

## 7.0 Convolutional Codes

### Background

- Elias [1954] introduced convolutional codes.
- Wozencraft [1957] presented sequential decoding.
- Massey [1963] introduced threshold decoding.
- Viterbi [1967] devised maximum likelihood decoder.
- Forney [1970] gave an algebraic formulation.
- Johaneson and Zigangirov [1999] published comprehensive book

### Comparison:

- Convolutional codes tend to be **heuristic** while block codes are often **algebraic**.
- Historically, algebra alone has not produced useful convolutional codes.

## Some Basic Notions

- A **convolutional code** adds **redundancy** to a message sequence in order to form a code sequence.
- **Input:** a sequence of  $k$ -tuples over some field (usually binary).
- **Output:** a sequence of  $n$ -tuples ( $n > k$ ) over (almost always) the same field
- Transition from input to output occurs over **one clock period**.
- Convolutional encoder is a **finite state sequential machine:** its output is a function of input and current “state,” or memory contents.
- The **total memory** ( $m$ ) of an encoder is the number of consecutive  $k$ -tuples that it can store.
- The **constraint length** of a code is the longest number of nonzero output blocks produced by a single, nonzero input block.

## Example 0

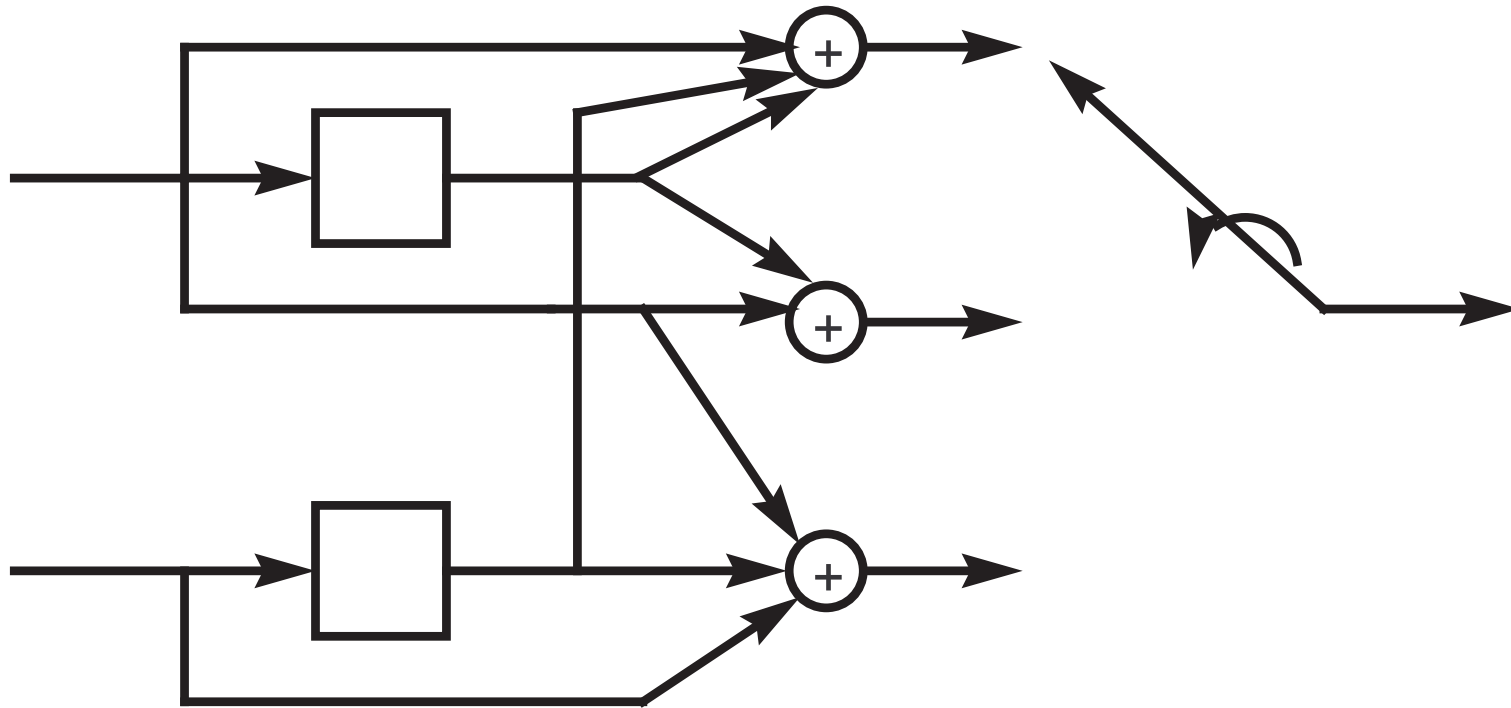


Figure 1: General, feedforward, convolutional encoder

## Impulse Response

- A convolutional encoder has an **impulse response** from each input to each output.
- It is found by determining the response at each output to the sequence  $(1\ 0\ 0\ \dots\ 0)$  applied at each input.
- **Can show** that the encoder output is the **convolution** of the input sequence (of  $k$ -tuples) with the matrix of impulse responses.

