Supplementary Lecture S11

Polar Codes

February 12, 2026

JOHN M. CIOFFI

Hitachi Professor Emeritus (recalled) of Engineering

Instructor EE379A – Winter 2026

Announcements & Agenda

- Agenda
  - Generator Recursions (& Relation to GDFE for 379B supplement)
  - Successive Decoding

Polar Codes
Erdal Arikan
2009

Generator Recursions

Section 8.3.4

Spreading Bandwidth in a Binary Field

- Polar codes invertibly spread encoder-input bits' influence uniformly.
- A simple 2-dimensional example is

$$G_2 = F = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \qquad [v_1 \quad v_0] = [u_1 \quad u_0] \cdot \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

  - Note this matrix is upper triangular monic.
  - $|G_2| = 1$. It is invertible, and it trivially preserves distance ($d_{free} = 1$, $r = 1$).
  - u_1 is spread into $[v_1 \quad v_0]$. u_0 is not spread (used once) and is less reliable.
  - Decoding u_1 is more reliable, while u_0 is less reliable → the bits are **polar**ized.

$$G_4 = F \otimes F \triangleq F^{\otimes 2} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

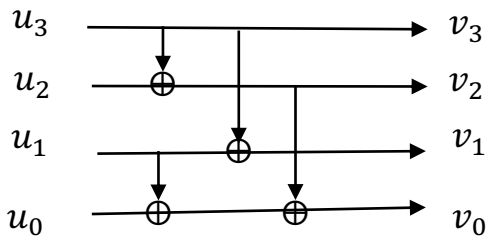
$$G_{2^m = n} = F^{\otimes m}$$

Generators are recursively constructed.

  - G_4 is also upper triangular monic.
  - Same statements apply but u_3 is most spread and most reliably decoded, u_0 is least reliable

Encoder Circuits

- $n = 4$

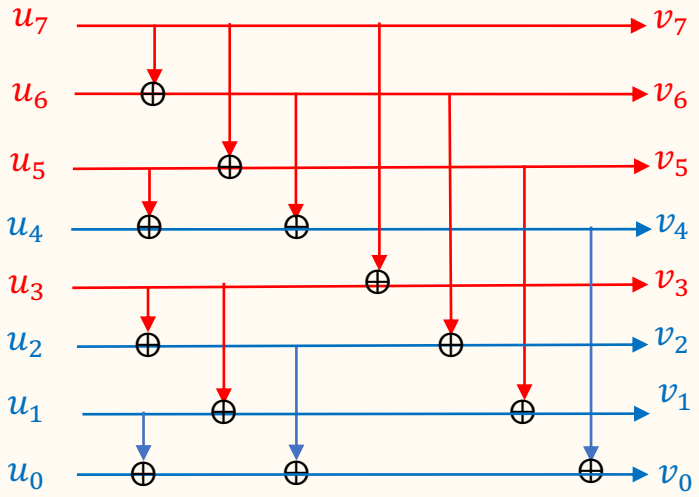


$$G = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- $n = 8$

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

Hot bits



Frozen bits

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

- Frozen bits are the $p = n - k$ least reliable, so set to zero and not transmitted.
- The receiver/decoder knows where these zeros are.

EE379B GDFE Analogies

- The bits have a “chain-rule” interpretation, just like users/dimensions in a GDFE.
- The upper triangular monic matrix lends itself (even in binary) to successive cancellation (SC).
  - SC (or “successive decoding”) is also a more general term for “decision feedback.”
- The GDFE achieves reliable transmission at any rate below capacity, even though not an ML decoder.
  - The GDFE benefits from each user using a good (AWGN-based) code with zero gap.
- Think of bits’ codeword positions as users’ positions in decoding order.
  - No “outer code” in polar code case, but mutual information/entropy are averages over all allowed codes anyway.
  - The arithmetic is binary (but the same chain-rule mutual information result is applicable)
  - For $n < \infty$, the polar codes’ canonical property is lost, but asymptotically they are canonical (achieve nearly zero error probability for rate less than capacity).
- The GDFE’s (optimized, e.g., with minPMAC) order is equivalent to finding frozen bits’ positions.
  - Both GDFE and polar codes have unequal (averaged) mutual information per position/dimension/user.
  - Early GDFE order positions have “low” rate/mutual-info, while late/last positions have high rate/mutual-info

