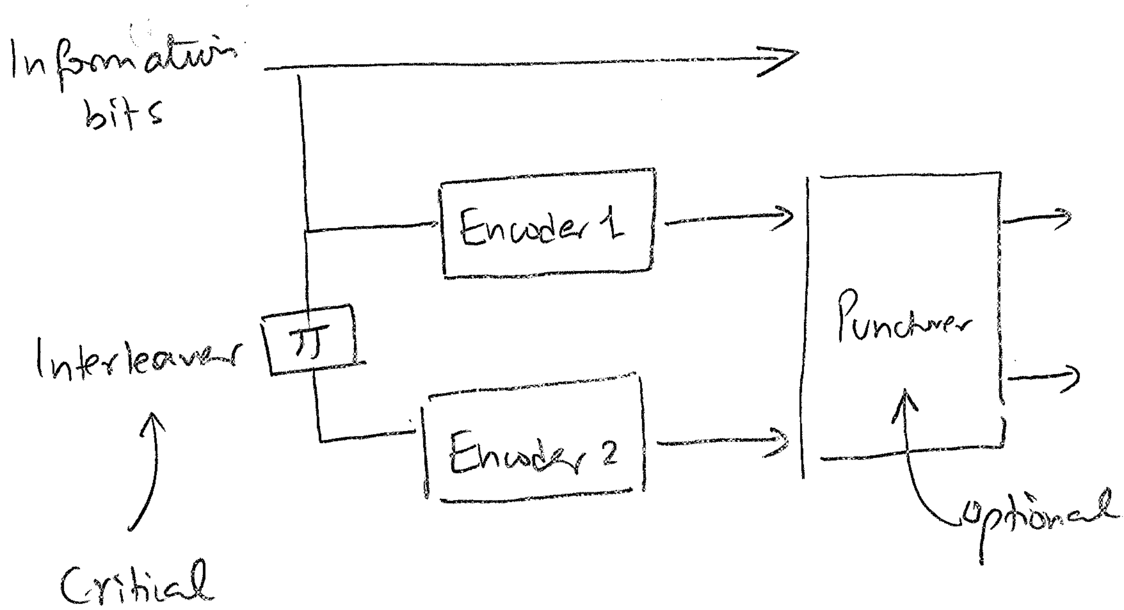
# TURBO CODES

Turbo codes are built from simpler codes as constituent codes and use an interleaver (randomly chosen, ideally) to create random-like codewords. In addition, they use a simple iterative decoding which is not ML but performs very well in practice.

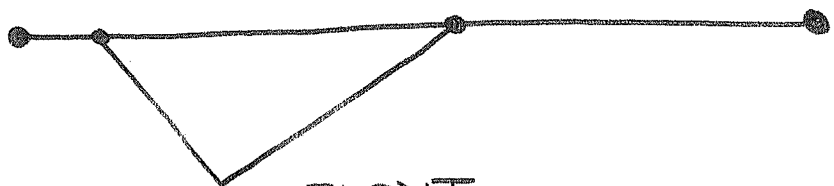
## Parallel Turbo Codes



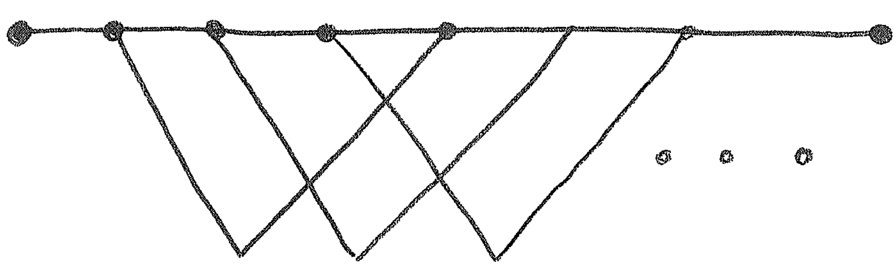

- Encoder 1 may or may not be equal to Encoder 2
- Length of interleaver determines the performance in addition to constituent codes (Encoders 1 & 2)

WHAT IS A PROBLEM WITH CONVOLUTIONAL ENCODERS

A: TIME INVARIANCE



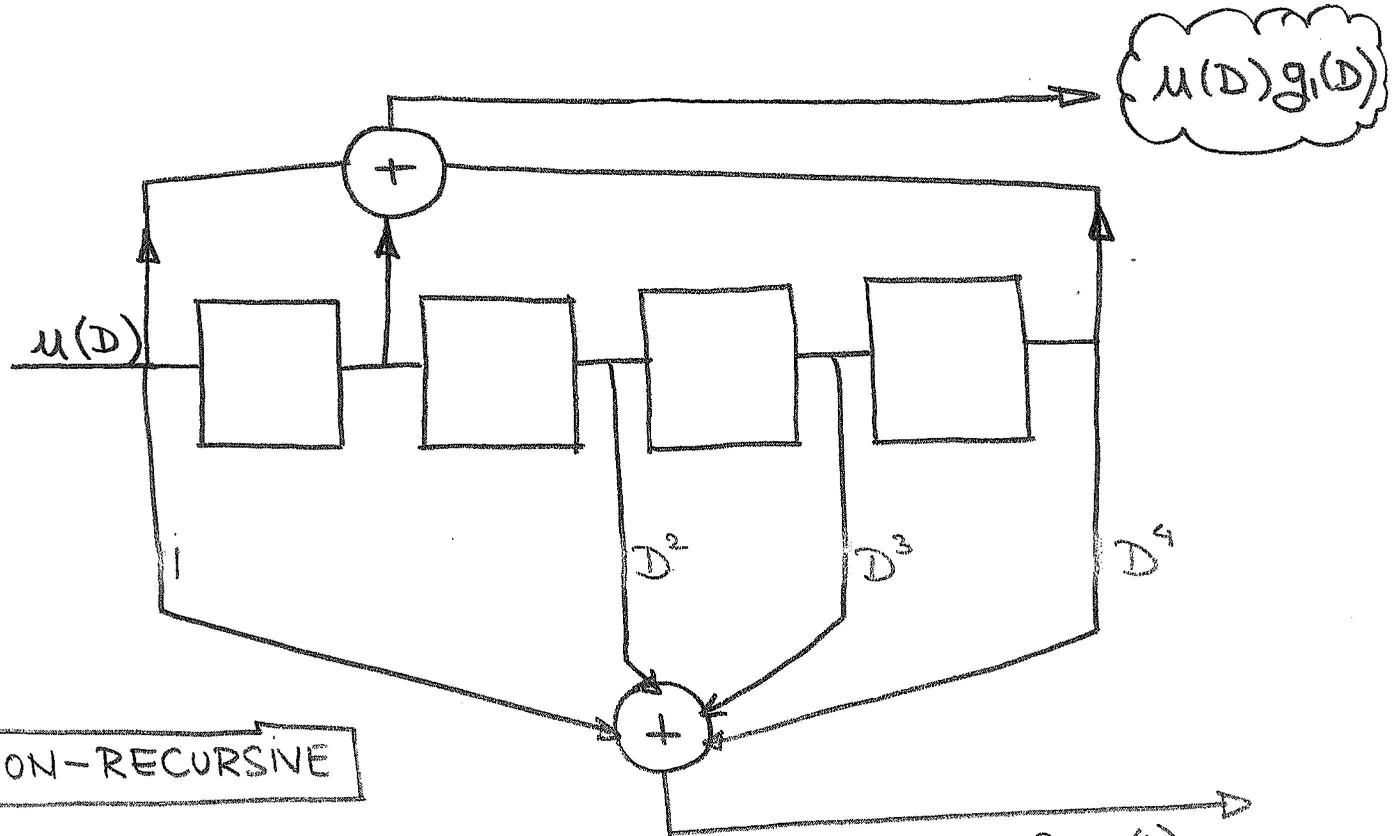
ERROR EVENT



TON OF ERROR EVENTS

ONCE WE HAVE AN ERROR EVENT, WE AUTOMATICALLY GET A TON OF THEM (JUST SHIFTED)