In general, there are  $kn$  impulse responses or **generators**,  $\{\mathbf{g}_i^{(j)}, i = 1, \dots, k, j = 1, \dots, n\}$ . **Example 0** of Fig. 1 has 6 generators. We write its input as:

$$\mathbf{i} = \left( i_0^{(1)} i_0^{(2)}, i_1^{(1)} i_1^{(2)}, \dots \right)$$

where

$$\mathbf{i}^{(1)} = \left( i_0^{(1)}, i_1^{(1)}, i_2^{(1)}, \dots \right)$$

$$\mathbf{i}^{(2)} = \left( i_0^{(2)}, i_1^{(2)}, i_2^{(2)}, \dots \right)$$

The impulse responses are:

$$\mathbf{g}_i^{(j)} = \left( g_{i,0}^{(j)}, g_{i,1}^{(j)}, \dots, g_{i,m}^{(j)} \right)$$

So, for Example 0,

$$\begin{array}{ll} \mathbf{g}_1^{(1)} = (11) & \mathbf{g}_2^{(1)} = (01) \\ \mathbf{g}_1^{(2)} = (11) & \mathbf{g}_2^{(2)} = (00) \\ \mathbf{g}_1^{(3)} = (10) & \mathbf{g}_2^{(3)} = (11) \end{array} \quad (1)$$

And,

$$\mathbf{c}^{(j)} = \sum_{i=1}^2 \mathbf{i}^{(i)} \star \mathbf{g}_i^{(j)}, \quad j = 1, 2, 3$$

So, e.g., can show

$$c_\ell^{(1)} = i_\ell^{(1)} + i_{\ell-1}^{(1)} + i_{\ell-2}^{(2)}$$

$$c_\ell^{(2)} = i_\ell^{(1)} + i_{\ell-1}^{(1)}$$

$$c_\ell^{(3)} = i_\ell^{(1)} + i_\ell^{(2)} + i_{\ell-1}^{(3)}$$

These relations can also be written using the **delay** operator, shown later.