Optimizing Polar Codes' Order

- Use the Bbound quantity for $\bar{P}_b$ (Arikan):
  - Smaller Z means lower $\bar{P}_b$ bound.
  - Recursion to double size

$$\boldsymbol{Z} = \begin{bmatrix} 2\boldsymbol{Z} - \boldsymbol{Z} \odot \boldsymbol{Z} & \boldsymbol{Z} \odot \boldsymbol{Z} \end{bmatrix}$$

$\odot$ is same as matlab's .*

$$\textbf{BSC}\ \ Z = \sqrt{4p(1-p)},$$

$$\textbf{BEC}\ \ Z = p, \text{ or}$$

$$\textbf{AWGN}\ \ Z = e^{-2 \cdot SNR}\ .$$

- The quantity Z also characterizes mutual information → larger is more reliable or higher $\mathcal{I}$
- Arikan's design: Apply the recursion m times and sort the $\boldsymbol{Z}$ elements. Keep k largest, rest are **frozen**.
- Arikan 2009 summary: if $r = k/n$ and $n \to \infty$, this ordering achieves capacity.
  - This also trivially follows from (379B) GDFE/chain-rule perspective.
  - Polar Codes are canonical at infinite codeword length and thus achieve capacity reliably.

Finite Block Length?

- $n < \infty$? Then, the polar code is **not** canonical and has significant degradation.
- Fixes include:
  - Augment small check (CRC, see 379A:L12) code over block length (slightly reduces rate).
  - Keep list of L best codeword matches, keep first in list that passes the check.
  - This improves polar codes (for similar block lengths) to perform at LDPC levels (for same k, n).
- 379A:L10's product codes with GRAND perform about 1dB better and hug capacity performance.
  - That is why polar codes are in a supplementary lecture here.
- Both LDPC and Polar are used in cellular standards.
  - LDPC codes are more heavily used in field.
- GRAND-product codes are 10 years newer, so now finding their way into wider use.

Successive Decoding

Section 8.3.4

Soft Bits

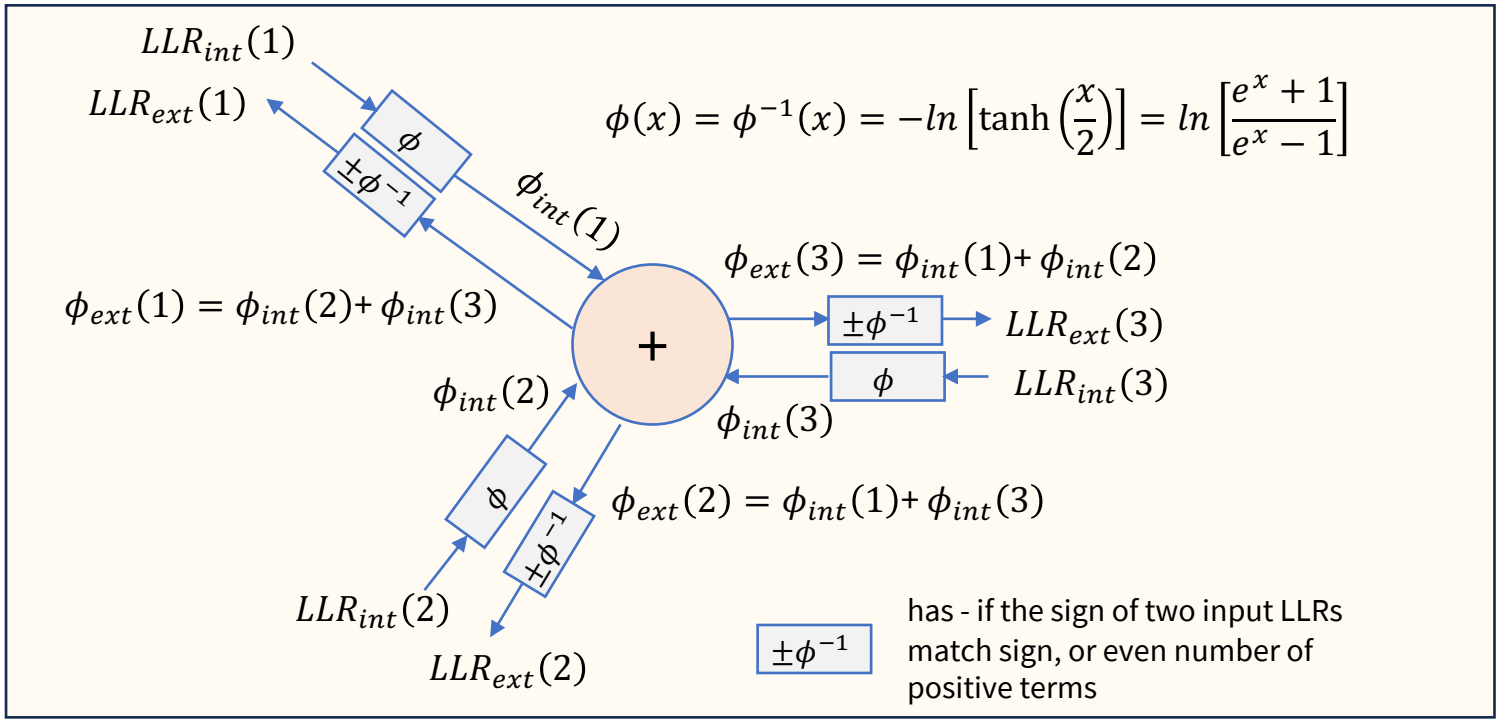
- The **soft bit** is $\chi_i = 2 \cdot Pr\{v_i = 0\} - 1 = 1 - 2 \cdot Pr\{v_i = 1\}$.
  - A soft bit accepts any probability (extrinsic , intrinsic, …) for $Pr\{v_i = 0\}$.

