Introduction to Binary Convolutional Codes [1]

Yunghsiang S. Han

Graduate Institute of Communication Engineering,
National Taipei University
Taiwan

E-mail: yshan@mail.ntpu.edu.tw

Binary Convolutional Codes

1. A binary convolutional code is denoted by a three-tuple (n, k, m).
2. n output bits are generated whenever k input bits are received.
3. The current n outputs are linear combinations of the present k input bits and the previous $m \times k$ input bits.
4. m designates the number of previous k-bit input blocks that must be memorized in the encoder.
5. m is called the *memory order* of the convolutional code.

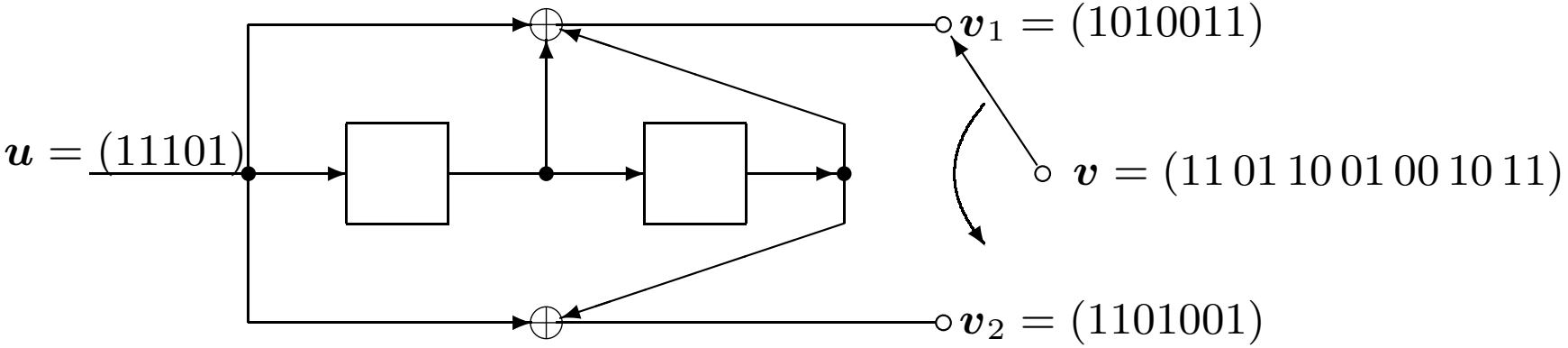
Encoders for the Convolutional Codes

1. A binary convolutional encoder is conveniently structured as a mechanism of shift registers and modulo-2 adders, where the output bits are modular-2 additions of selective shift register contents and present input bits.
2. n in the three-tuple notation is exactly the number of output sequences in the encoder.
3. k is the number of input sequences (and hence, the encoder consists of k shift registers).
4. m is the maximum length of the k shift registers (i.e., if the number of stages of the jth shift register is K_j, then $m = \max_{1 \le j \le k} K_j$).
5. $K = \sum_{j=1}^{k} K_j$ is the total memory in the encoder (K is sometimes called the *overall constraint lengths*).

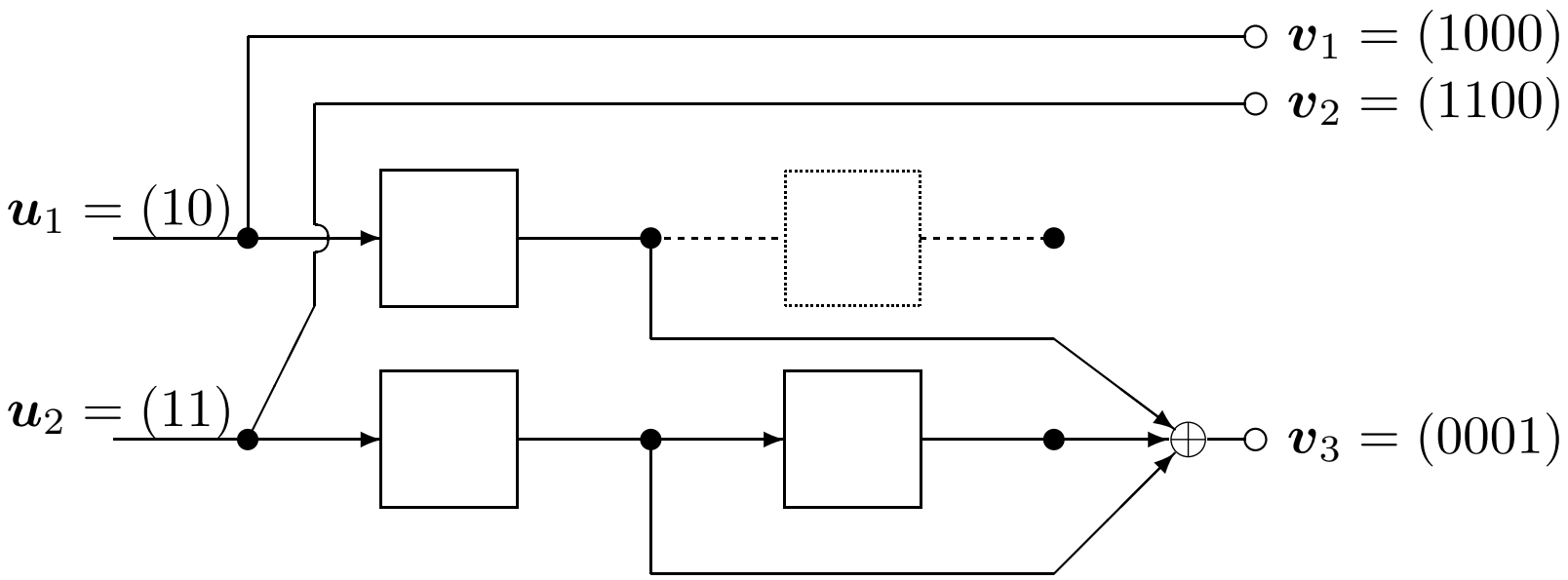
6. The definition of constraint length of a convolutional code is defined in several ways. The most popular one is $m + 1$.

Encoder for the Binary $(2, 1, 2)$ Convolutional Code

$\boldsymbol{u}$ is the information sequence and $\boldsymbol{v}$ is the corresponding code sequence (codeword).



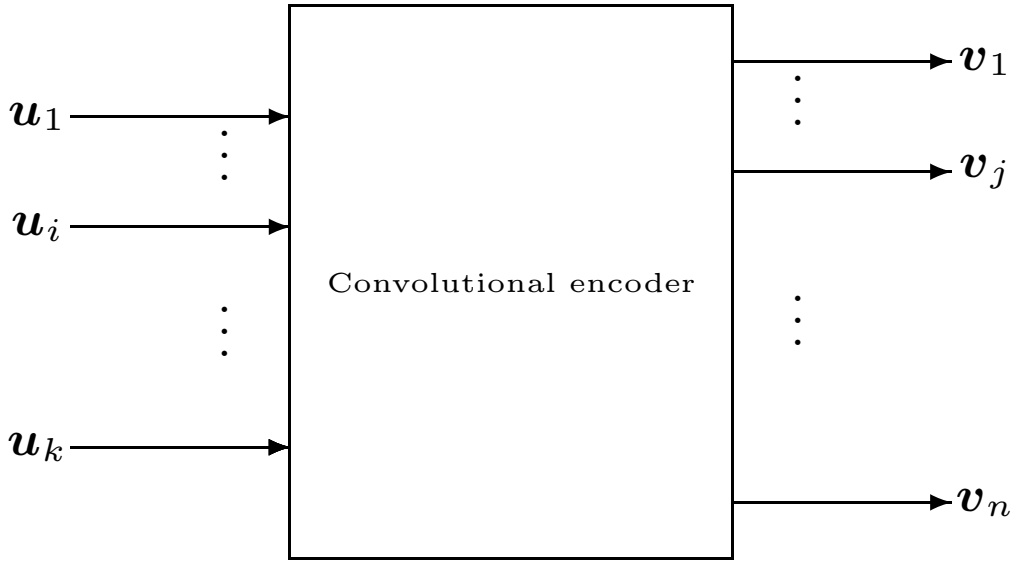
Encoder for the Binary $(3, 2, 2)$ Convolutional Code



$\boldsymbol{u} = (11\ 01)$

$\boldsymbol{v} = (110\ 010\ 000\ 001)$

Impose Response and Convolution



1. The encoders of convolutional codes can be represented by *linear time-invariant* (LTI) systems.

2.

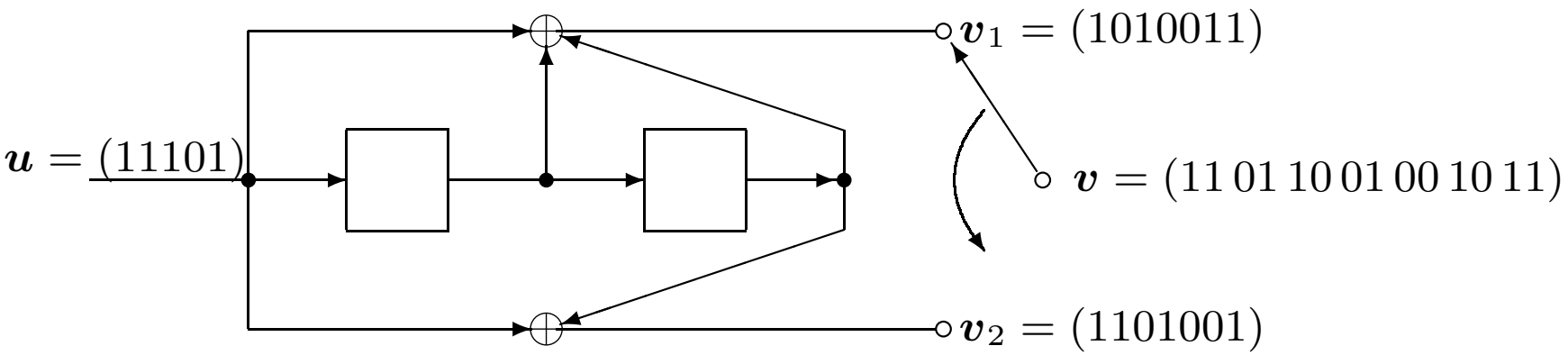
$$\boldsymbol{v}_j = \boldsymbol{u}_1 * \boldsymbol{g}_j^{(1)} + \boldsymbol{u}_2 * \boldsymbol{g}_j^{(2)} + \cdots + \boldsymbol{u}_k * \boldsymbol{g}_j^{(k)} = \sum_{i=1}^{k} \boldsymbol{u}_i * \boldsymbol{g}_j^{(k)},$$

where $*$ is the convolutional operation and $g_j^{(i)}$ is the impulse response of the ith input sequence with the response to the jth output.

