequence is given by

$$\mathbf{v}^{(3)}(D) = \mathbf{u}^{(1)}(D)\mathbf{g}_1^{(3)}(D) + \mathbf{u}^{(2)}(D)\mathbf{g}_2^{(3)}(D),$$



  - The (3, 2, 2) systematic encoder, and it requires only two stages of encoder memory rather than 4.

Structural Properties of Convolutional Codes

- A complete discussion of the minimal encoder memory required to realize a convolutional code is given by Forney.
- In most cases the straightforward realization requiring K states of shift register memory is most efficient.
- In the case of an (n,k,m) systematic code with $k>n-k$, a simpler realization usually exists.
- Another advantage of systematic codes is that no inverting circuit is needed for recovering the information sequence from the code word.

Structural Properties of Convolutional Codes

- Nonsystematic codes, on the other hand, require an inverter to recover the information sequence; that is, an $n \times k$ matrix $\mathbf{G}^{-1}(D)$ must exit such that

$$\mathbf{G}(D)\mathbf{G}^{-1}(D) = \mathbf{I}D^l$$

for some $l \geq 0$, where $\mathbf{I}$ is the $k \times k$ identity matrix.

- Since $\mathbf{V}(D) = \mathbf{U}(\mathrm{D})\mathbf{G}(\mathrm{D})$, we can obtain

$$\mathbf{V}(D)\mathbf{G}^{-1}(D) = \mathbf{U}(D)\mathbf{G}(D)\mathbf{G}^{-1}(D) = \mathbf{U}(D)D^l ,$$

and the information sequence can be recovered with an l-time-unit delay from the code word by letting $\mathbf{V}(D)$ be the input to the n-input, k-output linear sequential circuit whose transfer function matrix is $\mathbf{G}^{-1}(D)$.

Structural Properties of Convolutional Codes

- For an $(n, 1, m)$ code, a transfer function matrix $\mathbf{G}(D)$ has a feedforward inverse $\mathbf{G}^{-1}(D)$ of delay l if and only if

$$\text{GCD}[\mathbf{g}^{(1)}(D), \mathbf{g}^{(2)}(D), \ldots, \mathbf{g}^{(n)}(D)] = D^l$$

  for some $l \geq 0$, where GCD denotes the greatest common divisor.

- For an (n, k, m) code with $k > 1$, let $\Delta_i(D), \quad i = 1, 2, \cdots, \binom{n}{k}$, be the determinants of the $\binom{n}{k}$ distinct $k \times k$ submatrices of the transfer function matrix $\mathbf{G}(D)$.

- A feedforward inverse of delay l exits if and only if

$$\text{GCD}\left[\Delta_i(D) : i = 1, 2, \cdots, \binom{n}{k}\right] = D^l$$

  for some $l \geq 0$.

Structural Properties of Convolutional Codes

- Example (2,1,3) Code:
  - For the (2, 1, 3) code,

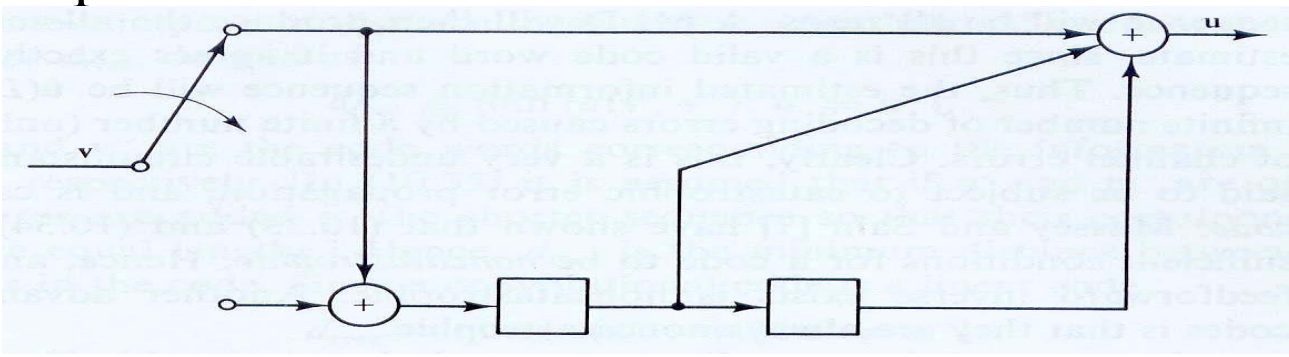
$$\text{GCD}\left[1+D^2+D^3,\ 1+D+D^2+D^3\right]=1=D^0$$

and the transfer function matrix

$$\mathbf{G}^{-1}(D)=\begin{bmatrix}1+D+D^2\\ D+D^2\end{bmatrix}$$

provides the required inverse of delay 0 [i.e., $\mathbf{G}(D)\mathbf{G}^{-1}(D)=1$].

  - The implementation of the inverse is shown below



Structural Properties of Convolutional Codes

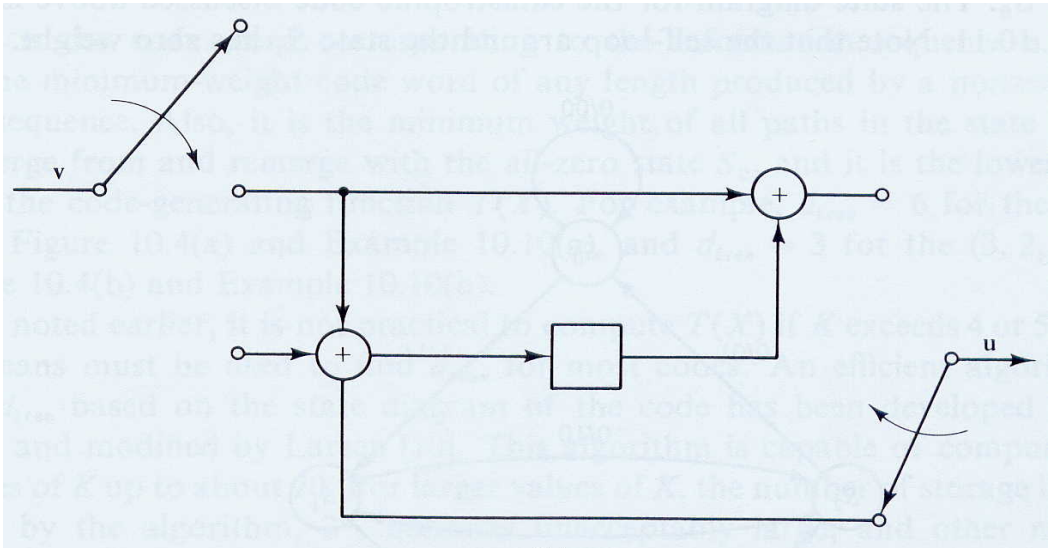
- Example (3,2,1) Code:
  - For the (3, 2, 1) code , the 2×2 submatrices of $\mathbf{G}(D)$ yield determinants $1 + D + D^2$, $1 + D^2$, and 1. Since

$$GCD\left[1+D+D^2,\ 1+D^2,\ 1\right]=1$$

there exists a feedforward inverse of delay 0.

