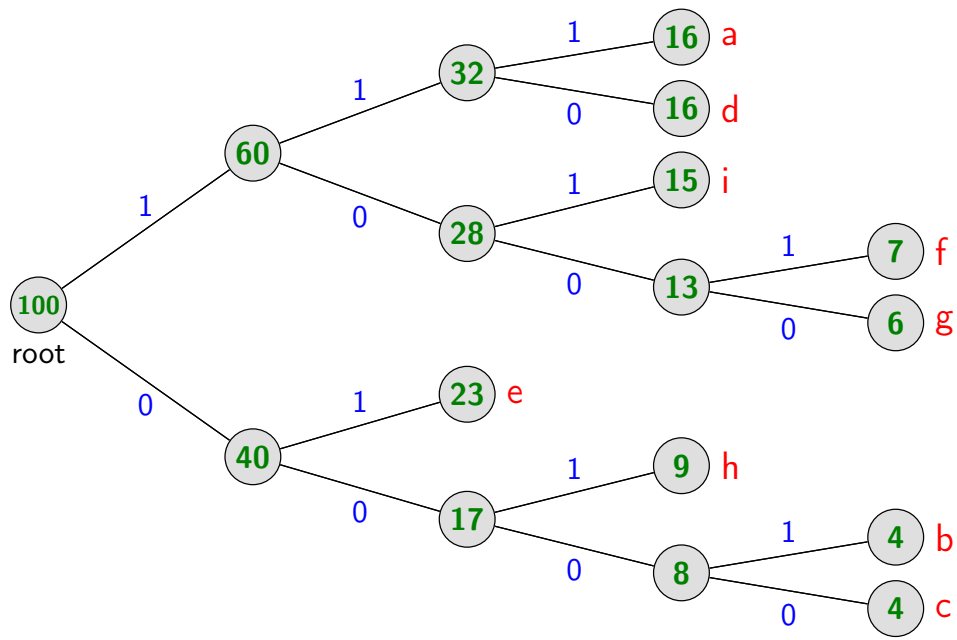
Lossless Coding IV

a_k	p_k	$\boldsymbol{b}_k$
a	0.16	111
b	0.04	0001
c	0.04	0000
d	0.16	110
e	0.23	01
f	0.07	1001
g	0.06	1000
h	0.09	001
i	0.15	101



Shannon-Fano-Elias Coding: Review

Shannon-Fano-Elias Coding: Special Block Code of size N

- Order of N-symbol sequences $\{\boldsymbol{s}_k\}$ is known by encoder and decoder
- N-th order pmf $p(\boldsymbol{s}_k) = \mathrm{P}(\boldsymbol{S} = \boldsymbol{s}_k)$ is known by encoder and decoder

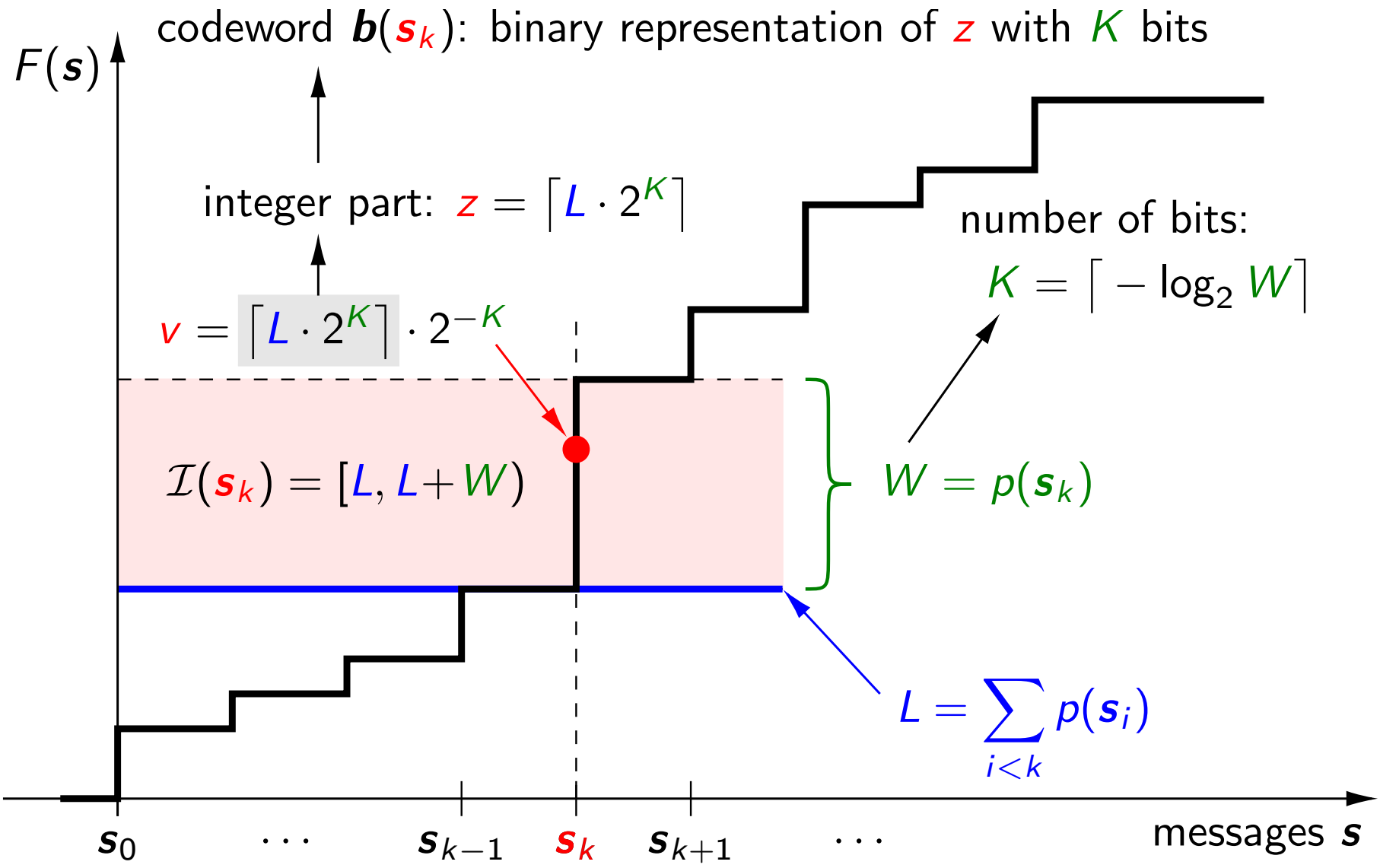
➡ On-the-fly encoding and decoding (no codeword table)