# RECURSIVE CONVOL. CODERS

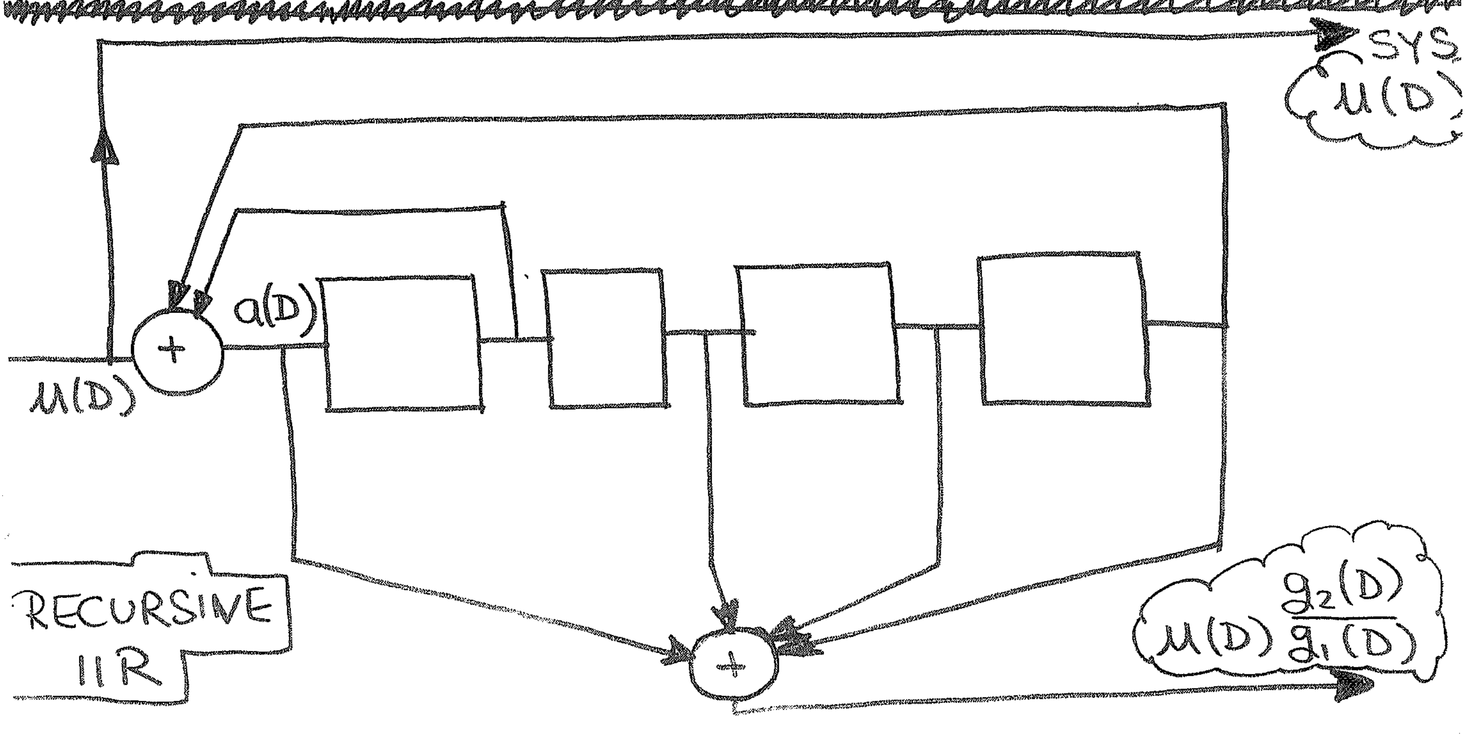



NON-RECURSIVE

$u(D)\,(1+D^2+D^3+D^4)$
$= u(D)\, g_2(D)$

(31, 27) WHY?

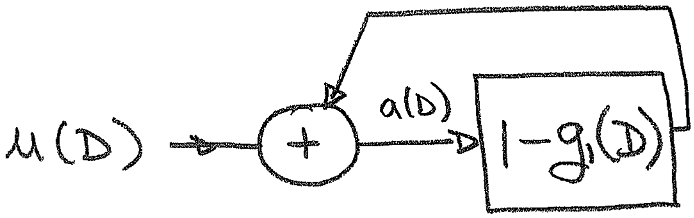
FIR




RECURSIVE
IIR

THIS BECOMES OBVIOUS WHEN WE WRITE EQUATION(S) FOR $a(D)$

— DISREGARD THE BOTTOM PART, AND TOP




$$a(D) = u(D) + a(D)\,(1 - g_1(D))$$

$$\Rightarrow \boxed{a(D) = u(D)/g_1(D)}$$

SO, IF $[g_1(D) \;\; g_2(D)]$ WAS THE NON-RECURSIVE CONVOLUTIONAL ENCODER, THEN

$$[1 \quad g_2(D)/g_1(D)]$$

IS THE "EQUIVALENT" RECURSIVE CONVOLUTIONAL ENCODER.

Q: IS IT SYSTEMATIC?

Q: WHY DO WE CALL IT "EQUIVALENT"

A: INPUT $u(D)$ TO THE NON-RECURSIVE ENCODER PRODUCES THE CODEWORD

$$\left[ u(D)\, g_1(D) \quad u(D)\, g_2(D) \right]$$

INPUT $u(D)\, g_1(D)$ TO THE RECURSIVE ENCODER PRODUCES THE CODEWORD

$$\left[ u(D)\, g_1(D) \cdot 1 \quad u(D)\, \cancel{g_1(D)}\, \frac{g_2(D)}{\cancel{g_1(D)}} \right]$$

$\therefore$ THE RESULTING SET OF CODEWODS IS THE SAME!

$\Rightarrow$ RECURSIVE ENCODER HAS INPUT PATTERNS WHICH PRODUCE FINITE-WEIGHT OUTPUTS

EG: ANY MULTIPLE OF $g_1(D)$ WILL PRODUCE FINITE-WEIGHT OUTPUT

---

FACT: $g_1(D)$ DIVIDES $D^i(1+D^j)$ FOR SOME $i,j$. HOMEWORK.