For detailed math/proofs, See end of Section 7.3.1

- The soft bit relates to LLR as $LLR_i = \ln \frac{1+\chi_i}{1-\chi_i}$ or $\chi_i = -tanh\left(\frac{LLR_i}{2}\right)$.
- By induction (with t_r = # of 1's in a parity check) $\chi_i = \prod_{\substack{j=1 \\ j \neq i}}^{t_r} \chi_j$.
- Use this soft bit with extrinsic information for all the "other" bits:
- Define the involution $$\phi(x) = \phi^{-1}(x) = -\ln\left[\tanh\left(\frac{x}{2}\right)\right] = \ln\left(\frac{e^x+1}{e^x-1}\right)$$
- So then $$\phi(LLR_{ext,i}) \triangleq +\ln\left(\frac{e^{LLR_{ext,i}}+1}{e^{LLR_{ext,i}}-1}\right) = -\ln\left(\tanh\left[\frac{LLR_{ext,i}}{2}\right]\right)$$
- And finally: $\chi_i \cdot \chi_j \leftrightarrow \phi(LLR_i) + \phi\left(LLR_j\right) \cdot [\text{sgn}(\llbracket LLR \rrbracket\_i) \cdot \text{sgn}(\llbracket LLR \rrbracket\_j)]$
- .
  - This means no multiplication, just adds and table look-up $\phi(x)$.
  - See text for algebra details

Illustration on next slide

Parity Constraint Soft-Information Flows



- So each bit, considered like a tiny code, sends receives extrinsic info and sends intrinsic info, to all others in E.
- For polar codes, input 1 views inputs 2 and 3 (their sum really) as the other bit and works only 2 bits at a time.

SC's combining functions f and g

- The polar "SC" decoder uses two functions. The first f corresponds to the box sum $\boxplus$ for parity sum.
  - One bit is a single bit, but the other is (often) a sum of bits.

$$f(LLR_i, LLR_j) \triangleq -\ln\left[\tanh\left(\frac{LLR_i}{2}\right)\cdot\tanh\left(\frac{LLR_j}{2}\right)\right]$$

- This f function corresponds to sum of parity-bits' ϕ functions, (when |LLR| is big, tanh = ± 1)