Basic Concept

- Unique mapping of N-symbol sequences to intervals $\mathcal{I}_k$ of cdf $F(\boldsymbol{s})$
- Half-open intervals $\mathcal{I}_k = [L_k, U_k) = [L_k, L_k + W_k)$ are characterized by
  - Interval width: $W_k = F(\boldsymbol{s}_k) - F(\boldsymbol{s}_{k-1}) = p(\boldsymbol{s}_k)$
  - Lower interval boundary: $L_k = F(\boldsymbol{s}_{k-1}) = \sum_{i<k} p(\boldsymbol{s}_i)$
- Codewords: Fractional bits of representive value v_k inside interval $\mathcal{I}_k$
  - Length of codeword: $K = \lceil -\log_2 W_k \rceil$
  - Representative value: $v_k = \lceil L_k \cdot 2^K \rceil \cdot 2^{-K} = z_k \cdot 2^{-K}$

  ➡ Codeword: Binary representation of $z_k = \lceil L_k \cdot 2^K \rceil$ with K bits

Shannon-Fano-Elias Encoding: Illustration



Shannon-Fano-Elias Encoding: Summary

Determination of Codewords

- Given: Ordered set of symbol sequences $\{\boldsymbol{s}_k\}$ with associated pmf $\{p_k\}$
- Construct codeword $\boldsymbol{b}_k = \boldsymbol{b}(\boldsymbol{s}_k)$ for any particular sequence $\boldsymbol{s}_k$ by

1. Determine interval width W_k and lower interval boundary L_k

$$W_k = p_k \tag{1}$$

$$L_k = \sum_{i<k} p_i \tag{2}$$

2. Determine codeword length K_k

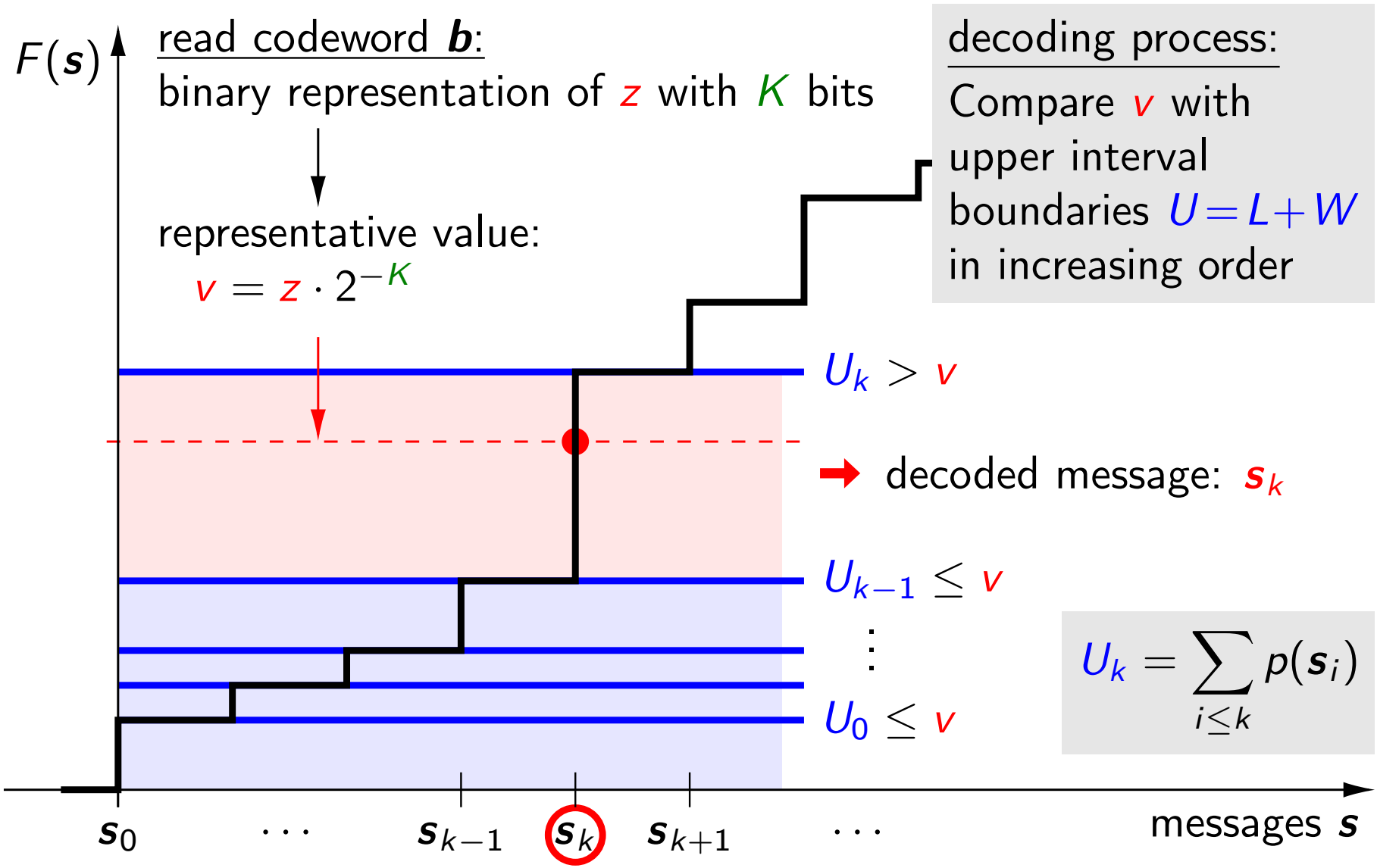
$$K_k = \lceil -\log_2 W_k \rceil \tag{3}$$