$\Rightarrow$ $\exists$ WEIGHT-2 INPUT WHICH PRODUCES FINITE OUTPUT, ON THE "PARITY BRANCH"

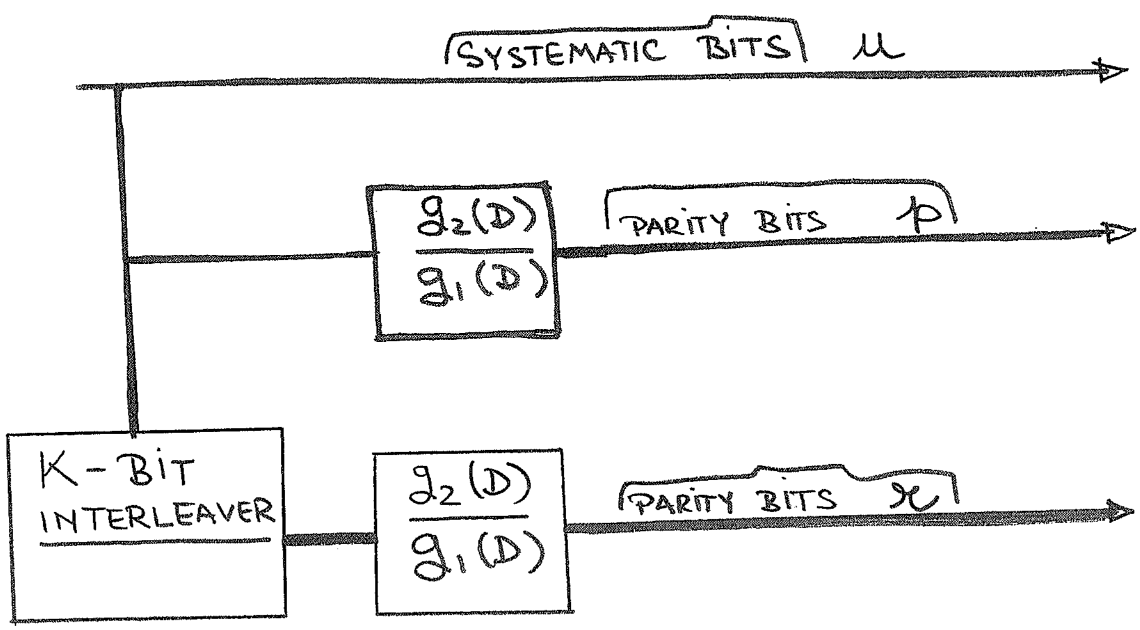
# TURBO ENCODER

- CONCATENATION OF TWO (OR MORE) CONVOLUTIONAL ENCODERS.

- THE NAME "TURBO" COMES FROM THE DECODING PROCESS

- ENCODER STRUCTURE

  2 BINARY, RATE 1/2 CONVOLUTIONAL ENCODERS, SEPARATED BY A K-BIT PSEUDO-RANDOM INTERLEAVER




PARALLEL TURBO ENCODER

# QUESTIONS FOR THE CLASS

1) IS TURBO ENCODER LINEAR ?
2) IS TURBO ENCODER TIME INVARIANT ?
3) TO EVALUATE ERROR PROBABILITY IS IT ENOUGH TO DO IT IN A REFERENCE TO ALL ZERO CODEWORD ?

## WHAT IS SO GREAT ABOUT THIS STRUCTURE?

- LET'S ANALYZE WEIGHTS OF CODEWORDS

THE ML DECODER WILL CHOOSE $k^{TH}$ CODEWORD WITH PROB.

$Q\left(\sqrt{2 d_k R E_b / N_0}\right) = P(0 \to k)$, WHERE

<u>$d_k$ IS THE WEIGHT OF THE $k^{TH}$ CODEWORD.</u>

<u>PROOF</u>: THINK HYPOTHESIS TESTING

NOISE VAR IS $N_0/2$.

ENERGY IN THE $k^{TH}$ CODEWORD IS

$E_b \times$ (ENERGY / INPUT BIT)

$R \times$ (INPUT BIT / OUTPUT BIT)

$d_k$ (# OF DIFFERING PLACES)

$\boxtimes$

<u>LET $W_k$ DENOTE THE INPUT WEIGHT OF THE $k$-TH CODEWORD</u>

## UNION BOUND

$$\boxed{P\left(\bigcup_i A_i\right) \le \sum_i P(A_i)}$$

$$\text{FER} \le \sum_{k=1}^{2^K-1} P(0 \to k)$$

$$= \sum_{w=1}^{K} \sum_{v=1}^{\binom{K}{w}} P(0 \to (w,v))$$

HERE WE HAVE CHANGED THE "VARIABLES" FROM CODEWORD $k$ TO THE "JOINT" VARIABLE DETERMINED BY INPUT WEIGHT $W$ AND ITS "POSITION", HAVING $\binom{K}{W}$ POSSIBILITIE

EXAMPLE $K=3$, LIST INPUTS

| | | | |
|---|---|---|---|
| 0 0 1 | $k=1$ | $W=1$ | $v=1$ |
| 0 1 0 | $k=2$ | $W=1$ | $v=$ |
| 0 1 1 | $k=3$ | $W=2$ | $v=$ |
| 1 0 0 | $k=4$ | $W=1$ | $v=$ |
| 1 0 1 | $k=5$ | $W=2$ | $v=$ |
| 1 1 0 | $k=6$ | $W=2$ | $v=$ |
| 1 1 1 | $k=7$ | $W=3$ | $v=$ |

---

HOMEWORK

$$2^K = \binom{K}{0} + \binom{K}{1} + \dots + \binom{K}{K}$$

SAME # OF TERMS !!!

$\underline{W=1}$ BOTH RECURSIVE ENCODERS PRODUCE "INFINITE" WEIGHT OUTPUTS.

$$P(0 \to (1, v)) \simeq Q(\sqrt{\infty}\quad) = 0$$

GOOD!

$\underline{W=2}$ $\boxed{\text{THIS IS IT!}}$

$\binom{K}{2}$ POSSIBLE INPUTS, SOME $(\approx K)$ OF THEM ARE OF THE FORM

$D^i(1+D^j)$

(SEE NOTES ON RECURSIVE)

$\Rightarrow$ PRODUCE FINITE WEIGHT OUTPUTS (ON THE ENCODER 1).

| |
|---|
| USUALLY, $i=0$ AND ONY SPECIFIC VALUES OF $j$ PRODUCE FINITE WEIGHT |

HOMEWORK: IF $g_1(D)$ IS PRIMITIVE, DEG$=m$ SAY $i=0$, AND WHAT IS THE SMALLEST VALUE OF $j$ PRODUCING FINITE WEIGHT?

AFTER INTERLEAVING, THESE POSITIONS OF $j$'S CHANGE.
IN MOST OF THE CASES, THE SECOND ENCODER WILL PRODUCE INFINITE WEIGHT OUTPUTS.

THIS IS CALLED "SPECTRAL THINNING"

IN GENERAL: THE NUMBER OF INPUT PATTERNS THAT PRODUCE "FINITE" OUTPUTS FOR BOTH ENCODERS IS $\ll K$.

$w \geq 3$ SIMILAR ARGUMENT

NOTE: $d_{w,v}$ IS A FUNCTION OF A PARTICULAR INTERLEAVER.
USUALLY, ANALYSIS IS DONE ON THE AVERAGE (OVER ALL INTERLEAVER POSSIBILITIES) AND IS INVOLVED

Q: WOULD NON-RECURSIVE CODERS WORK WITH THIS ARGUMENT?