3. $\boldsymbol{g}_j^{(i)}$ can be found by stimulating the encoder with the discrete impulse $(1, 0, 0, \ldots)$ at the ith input and by observing the jth output when all other inputs are fed the zero sequence $(0, 0, 0, \ldots)$.

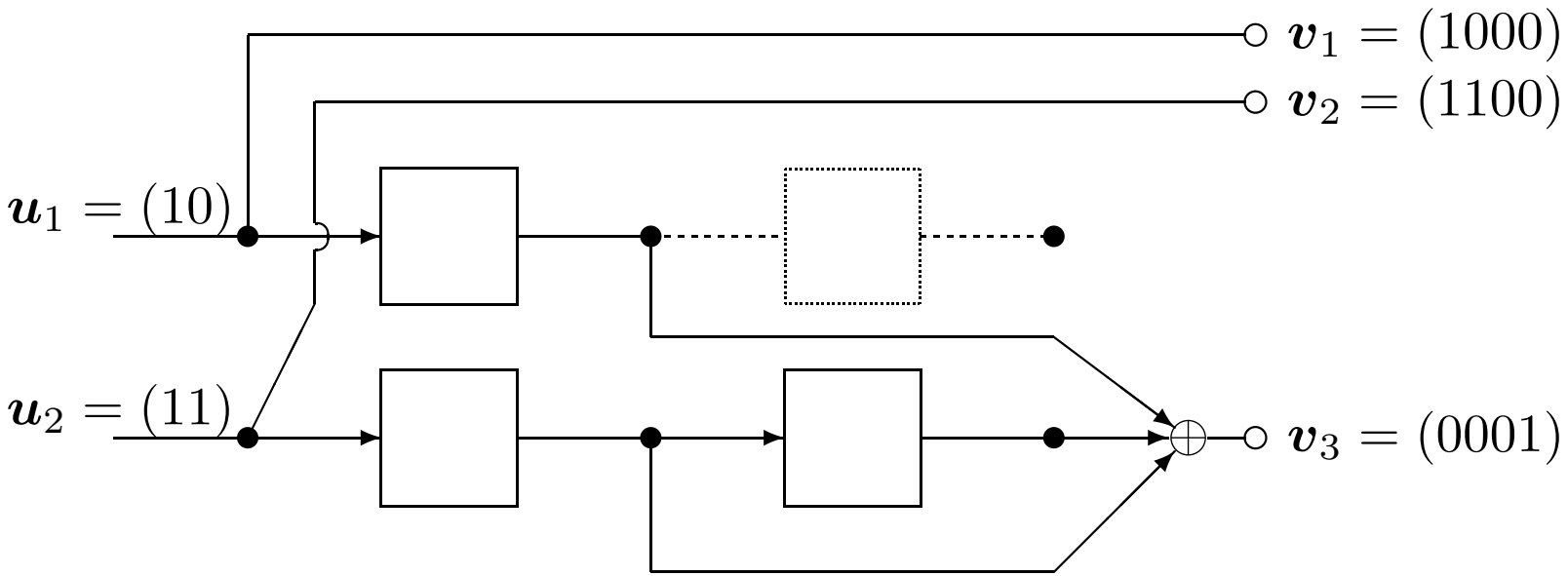
4. The impulse responses are called *generator sequences* of the encoder.

Impulse Response for the Binary $(2, 1, 2)$ Convolutional Code



$$\begin{aligned}
\boldsymbol{g}_1 &= (1,1,1,0,\ldots) = (1,1,1),\ \boldsymbol{g}_2 = (1,0,1,0,\ldots) = (1,0,1) \\
\boldsymbol{v}_1 &= (1,1,1,0,1) * (1,1,1) = (1,0,1,0,0,1,1) \\
\boldsymbol{v}_2 &= (1,1,1,0,1) * (1,0,1) = (1,1,0,1,0,0,1)
\end{aligned}$$

Impulse Response for the $(3, 2, 2)$ Convolutional Code



$\boldsymbol{u} = (11\ 01)$

$\boldsymbol{v} = (110\ 010\ 000\ 001)$

$\boldsymbol{g}_1^{(1)} = 4$ (octal)$= (1, 0, 0)$, $\boldsymbol{g}_1^{(2)} = 0$ (octal), $\boldsymbol{g}_2^{(1)} = 0$ (octal), $\boldsymbol{g}_2^{(2)} = 4$ (octal), $\boldsymbol{g}_3^{(1)} = 2$ (octal), $\boldsymbol{g}_3^{(2)} = 3$ (octal)

Generator Matrix in the Time Domain

1. The convolutional codes can be generated by a generator matrix multiplied by the information sequences.
2. Let $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_k$ are the information sequences and $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n$ the output sequences.
3. Arrange the information sequences as

$$\begin{aligned} \boldsymbol{u} &= (u_{1,0}, u_{2,0}, \ldots, u_{k,0}, u_{1,1}, u_{2,1}, \ldots, u_{k,1}, \\ &\quad \ldots, u_{1,\ell}, u_{2,\ell}, \ldots, u_{k,\ell}, \ldots) \\ &= (\boldsymbol{w}_0, \boldsymbol{w}_1, \ldots, \boldsymbol{w}_\ell, \ldots), \end{aligned}$$

and the output sequences as

$$\begin{aligned} \boldsymbol{v} &= (v_{1,0}, v_{2,0}, \ldots, v_{n,0}, v_{1,1}, v_{2,1}, \ldots, v_{n,1}, \\ &\quad \ldots, v_{1,\ell}, v_{2,\ell}, \ldots, v_{n,\ell}, \ldots) \\ &= (\boldsymbol{z}_0, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_\ell, \ldots) \end{aligned}$$

4. $\boldsymbol{v}$ is called a codeword or code sequence.

5. The relation between $\boldsymbol{v}$ and $\boldsymbol{u}$ can characterized as

$$\boldsymbol{v} = \boldsymbol{u} \cdot \boldsymbol{G},$$

where $\boldsymbol{G}$ is the generator matrix of the code.

6. The generator matrix is

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

with the $k \times n$ submatrices

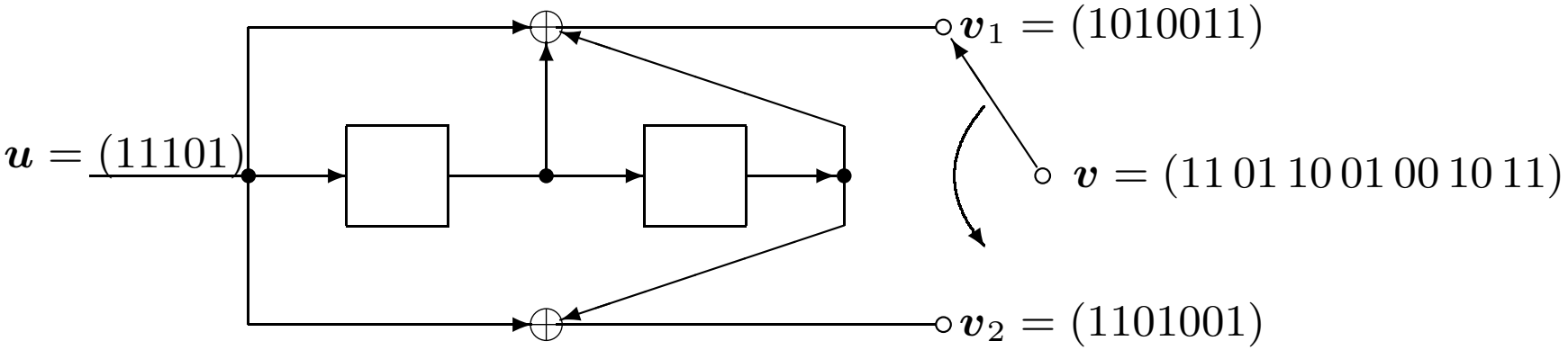
$$\boldsymbol{G}_\ell = \begin{pmatrix} g_{1,\ell}^{(1)} & g_{2,\ell}^{(1)} & \cdots & g_{n,\ell}^{(1)} \\ g_{1,\ell}^{(2)} & g_{2,\ell}^{(2)} & \cdots & g_{n,\ell}^{(2)} \\ \vdots & \vdots & & \vdots \\ g_{1,\ell}^{(k)} & g_{2,\ell}^{(k)} & \cdots & g_{n,\ell}^{(k)} \end{pmatrix}.$$

7. The element $g_{j,\ell}^{(i)}$, for $i \in [1, k]$ and $j \in [1, n]$, are the impulse

response of the ith input with respect to jth output:

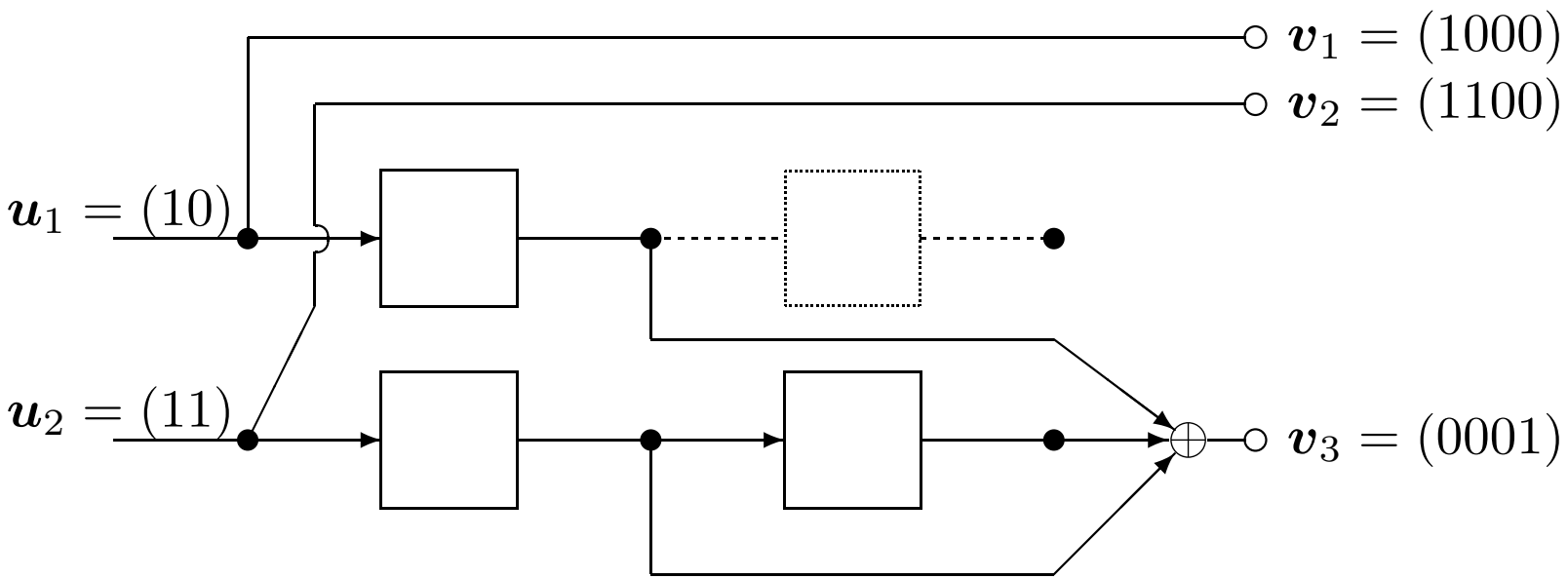
$$\boldsymbol{g}_j^{(i)} = (g_{j,0}^{(i)}, g_{j,1}^{(i)}, \ldots, g_{j,\ell}^{(i)}, \ldots, g_{j,m}^{(i)}).$$