  - The required transfer function matrix is given by:

$$\mathbf{G}^{-1}(D)=\begin{bmatrix}0 & 0\\ 1 & 1+D\\ 1 & D\end{bmatrix}$$



Structural Properties of Convolutional Codes

- To understand what happens when a feedforward inverse does not exist, it is best to consider an example.
  - For the (2, 1, 2) code with $\mathbf{g}^{(1)}(D) = 1 + D$ and $\mathbf{g}^{(2)}(D) = 1 + D^2$,

$$\mathrm{GCD}\left[1+D,\ 1+D^2\right]=1+D,$$

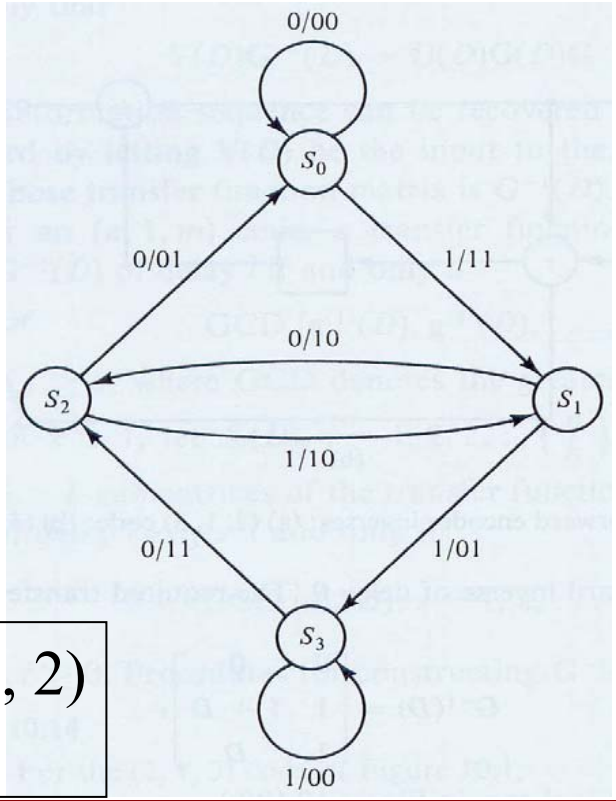
    and a feedforward inverse does not exist.
  - If the information sequence is $\mathbf{u}(D) = 1/(1 + D) = 1 + D + D^2 + \ldots$, the output sequences are $\mathbf{v}^{(1)}(D) = 1$ and $\mathbf{v}^{(2)}(D) = 1 + D$; that is, the code word contains only three nonzero bits even though the information sequence has infinite weight.
  - If this code word is transmitted over a BSC, and the three nonzero bits are changed to zeros by the channel noise, the received sequence will be all zeros.

Structural Properties of Convolutional Codes

- A MLD will then produce the all-zero code word as its estimated, since this is a valid code word and it agrees exactly with the received sequence.
- The estimated information sequence will be $\hat{\mathbf{u}}(D) = 0$, implying an infinite number of decoding errors caused by a finite number (only three in this case) of channel errors.
- Clearly, this is a very undesirable circumstance, and the code is said to be subject to catastrophic error propagation, and is called a *catastrophic code*.
- $\mathrm{GCD}\left[\mathbf{g}^{(1)}(D), \mathbf{g}^{(2)}(D), \ldots, \mathbf{g}^{(n)}(D)\right] = D^l$ and $\mathrm{GCD}\left[\Delta_i(D) : i = 1, 2, \ldots, \binom{n}{k}\right] = D^l$ are necessary and sufficient conditions for a code to be *noncatastrophic*.

Structural Properties of Convolutional Codes

- Any code for which a feedforward inverse exists is noncatastrophic.
- Another advantage of systematic codes is that they are always noncatastrophic.
- A code is catastrophic if and only if the state diagram contains a loop of zero weight other than the self-loop around the state S_0.
- Note that the self-loop around the state S_3 has zero weight.



State diagram of a (2, 1, 2) catastrophic code.

Structural Properties of Convolutional Codes

- In choosing nonsystematic codes for use in a communication system, it is important to avoid the selection of catastrophic codes.
- Only a fraction $1/(2^n - 1)$ of $(n, 1, m)$ nonsystematic codes are catastrophic.
- A similar result for (n, k, m) codes with $k > 1$ is still lacking.

Convolutional Decoder and Its Applications

Introduction

- In 1967, Viterbi introduced a decoding algorithm for convolutional codes which has since become known as Viterbi algorithm.
- Later, Omura showed that the Viterbi algorithm was equivalent to finding the shortest path through a weighted graph.
- Forney recognized that it was in fact a maximum likelihood decoding algorithm for convolutional codes; that is, the decoder output selected is always the code word that gives the largest value of the log-likelihood function.

The Viterbi Algorithm

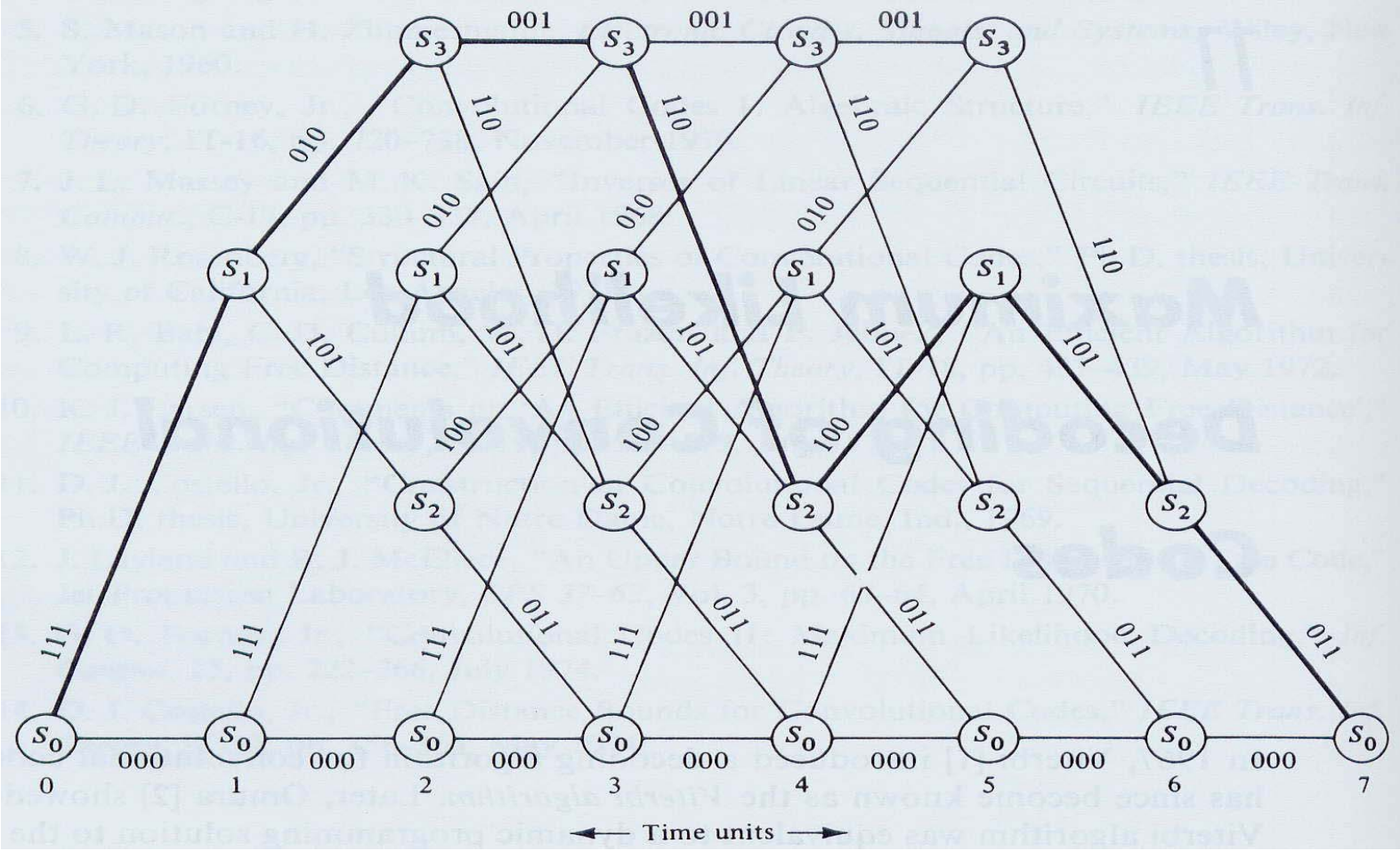
- In order to understand Viterbi's decoding algorithm, it is convenient to expand the state diagram of the encoder in time (i.e., to represent each time unit with a separate state diagram).
- The resulting structure is called a *trellis diagram*, and is shown in Figure (a) for the (3, 1, 2) code with

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

  and an information sequence of length $L=5$.
- The trellis diagram contains $L+m+1$ time units or levels, and these are labeled from 0 to $L+m$ in Figure (a).
- Assuming that the encoder always starts in state S_0 and returns to state S_0, the first m time units correspond to the encoder's departure from state S_0, and the last m time units correspond to the encoder's return to state S_0.