TABLE 16.2: Weight spectra of two (64, 28) codes.

| (a) Terminated convolutional | | | | (b) Parallel concatenated | | | |
|---|---|---|---|---|---|---|---|
| **Weight** | **Multiplicity** | **Weight** | **Multiplicity** | **Weight** | **Multiplicity** | **Weight** | **Multiplicity** |
| 0 | 1 | 33 | 25431436 | 0 | 1 | 33 | [illegible] |
| 1 | 0 | 34 | 23509909 | 1 | 0 | 34 | [illegible] |
| 2 | 0 | 35 | 20436392 | 2 | 0 | 35 | [illegible] |
| 3 | 0 | 36 | 16674749 | 3 | 0 | 36 | [illegible] |
| 4 | 0 | 37 | 12757248 | 4 | 0 | 37 | [illegible] |
| 5 | 0 | 38 | 9168248 | 5 | 0 | 38 | [illegible] |
| 6 | 27 | 39 | 6179244 | 6 | 6 | 39 | [illegible] |
| 7 | 28 | 40 | 3888210 | 7 | 9 | 40 | [illegible] |
| 8 | 71 | 41 | 2271250 | 8 | 15 | 41 | [illegible] |
| 9 | 118 | 42 | 1226350 | 9 | 9 | 42 | [illegible] |
| 10 | 253 | 43 | 615942 | 10 | 80 | 43 | [illegible] |
| 11 | 558 | 44 | 287487 | 11 | 119 | 44 | [illegible] |
| 12 | 1372 | 45 | 124728 | 12 | 484 | 45 | [illegible] |
| 13 | 3028 | 46 | 50466 | 13 | 1027 | 46 | [illegible] |
| 14 | 6573 | 47 | 19092 | 14 | 3007 | 47 | [illegible] |
| 15 | 14036 | 48 | 6888 | 15 | 6852 | 48 | [illegible] |
| 16 | 29171 | 49 | 2172 | 16 | 17408 | 49 | [illegible] |
| 17 | 60664 | 50 | 642 | 17 | 40616 | 50 | [illegible] |
| 18 | 122093 | 51 | 140 | 18 | 90244 | 51 | [illegible] |
| 19 | 240636 | 52 | 35 | 19 | 193196 | 52 | [illegible] |
| 20 | 457660 | 53 | 6 | 20 | 390392 | 53 | [illegible] |
| 21 | 838810 | 54 | 2 | 21 | 754819 | 54 | [illegible] |
| 22 | 1476615 | 55 | 0 | 22 | 1368864 | 55 | [illegible] |
| 23 | 2484952 | 56 | 0 | 23 | 2367949 | 56 | [illegible] |
| 24 | 3991923 | 57 | 0 | 24 | 3874836 | 57 | [illegible] |
| 25 | 6098296 | 58 | 0 | 25 | 5988326 | 58 | 0 |
| 26 | 8845265 | 59 | 0 | 26 | 8778945 | 59 | 0 |
| 27 | 12167068 | 60 | 0 | 27 | 12149055 | 60 | 0 |
| 28 | 15844169 | 61 | 0 | 28 | 15907872 | 61 | 0 |
| 29 | 19504724 | 62 | 0 | 29 | 19684668 | 62 | 0 |
| 30 | 22702421 | 63 | 0 | 30 | 22978613 | 63 | 0 |
| 31 | 24967160 | 64 | 0 | 31 | 25318411 | 64 | 0 |
| 32 | 25927128 | | | 32 | 26289667 | | |

- Spectral thinning has little effect on the minimum free distance, but it greatly reduces the multiplicities of the low-weight codewords.
- As the block length and corresponding interleaver size $K$ increase, the weight spectrum of parallel concatenated convolutional codes begins to approximate a randomlike distribution, that is, the distribution that would result if each bit in every codeword were selected randomly from an independent and identically distributed probability distribution.
- There is only a small spectral thinning effect if feedforward constituent encoders are used, as will be seen in the next section.
- One can explain the superiority of feedback encoders in parallel concatenation as a consequence of their being IIR, rather than FIR, filters, that is, their response to single input 1's is not localized to the constraint length of the code but extends over the entire block length. This property of feedback encoders is exploited by a pseudorandom interleaver to produce the spectral thinning effect.

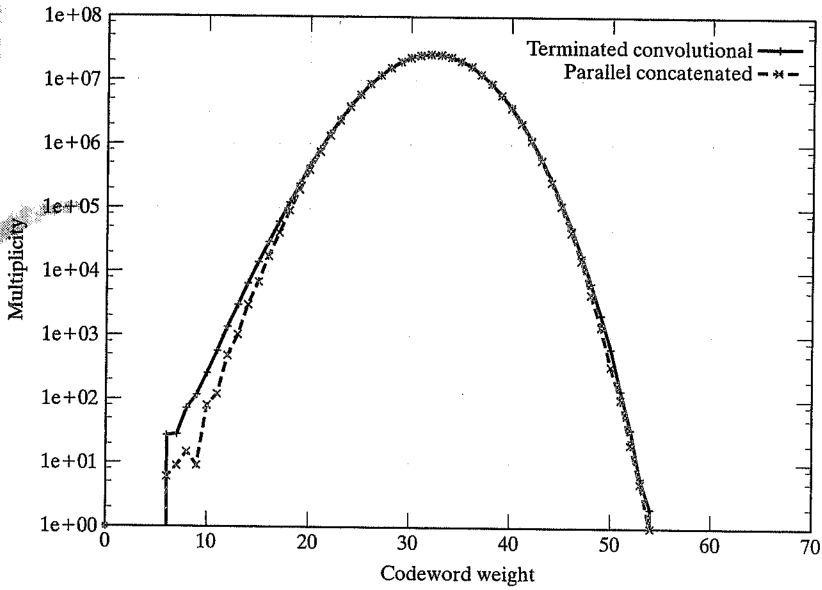



FIGURE 16.4: An illustration of spectral thinning.

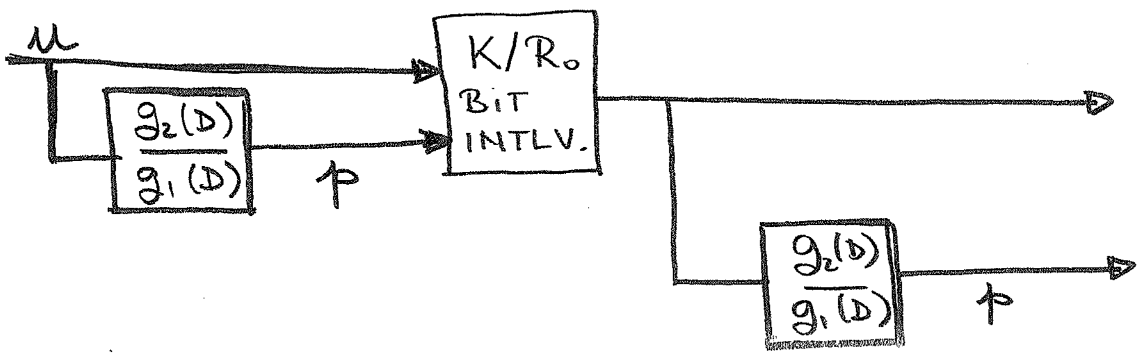
It is also worth noting that parallel concatenated codes are no longer time-invariant. This can easily be seen by considering the effect when the input sequence in Example 16.2 is delayed by one time unit; that is, consider the input sequence $\mathbf{u} = [0100001000000000]$. The first parity sequence $\mathbf{v}^{(1)} = [0110011000000000]$ is also delayed by one time unit, but the interleaved input sequence is now $\mathbf{u}' = [0000000010010000]$, which produces the second parity sequence $\mathbf{v}^{(2)} = [0000000011010011]$. This is clearly not a delayed version of the $\mathbf{v}^{(2)}$ in Example 16.2. In other words, the interleaver has broken the time-invariant property of the code, resulting in a *time-varying* code. To summarize, to achieve the spectral thinning effect of parallel concatenation, it is necessary both to generate a time-varying code (via interleaving) and to employ feedback, that is, IIR, encoders.