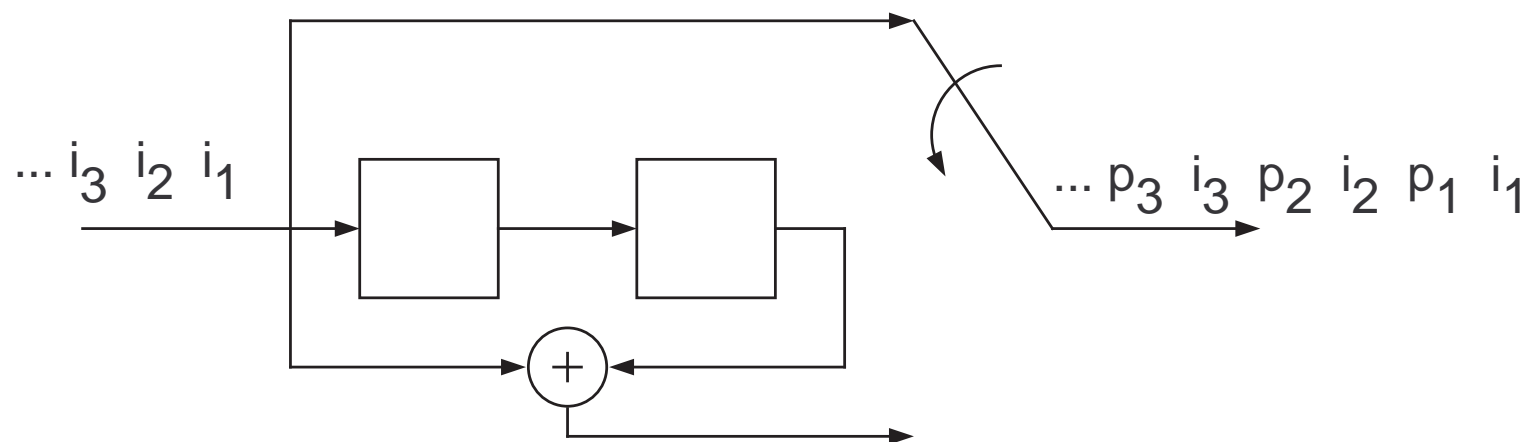
**Example 1.**

Figure 2: 4-state feedforward, systematic encoder

$$m = 2$$

$$k = 1$$

$$n = 2$$

$$R = 1/2$$

**Example 1** provides a simpler problem:

$$\mathbf{c}^{(j)} = \mathbf{i} \star \mathbf{g}^{(j)}, \quad j = 1, 2$$

$$\mathbf{i} = (i_0, i_1, \dots, )$$

$$\mathbf{c} = (\mathbf{c}^{(1)} \mathbf{c}^{(2)})$$

So at time  $\ell$ , we infer from the encoder block diagram that

$$c_\ell^{(1)} = i_\ell g_0^{(1)} + i_{\ell-1} g_1^{(1)} + i_{\ell-2} g_2^{(1)}$$

$$c_\ell^{(2)} = i_\ell g_0^{(2)} + i_{\ell-2} g_2^{(2)}$$

Further,

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



Substitute from (1):

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

So, for  $\mathbf{i} = (11, 10, 010, \mathbf{c} = (001, 011, 111, 110)$ .

## Example 2.

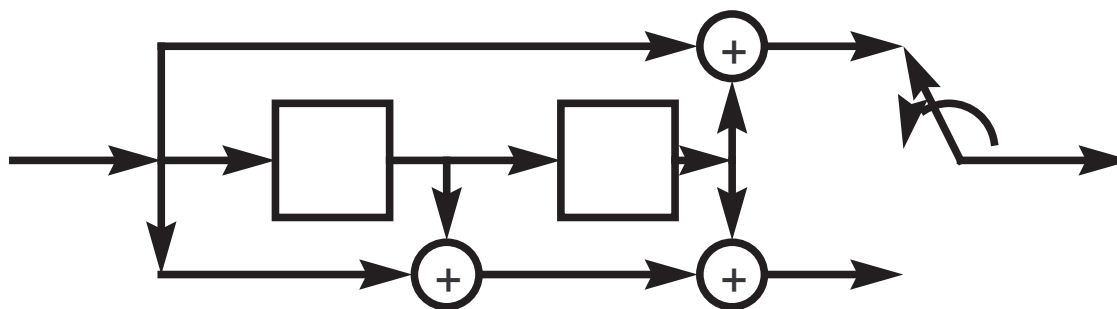


Figure 3: 4-state feedforward, non-systematic encoder

Parameters  $k, m, n, R$  are as in previous. Clearly, this encoder is not systematic.

## Transform Domain Expressions for Convolutional Codes

Let  $k = 1$  (for simplicity). Then  $\mathbf{i} = i_0, i_1, i_2, \dots$  can be written,

$$i(D) = i_0 + i_1 D + i_2 D^2 + \dots$$

where  $D$  represents a unit of time or clock interval. Now for  $\mathbf{g}_i^{(j)}$  let

$$g_i^{(j)}(D) = g_0^{(j)} + g_1^{(j)} D + g_2^{(j)} D^2 + \dots + g_m^{(j)} D^m$$

Now form

$$\begin{aligned} c^{(j)}(D) &\triangleq i(D) \cdot g^{(j)}(D) \\ &= v_0^{(j)} + v_1^{(j)} D + v_2^{(j)} D^2 + \dots \\ &= i_0 g_0^{(j)} + \left( i_0 g_1^{(j)} + i_1 g_0^{(j)} \right) D + \left( i_0 g_2^{(j)} + i_1 g_1^{(j)} + i_2 g_0^{(j)} \right) D^2 \\ &\quad + \dots + \sum_{\ell=0}^r i_\ell g_{r-\ell}^{(j)} D^r + \dots \end{aligned}$$