3. Determine representative integer z_k

$$z_k = \lceil L_k \cdot 2^{K_k} \rceil \tag{4}$$

4. Codeword $\boldsymbol{b}_k$: Binary representation of z_k with K_k bits

Shannon-Fano-Elias Decoding: Illustration



Shannon-Fano-Elias Decoding: Summary

Decoding of a Symbol Sequence

- Given: Ordered set of symbol sequences $\{\boldsymbol{s}_k\}$ with associated pmf $\{p_k\}$

1. Read codeword $\boldsymbol{b}$:
   Codeword $\boldsymbol{b}$ has K bits and represents the binary value of the integer z
2. Determine the representattive value v according to
$$v = z \cdot 2^{-K} \tag{5}$$
3. Initilization: Set index $k = 0$ and upper interval boundary $U_0 = p_0$
4. Compare v with U_k
   - If $v < U_k$
     - ➡ Output decoded symbol sequence $\boldsymbol{s}_k$
     - ➡ Terminate decoding
   - Otherwise ($v \geq U_k$)
     - ➡ Set $k = k + 1$ and $U_k = U_{k-1} + p_k$
     - ➡ Goto step 4

Example for a Shannon-Fano-Elias Code

Blocks of three Symbols for a Binary IID Source

➡ **Binary iid source with alphabet $\mathcal{A} = \{a, b\}$ and pmf $p = \{0.8, 0.2\}$**

joint pmf		*intervals*		*number*		*codeword*
s_k	p_k	W_k	L_k	K_k	z_k	$\boldsymbol{b}_k$
aaa	0.512	0.512	0.000	1	0	0
aab	0.128	0.128	0.512	3	5	101
aba	0.128	0.128	0.640	3	6	110
abb	0.032	0.032	0.768	5	25	11001
baa	0.128	0.128	0.800	3	7	111
bab	0.032	0.032	0.928	5	30	11110
bba	0.032	0.032	0.960	5	31	11111
bbb	0.008	0.008	0.992	7	127	1111111

average codeword length: $\bar{\ell} = 0.733$

block Huffman code: $\bar{\ell} = 0.728$

$$W_k = p_k$$

$$L_k = \sum_{i<k} p_i$$

$$K_k = \lceil -\log_2 W_k \rceil$$

$$z_k = \lceil L_k \cdot 2^{K_k} \rceil$$

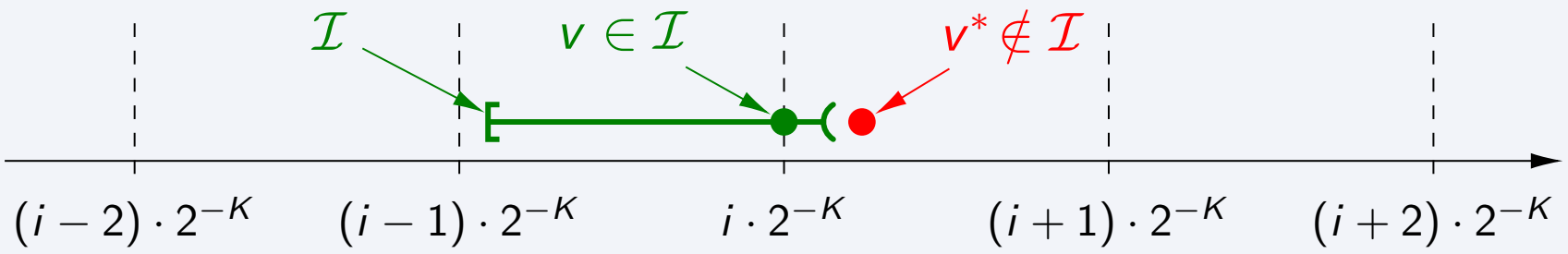
$\boldsymbol{b}_k$: z_k with K_k bits

➡ Worse than block Huffman code for same block size ($N = 3$)

➡ **Code is not prefix-free !** ➡ Can be a problem (depends on application) !

Why Is The Code Not Prefix-Free ?

Effect of Codeword Concatenation



- Encoder transmits codeword $\boldsymbol{b} = \{b_0, b_1, b_2, \cdots, b_{K-1}\}$ of K bits, signaling the binary fraction $v \in \mathcal{I}$

$$v = (0.b_0 b_1 b_2 \cdots b_{K-1})_{\mathrm{b}}$$

- Decoder sees a modified binary fraction v^* given by

$$v^* = (0.b_0 b_1 b_2 \cdots b_{K-1} b_K b_{K+1} b_{K+2} \cdots)_{\mathrm{b}}$$

  where $\{b_K b_{K+1} b_{K+2} \cdots\}$ are the bits of following codewords

➡ Depending on the location of v inside the interval $\mathcal{I}$ and the values of the following bits, v^* can lay outside the interval $\mathcal{I}$

Prefix-Free Shannon-Fano-Elias Code

Prefix Code Construction

- Ensure that binary fraction v^* (seen by decoder) lies inside interval
  - Worst case: All following bits are equal to 1

$$v^* = v + \sum_{i=K+1}^{\infty} 2^{-i} < L + W \tag{6}$$

  - Since the sum in above equation is less than 2^{-K}, we require

$$v^* < v + 2^{-K} \leq L + W \tag{7}$$

  - Question: How many bits K do we need for representing v according to

$$v = \lceil L \cdot 2^K \rceil \cdot 2^{-K}$$

- Have to choose K so that the following inequality is fulfilled

$$v + 2^{-K} = \lceil L \cdot 2^K \rceil \cdot 2^{-K} + 2^{-K} \leq L + W \tag{8}$$

Prefix-Free Shannon-Fano-Elias Code

Prefix Code Construction (continued)

- Since $\lceil x \rceil < x + 1$, the inequality

$$\left\lceil L \cdot 2^K \right\rceil \cdot 2^{-K} + 2^{-K} \leq L + W$$

is always fulfilled if we have

$$\begin{aligned} \left(L \cdot 2^K + 1\right) \cdot 2^{-K} + 2^{-K} &\leq L + W \\ L + 2 \cdot 2^{-K} &\leq L + W \\ 2^{1-K} &\leq W \\ 1 - K &\leq \log_2 W \\ K &\geq 1 - \log_2 W \end{aligned} \tag{9}$$