It is clear from the preceding examples that the interleaver plays a key role in turbo coding. As we shall now briefly discuss, it is important that the interleaver has pseudorandom properties. Traditional block or convolutional interleavers do not work well in turbo coding, particularly when the block length is large. What is important is that the low-weight parity sequences from the first encoder get matched with high-weight parity sequences from the second encoder almost all the time. This requires that the interleaver break the patterns in the input sequences that produce low-weight parity sequences. Interleavers with structure, such as block or convolutional interleavers, tend to preserve too many of these "bad" input patterns, resulting in poor matching properties and limited spectral thinning. Pseudorandom interleavers, on the other hand, break up almost all the bad patterns and thus achieve the full effect of spectral thinning. In Example 16.2, the 11 input sequences

$$\mathbf{u}(D) = D^l\left(1 + D^5\right), \quad l = 0, 1, \cdots, 10, \tag{16.11}$$

## OTHER NOTES:

SERIAL CONCATENATION:




## PUNCTURING:

WE CAN CHOOSE NOT TO TRANSMIT SOME OF THE PARITY BITS IN SOME FASHION.

THIS INCREASES THE DATA RATE BUT DECREASES RELIABILITY.

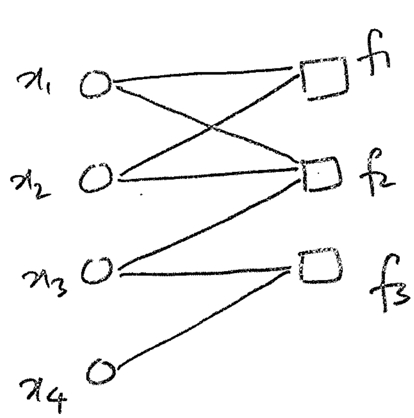
# DECODING OF TURBO CODES

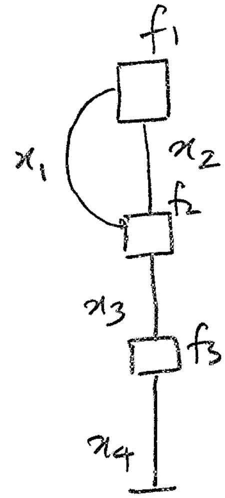
We will use factor graphs and message-passing to decode turbo codes.

Steps in our approach

- An alternate factor graph representation. (Reduces # of nodes but equivalent)
- Decoding component convolutional code on this factor graph
- Decoding turbo code (turbo decoder as special case)

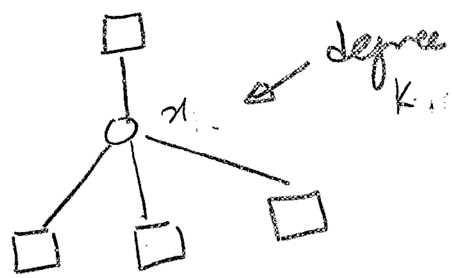
# Forney-style Factor Graphs

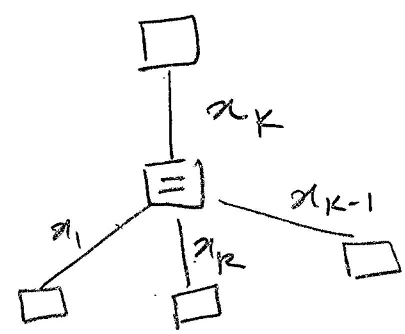



variables are like (half) edges

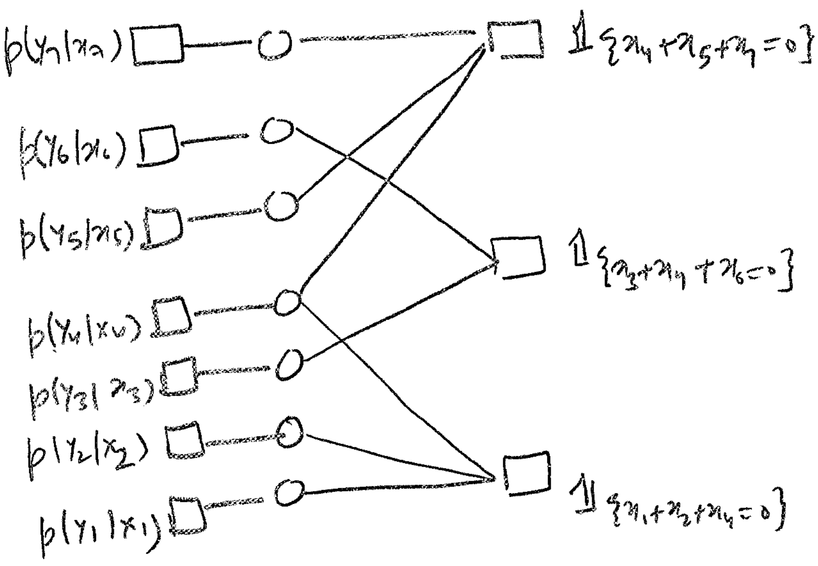
- Draw only factor nodes
- Edge between nodes exist if they share a variable
- converts bipartite graph to a regular graph



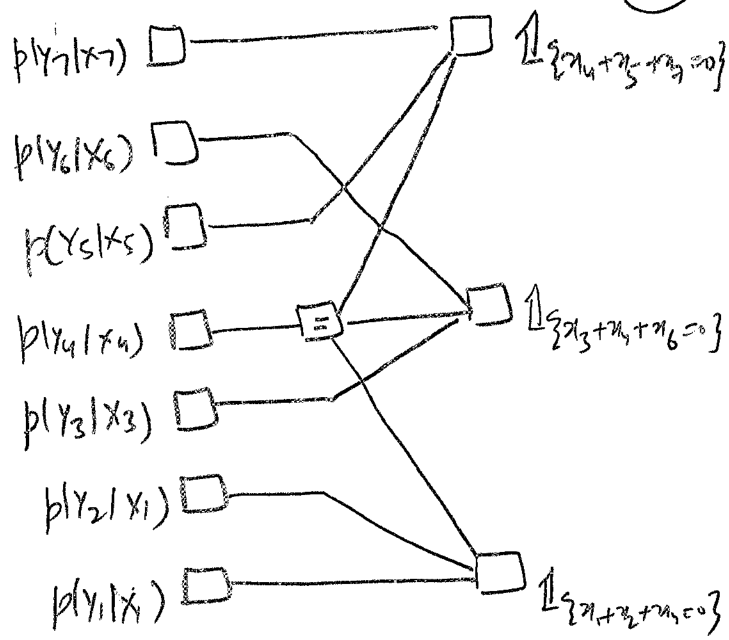




If a variable node has a degree greater than 2, replicate it sufficient number of times.

$$x = x_1 = x_2 = \quad \cdots \quad = x_K$$




Standard Factor Graph




FSFG

Message rules carry over verbatim

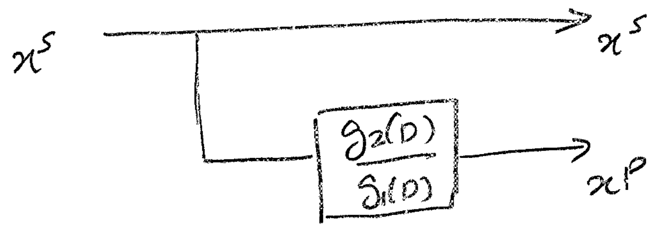
- We do not need variable node rules
- only factor nodes are sufficient
- for a generic node of degree $J+1$, the outgoing message along edge $x$ is

$$\mu(x) = \sum_{\sim x} f(x, x_1, \ldots x_J) \prod_{j=1}^{J} \mu_j(x_j)$$

- For final steps, multiply the incoming messages are multiplied

- FGFGs have fewer nodes and are typically simpler.
- "Internal" edges are "state" variables. "Half-edges" are "external" variables.