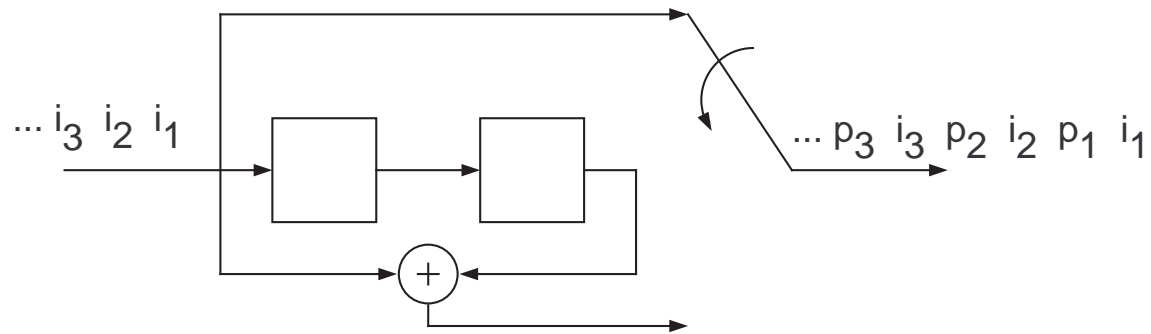
Hence,

$$\begin{aligned}v_r^{(j)} &= \sum_{\ell=0}^r i_\ell g_{r-\ell}^{(j)} \\ &= \mathbf{i} \star \mathbf{g}_r^{(j)}\end{aligned}$$

- This is the **convolution** that defines the linear convolutional encoder.
- So, polynomial multiplication in the  $D$ -domain is equivalent to convolution in the bit-serial domain.

## Structural Properties of Convolutional Codes

We revisit **Example 1**:

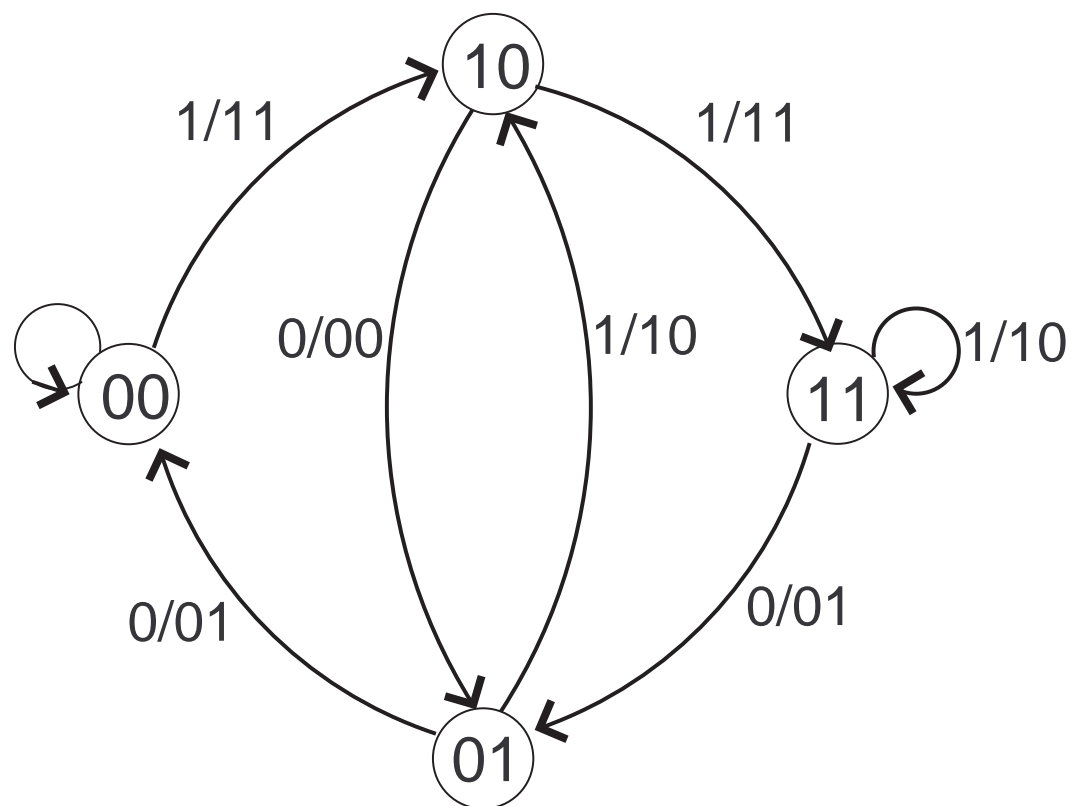


1. initialize to zero;
2. clock in next symbol;
3. compute next state and output;
4. clock in next symbol;
5. continue.

input	state	output
	00	00
0	00	00
1	00	11
0	10	00
1	01	10
1	10	11
0	11	01
0	01	01
1	00	11

## State Transition Diagram for Example 1

State transitions are deduced from the encoder or the table. For branch label  $m/ij$ ,  $m$  = input bit and  $ij$  = output symbol pair.