➡ Unique decodability is guaranteed, if we choose

$$K = \left\lceil 1 - \log_2 W \right\rceil = \left\lceil -\log_2 W \right\rceil + 1 \tag{10}$$

➡ One additional bit per codeword (compared to non-prefix-free version)

Example for a Prefix-Free Shannon-Fano-Elias Code

Repeated Example: Blocks of three Symbols for a Binary IID Source

➡ **Binary iid source with alphabet $\mathcal{A} = \{a, b\}$ and pmf $p = \{0.8, 0.2\}$**

joint pmf		*intervals*		*number*		*codewords*
s_k	p_k	W_k	L_k	K_k	z_k	$\boldsymbol{b}_k$
aaa	0.512	0.512	0.000	2	0	00
aab	0.128	0.128	0.512	4	9	1001
aba	0.128	0.128	0.640	4	11	1011
abb	0.032	0.032	0.768	6	50	110010
baa	0.128	0.128	0.800	4	13	1101
bab	0.032	0.032	0.928	6	60	111100
bba	0.032	0.032	0.960	6	62	111110
bbb	0.008	0.008	0.992	8	254	11111110

average codeword length: $\bar{\ell} = 1.067$

block Huffman code: $\bar{\ell} = 0.728$

$$W_k = p_k$$

$$L_k = \sum_{i<k} p_i$$

$$K_k = \lceil 1 - \log_2 W_k \rceil$$

$$z_k = \lceil L_k \cdot 2^{K_k} \rceil$$

$\boldsymbol{b}_k$: z_k with K_k bits

➡ Additional bit ensures that code becomes a **prefix code**

➡ Worse than block Huffman code (several redundant bits)

Efficiency of Shannon-Fano-Elias Codes

Average Codeword Length

- Average codeword length $\bar{\ell}$ per symbol

$$\bar{\ell} = \frac{\mathrm{E}\{\, K(\boldsymbol{S}) \,\}}{N} = \frac{\mathrm{E}\{\, \lceil A - \log_2 p_N(\boldsymbol{S}) \rceil \,\}}{N} \quad \text{with} \quad A = \begin{cases} 1 & : \text{ prefix-free} \\ 0 & : \text{ otherwise} \end{cases} \tag{11}$$

Bounds on Average Codeword Length

- Using inequalities $x \le \lceil x \rceil$ and $\lceil x \rceil < x + 1$, we obtain

$$\frac{\mathrm{E}\{\, -\log_2 p_N(\boldsymbol{S}) \,\}}{N} + \frac{A}{N} \le \bar{\ell} < \frac{\mathrm{E}\{\, -\log_2 p_N(\boldsymbol{S}) \,\}}{N} + \frac{1+A}{N}$$

$$\frac{H_N(\boldsymbol{S})}{N} + \frac{A}{N} \le \bar{\ell} < \frac{H_N(\boldsymbol{S})}{N} + \frac{1+A}{N} \tag{12}$$

➡ Non-prefix-free version ($A = 0$): Same bounds as for block Huffman coding

➡ Both versions: Close to entropy rate for $N \gg 1$ (for typical sources)

Shannon-Fano-Elias Coding: Intermediate Results

Shannon-Fano-Elias Code

- Special block code (for given number of symbols N)
- Worse than block Huffman code of same size N
- Still close to entropy bound H_N/N for $N \gg 1$
- **No need to store codeword table !**
- **Have to store N-th order pmf (or N-th order cdf) !**

What is the advantage ?

Iterative Coding

- Can define a suitable order for sequences of N symbols

➡ Probability intervals are nested

➡ Iterative calculation of interval boundaries

➡ Iterative codeword construction

Sorting of Symbol Sequences

Lexicographical Order

- Define any order for single symbols s_k (sorted symbol alphabet)
- Consider two sequences of N symbols

$$\boldsymbol{s}_a = \{s_0^a, s_1^a, \cdots, s_{N-1}^a\} \qquad \text{and} \qquad \boldsymbol{s}_b = \{s_0^b, s_1^b, \cdots, s_{N-1}^b\}$$

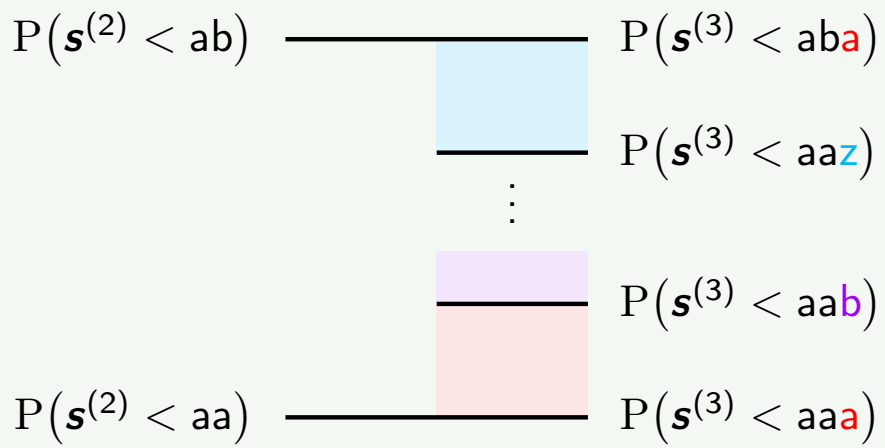
➡ Lexicographical order for sequences of N symbols

$$\boldsymbol{s}_a < \boldsymbol{s}_b \iff \exists n < N : \left(\forall k < n : \; s_k^a = s_k^b\right) \wedge \left(s_n^a < s_n^b\right) \tag{13}$$