Now consider recursive convolutional code :




Let input $x^s = (x_1^s, x_2^s, \ldots, x_n^s, \underbrace{0, 0, 0 \ldots 0}_{m \text{ times}})$

$$m = \max\{\deg(g_1(D)), \deg(g_2(D))\}$$

parity $x^p = (x_1^p, x_2^p, \ldots, x_{n+m}^p)$

For the first $n$ bits, the filter $\frac{g_2(D)}{g_1(D)}$ is used

last $m$ bits, use $g_2(D)$ to ensure termination

---

Use state-space model to represent the encoding

- Let $x_i^s$ denote the $i^{th}$ component of input
  $i = 1, \ldots, n+m$
- Let $\sigma_i$ denote state of system at time $i$
  $i = 0, 1, \ldots, n+m$

$\sigma_{i-1}$ : state of shift register before $x_i^s$

$\sigma_i$ : state after $x_i^s$

$$(\sigma_{i-1}, x_i^s) \longmapsto (\sigma_i, x_i^p)$$

$\sigma_i$ : binary $m$-tuple

$\sigma_0 = (0, \ldots, 0)$

---

Assume that a codeword is transmitted over a binary memoryless channel $p(y|x)$.

$$(X^s, X^p) \longrightarrow \boxed{p(y|x)} \longrightarrow (Y^s, Y^p)$$

Consider bit-wise MAP decoder

$$\hat{x}_i^s = \arg\max_{x_i^s \in \{0,1\}} p(x_i^s \mid y^s, y^p)$$

$$= \arg\max_{x_i^s \in \{0,1\}} \sum_{\sim x_i^s} p(x^s, x^p, \sigma \mid y^s, y^p)$$

$$= \arg\max_{x_i^s \in \{0,1\}} \sum_{\sim x_i^s} p(y^s, y^p \mid x^s, x^p, \sigma)\, p(x^s, x^p, \sigma)$$

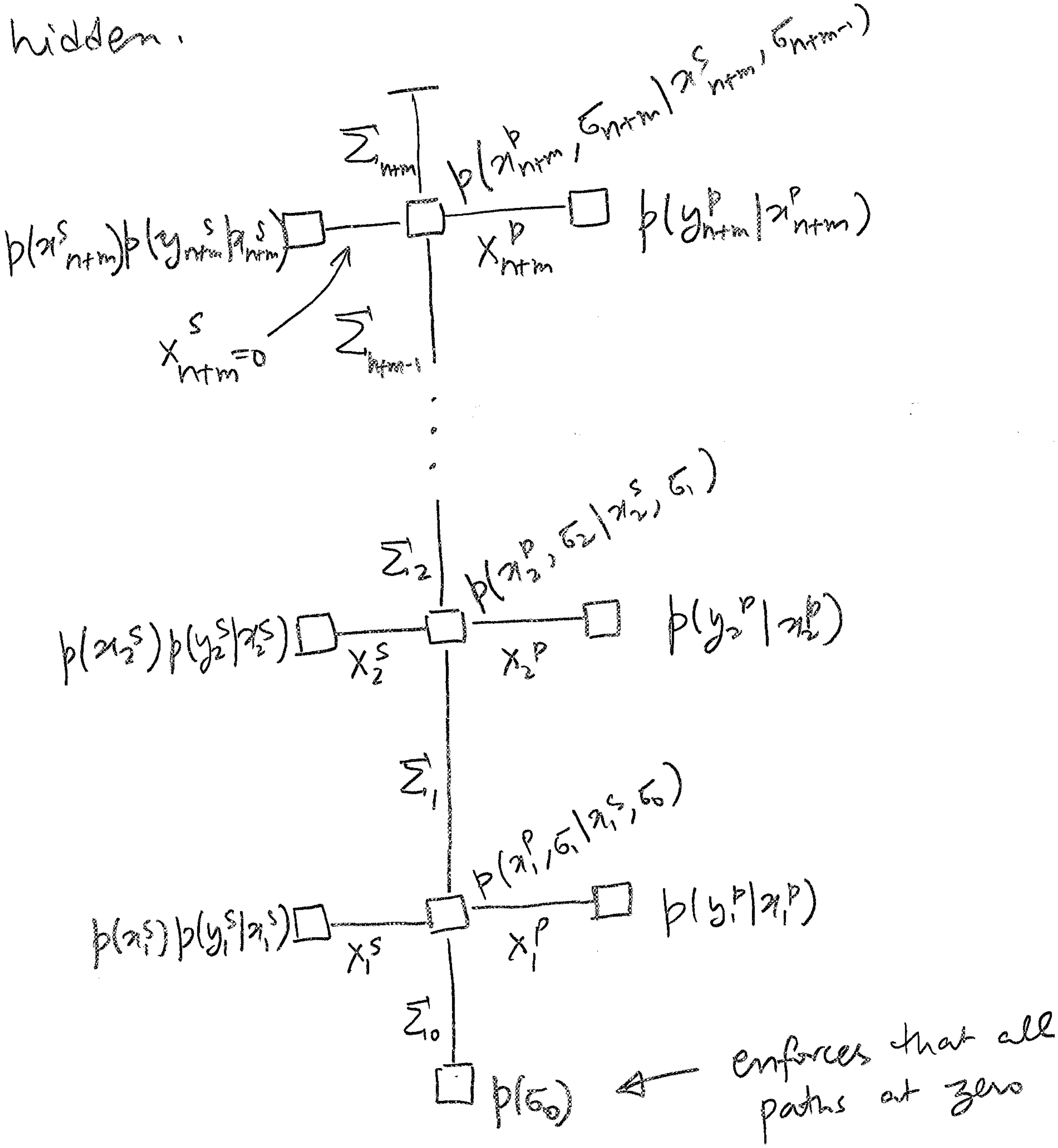
$$= \arg\max_{x_i^s \in \{0,1\}} \sum_{\sim x_i^s} p(\sigma_0) \prod_{j=1}^{n+m} \underbrace{p(y_j^s \mid x_j^s)\, p(y_j^p \mid x_j^p)}_{\text{channel}}$$

$$\underbrace{p(x_j^s)}_{\text{prior}} \; \underbrace{p(x_j^p, \sigma_j \mid x_j^s, \sigma_{j-1})}_{\text{allowed transitions}}$$

$p(\sigma_0)$ is a zero-one function.

$$p(\sigma_0 = \underline{0}) = 1 \;,\; \underbrace{p(\sigma_0 \neq \underline{0}) = 0}_{\text{zero initial state}}$$

Note that state sequence $\sigma_j$ is a variable but hidden.




FSFG for MAP decoding of Conv. code

This FSFG is a tree

$\Rightarrow$ message-passing algorithm is exact.