Generator Matrix of the Binary $(2, 1, 2)$ Convolutional Code



$$
\begin{aligned}
\boldsymbol{v} &= \boldsymbol{u} \cdot \boldsymbol{G} \\
&= (1,1,1,0,1) \cdot \begin{pmatrix}
11 & 10 & 11 & & & & \\
 & 11 & 10 & 11 & & & \\
 & & 11 & 10 & 11 & & \\
 & & & 11 & 10 & 11 & \\
 & & & & 11 & 10 & 11
\end{pmatrix} \\
&= (11,01,10,01,00,10,11)
\end{aligned}
$$

Generator Matrix of the Binary $(3, 2, 2)$ Convolutional Code



$\boldsymbol{u} = (11\ 01)$

$\boldsymbol{v} = (110\ 010\ 000\ 001)$

$\boldsymbol{g}_1^{(1)} = 4$ (octal)$= (1, 0, 0)$, $\boldsymbol{g}_1^{(2)} = 0$ (octal), $\boldsymbol{g}_2^{(1)} = 0$ (octal), $\boldsymbol{g}_2^{(2)} = 4$

(octal), $\boldsymbol{g}_3^{(1)} = 2$ (octal), $\boldsymbol{g}_3^{(2)} = 3$ (octal)

$$
\begin{aligned}
\boldsymbol{v} &= \boldsymbol{u} \cdot \boldsymbol{G} \\
&= (11, 01) \cdot \begin{pmatrix} 100 & 001 & 000 & \\ 010 & 001 & 001 & \\ & 100 & 001 & 000 \\ & 010 & 001 & 001 \end{pmatrix} \\
&= (110, 010, 000, 001)
\end{aligned}
$$

Generator Matrix in the Z Domain

1. According to the Z transform,

$$\boldsymbol{u}_i \quad \multimap \quad U_i(D) = \sum_{t=0}^{\infty} u_{i,t} D^t$$

$$\boldsymbol{v}_j \quad \multimap \quad V_j(D) = \sum_{t=0}^{\infty} v_{j,t} D^t$$

$$\boldsymbol{g}_j^{(i)} \quad \multimap \quad G_{i,j}(D) = \sum_{t=0}^{\infty} g_{j,t}^{(i)} D^t$$

2. The convolutional relation of the Z transform $Z\{\boldsymbol{u} * \boldsymbol{g}\} = U(D)G(D)$ is used to transform the convolution of input sequences and generator sequences to a multiplication in the Z domain.

3. $V_j(D) = \sum_{i=1}^{k} U_i(D) \cdot G_{i,j}(D)$.

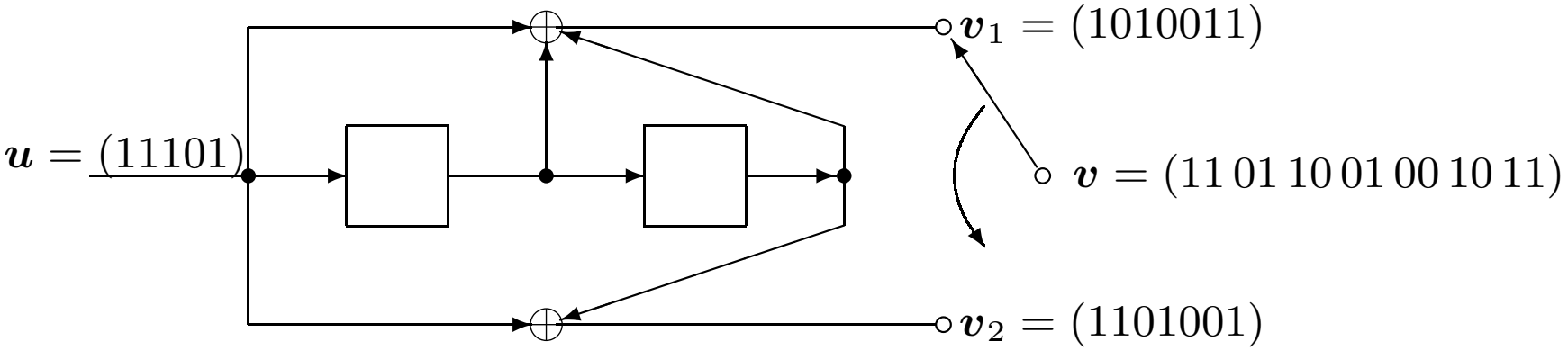
4. We can write the above equations into a matrix multiplication:

$$\boldsymbol{V}(D) = \boldsymbol{U}(D) \cdot \boldsymbol{G}(D),$$

where

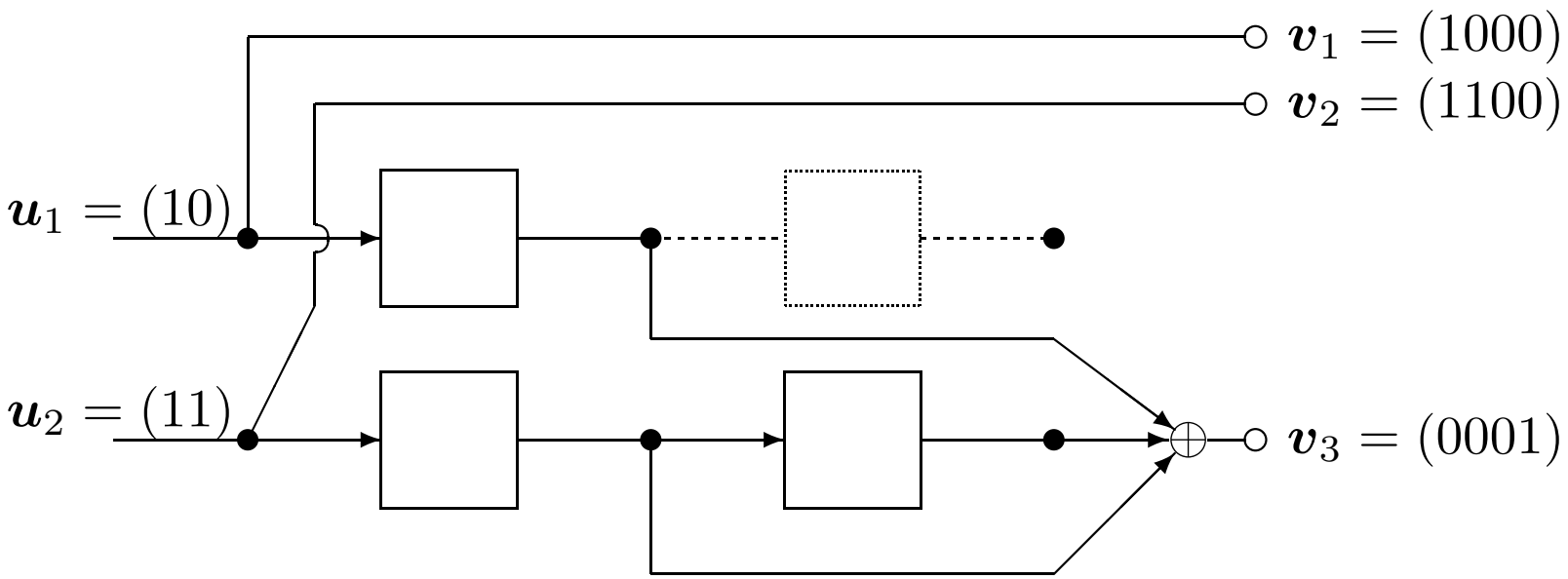
$$\begin{aligned} \boldsymbol{U}(D) &= (U_1(D), U_2(D), \ldots, U_k(D)) \\ \boldsymbol{V}(D) &= (V_1(D), V_2(D), \ldots, V_n(D)) \\ \boldsymbol{G}(D) &= \begin{pmatrix} & G_{i,j} & \\ & & \end{pmatrix} \end{aligned}$$

Generator Matrix of the Binary $(2, 1, 2)$ Convolutional Code



$$
\begin{aligned}
\boldsymbol{g}_1 &= (1,1,1,0,\ldots) = (1,1,1), \\
\boldsymbol{g}_2 &= (1,0,1,0,\ldots) = (1,0,1) \\
G_{1,1}(D) &= 1 + D + D^2 \\
G_{1,2}(D) &= 1 + D^2 \\
U_1(D) &= 1 + D + D^2 + D^4 \\
V_1(D) &= 1 + D^2 + D^5 + D^6 \\
V_2(D) &= 1 + D + D^3 + D^6
\end{aligned}
$$