Example: Alphabetical Order

$\mathrm{a} < \mathrm{b} < \cdots < \mathrm{z} \implies$

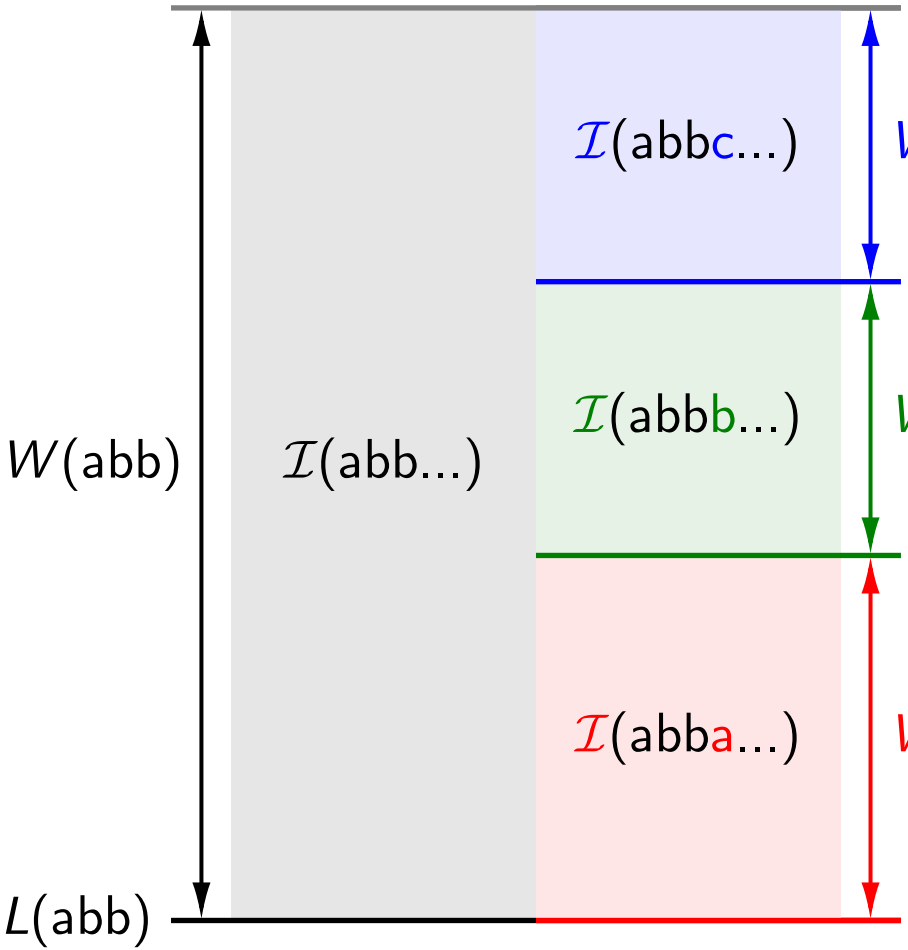
$\ldots\,\mathrm{aaaa}$
$\ldots\,\mathrm{aaab}$
$\vdots$
$\ldots\,\mathrm{aaaz}$
$\ldots\,\mathrm{aaba}$
$\ldots\,\mathrm{aabb}$
$\vdots$



Iterative Interval Refinement: Illustration

Example: 3-symbol alphabet $\mathcal{A} = \{a, b, c\}$

$$c(z\,|\,..) = \sum_{\forall x<z} p(x\,|\,...)$$



$$W(\text{abbc}) = W(\text{abb}) \cdot p(\text{c}\,|\,\text{abb})$$

$$L(\text{abbc}) = L(\text{abb}) + W(\text{abb}) \cdot c(\text{c}\,|\,\text{abb})$$

$$W(\text{abbb}) = W(\text{abb}) \cdot p(\text{b}\,|\,\text{abb})$$

$$L(\text{abbb}) = L(\text{abb}) + W(\text{abb}) \cdot c(\text{b}\,|\,\text{abb})$$

$$W(\text{abba}) = W(\text{abb}) \cdot p(\text{a}\,|\,\text{abb})$$

$$L(\text{abba}) = L(\text{abb}) + W(\text{abb}) \cdot c(\text{a}\,|\,\text{abb})$$

Important: Probability intervals are nested!

Iterative Interval Refinement: Interval Width

Interval Refinement

- Consider sub-sequences $\boldsymbol{s}^{(n)} = \{s_0, s_1, \cdots, s_{n-1}\}$ with $n < N$

➡ Determine interval $\mathcal{I}_{n+1} = [L_{n+1}, L_{n+1} + W_{n+1})$ for $\boldsymbol{s}^{(n+1)} = \{\boldsymbol{s}^{(n)}, s_n\}$ based on interval $\mathcal{I}_n = [L_n, L_n + W_n)$ for prefix sequence $\boldsymbol{s}^{(n)}$

Refinement of Interval Width

- Interval width W_{n+1} for sub-sequence $\boldsymbol{s}^{(n+1)} = \{\boldsymbol{s}^{(n)}, s_n\}$

$$\begin{aligned} W_{n+1} &= \mathrm{P}\Big(\boldsymbol{S}^{(n+1)} = \boldsymbol{s}^{(n+1)}\Big) \\ &= \mathrm{P}\Big(\boldsymbol{S}^{(n)} = \boldsymbol{s}^{(n)},\, S_n = s_n\Big) \\ &= \mathrm{P}\Big(\boldsymbol{S}^{(n)} = \boldsymbol{s}^{(n)}\Big) \cdot \mathrm{P}\Big(S_n = s_n \,|\, \boldsymbol{S}^{(n)} = \boldsymbol{s}^{(n)}\Big) \end{aligned} \tag{14}$$

➡ **Iteration rule for interval width**

$$W_{n+1} = W_n \cdot p(s_n \,|\, s_{n-1}, s_{n-2}, \cdots, s_0) \tag{15}$$

Iterative Interval Refinement: Lower Interval Boundary

Refinement of Lower Interval Boundary

- Lower interval boundary L_{n+1} for sub-sequence $\boldsymbol{s}^{(n+1)} = \{\boldsymbol{s}^{(n)}, s_n\}$

$$\begin{aligned} L_{n+1} &= \mathrm{P}\Big(\boldsymbol{S}^{(n+1)} < \boldsymbol{s}^{(n+1)}\Big) \\ &= \mathrm{P}\Big(\boldsymbol{S}^{(n)} < \boldsymbol{s}^{(n)}\Big) + \mathrm{P}\Big(\boldsymbol{S}^{(n)} = \boldsymbol{s}^{(n)},\ S_n < s_n\Big) \\ &= \mathrm{P}\Big(\boldsymbol{S}^{(n)} < \boldsymbol{s}^{(n)}\Big) + \mathrm{P}\Big(\boldsymbol{S}^{(n)} = \boldsymbol{s}^{(n)}\Big) \cdot \mathrm{P}\Big(S_n < s_n \,|\, \boldsymbol{S}^{(n)} = \boldsymbol{s}^{(n)}\Big) \end{aligned} \tag{16}$$