The Viterbi Algorithm

- Figure (a): Trellis diagram for a (3, 1, 2) code with L=5.



The Viterbi Algorithm

- Not all states can be reached in the first m or the last m time units. However, in the center portion of the trellis, all states are possible, and each time unit contains a replica of the state diagram.
- There are two branches leaving and entering each state.
  - The upper branch leaving each state at time unit i represents the input $u_i = 1$, while the lower branch represents $u_i = 0$.
- Each branch is labeled with the n corresponding outputs v_i, and each of the 2^L code words of length $N = n(L + m)$ is represented by a unique path through the trellis.
- For example, the code word corresponding to the information sequence u = (1 1 1 0 1) is shown highlighted in Figure (a).

The Viterbi Algorithm

- In the general case of an (n, k, m) code and an information sequence of length kL, there are 2^k branches leaving and entering each state, and 2^{kL} distinct paths through the trellis corresponding to the 2^{kL} code words.
- Now assume that an information sequence $\mathbf{u} = (\mathbf{u}_0, \ldots, \mathbf{u}_{L-1})$ of length kL is encoded into a code word $\mathbf{v} = (\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{L+m-1})$ of length $N = n(L + m)$, and that a sequence $\mathbf{r} = (\mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_{L+m-1})$ is received over a *discrete memoryless channel* (DMC).
- Alternatively, these sequences can be written as $\mathbf{u} = (u_0, \ldots, u_{kL-1})$, $\mathbf{v} = (v_0, v_1, \ldots, v_{N-1})$, $\mathbf{r} = (r_0, r_1, \ldots, r_{N-1})$, where the subscripts now simply represent the ordering of the symbols in each sequence.

The Viterbi Algorithm

- As a general rule of detection, the decoder must produce an estimate $\hat{\mathbf{v}}$ of the code word $\mathbf{v}$ based on the received sequence $\mathbf{r}$.
- A *maximum likelihood decoder* (MLD) for a DMC chooses $\hat{\mathbf{v}}$ as the code word $\mathbf{v}$ which maximizes the *log-likelihood function* $\log P(\mathbf{r}\,|\,\mathbf{v})$.
- Since for a DMC

$$P(\mathbf{r}\,|\,\mathbf{v}) = \prod_{i=0}^{L+m-1} P(\mathbf{r}_i\,|\,\mathbf{v}_i) = \prod_{i=0}^{N-1} P(r_i\,|\,v_i),$$

it follows that

$$\log P(\mathbf{r}\,|\,\mathbf{v}) = \sum_{i=0}^{L+m-1} \log P(\mathbf{r}_i\,|\,\mathbf{v}_i) = \sum_{i=0}^{N-1} \log P(r_i\,|\,v_i) \text{------ (A)}$$

where $P(r_i\,|\,v_i)$ is a *channel transition probability*.

- This is a minimum error probability decoding rule when all code words are equally likely.

The Viterbi Algorithm

- The log-likelihood function $\log P(\mathbf{r}\,|\mathbf{v})$ is called the *metric* associated with the path $\mathbf{v}$, and is denoted $M(\mathbf{r}\,|\mathbf{v})$.
- The terms $\log P(\mathbf{r}_i\,|\mathbf{v}_i)$ in the sum of Equation (A) are called *branch metrics*, and are denoted $M(\mathbf{r}_i\,|\mathbf{v}_i)$, whereas the terms $\log P(r_i\,|v_i)$ are called *bit metrics*, and are denoted $M(r_i\,|v_i)$.
- The path metric $M(\mathbf{r}\,|\mathbf{v})$ can be written as

$$M(\mathbf{r}\,|\,\mathbf{v}) = \sum_{i=0}^{L+m-1} M(\mathbf{r}_i\,|\,\mathbf{v}_i) = \sum_{i=0}^{N-1} M(r_i\,|\,v_i).$$

- The decision made by the log-likelihood function is called the *soft-decision*.
- If the channel is added with AWGN, soft-decision decoding leads to finding the path with minimum Euclidean distance.

The Viterbi Algorithm

- A partial path metric for the first j branches of a path can now be expressed as

$$M\left(\left[\mathbf{r} \mid \mathbf{v}\right]_j\right) = \sum_{i=0}^{j-1} M\left(\mathbf{r}_i \mid \mathbf{v}_i\right).$$

- The following algorithm, when applied to the received sequence **r** from a DMC, finds the path through the trellis with the largest metric (i.e., the *maximum likelihood path*).
- The algorithm processes **r** in an iterative manner.
- At each step, it compares the metrics of all paths entering each state, and stores the path with the largest metric, called the *survivor*, together with its metric.

The Viterbi Algorithm

- The Viterbi Algorithm

Step 1. Beginning at time unit $j = m$, compute the partial metric for the single path entering each state. Store the path (the survivor) and its metric for each state.

Step 2. Increase j by 1. Compute the partial metric for all the paths entering a state by adding the branch metric entering that state to the metric of the connecting survivor at the preceding time unit. For each state, store the path with the largest metric (the survivor), together with its metric, and eliminate all other paths.

Step 3. If $j < L + m$, repeat step 2. Otherwise, stop.

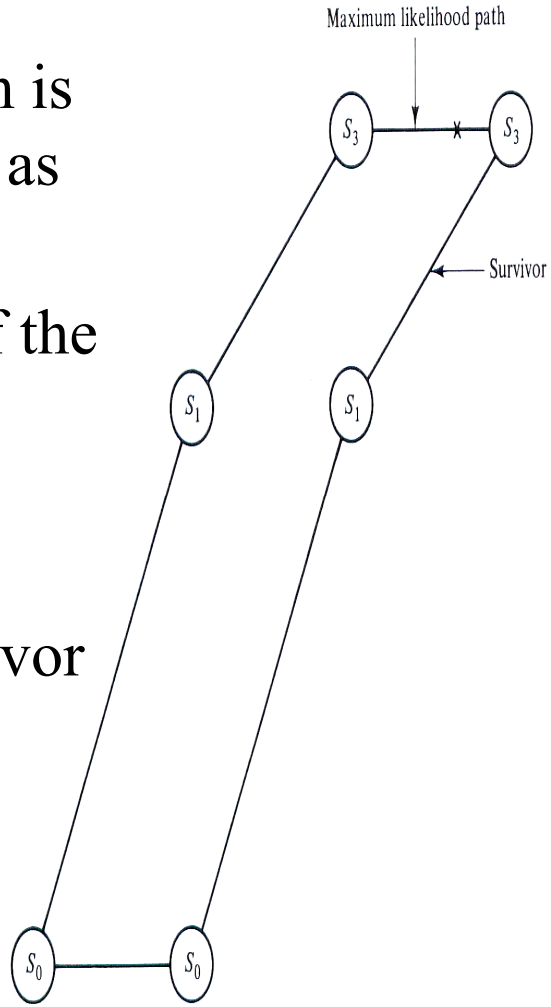
The Viterbi Algorithm

- There are 2^K survivors from time unit m through time unit L, one for each of the 2^K states. (K is the total number of registers)
- After time unit L there are fewer survivors, since there are fewer states while the encoder is returning to the all-zero state.
- Finally, at time unit $L + m$, there is only one state, the all-zero state, and hence only one survivor, and the algorithm terminates.
- **Theorem** The final survivor $\hat{\mathbf{v}}$ in the Viterbi algorithm is maximum likelihood path, that is,

$$M\left(\mathbf{r} \mid \hat{\mathbf{v}}\right) \geq M\left(\mathbf{r} \mid \mathbf{v}\right), \qquad \text{all } \mathbf{v} \neq \hat{\mathbf{v}}.$$