Generator Matrix of the Binary $(3, 2, 2)$ Convolutional Code



$\boldsymbol{u} = (11\ 01)$

$\boldsymbol{v} = (110\ 010\ 000\ 001)$

$$
\begin{aligned}
\boldsymbol{g}_1^{(1)} &= (1,0,0),\ \boldsymbol{g}_1^{(2)} = (0,0,0),\ \boldsymbol{g}_2^{(1)} = (0,0,0) \\
\boldsymbol{g}_2^{(2)} &= (1,0,0),\ \boldsymbol{g}_3^{(1)} = (0,1,0),\ \boldsymbol{g}_3^{(2)} = (0,1,1) \\
G_{1,1}(D) &= 1,\ G_{1,2}(D) = 0,\ G_{1,3}(D) = D \\
G_{2,1}(D) &= 0,\ G_{2,2} = 1,\ G_{2,3}(D) = D + D^2 \\
U_1(D) &= 1,\ U_2(D) = 1 + D \\
V_1(D) &= 1,\ V_2(D) = 1 + D,\ V_3(D) = D^3
\end{aligned}
$$

Termination

1. The *effective code rate*, $R_{\text{effective}}$, is defined as the average number of input bits carried by an output bit.
2. In practice, the input sequences are with finite length.
3. In order to terminate a convolutional code, some bits are appended onto the information sequence such that the shift registers return to the zero.
4. Each of the k input sequences of length L bits is padded with m zeros, and these k input sequences jointly induce $n(L+m)$ output bits.
5. The effective rate of the terminated convolutional code is now

$$R_{\text{effective}} = \frac{kL}{n(L+m)} = R\frac{L}{L+m},$$

where $\frac{L}{L+m}$ is called the *fractional rate loss.*

6. When L is large, $R_{\text{effective}} \approx R$.

7. All examples presented are terminated convolutional codes.

Truncation

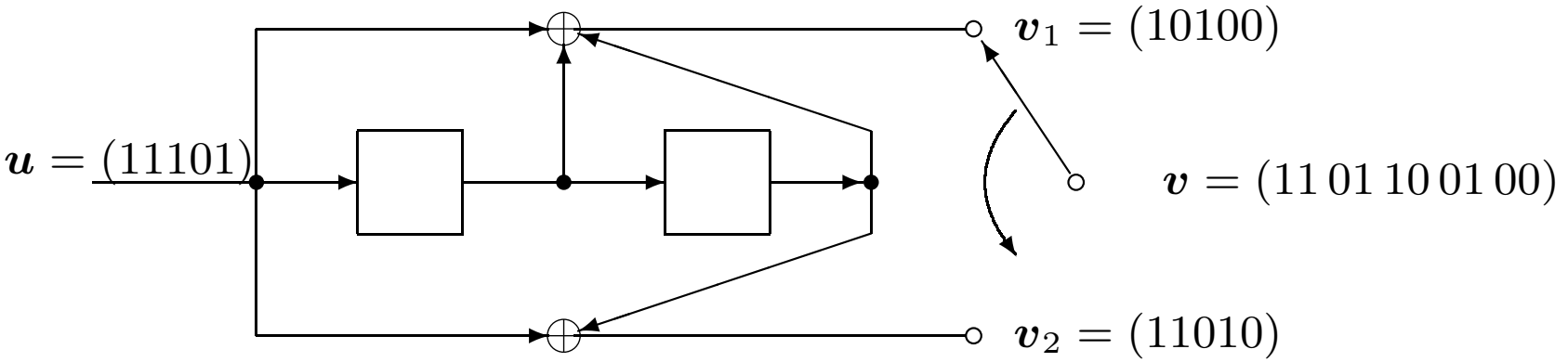
1. The second option to terminate a convolutional code is to stop for $t > L$ no matter what contents of shift registers have.
2. The effective code rate is still R.
3. The generator matrix is clipped after the Lth column:

$$\boldsymbol{G}^c_{[L]} = \begin{pmatrix} \boldsymbol{G}_0 & \boldsymbol{G}_1 & \cdots & \boldsymbol{G}_m & & & \\ & \boldsymbol{G}_0 & \boldsymbol{G}_1 & \cdots & \boldsymbol{G}_m & & \\ & & \ddots & & \ddots & & \\ & & & \boldsymbol{G}_0 & & & \boldsymbol{G}_m \\ & & & & \ddots & & \vdots \\ & & & & & \boldsymbol{G}_0 & \boldsymbol{G}_1 \\ & & & & & & \boldsymbol{G}_0 \end{pmatrix},$$

where $\boldsymbol{G}^c_{[L]}$ is an $(k \cdot L) \times (n \cdot L)$ matrix.

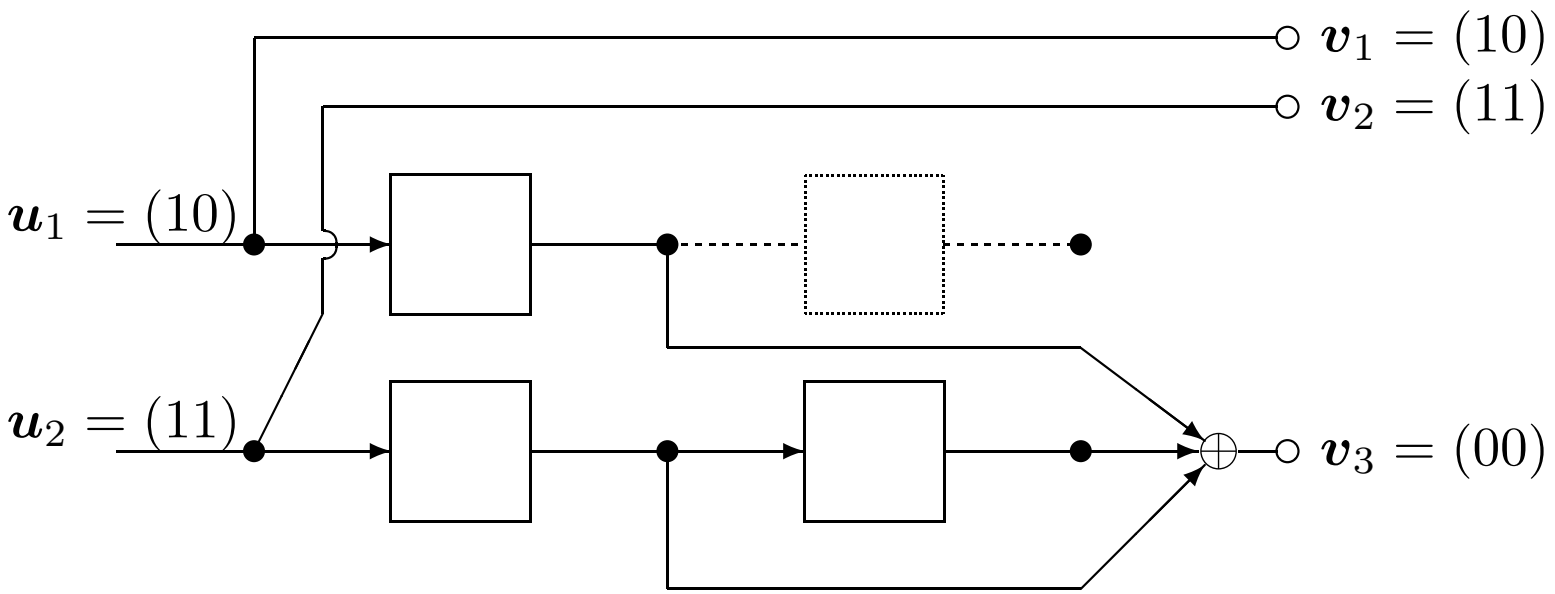
4. The drawback of truncation method is that the last few blocks of information sequences are less protected.

Generator Matrix of the Truncation Binary $(2, 1, 2)$ Convolutional Code



$$
\begin{aligned}
\boldsymbol{v} &= \boldsymbol{u} \cdot \boldsymbol{G}^{c}_{[5]} \\
&= (1,1,1,0,1) \cdot \begin{pmatrix}
11 & 10 & 11 & & \\
 & 11 & 10 & 11 & \\
 & & 11 & 10 & 11 \\
 & & & 11 & 10 \\
 & & & & 11
\end{pmatrix} \\
&= (11,01,10,01,00)
\end{aligned}
$$

Generator Matrix of the Truncation Binary $(3, 2, 2)$ Convolutional Code



$\boldsymbol{u} = (11\ 01)$

$\boldsymbol{v} = (110\ 010)$

$$
\begin{aligned}
\boldsymbol{v} &= \boldsymbol{u} \cdot \boldsymbol{G}^{c}_{[2]} \\
&= (11, 01) \cdot \begin{pmatrix} 100 & 001 \\ 010 & 001 \\ & 100 \\ & 010 \end{pmatrix} \\
&= (110, 010)
\end{aligned}
$$

Tail Biting

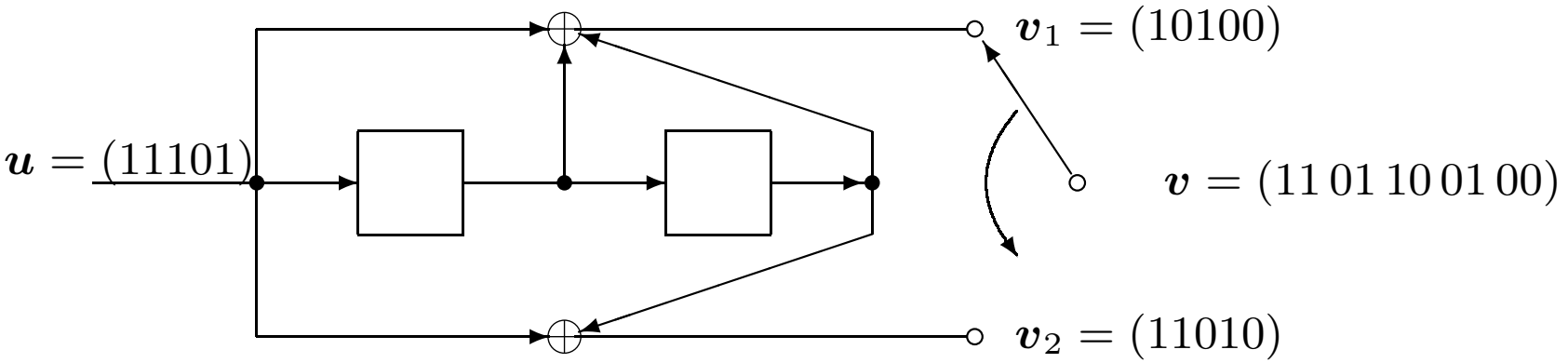
1. The third possible method to generate finite code sequences is called tail biting.
2. Tail biting is to start the convolutional encoder in the same contents of all shift registers (state) where it will stop after the input of L information blocks.
3. Equal protection of all information bits of the entire information sequences is possible.
4. The effective rate of the code is still R.
5. The generator matrix has to be clipped after the Lth column and

manipulated as follows:

$$\tilde{\boldsymbol{G}}^{c}_{[L]} = \begin{pmatrix} \boldsymbol{G}_0 & \boldsymbol{G}_1 & \cdots & \boldsymbol{G}_m & & & \\ & \boldsymbol{G}_0 & \boldsymbol{G}_1 & \cdots & \boldsymbol{G}_m & & \\ & & \ddots & & \ddots & & \\ & & & \boldsymbol{G}_0 & & & \boldsymbol{G}_m \\ \boldsymbol{G}_m & & & & \ddots & & \vdots \\ \vdots & \ddots & & & & \boldsymbol{G}_0 & \boldsymbol{G}_1 \\ \boldsymbol{G}_1 & \cdots & \boldsymbol{G}_m & & & & \boldsymbol{G}_0 \end{pmatrix},$$