- Define **modified cmf** $c(.)$, which excludes current symbol

$$c(s_n \,|\, s_{n-1}, \cdots, s_0) = \mathrm{P}\Big(S_n < s_n \,|\, \boldsymbol{S}^{(n)} = \boldsymbol{s}^{(n)}\Big) = \sum_{\forall a < s_n} p(a \,|\, s_{n-1}, \cdots, s_0) \tag{17}$$

➡ **Iteration rule for lower interval boundary**

$$L_{n+1} = L_n + W_n \cdot c(s_n \,|\, s_{n-1}, s_{n-2}, \cdots, s_0) \tag{18}$$

Nested Probability Intervals: Verification

Intervals are Nested

- Lower interval boundary

$$L_{n+1} = L_n + W_n \cdot \mathrm{P}\Big(S_n < s_n \,|\, \boldsymbol{S}^{(n)} = \boldsymbol{s}^{(n)}\Big) \geq L_n \tag{19}$$

- Upper interval boundary

$$\begin{aligned} L_{n+1} + W_{n+1} &= L_n + W_n \cdot \mathrm{P}\Big(S_n < s_n \,|\, \boldsymbol{S}^{(n)} = \boldsymbol{s}^{(n)}\Big) \\ &\quad + W_n \cdot \mathrm{P}\Big(S_n = s_n \,|\, \boldsymbol{S}^{(n)} = \boldsymbol{s}^{(n)}\Big) \\ &= L_n + W_n \cdot \mathrm{P}\Big(S_n \leq s_n \,|\, \boldsymbol{S}^{(n)} = \boldsymbol{s}^{(n)}\Big) \\ &= L_n + W_n - W_n \cdot \mathrm{P}\Big(S_n > s_n \,|\, \boldsymbol{S}^{(n)} = \boldsymbol{s}^{(n)}\Big) \\ &\leq L_n + W_n \end{aligned} \tag{20}$$

➡ Interval for $\boldsymbol{s}^{(n+1)} = \{\boldsymbol{s}^{(n)}, s_n\}$ is fully included in interval for $\boldsymbol{s}^{(n)}$

Iterative Interval Refinement

Iterative Algorithm for Calculation of Interval Boundaries

- Initialization:

$$W_0 = 1 \tag{21}$$

$$L_0 = 0 \tag{22}$$

- Iteration Step:

$$W_{n+1} = W_n \cdot p(s_n \,|\, s_{n-1}, \cdots, s_0) \tag{23}$$

$$L_{n+1} = L_n + W_n \cdot c(s_n \,|\, s_{n-1}, \cdots, s_0) \tag{24}$$

Advantage of Interval Refinement ?

- Derived iteration approach:
  - Conditional pmf $p(s_n \,|\, s_{n-1}, \cdots)$ instead of joint pmf $p(s_0, \cdots, s_{N-1})$

  ➡ Need to store same amount of data

➡ But: Conditional pmfs can be well approximated using simple models

- IID model: $p(s_n \,|\, s_{n-1}, \cdots, s_0) = p(s_n)$
- Markov model: $p(s_n \,|\, s_{n-1}, \cdots, s_0) = p(s_n \,|\, s_{n-1})$

Practical Iterative Interval Refinement

Simplified Iteration

- IID Model:

$$W_{n+1} = W_n \cdot p(s_n) \tag{25}$$

$$L_{n+1} = L_n + W_n \cdot c(s_n) \tag{26}$$

- Markov Model:

$$W_{n+1} = W_n \cdot p(s_n \,|\, s_{n-1}) \tag{27}$$

$$L_{n+1} = L_n + W_n \cdot c(s_n \,|\, s_{n-1}) \tag{28}$$

- Many other simple models possible: Condition $= f(s_{n-1}, \cdots)$

Other Aspects

- Switching between symbol alphabets possible
  - ➡ Suitable for complicated syntax (as for prefix codes)
- Adaptation of probability models
  - Probabilities can be estimated during encoding and decoding
  - ➡ Elegant way to deal with instationary or unkown sources

Iterative Encoding Algorithm

Iterative Shannon-Fano-Elias Encoding

1. Given is a sequence $\boldsymbol{s} = \{s_0, s_1, s_2, \cdots, s_{N-1}\}$ of N symbols
2. Initialization of probability interval

$$W_0 = 1 \qquad \text{and} \qquad L_0 = 0 \tag{29}$$

3. Determine probability interval: For each $n = 0, 1, \cdots, N-1$, do

$$W_{n+1} = W_n \cdot p(\, s_n \,|\, \cdots) \tag{30}$$

$$L_{n+1} = L_n + W_n \cdot c(\, s_n \,|\, \cdots) \tag{31}$$

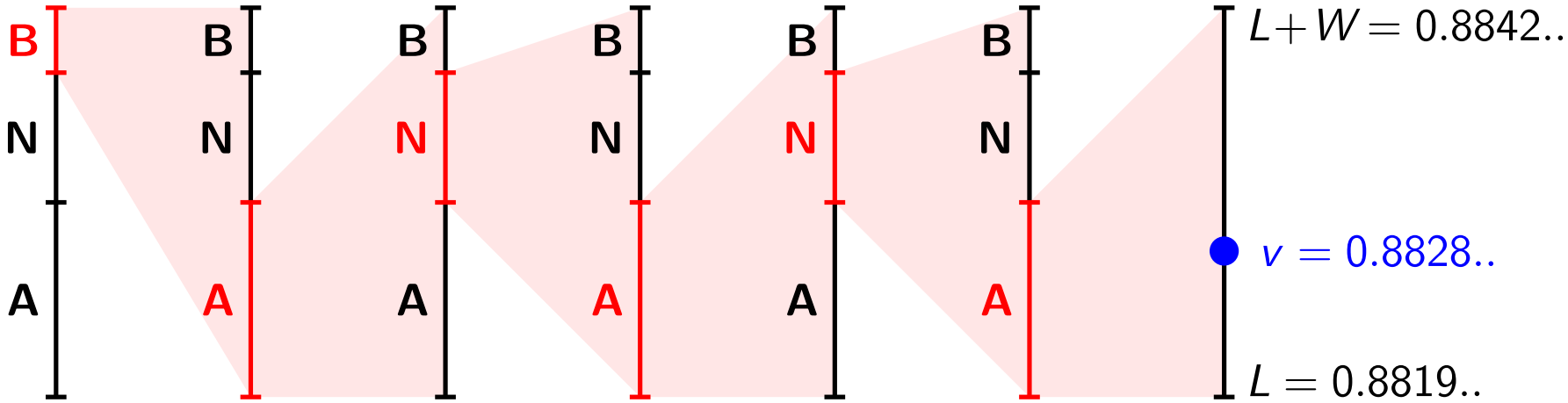
4. Determine codeword length and codeword value