The Viterbi Algorithm

- ***Proof.***
  - Assume that the maximum likelihood path is eliminated by the algorithm at time unit j, as illustrated in figure.
  - This implies that the partial path metric of the survivor exceeds that of the maximum likelihood path at this point.
  - If the remaining portion of the maximum likelihood path is appended onto the survivor at time unit j, the total metric of this path will exceed the total metric of the maximum likelihood path.



The Viterbi Algorithm

- But this contradicts the definition of the maximum likelihood path as the path with the largest metric. Hence, the maximum likelihood path cannot be eliminated by the algorithm, and must be the final survivor.
- Therefore, the Viterbi algorithm is optimum in the sense that it always finds the maximum likelihood path through the trellis.

- In the special case of a *binary symmetric channel* (BSC) with transition probability $p < ½$, the received sequence $\mathbf{r}$ is binary ($Q = 2$) and the log-likelihood function becomes (Eq 1.11):

$$\log P(\mathbf{r} \mid \mathbf{v}) = d(\mathbf{r}, \mathbf{v}) \log \frac{p}{1-p} + N \log(1-p)$$

where $d(\mathbf{r}, \mathbf{v})$ is the Hamming distance between $\mathbf{r}$ and $\mathbf{v}$.

The Viterbi Algorithm

- Since $\log\left[p/(1-p)\right] < 0$ and $N\log(1-p)$ is a constant for all $\mathbf{v}$, an MLD for BSC chooses $\mathbf{v}$ as the code word $\hat{\mathbf{v}}$ that minimizes the Hamming distance

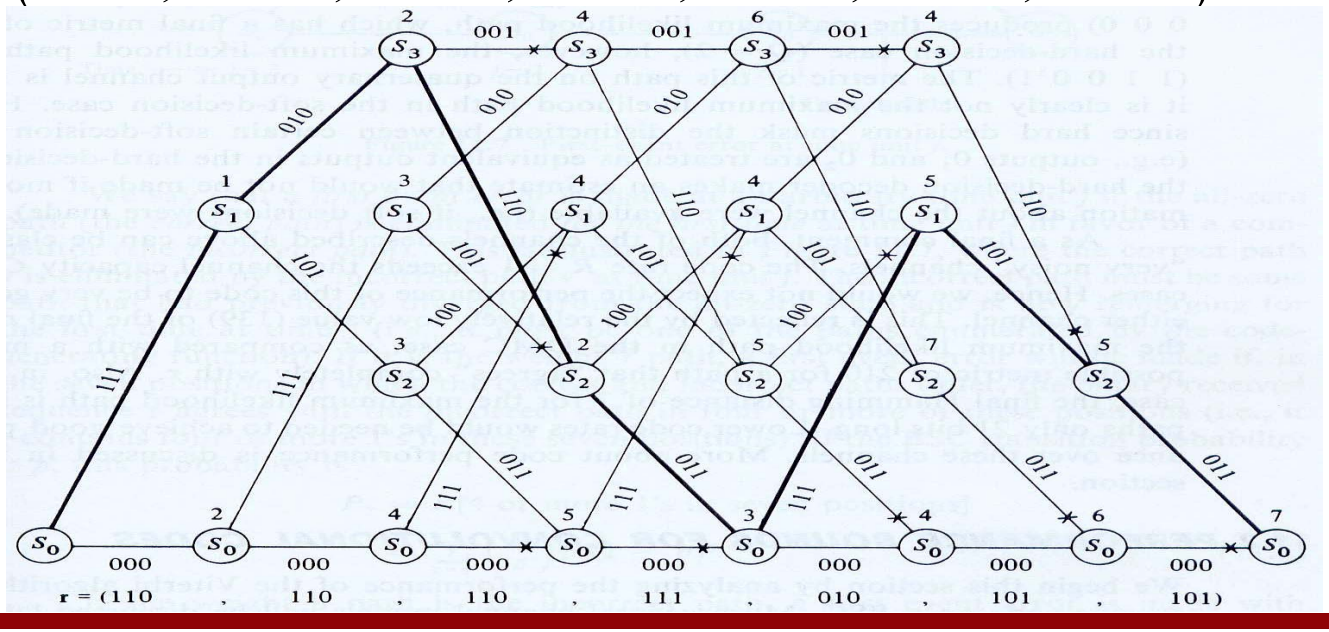
$$d(\mathbf{r},\mathbf{v}) = \sum_{i=0}^{L+m-1} d(\mathbf{r}_i,\mathbf{v}_i) = \sum_{i=0}^{N-1} d(r_i,v_i).$$

- In applying the Viterbi algorithm to the BSC, $d(\mathbf{r}_i, \mathbf{v}_i)$ becomes the branch metric, $d(r_i, v_i)$ becomes the bit metric, and the algorithm must find the path through the trellis with the smallest metric (i.e., the path closest to $\mathbf{r}$ in Hamming distance).
- The detail of the algorithm are exactly the same, except that the Hamming distance replaces the log-likelihood function as the metric and the survivor at each state is the path with the smallest metric. This kind of decoding is called *hard-decision*.

The Viterbi Algorithm

- Example 11.2:
  - The application of the Viterbi algorithm to a BSC is shown in the following figure. Assume that a code word from the trellis diagram of the (3, 1, 2) code of Figure (a) is transmitted over a BSC and that the received sequence is

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



The Viterbi Algorithm

- Example 11.2 (cont.):
  - The final survivor
    $\hat{\mathbf{v}} = (1\ 1\ 1,\ 0\ 1\ 0,\ 1\ 1\ 0,\ 0\ 1\ 1,\ 1\ 1\ 1,\ 1\ 0\ 1,\ 0\ 1\ 1)$
    is shown as the highlighted path in the figure, and the decoded information sequence is $\hat{\mathbf{u}} = (1\ 1\ 0\ 0\ 1)$.
  - The fact that the final survivor has a metric of 7 means that no other path through the trellis differs from **r** in fewer than seven positions.
  - Note that at some states neither path is crossed out. This indicates a tie in the metric values of the two paths entering that state.
  - If the final survivor goes through any of these states, there is more than one maximum likelihood path (i.e., more than one path whose distance from **r** is minimum).

The Viterbi Algorithm

- Example 11.2 (cont.):
  - From an implementation point of view, whenever a tie in metric values occurs, one path is arbitrarily selected as the survivor, because of the impracticality of storing a variable number of paths.
  - This arbitrary resolution of ties has no effect on the decoding error probability.

Punctured Convolutional Codes

- In some practical applications, there is a need to employ high-rate convolutional codes, e.g., rate of $(n\text{-}1)/n$.
- The trellis for such high-rate codes has 2^{n-1} branches that enter each state. Consequently, there are 2^{n-1} metric computations per state that must be performed in implementing the Viterbi algorithm and as many comparisons of the updated metrics in order to select the best path at each state.
- The computational complexity inherent in the implementation of the decoder of a high-rate convolutional code can be avoided by designing the high-rate code from a low-rate code in which some of the coded bits are deleted (*punctured*) from transmission.
- Puncturing a code reduces the free distance.