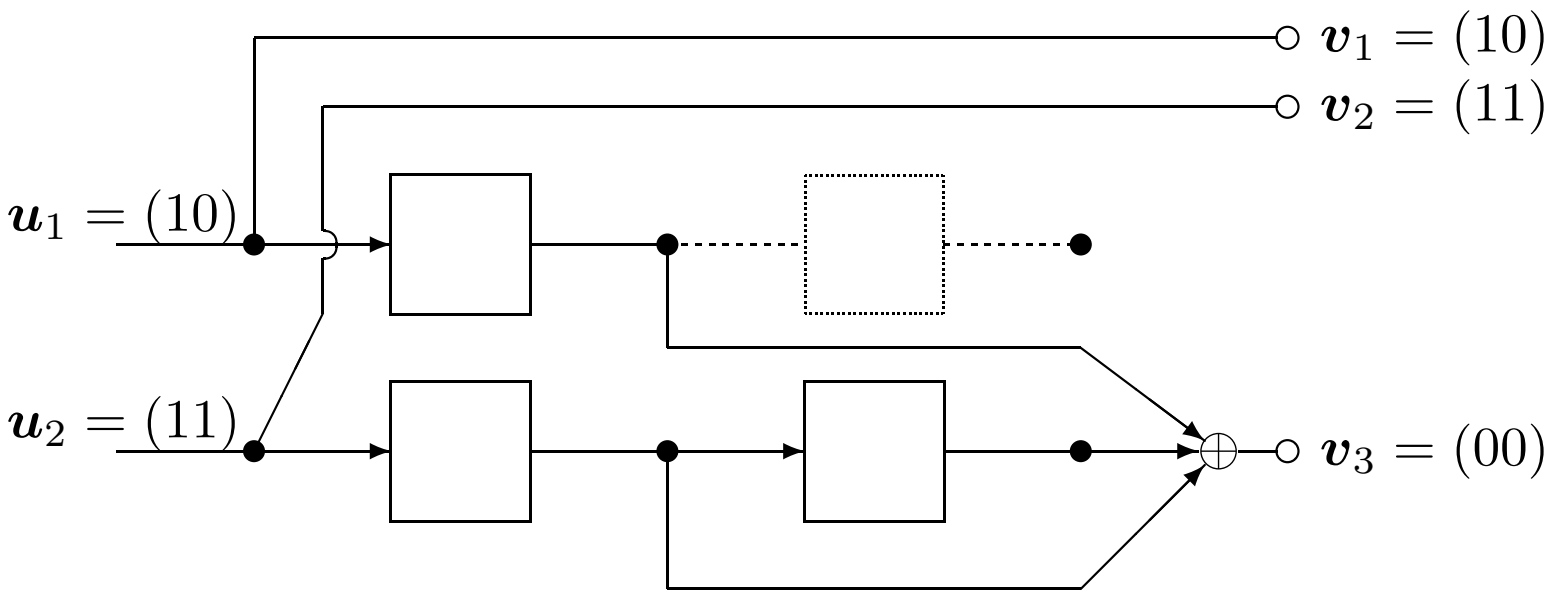
where $\tilde{\boldsymbol{G}}^{c}_{[L]}$ is an $(k \cdot L) \times (n \cdot L)$ matrix.

Generator Matrix of the Tail Biting Binary $(2, 1, 2)$ Convolutional Code



$$
\begin{aligned}
\boldsymbol{v} &= \boldsymbol{u} \cdot \tilde{\boldsymbol{G}}^{c}_{[5]} \\
&= (1,1,1,0,1) \cdot \begin{pmatrix}
11 & 10 & 11 & & \\
 & 11 & 10 & 11 & \\
 & & 11 & 10 & 11 \\
11 & & & 11 & 10 \\
10 & 11 & & & 11
\end{pmatrix} \\
&= (01,10,10,01,00)
\end{aligned}
$$

Generator Matrix of the Tail Biting Binary $(3, 2, 2)$ Convolutional Code

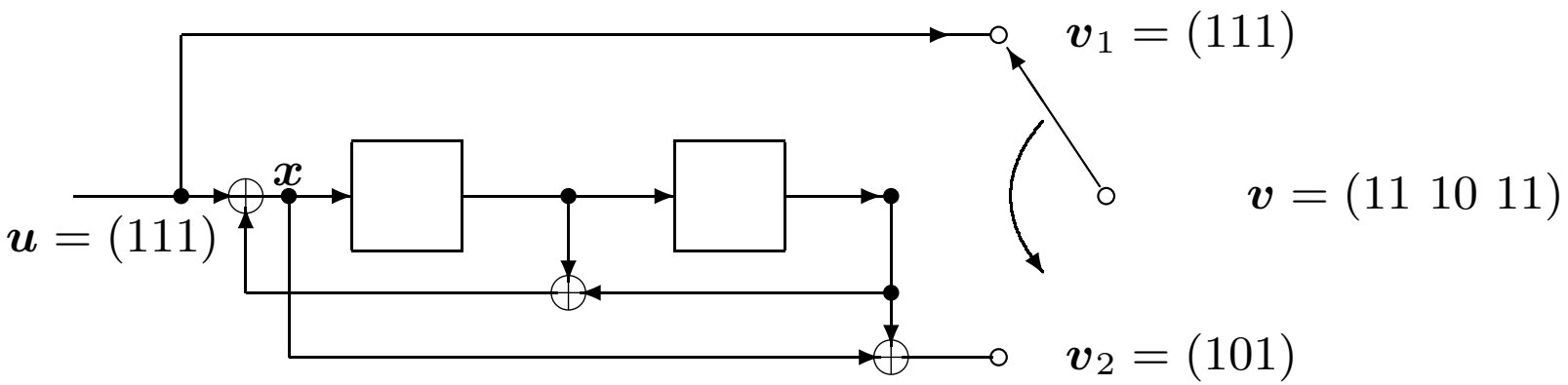


$\boldsymbol{u} = (11\ 01)$

$\boldsymbol{v} = (110\ 010)$

$$
\begin{aligned}
\boldsymbol{v} &= \boldsymbol{u} \cdot \tilde{\boldsymbol{G}}^{c}_{[2]} \\
&= (11, 01) \cdot \begin{pmatrix} 100 & 001 \\ 010 & 001 \\ 001 & 100 \\ 001 & 010 \end{pmatrix} \\
&= (111, 010)
\end{aligned}
$$

FIR and IIR systems



1. All examples in previous slides are *finite impulse response* (FIR) systems that are with finite impulse responses.
2. The above example is an *infinite impulse response* (IIR) system that is with infinite impulse response.

3. The generator sequences of the above example are

$$\begin{aligned} g_1^{(1)} &= (1) \\ g_2^{(1)} &= (1,1,1,0,1,1,0,1,1,0,\ldots). \end{aligned}$$

4. The infinite sequence of $g_2^{(1)}$ is caused by the recursive structure of the encoder.

5. By introducing the variable $\boldsymbol{x}$, we have

$$\begin{aligned} x_t &= u_t + x_{t-1} + x_{t-2} \\ v_{2,t} &= x_t + x_{t-2}. \end{aligned}$$

Accordingly, we can have the following difference equations:

$$\begin{aligned} v_{1,t} &= u_t \\ v_{2,t} + v_{2,t-1} + v_{2,t-2} &= u_t + u_{t-2} \end{aligned}$$

6. We then apply the z transform to the second equation:

$$\begin{aligned}\sum_{t=0}^{\infty} v_{2,t}D^t + \sum_{t=0}^{\infty} v_{2,t-1}D^t + \sum_{t=0}^{\infty} v_{2,t-2}D^t &= \sum_{t=0}^{\infty} u_t D^t + \sum_{t=0}^{\infty} u_{t-2}D^t, \\ V_2(D) + DV_2(D) + D^2V_2(D) &= U(D) + D^2U(D).\end{aligned}$$

7. The system function is then

$$V_2(D) = \frac{1+D^2}{1+D+D^2}U(D) = G_{12}(D)U(D).$$

8. The generator matrix is obtained:

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

9.

$$\begin{aligned} \boldsymbol{V}(D) &= \boldsymbol{U}(D) \cdot \boldsymbol{G}(D) \\ &= \left(1 + D + D^2\right) \begin{pmatrix} 1 & \frac{1+D^2}{1+D+D^2} \end{pmatrix} \\ &= \left(1 + D + D^2 \; 1 + D^2\right) \end{aligned}$$

10. The time diagram:

t	$\boldsymbol{\sigma}_t$	$\boldsymbol{\sigma}_{t+1}$	$\boldsymbol{v}_1$	$\boldsymbol{v}_2$
0	00	10	1	1
1	10	01	1	0
2	01	00	1	1

State Diagram

1. A convolutional encoder can be treated as a finite state machine.
2. The contents of the shift registers represent the states. The output of a code block $\boldsymbol{v}_t$ at time t depends on the current state $\boldsymbol{\sigma}_t$ and the information block $\boldsymbol{u}_t$.
3. Each change of state $\boldsymbol{\sigma}_t \rightarrow \boldsymbol{\sigma}_{t+1}$ is associated with the input of an information block and the output of a code block.
4. The *state diagram* is obtained by drawing a graph. In this graph, nodes are possible states and the state transitions are labelled with the appropriate inputs and outputs $(\boldsymbol{u}_t/\boldsymbol{v}_t)$. In this course we only consider the convolutional encoder with state diagrams that do not have parallel transitions.
5. The state of the encoder can be expressed as k-tuple of the

memory values:

$$\boldsymbol{\sigma}_t = (u_{1,t-1} \ldots u_{1,t-K_1}, u_{2,t-1} \ldots u_{2,t-K_2}, \ldots, u_{k,t-1} \ldots u_{k,t-K_k}).$$