$$K = \lceil -\log_2 W_N \rceil \qquad \text{(for prefix code: } K \rightarrow K+1\text{)} \tag{32}$$

$$z = \lceil L_N \cdot 2^K \rceil \tag{33}$$

5. Transmit codeword $\boldsymbol{b}(\boldsymbol{s})$: Binary representation of z with K bits

Iterative Encoding Example: IID Source



a	$p(a)$	$c(a)$
A	$\frac{1}{2}$	0
N	$\frac{1}{3}$	$\frac{1}{2}$
B	$\frac{1}{6}$	$\frac{5}{6}$

	init	**B**	**A**	**N**	**A**	**N**	**A**
W_n	1	$\frac{1}{6}$	$\frac{1}{12}$	$\frac{1}{36}$	$\frac{1}{72}$	$\frac{1}{216}$	$\frac{1}{432}$
L_n	0	$\frac{5}{6}$	$\frac{5}{6}$	$\frac{21}{24}$	$\frac{21}{24}$	$\frac{127}{144}$	$\frac{127}{144}$

$$W_{n+1} = W_n \, p(s_n)$$

$$L_{n+1} = L_n + W_n \, c(s_n)$$

$$K = \lceil -\log_2 W \rceil = \lceil \log_2 432 \rceil = 9$$

$$z = \left\lceil L \cdot 2^K \right\rceil = \left\lceil \frac{127}{144} \cdot 512 \right\rceil = 452 \qquad \left(v = \frac{452}{512} \right)$$

$$\boldsymbol{b} = \text{"}\mathbf{111000100}\text{"} \qquad (z = 452 \text{ with } K = 9 \text{ bits})$$

Iterative Decoding Algorithm

Iterative Shannon-Fano-Elias Decoding

1. Given:
   - Bitstream $\boldsymbol{b} = \{b_0, b_1, \cdots, b_{K^*-1}\}$ of $K^* \geq K$ bits
   - Number N of symbols to be decoded
2. Determine interval representative: $v = (0.b_0 b_1 \cdots b_{K^*-1})_{\mathrm{b}} = z \cdot 2^{-K^*}$
3. Initialization of probability interval: $W_0 = 1$ and $L_0 = 0$
4. Iterative decoding: For each $n = 0, 1, \cdots, N-1$, do
   a. Calculate symbol intervals: For each $a \in \mathcal{A}_n$, calculate

$$W_{n+1}(a) = W_n \cdot p(\, a \,|\, \cdots) \tag{34}$$

$$L_{n+1}(a) = L_n + W_n \cdot c(\, a \,|\, \cdots) \tag{35}$$

   b. Compare v with upper interval boundaries in increasing symbol order and output symbol s_n with

$$L_{n+1}(s_n) \leq v < U_{n+1}(s_n) = L_{n+1}(s_n) + W_{n+1}(s_n) \tag{36}$$

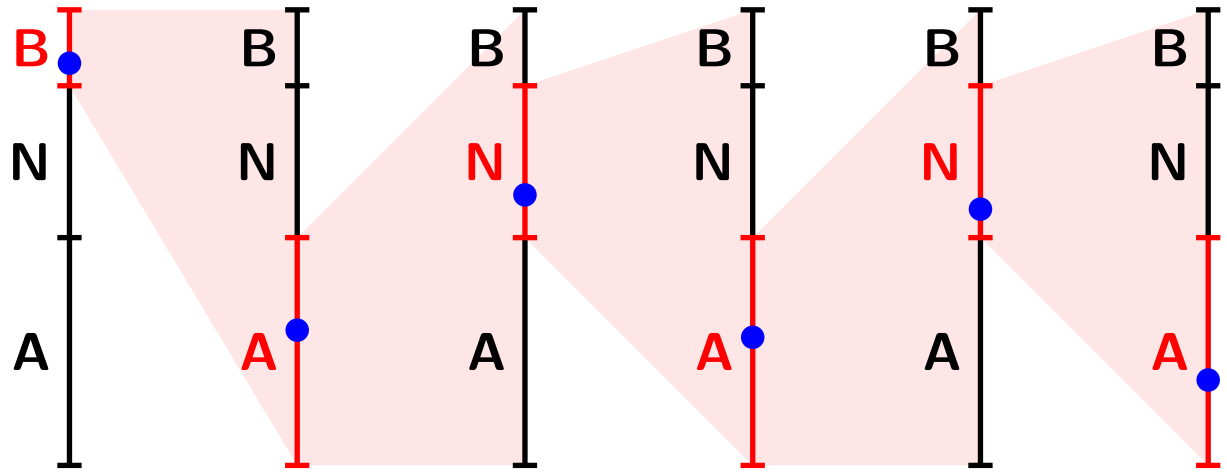
   c. Update interval boundary and interval width

$$W_{n+1} = W_{n+1}(s_n) \quad \text{and} \quad L_{n+1} = L_{n+1}(s_n) \tag{37}$$