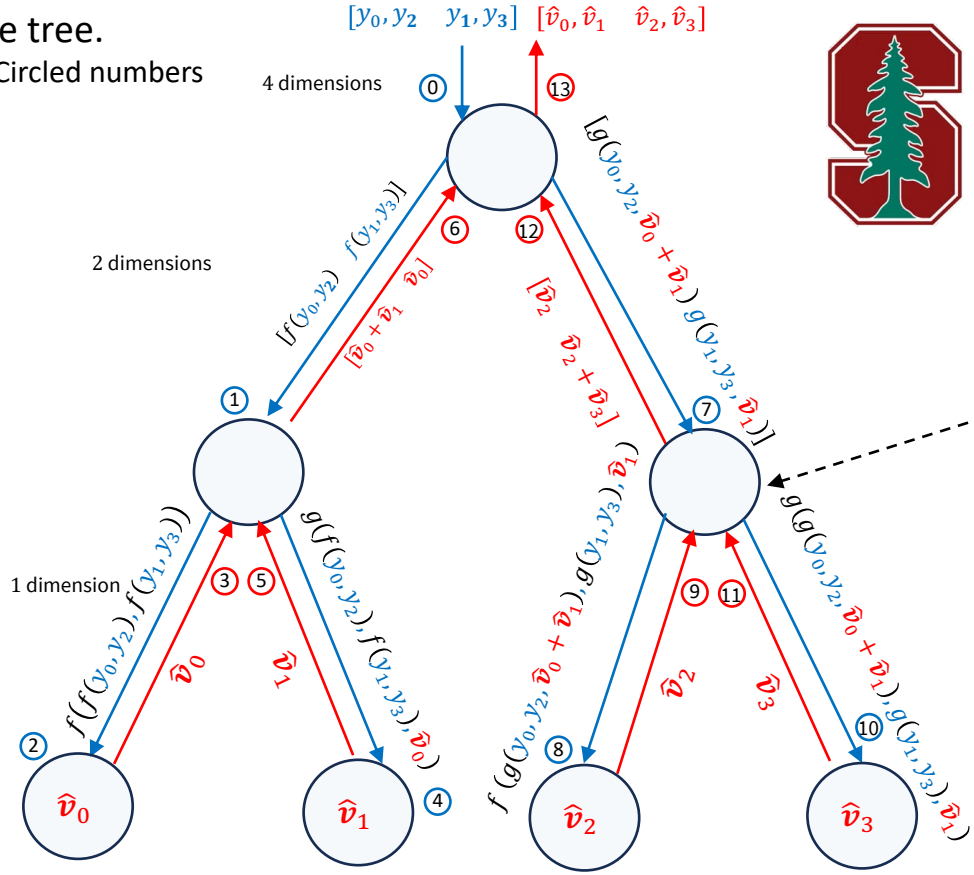
$$\phi(LLR) = [\phi(LLR_i) + \phi(LLR_j)]\cdot[\mathrm{sgn}(LLR_i)\cdot\mathrm{sgn}(LLR_j)] = \pm\phi(LLR_i \boxplus LLR_j)$$

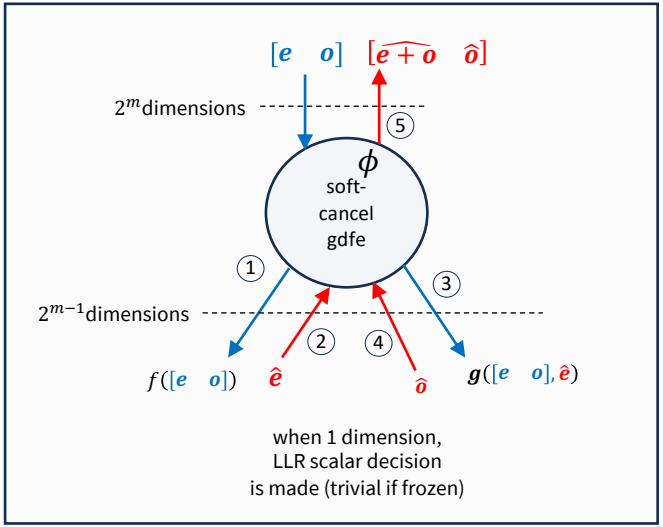
- The "gdfe" function removes a decision's influence on LLR inputs x and y.