This algorithm is actually called BCJR algorithm.
(Bahl-Cocke-Jelinek-Raviv)

Note that the graph is essentially a line

$\Rightarrow$ message-passing has two flows of information

The flow starts at the bottom & goes to top

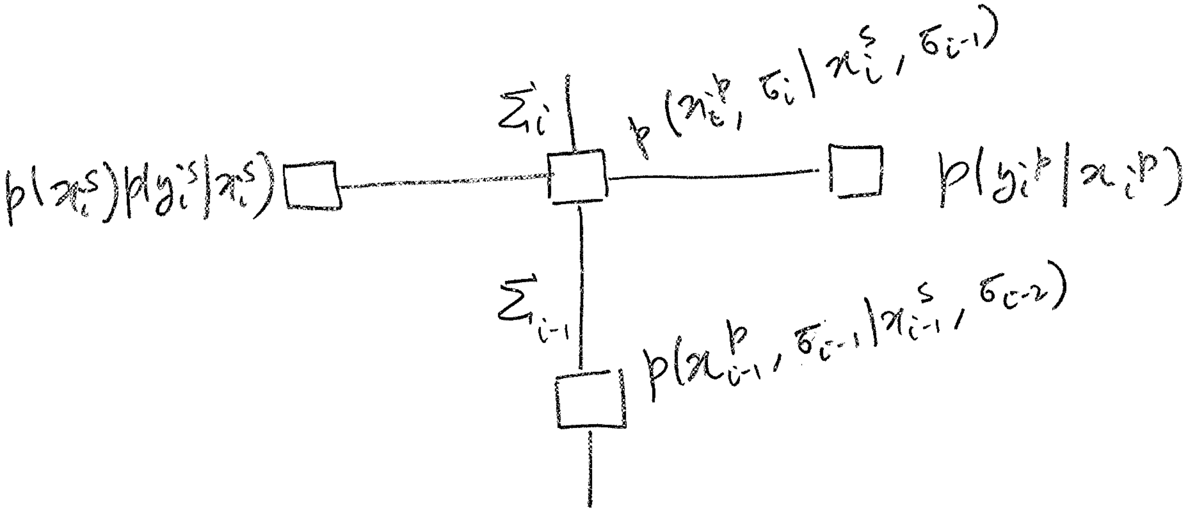
$\rightarrow$ this is called $\alpha$-recursion

And then comes back from top to bottom

$\rightarrow$ this is called $\beta$-recursion

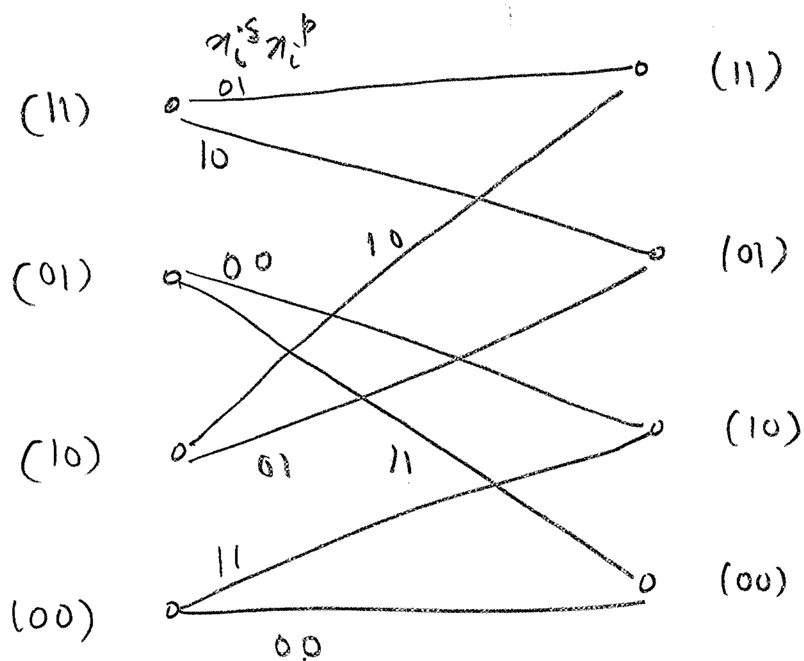
Finally there is a decision step

$\rightarrow$ this is called $\gamma$-step




$p(x_i^p, \sigma_i \mid x_i^s, \sigma_{i-1})$ is a $\{0,1\}$ function

Given the encoder is in state $\sigma_{i-1}$ and the input is $x_i^s$, the function describes next state $\sigma_i$ & corresponding output bit $x_i^p$.




$$\frac{g_2}{g_1} = \frac{1 + D + D^2}{1 + D^2}$$

## $\alpha$-recursion

Message sent on edge labeled $\Sigma'_i$

$$\alpha_{\Sigma'_i}(\sigma_i) = \sum_{x_i^s, x_i^p, \sigma_{i-1}}' \underbrace{p(x_i^p, \sigma_i \mid x_i^s, \sigma_{i-1})}_{\text{Kernel}} \underbrace{p(x_i^s)\, p(y_i^s \mid x_i^s)\, p(y_i^p \mid x_i^p)\, \alpha_{\Sigma'_{i-1}}(\sigma_{i-1})}_{\text{product of incoming messages}}$$

**Claim**: $\alpha_{\Sigma_i}(\sigma_i) = p(\sigma_i, y_1^s, \ldots, y_i^s, y_1^p, \ldots, y_i^p)$

**Proof by induction**: $\alpha_{\Sigma'_0}(\sigma_0) = p_{\Sigma'_0}(\sigma_0)$

For $i > 0$

$$p(\sigma_i, y_1^s, \ldots, y_i^s, y_1^p, \ldots, y_i^p)$$

$$= \sum_{x_i^s, x_i^p, \sigma_{i-1}}' p(\sigma_{i-1}, y_1^s, \ldots, y_{i-1}^s, y_1^p, \ldots, y_{i-1}^p, x_i^s, x_i^p, y_i^s, y_i^p, \sigma_i)$$

$$= \sum_{x_i^s, x_i^p, \sigma_{i-1}}' p(\sigma_{i-1}, y_1^s, \ldots, y_{i-1}^s, y_1^p, \ldots, y_{i-1}^p)\, p(x_i^s)\, p(x_i^p, \sigma_i \mid x_i^s, \sigma_{i-1})\, p(y_i^s \mid x_i^s)\, p(y_i^p \mid x_i^p)$$

$$= \sum_{x_i^s, x_i^p, \sigma_{i-1}}' p(x_i^p, \sigma_i \mid x_i^s, \sigma_{i-1})\, p(x_i^s)\, p(y_i^s \mid x_i^s)\, p(y_i^p \mid x_i^p)\, \alpha_{\Sigma'_{i-1}}(\sigma_{i-1})$$

using induction hypothesis

∎

## $\beta$-recursion

$$\beta_{\Sigma'_{m+n-1}}(\sigma_{m+n-1}) = \sum'_{x^s_{m+n-1},\, x^p_{m+n-1},\, \sigma_{m+n-1}} \underbrace{p(x^p_{m+n-1}, \sigma_{m+n-1} \mid x^s_{m+n-1}, \sigma_{m+n-2})}_{\text{kernel}}$$

$$\underbrace{p(x^s_{m+n-1})\, p(y^s_{m+n-1} \mid x^s_{m+n-1})\, p(y^p_{m+n-1} \mid x^p_{m+n-1})}_{\text{product of incoming messages}}$$

$$= p(y^s_{n+m}, y^p_{n+m} \mid \sigma_{m+n-1})$$

More generally for $i = m+n-1, \ldots, 0$

$$\beta_{\Sigma_i}(\sigma_i) = p(y^s_{i+1}, \ldots, y^s_{n+m}, y^p_{i+1}, \ldots, y^p_{n+m} \mid \sigma_i)$$