Punctured Convolutional Code

- The puncturing process may be described as periodically deleting selected bits from the output of the encoder, thus, creating a periodically time varying trellis code.
- We begin with a rate $1/n$ parent code and define a puncturing period P, corresponding to P input information bits to the encoder.
  - In one period, the encoder outputs nP coded bits.
  - Associated with the nP encoded bits is a *puncturing matrix* **P** of the form:

$$\mathbf{P} = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1p} \\ p_{21} & p_{22} & \cdots & p_{2p} \\ \vdots & \vdots & & \vdots \\ p_{n1} & p_{n2} & \cdots & p_{np} \end{bmatrix}$$

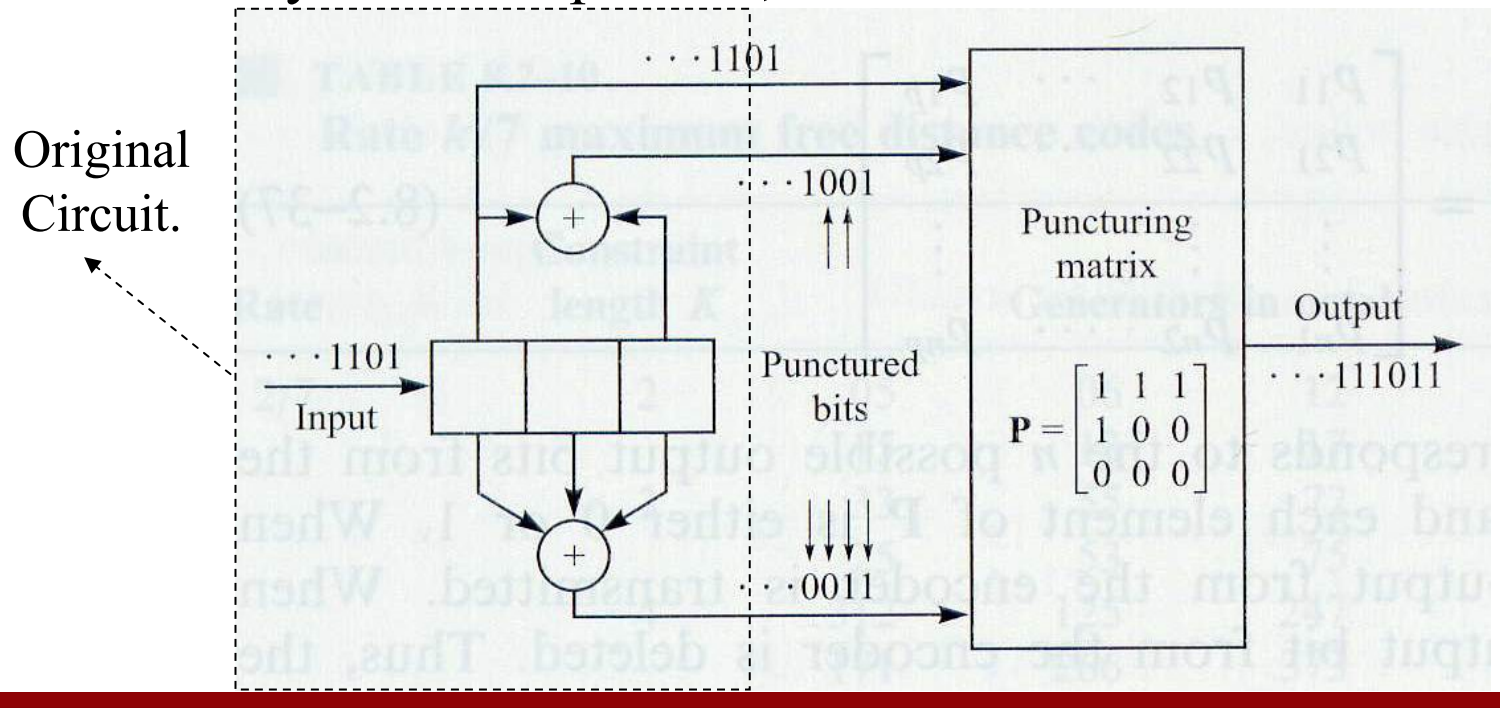
Punctured Convolutional Code

- Each column of **P** corresponds to the n possible output bits from the encoder for each input bit and each element of **P** is either 0 or 1.
  - When p_{ij}=1, the corresponding output bit from the encoder is transmitted.
  - When p_{ij}=0, the corresponding output bit from the encoder is deleted.
  - The code rate is determined by the period P and the number of bits deleted.
- If we delete N bits out of nP, the code rate is $P/(nP\text{-}N)$, where N may take any integer value in the range 0 to $(n\text{-}1)P\text{-}1$. That is:

$$\text{Achievable code rates are}: R_c = \frac{P}{P+M}, \quad M = 1,2,\ldots(n-1)P$$