$$g(x, y, \hat{u}) = y + (2\cdot\hat{u} - 1)\cdot x$$

Fear the Tree 😀 Successive Decoder & gdfe

- Trace tree.
  - Circled numbers

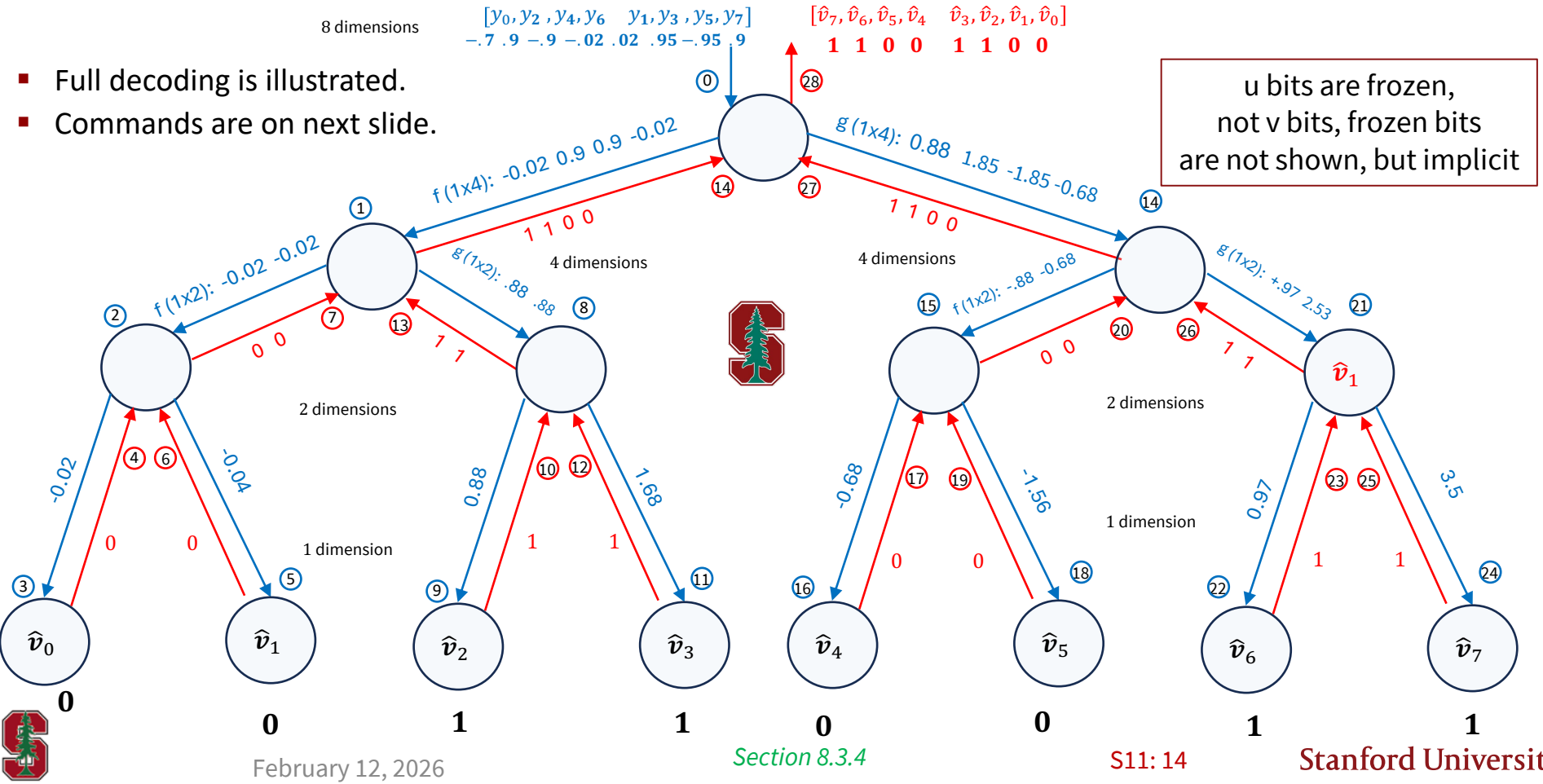




Bottom nodes invert ϕ and take polarity of resultant LLR for decision

Polar (also Hamming) – (8,4,4) code example

- Full decoding is illustrated.
- Commands are on next slide.



Commands

- ChatGPT matlab program
  - Available from instructor on request.
- SC decoder, prints out f and g values also.
- These are placed on the tree on S11:14.
  - Some ChatGPT sign errors, but obvious and fixed.

```
>> u = [   1   0   1   0];
>> v  = [   1   1   0   0   1   1   0   0 ];
>> vreal = [   1   1  -1  -1   1   1  -1  -1];

>> noise = [  -0.1000  -1.0200   0.0500   0.1000  -0.0500  -0.1000   1.0200   0.3000]
>> y=vreal+noise;
>> [u_info,~,~] = polar_sc_decode_8_4(y', [])

u_info' = 1   0   1   0

SC per-bit LLR paths (bit-reversed order, your info order [u8 u7 u6 u4]):
  u_7 path: -0.68  -2.53  -3.5  -3.5
  u_6 path: -0.02  +0.88  +1.76  +1.76
  u_5 path: +1.85  -0.68  -1.56  -1.56
  u_3 path: -1.85  -0.97  +0.97  +0.97

   L1 (top):
  left_f (1x4):  -0.02 +0.9 +0.9 -0.02
  right_g (1x4): +0.88 +1.85 -1.85 -0.68

L2 (size-4 blocks split to 2+2):
  LL.f (1x2): -0.02 -0.02
  LR.g (1x2): +0.88 +0.88
  RL.f (1x2): -0.88 -0.68
  RR.g (1x2): -0.97 -2.53

L3 (size-2 blocks split to 1+1):
  LL.f: +0.02   LL.g: -0.04
  LR.f: +0.88   LR.g: +1.76
  RL.f: +0.68   RL.g: -1.56
  RR.f: +0.97   RR.g: -3.5
```

End Lecture S11