[Prove by induction]

$\gamma$-step : We are only interested in $X_i^s$

$$p(x_i^s)\, p(y_i^s \mid x_i^s) \sum_{\sim x_i^s} p(x_i^p, \sigma_i \mid x_i^s, \sigma_{i-1})\, p(y_i^p \mid x_i^p)\, \beta_{\Sigma_i}(\sigma_i)\, \alpha_{\Sigma_{i-1}}(\sigma_{i-1})$$

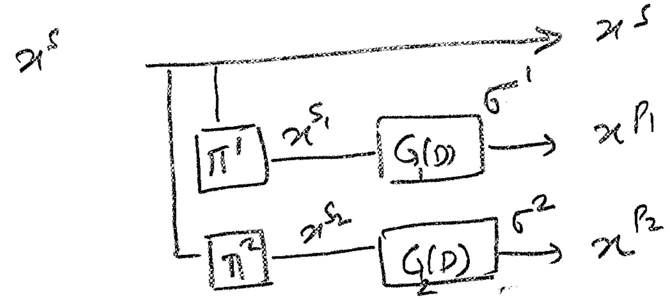
If you insert the expressions for $\beta_{\Sigma_i}(\sigma_i)$ & $\alpha_{\Sigma_{i-1}}(\sigma_{i-1})$, we get

$$p(x_i^s, y_1^s, \ldots, y^s_{n+m}, y^p_1, \ldots, y^p_{n+m})$$

This gives $p(x_i^s \mid y^s, y^p)$.

# NOW TURBO DECODING

Let $x^{s_1} = \Pi^1(x^s)$ , $x^{s_2} = \Pi^2(x^s)$




[$\Pi^1$ = identity]

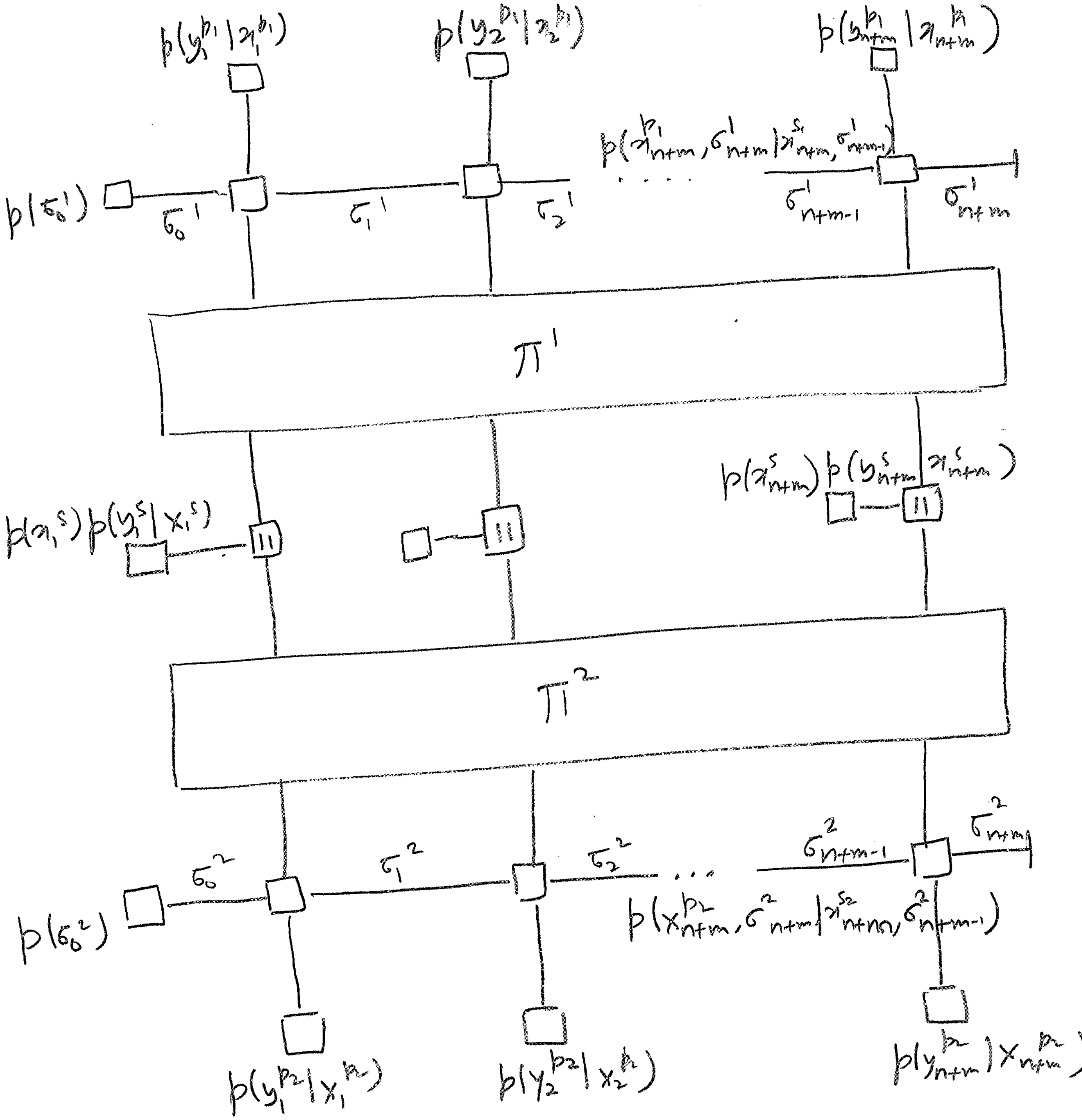
bit-wise MAP decoder

$$\hat{x}_i^{MAP}(y^s, y^{p_1}, y^{p_2}) = \arg\max_{x_i^s \in \{0,1\}} p(x_i^s \mid y^s, y^{p_1}, y^{p_2})$$

$$= \arg\max_{x_i^s \in \{0,1\}} \sum_{\sim x_i^s} p(x^s, x^{p_1}, x^{p_2}, \sigma^1, \sigma^2, y^s, y^{p_1}, y^{p_2})$$

$$= \arg\max_{x_i^s \in \{0,1\}} \sum_{\sim x_i^s} \Bigg( \prod_{j=1}^{n+m} \underbrace{p(x_j^s)}_{\text{prior}} \underbrace{p(y_j^s \mid x_j^s)\, p(y_j^{p_1} \mid x_j^{p_1})\, p(y_j^{p_2} \mid x_j^{p_2})}_{\text{channel}} \Bigg)$$

$$p(\sigma_0^1)\, p(\sigma_0^2) \Bigg( \prod_{j=1}^{n+m} \underbrace{p(x_j^{p_1}, \sigma_j^1 \mid x_j^{s_1}, \sigma_{j-1}^1)}_{\text{code 1}} \underbrace{p(x_j^{p_2}, \sigma_j^2 \mid x_j^{s_2}, \sigma_{j-1}^2)}_{\text{code 2}} \Bigg)$$




- Not a tree, so the schedule matters
- We can start on the leaf nodes and apply the usual factor graph rules

- Turbo schedule:
  - Freeze one component
  - Run BCJR on the second
  - Changes priors for the second
  - Then run BCJR on first