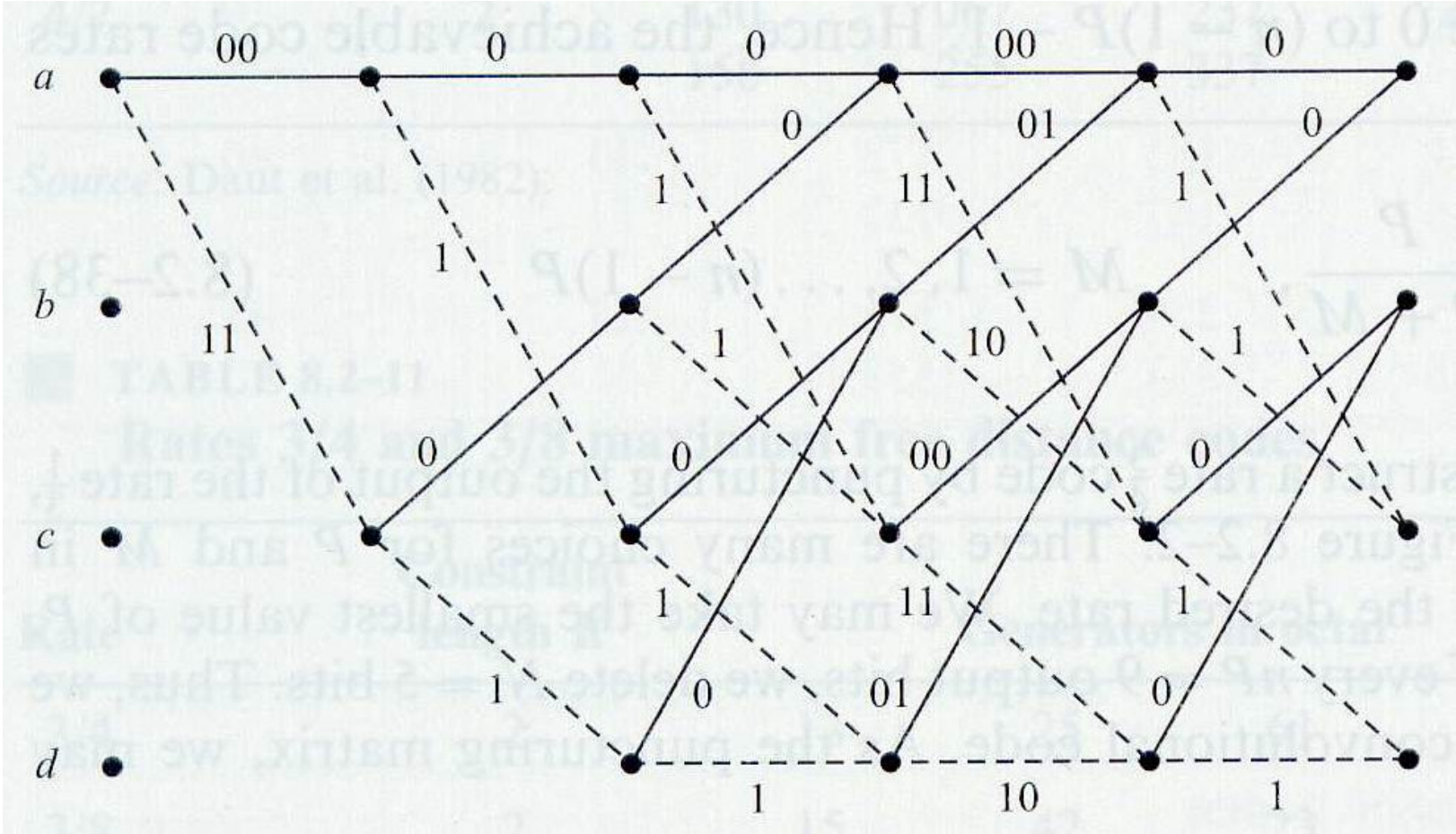
Punctured Convolutional Code

- Example: Constructing a rate ¾ code by puncturing the output of the rate 1/3, K=3 encoder as shown in the following figure.
- There are many choices for P and M to achieve the desired rate. We may take the smallest value of P, namely, P=3.
- Out of every nP=9 output bits, we delete N=5 bits.



Punctured Convolutional Code

- Example: Trellis of a rate ¾ convolutional code by puncturing the output of the rate 1/3, K=3 encoder .



Punctured Convolutional Code

- Some puncturing matrices are better than others in that the trellis paths have better Hamming distance properties.
- A computer search is usually employed to find good puncturing matrices.
- Generally, the high-rate punctured convolutional codes generated in this manner have a free distance that is either equal to or 1 bit less than the best same high-rate convolutional code obtained directly without puncturing.
- In general, high-rate codes with good distance properties are obtained by puncturing rate ½ maximum free distance codes.

Punctured Convolutional Code

Puncturing matrices for code rates of $2/3 \leqslant R_c \leqslant 7/8$ from rate 1/2 code

K	Rate 2/3		Rate 3/4		Rate 4/5		Rate 5/6		Rate 6/7		Rate 7/8	
	P	d_{free}	P	d_{free}	P	d_{free}	P	d_{free}	P	d_{free}	P	d_{free}
3	10 11	3	101 110	3	1011 1100	2	10111 11000	2	101111 110000	2	1011111 1100000	2
4	11 10	4	110 101	4	1011 1100	3	10100 11011	3	100011 111100	2	1000010 1111101	2
5	11 10	4	101 110	3	1010 1101	3	10111 11000	3	101010 110101	3	1010011 1101100	3
6	10 11	6	100 111	4	1000 1111	4	10000 11111	4	110110 101001	3	1011101 1100010	3
7	11 10	6	110 101	5	1111 1000	4	11010 10101	4	111010 100101	3	1111010 1000101	3
8	10 11	7	110 101	6	1010 1101	5	11100 10011	4	101001 110110	4	1010100 1101011	4
9	11 10	7	111 100	6	1101 1010	5	10110 11001	5	110110 101001	4	1101011 1010100	4

Punctured Convolutional Code

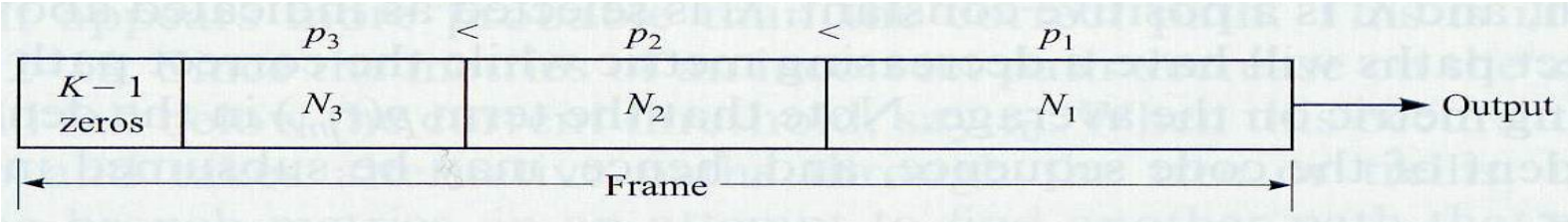
- The decoding of punctured convolutional codes is performed in the same manner as the decoding of the low-rate $1/n$ parent code, using the trellis of the $1/n$ code.
- When one or more bits in a branch are punctured, the corresponding branch metric increment is computed based on the non-punctured bits, thus, the punctured bits do not contribute to the branch metrics.
- Error events in a punctured code are generally longer than error events in the low rate $1/n$ parent code.
- Consequently, the decoder must wait longer than five constraint lengths before making final decisions on the received bits.

Rate-compatible Punctured Convolutional Codes

- In the transmission of compressed digital speech signals and in some other applications, there is a need to transmit some groups of information bits with more redundancy than others.
- In other words, the different groups of information bits require unequal error protection to be provided in the transmission of the information sequence, where the more important bits are transmitted with more redundancy.
- Instead of using separate codes to encode the different groups of bits, it is desirable to use a single code that has variable redundancy. This can be accomplished by puncturing the same low rate $1/n$ convolutional code by different amounts as described by Hagenauer (1988).

Rate-compatible Punctured Convolutional Codes

- The puncturing matrices are selected to satisfy a rate-compatibility criterion, where the basic requirement is that lower-rate codes (higher redundancy) should transmit the same coded bits as all higher-rate codes plus additional bits.
- The resulting codes obtained from a single rate $1/n$ convolutional code are called *rate-compatible punctured convolutional* (RCPC) code.
- In applying RCPC codes to systems that require unequal error protection of the information sequence, we may format the groups of bits into a frame structure and each frame is terminated after the last group of information bits by K-1 zeros.