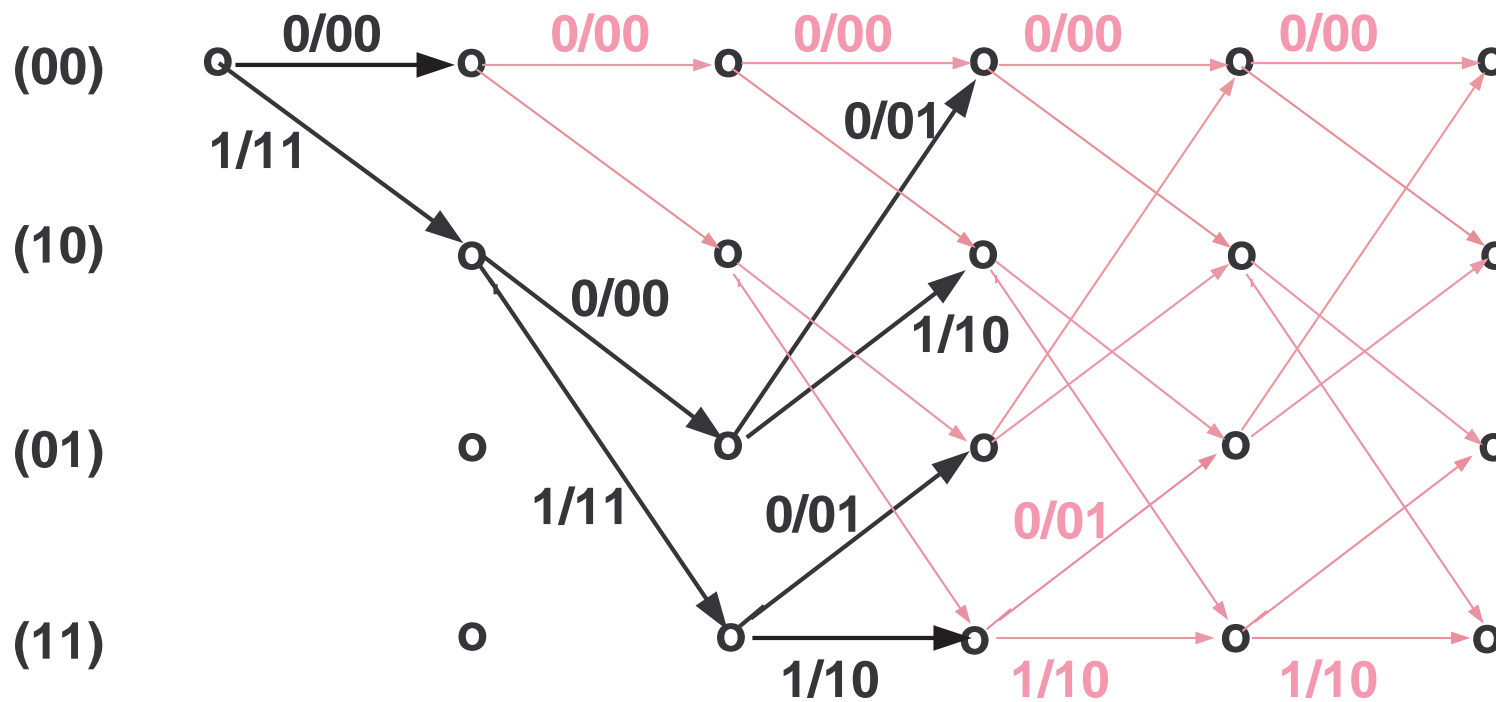


Figure 4: Trellis for CC Encoder of Example 1.

## The Code Tree

- a graphical statement of the encoding rule;
- a regular, branching structure;
- permits “hand encoding;”
- displays graphically the recursive nature of the CC.
- construct directly from the table.