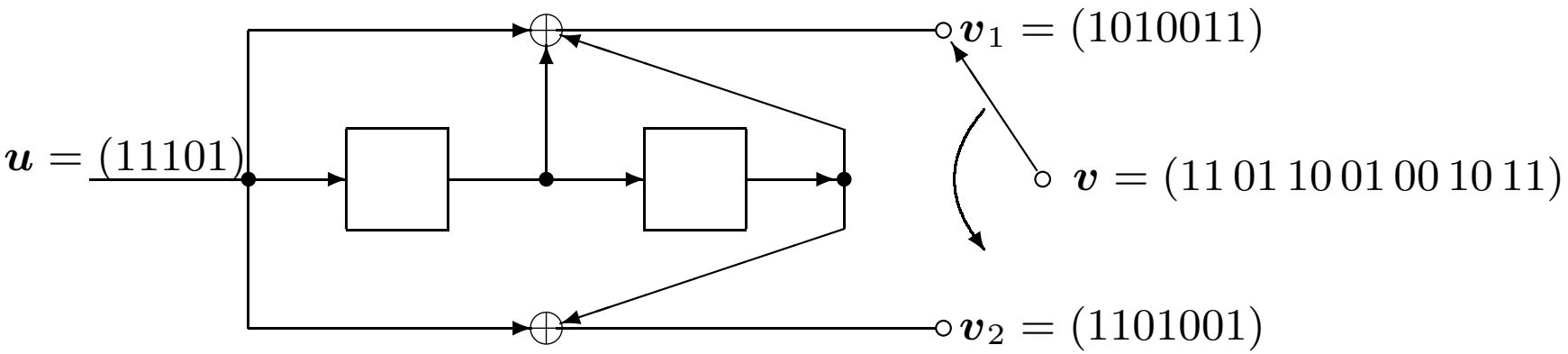
6. The *state sequence* is defined as

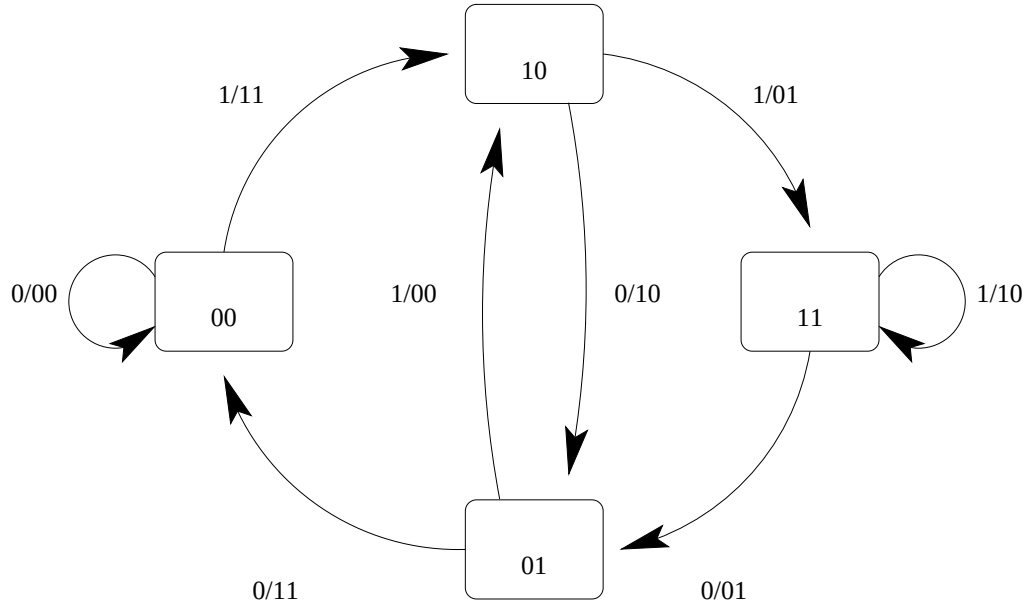
$$\boldsymbol{S} = (\boldsymbol{\sigma}_0, \boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_t, \ldots),$$

where $\boldsymbol{\sigma}_0$ is the initial state of the encoder.

7. If no parallel transmissions exists then there is a one-to-one correspondence between code sequences and state sequences.