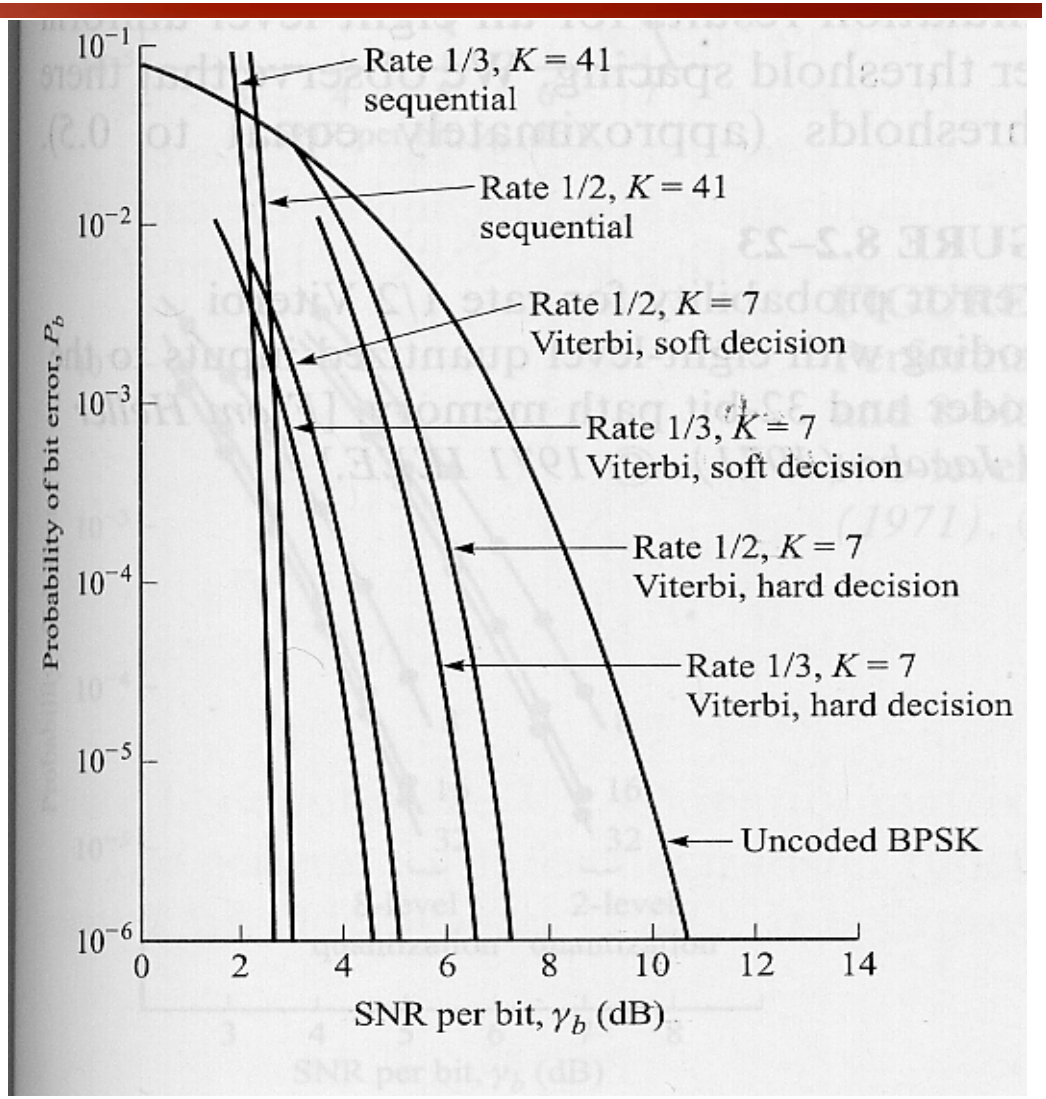
Rate-compatible Punctured Convolutional Codes

- Example: Constructing a RCPC code from a rate 1/3, K=4 maximum free distance convolutional code.
  - Note that when a 1 appears in a puncturing matrix of a high-rate code, a 1 also appears in the same position for all lower-rate codes.

Rate	Puncturing matrix P
$\frac{1}{3}$	$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$
$\frac{4}{11}$	$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$
$\frac{2}{5}$	$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$
$\frac{4}{9}$	$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$
$\frac{1}{2}$	$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$
$\frac{4}{7}$	$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$
$\frac{4}{6}$	$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$
$\frac{4}{5}$	$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$
$\frac{8}{9}$	$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

Practical Considerations

- The minimum free distance d_{free} can be increased either by decreasing the code rate or by increasing the constraint length, or both.
- If hard-decision decoding is used, the performances (coding gains) are reduced by approximately 2 dB for the AWGN channel when compared with the soft-decision decoding.

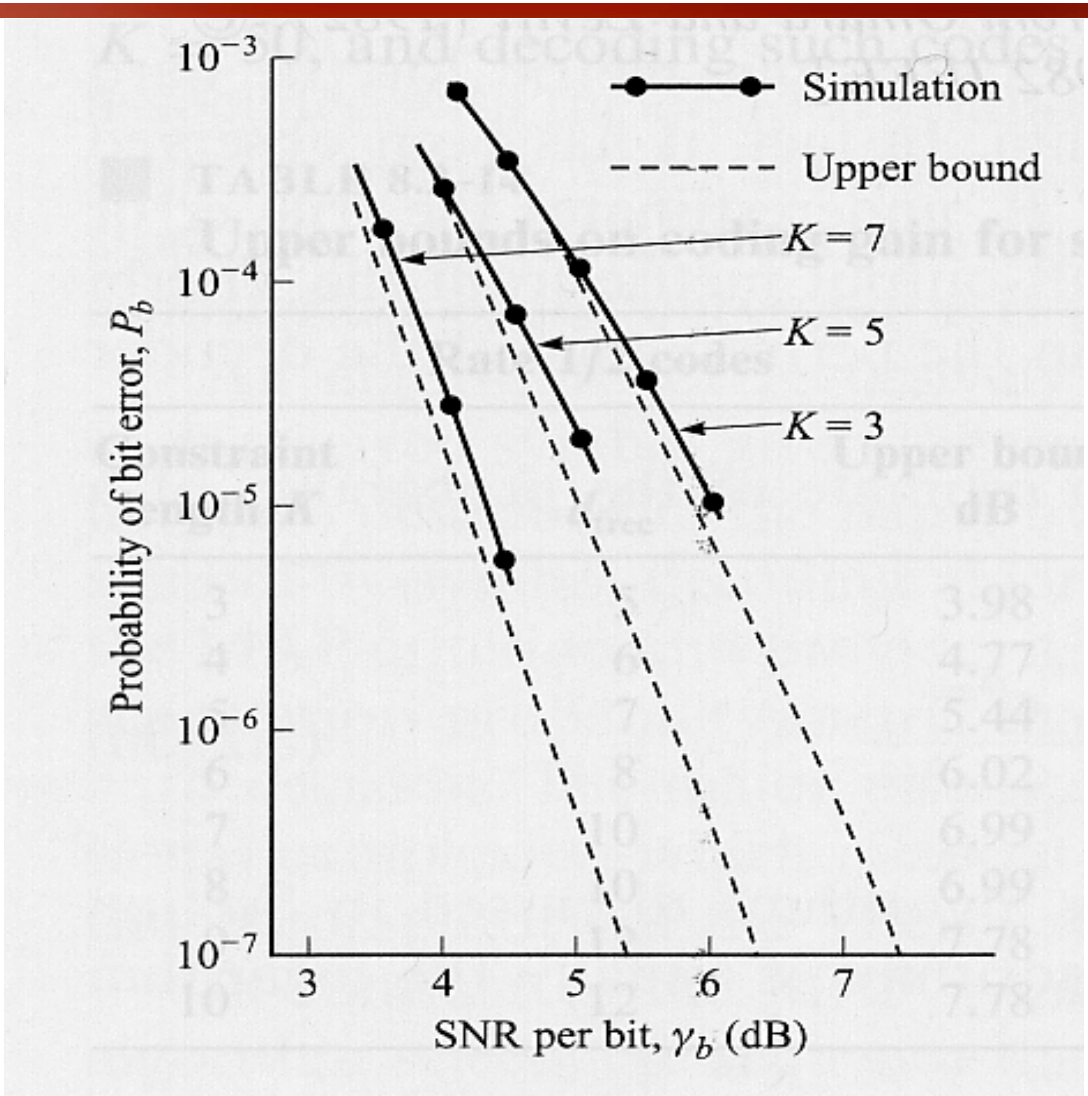


Practical Considerations

- Two important issues in the implementation of Viterbi decoding are:
  - The effect of path memory truncation, which is a desirable feature that ensures a fixed decoding delay.
  - The degree of quantization of the input signal to the Viterbi decoder.
- The path memory truncation to about five constraint lengths has been found to result in negligible performance loss.