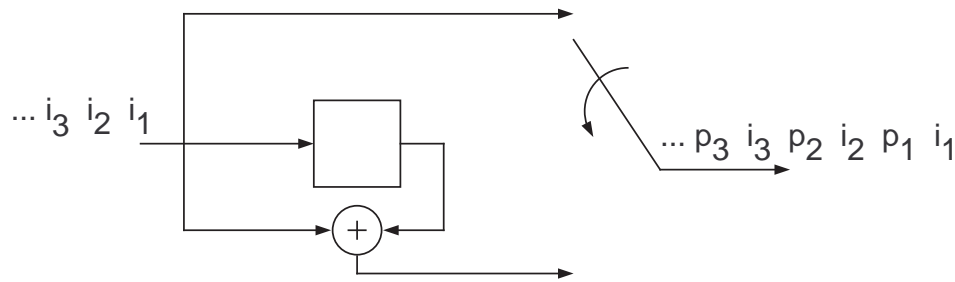
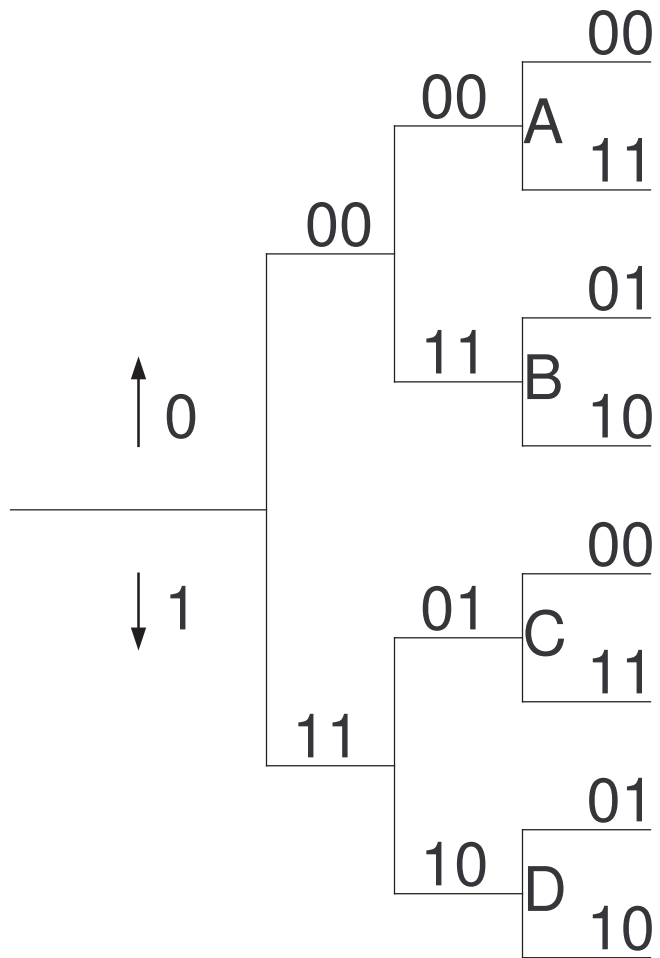


Figure 5: CC for  $m = 1$



I/O/State		
input	state	output
0	0	00
1	0	11
0	1	01
1	0	11
1	1	10
0	1	01
0	1	01
0	0	00

Figure 6: Code Tree for CC Encoder of Figure 5.

## Code Trellis from the Tree

- The output for a given input  $i_j$  depends upon present location in the tree (implicitly, the state).
- The *semi-infinite* tree is cyclic, repeating at various points. e.g.,
  - Sub-tree emanating from **A** is identical to that from **C**.
  - Hence, we can *fold* the tree so that **A** coincides with **C**.
  - Similarly, we can fold the tree so that **B** and **D** coincide.
- Such modifications transform the tree into a trellis.

## Finite-Length Output

1. Usually don't have  $\infty$ -length messages, or messages usually have a *block structure* in application.
2. So CC must be made to have *finite block length*. Two ways:
  - (a) *Truncation*: discard output terms beyond some code block, even if encoder remains in non-zero state.
    - $R = k/n$  but last data block has less protection.
  - (b) *Termination*:
    - Use  $L < \infty$  input data blocks; equivalent to appending  $m - 1$  blocks of zeros (*tail bits*) to the information.
    - “Flush” encoder: all data blocks have same protection.

- But rate for terminated code is:

$$R_{equiv} = \frac{Lk}{n(L+m)} < k/n$$

and we say that the **fractional rate loss** is

$$\begin{aligned} frl &= \frac{R - R_{equiv}}{R} \\ &= \frac{k/n - Lk/n(L+m)}{k/n} \\ &= \frac{m}{L+m} \end{aligned}$$

which becomes negligible for  $L \gg m$ .

## Coming soon!!

- Weight Enumerator Functions – What, why, and how?
- Catastrophic codes
- Distance properties of convolutional codes