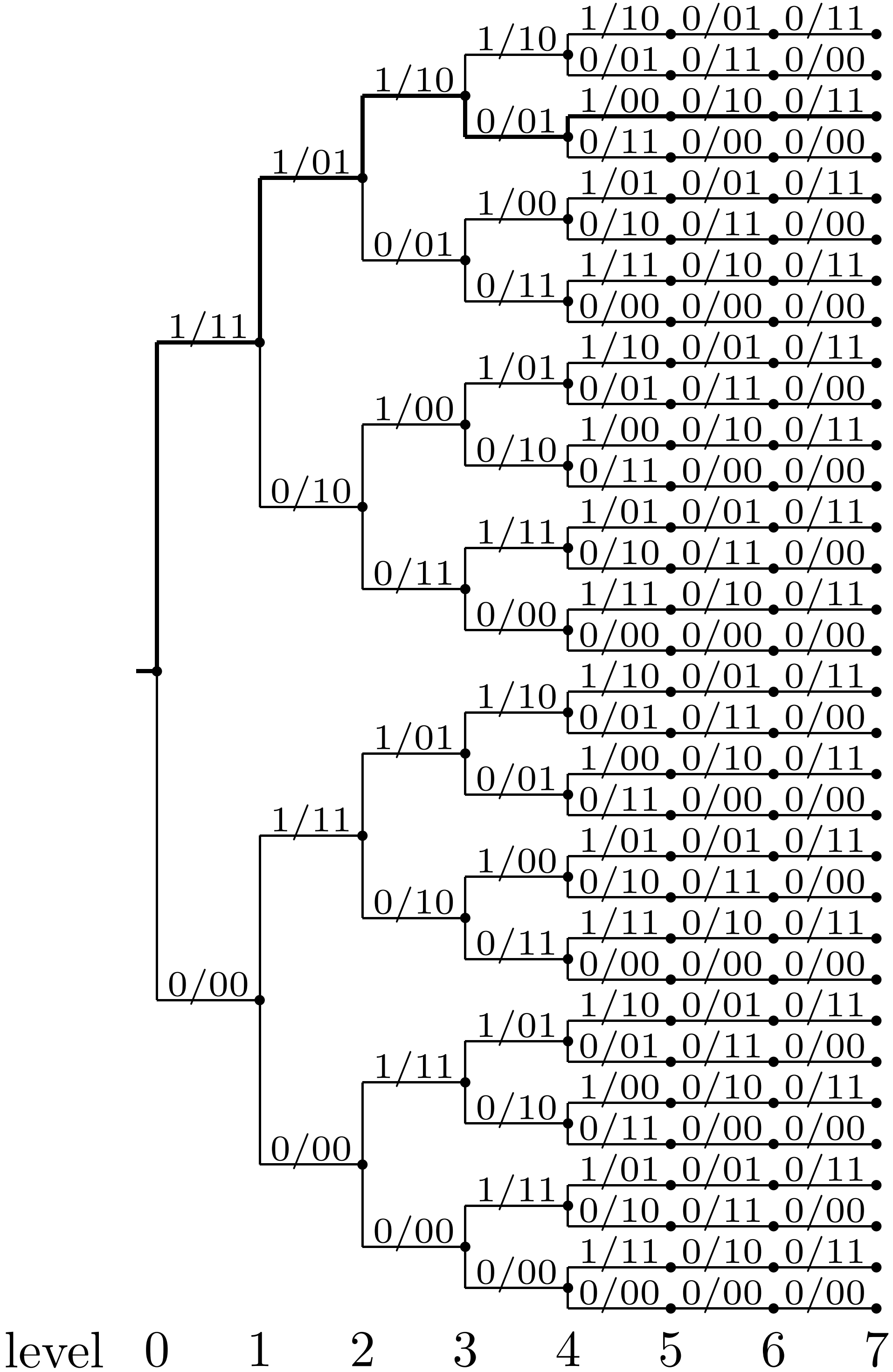
State Diagram: Example




10
1/11
1/01
0/00
00
1/00
0/10
11
1/10
01
0/11
0/01

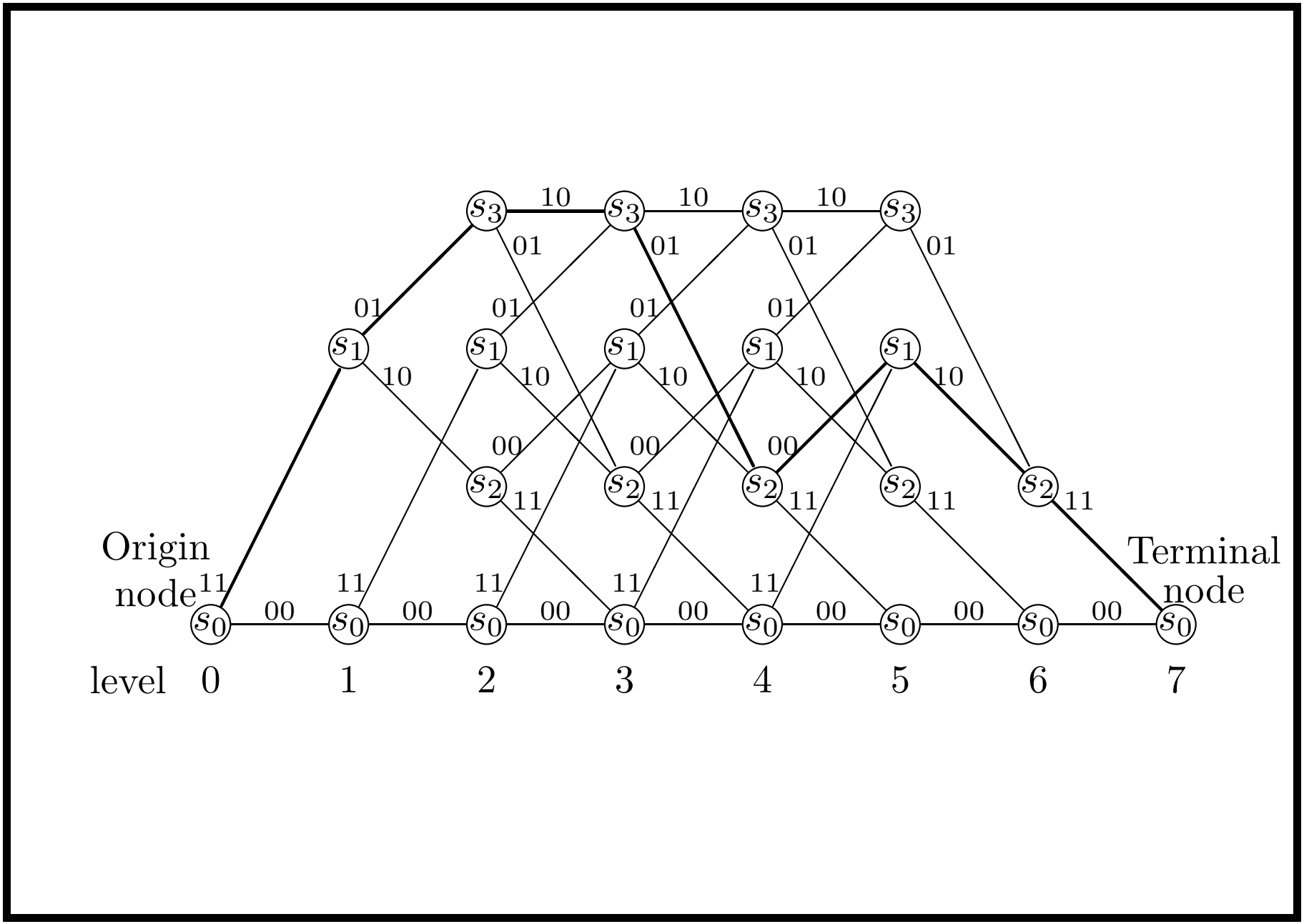
Code Trees of convolutional codes

1. A *code tree* of a binary (n, k, m) convolutional code presents every codeword as a path on a tree.
2. For input sequences of length L bits, the code tree consists of $(L + m + 1)$ levels. The single leftmost node at level 0 is called the *origin node*.
3. At the first L levels, there are exactly 2^k branches leaving each node. For those nodes located at levels L through $(L + m)$, only one branch remains. The 2^{kL} rightmost nodes at level $(L + m)$ are called the *terminal nodes*.
4. As expected, a path from the single origin node to a terminal node represents a codeword; therefore, it is named the *code path* corresponding to the codeword.


1/11
0/00
1/01
0/10
1/11
0/00
1/10
0/01
1/00
0/11
1/01
0/10
1/11
0/00
1/10
0/01
1/00
0/11
1/01
0/10
1/11
0/00
1/10
0/01
1/00
0/11
1/01
0/10
1/11
0/00
1/10 0/01 0/11
0/01 0/11 0/00
1/00 0/10 0/11
0/11 0/00 0/00
1/01 0/01 0/11
0/10 0/11 0/00
1/11 0/10 0/11
0/00 0/00 0/00
1/10 0/01 0/11
0/01 0/11 0/00
1/00 0/10 0/11
0/11 0/00 0/00
1/01 0/01 0/11
0/10 0/11 0/00
1/11 0/10 0/11
0/00 0/00 0/00
1/10 0/01 0/11
0/01 0/11 0/00
1/00 0/10 0/11
0/11 0/00 0/00
1/01 0/01 0/11
0/10 0/11 0/00
1/11 0/10 0/11
0/00 0/00 0/00
1/10 0/01 0/11
0/01 0/11 0/00
1/00 0/10 0/11
0/11 0/00 0/00
1/01 0/01 0/11
0/10 0/11 0/00
1/11 0/10 0/11
0/00 0/00 0/00
level 0 1 2 3 4 5 6 7

Trellises of Convolutional Codes

1. a code *trellis* as termed by Forney is a structure obtained from a code tree by merging those nodes in the same *state*.
2. Recall that the *state* associated with a node is determined by the associated shift-register contents.
3. For a binary (n, k, m) convolutional code, the number of states at levels m through L is 2^K, where $K = \sum_{j=1}^{k} K_j$ and K_j is the length of the jth shift register in the encoder; hence, there are 2^K nodes on these levels.
4. Due to node merging, only one terminal node remains in a trellis.
5. Analogous to a code tree, a path from the single origin node to the single terminal node in a trellis also mirrors a codeword.


Origin node
Terminal node
level 0 1 2 3 4 5 6 7

Column Distance Function

1. Let $\boldsymbol{v}_{(a,b)} = (v_a, v_{a+1}, \ldots, v_b)$ denote a portion of codeword $\boldsymbol{v}$, and abbreviate $\boldsymbol{v}_{(0,b)}$ by $\boldsymbol{v}_{(b)}$. The Hamming distance between the first rn bits of codewords $\boldsymbol{v}$ and $\boldsymbol{z}$ is given by:

$$d_H\left(\boldsymbol{v}_{(rn-1)}, \boldsymbol{z}_{(rn-1)}\right) = \sum_{i=0}^{rn-1} (v_i + z_i).$$

2. The Hamming weight of the first rn bits of codeword $\boldsymbol{v}$ thus equals $d_H(\boldsymbol{v}_{(rn-1)}, \mathbf{0}_{(rn-1)})$, where $\mathbf{0}$ represents the all-zero codeword.

3. The *column distance function* (CDF) $d_c(r)$ of a binary (n, k, m) convolutional code is defined as the minimum Hamming distance between the first rn bits of any two codewords whose first n bits

are distinct, i.e.,

$$d_c(r) = \min\left\{d_H(\boldsymbol{v}_{(rn-1)}, \boldsymbol{z}_{(rn-1)}) : \boldsymbol{v}_{(n-1)} \neq \boldsymbol{z}_{(n-1)} \text{ for } \boldsymbol{v}, \boldsymbol{z} \in \mathcal{C}\right\},$$

where $\mathcal{C}$ is the set of all codewords.

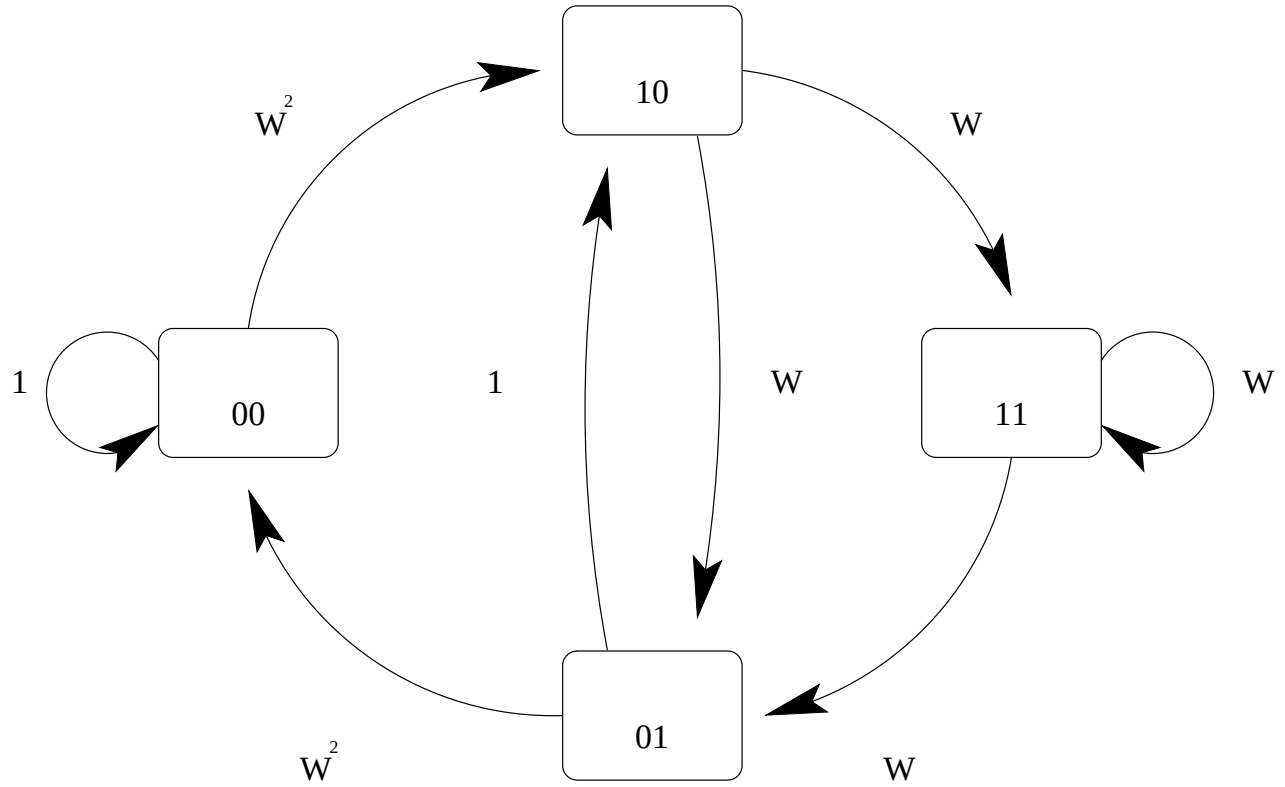
4. Function $d_c(r)$ is clearly nondecreasing in r.
5. Two cases of CDFs are of specific interest: $r = m + 1$ and $r = \infty$. In the latter case, the input sequences are considered infinite in length. Usually, $d_c(r)$ for an (n, k, m) convolutional code reaches its largest value $d_c(\infty)$ when r is a little beyond $5 \times m$; this property facilitates the determination of $d_c(\infty)$.
6. Terminologically, $d_c(m + 1)$ and $d_c(\infty)$ (or d_{free} in general) are called the *minimum distance* and the *free distance* of the convolutional code, respectively.
7. When a sufficiently large codeword length is taken, and an

optimal (i.e., maximum-likelihood) decoder is employed, the error-correcting capability of a convolutional code is generally characterized by d_{free}.

8. In case a decoder figures the transmitted bits only based on the first $n(m+1)$ received bits (as in, for example, the majority-logic decoding), $d_c(m+1)$ can be used instead to characterize the code error-correcting capability.

9. As for the sequential decoding algorithm that requires a rapid initial growth of column distance functions, the decoding computational complexity, defined as the number of metric computations performed, is determined by the CDF of the code being applied.