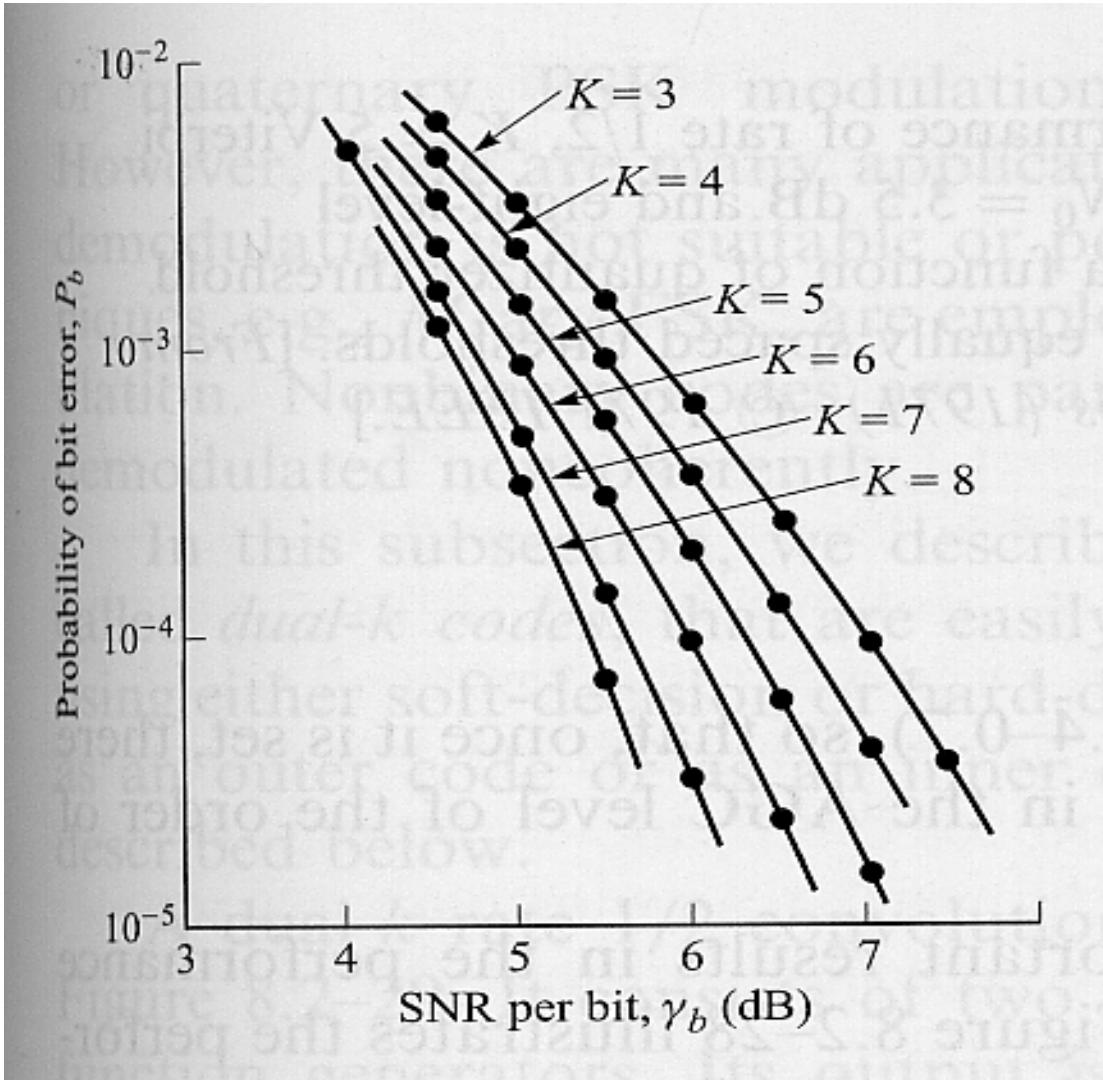
Practical Considerations

- Bit error probability for rate ½ Viterbi decoding with eight-level quantized inputs to the decoder and 32-bit path memory.



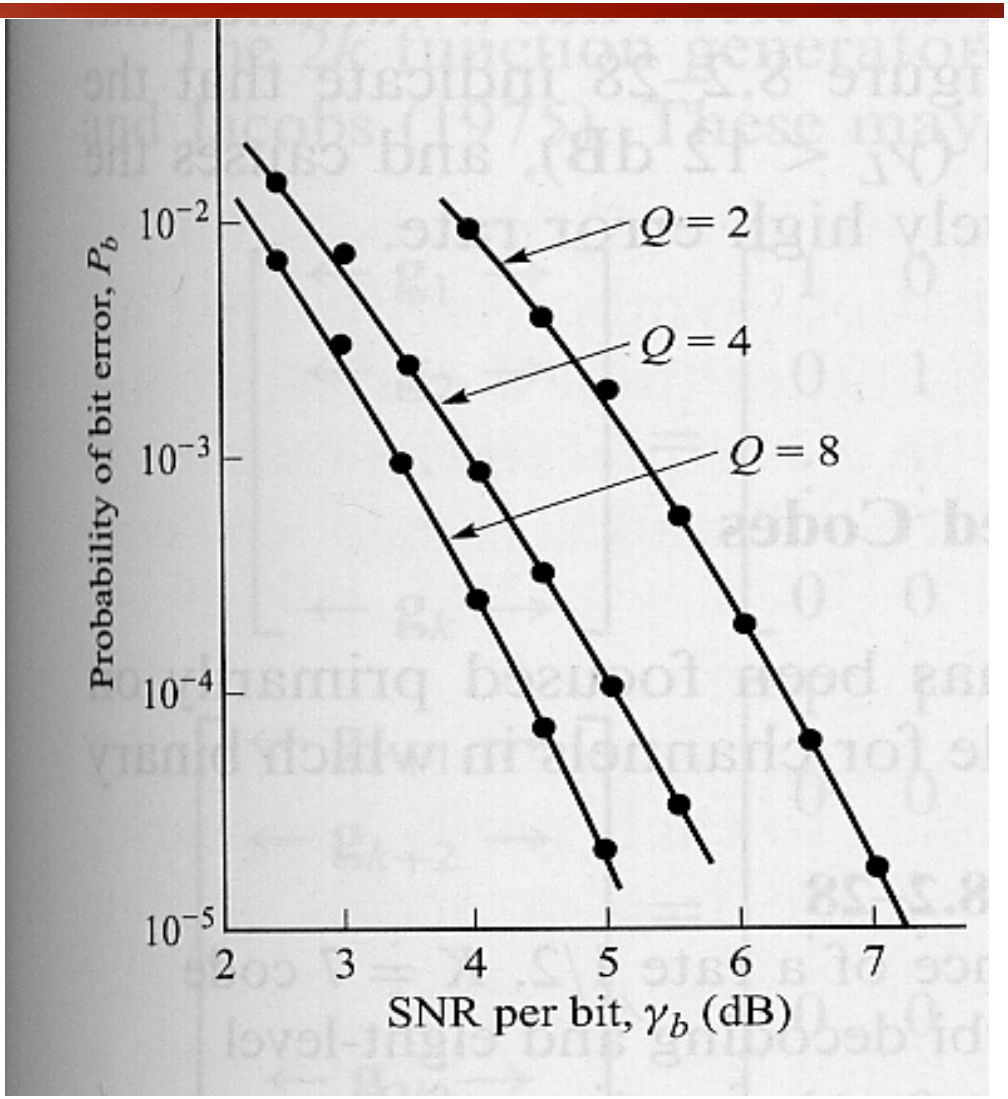
Practical Considerations

- Performance of rate ½ codes with hard-decision Viterbi decoding and 32-bit path memory truncation



Practical Considerations

- Performance of rate ½, K=5 code with eight-, four-, and two-level quantization at the input to the Viterbi decoder. Path truncation length=32 bits.



Practical Considerations

- Performance of rate ½, K=5 code with 32-, 16-, and 8-biy path memory truncation and eight- and two-level quantization.