Path Enumerators



1. The power of the operator W corresponds to the Hamming weight of the code block associated with the transition.

2. We consider that all possible code sequences of the code start at time $t = 0$ and state $\boldsymbol{\sigma}_0 = (00)$, and run through an arbitrary state sequence $S = (\boldsymbol{\sigma}_0, \boldsymbol{\sigma}_1, \ldots)$.

3. The Hamming weight of the code sequence

$$\boldsymbol{S} = (00, 10, 11, 01, 00, 00, \ldots)$$

is 6 since

$$W^2WWW^2 = W^6.$$

4. The calculation of the path enumerators is based on a subset of the set of all possible code sequences.

5. All sequences of this subset leave the zero state at time $t = 0$ and remain there after reaching it again.

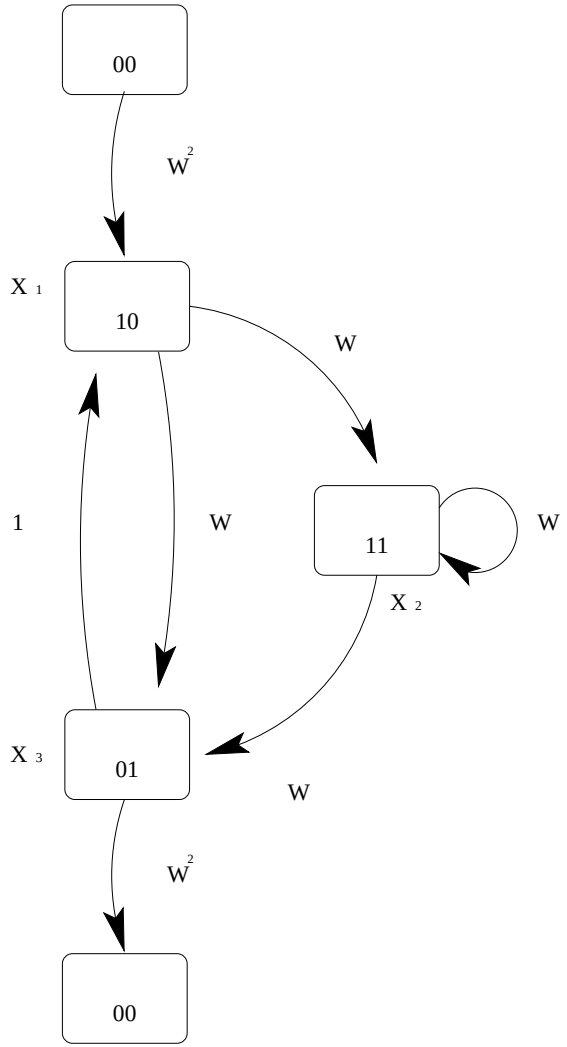
6. The above sequences have the state sequences

$$\boldsymbol{S} = (\mathbf{0}, \boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2, \ldots, \boldsymbol{\sigma}_{\ell-1}, \mathbf{0}, \mathbf{0}, \ldots),$$

with $\boldsymbol{\sigma}_t \neq \mathbf{0}$ for $t \in [1, \ell - 1]$.

7. ℓ must be greater than m since the zero state cannot be reached before $m + 1$ transitions.

8. A procedure from signal flow theory is used to calculate the path enumerator.


00
W2
X 1
10
W
1
W
11
W
X 2
X 3
01
W
W2
00

1. If the variables X_i for $i \in [1, 3]$ are assigned to the nodes between source and sink as shown in the figure then we have

$$\begin{aligned} X_1 &= X_3 + W^2 \\ X_2 &= WX_1 + WX_2 \\ X_3 &= WX_1 + WX_2 \end{aligned}$$

and

$$T(W) = W^2 X_3.$$

2. Each X_i is obtained by adding up all branches that go into the node.

3. The weights of the branches are multiplied by the variable and are assigned to the original node.

4. By solving the above equations, we have

$$T(W) = \frac{W^5}{1-2W} = W^5 + 2W^6 + \cdots + 2^j W^{j+5} + \cdots ,$$

where $T(W)$ is expanded in a series with positive powers of W.

5. In this series, the coefficients are equal to the number of code sequences with corresponding weight. For example, there are 2^j code sequences with weight $j + 5$.

6. The power of the first member of this series corresponds to the free distance of the code. In the previous example, it is 5.

Systematic Matrix

1. An important subclass of convolutional codes is the *systematic codes*, in which k out of n output sequences retain the values of the k input sequences. In other words, these outputs are directly connected to the k inputs in the encoder.

2. A generator matrix of a systematic code has the following structure:

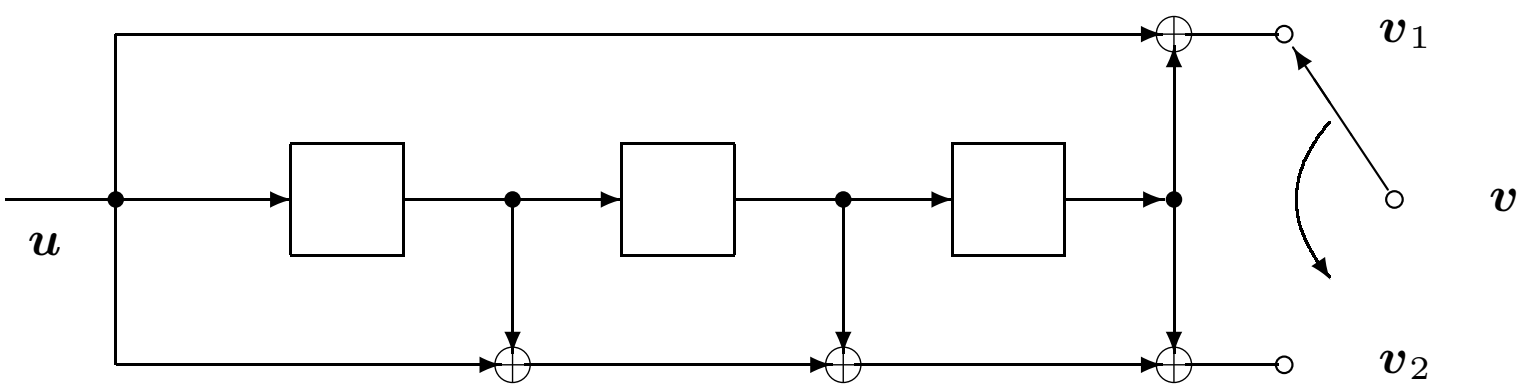
$$\begin{aligned}\boldsymbol{G}(D) &= \begin{pmatrix} 1 & & & G_{1,k+1}(D) & \cdots & G_{1,n}(D) \\ & 1 & & \vdots & & \vdots \\ & & \ddots & & & \\ & & & 1 \quad G_{k,k+1}(D) & & G_{k,n}(D) \end{pmatrix} \\ &= (\boldsymbol{I}_k \mid \boldsymbol{R}(D)),\end{aligned}$$

where $\boldsymbol{I}_k$ is the $k \times k$ identity matrix and $\boldsymbol{R}(D)$ is a $k \times (n-k)$ matrix with rational elements.

3. The input sequences do not need to be equal to the first k output sequences. Any permutation is allowed.

Catastrophic Generator Matrix

1. A catastrophic matrix maps information sequences with infinite Hamming weight to code sequences with finite Hamming weight.
2. For a catastrophic code, a finite number of transmission errors can cause an infinite number of errors in the decoded information sequence. Hence, theses codes should be avoided in practice.
3. It is easy to see that systematic generator matrices are never catastrophic.
4. It can be shown that a loop in the state diagram that produces a zero output for a non-zero input is a necessary and sufficient condition for a catastrophic matrix. A loop is defined as a closed path in the state diagram.
5. The (111) state has a loop associated with 1/00 in the following example.


v_1
v
u
v_2

Punctured Convolutional Codes

1. Punctured convolutional codes are derived from the mother code by periodically deleting certain code bits in the code sequence of the mother code:

$$(n_m, k_m, m_m) \xrightarrow{\boldsymbol{P}} (n_p, k_p, m_p),$$

   where $\boldsymbol{P}$ is an $n_m \times k_p/k_m$ puncturing matrix with elements $p_{ij} \in \{0, 1\}$.

2. A 0 in $\boldsymbol{P}$ means the deletion of a bit and a 1 means that this position is remained the same in the puncture sequence.

3. Let $p = n_m k_p/k_m$ be the puncture period (the number of all elements in $\boldsymbol{P}$) and $w \leq p$ be the number of 1 elements in $\boldsymbol{P}$. It

is clear that $w = n_p$. The rate of the puncture code is now

$$R_p = \frac{k_p}{n_p} = \frac{k_m p}{n_m} \frac{1}{w} = \frac{p}{w} R_m.$$

4. Since $w \leq p$, we have $R_m \leq R_p$.

5. w can not be too small in order to assure that $R_p \leq 1$.

Example of the Punctured Convolutional Code

1. Consider the $(2, 1, 2)$ code with the generator matrix

$$\boldsymbol{G} = \begin{pmatrix} 11 & 10 & 11 & & \\ & 11 & 10 & 11 & \\ & & \ddots & & \ddots \end{pmatrix}.$$

2. Let

$$\boldsymbol{P} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

be the puncture matrix.

3. The code rate of the punctured code is $\frac{k_p}{n_p} = \frac{6}{7}$.

4. If ℓth code bit is removed then the respective column of the

generator matrix of the mother code must be deleted.

5.

11	01	01	01	01	10	11	01	01	01	01	10	11	01
11	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	**11**							
					11	**10**	11						
						11	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	**11**	
											11	**10**	11

6. We can obtain the generator matrix of the punctured code by deleting those un-bold columns in the above matrix:

$$\boldsymbol{G}_p = \left(\begin{array}{cccccc|cccccc}
1 & 1 & 0 & 1 & & & & & & & & \\
 & & 1 & 0 & 1 & & & & & & & \\
 & & & 1 & 0 & 1 & & & & & & \\
 & & & & 1 & 0 & 1 & & & & & \\
 & & & & & 1 & 1 & 1 & 1 & & & \\
 & & & & & & 1 & 1 & 0 & 1 & & \\
\hline
 & & & & & & & 1 & 1 & 0 & 1 & \\
 & & & & & & & & & 1 & 0 & 1 \\
 & & & & & & & & & & \ddots & \ddots
\end{array}\right).$$

7. The above punctured code has $m_p = 2$.
8. Punctured convolutional codes are very important in practical applications, and are used in many areas.

References

[1] M. Bossert, *Channel Coding for Telecommunications*. New York, NY: John Wiley and Sons, 1999.