Iterative Decoding Example: IID Source

$$L_{n+1}(a) = L_n + W_n\, c(a)$$

$$W_{n+1}(a) = W_n\, p(a)$$



(L_n, W_n)	$0,1$	$\frac{5}{6},\frac{1}{6}$	$\frac{5}{6},\frac{1}{12}$	$\frac{21}{24},\frac{1}{36}$	$\frac{21}{24},\frac{1}{72}$	$\frac{127}{144},\frac{1}{216}$
$(L_{n+1}, W_{n+1})\,(\mathrm{A})$	$0,\frac{1}{2}$	$\frac{5}{6},\frac{1}{12}$	$\frac{5}{6},\frac{1}{24}$	$\frac{21}{24},\frac{1}{72}$	$\frac{21}{24},\frac{1}{144}$	$\frac{127}{144},\frac{1}{432}$
$(L_{n+1}, W_{n+1})\,(\mathrm{N})$	$\frac{1}{2},\frac{1}{3}$	$\frac{11}{12},\frac{1}{18}$	$\frac{21}{24},\frac{1}{36}$	$\frac{8}{9},\frac{1}{108}$	$\frac{127}{144},\frac{1}{216}$	$\frac{191}{216},\frac{1}{648}$
$(L_{n+1}, W_{n+1})\,(\mathrm{B})$	$\frac{5}{6},\frac{1}{6}$	$\frac{35}{36},\frac{1}{36}$	$\frac{65}{72},\frac{1}{72}$	$\frac{97}{108},\frac{1}{216}$	$\frac{383}{432},\frac{1}{432}$	$\frac{287}{324},\frac{1}{1296}$
symbol s_n	B	A	N	A	N	A

$v = \frac{452}{512}$ $\qquad \boldsymbol{b} = \text{"}111000100\text{"} \quad \Longrightarrow \quad \boldsymbol{s} = \text{"BANANA"}$

Arithmetic Coding

Iterative Shannon-Fano-Elias Coding

➡ **Iterative Encoding and Decoding**

- Iterative interval refinement
- Simple codeword construction

➡ **Precision requirements and delay for larger N**

- Require extremely high precision for W_n, L_n, and v
- Encoder: Complete codeword written at end of encoding
- Decoder: Complete codeword read at start of decoding

Arithmetic Coding

- Fixed-precision approximation of Shannon-Fano-Elias coding

➡ Represent pmf(s) with fixed-precision integers

➡ Represent interval width with fixed-precision integers

➡ Output bits as soon as possible

Quantization of PMF(s)

Fixed-Precision Approximation of Pmf(s)

- Choose number of bits V for representing probability masses
- Represent probability masses $p(a)$ by **V-bit integers** $p_V(a)$

$$p(a) = p_V(a) \cdot 2^{-V} \tag{38}$$

➡ Resulting modified cmf $c(a)$ can also be represented by **V-bit integers** $c_V(a)$

$$c(a) = \sum_{\forall b<a} p(b) = \left(\sum_{\forall b<a} p_V(b) \right) \cdot 2^{-V} = c_V(a) \cdot 2^{-V} \tag{39}$$

Requirements on Pmf Approximation

- **Probability masses must be non-zero** and **pmf must be valid**

$$\forall a : p_V(a) > 0 \qquad \text{and} \qquad \sum_{\forall a} p_V(a) \leq 2^V \tag{40}$$

Quantization of Interval Width

Fixed-Precision Approximation of Interval Width

- Represent interval width W_n by U-**bit integer** A_n and **counter** $z_n \geq U$

$$W_n = A_n \cdot 2^{-z_n} \tag{41}$$

➡ Use maximum possible precision for A_n

➡ Restrict A_n according to

$$2^{U-1} \leq A_n < 2^U \tag{42}$$

➡ Use following initialization

$$A_0 = 2^U - 1, \quad z_0 = U \qquad \Longleftrightarrow \qquad W_0 = 1 - 2^{-U} \tag{43}$$

➡ Binary representation of interval width $W_n = A_n \cdot 2^{-z_n}$

$$W_n = 0.\overbrace{\underbrace{00000\cdots0}_{z_n - U \text{ bits}}\underbrace{1xx\cdots x}_{U \text{ bits}}}^{z_n \text{ bits}}000\cdots$$

Rounding in Interval Refinement

Conventional Refinement of Interval Width

- Refinement of interval width

$$W_{n+1} = W_n \cdot p(s_n) \tag{44}$$

$$A_{n+1} \cdot 2^{-z_{n+1}} = \underbrace{\big(A_n \cdot p_V(s_n)\big)}_{(U+V)\text{-bit integer}} \cdot 2^{-(z_n+V)} \tag{45}$$

- In general: $W_n \cdot p(s_n)$ cannot be represented using a U-bit integer

➡ **What can we do?**

Requirement for Unique Decodability

- Code remains uniquely decodable if we ensure

$$0 < W_{n+1} \leq W_n \cdot p(s_n) \tag{46}$$

➡ **Solution:** Rounding down of $W_n \cdot p(s_n)$ in each iteration so that W_{n+1} can represented using $A_{n+1} \cdot 2^{-z_{n+1}}$ with $2^{U-1} \leq A_{n+1} < 2^U$

Refinement of Interval Width

Product of Interval Width and Pmf Entry

- Binary representations of $W_n = A_n \cdot 2^{-z_n}$ and $p(s_n) = p_V(s_n) \cdot 2^{-V}$

$$W_n = 0.\overbrace{00000\cdots0}^{z_n-U \text{ bits}}\overbrace{1xx\cdots x}^{U \text{ bits}}000\cdots$$
$$p(s_n) = 0.\underbrace{xxx\cdots x}_{V \text{ bits}}000\cdots$$

- Interval refinement $W_n \cdot p(s_n) \mapsto W_{n+1}$

$$W_n \cdot p(s_n) = 0.\underbrace{00000\cdots0}_{z_n-U \text{ bits}}\overbrace{00\cdots0}^{\Delta z \text{ bits}}\overbrace{1x\cdots x}^{U \text{ bits}}\overbrace{xx\cdots x}^{V-\Delta z \text{ bits}}000\cdots$$
$$\Big\downarrow \qquad \underbrace{\qquad\qquad\qquad\qquad}_{U+V \text{ bits: } A_n \cdot p_V(s_n)}$$
$$W_{n+1} = 0.\underbrace{00000\cdots0}_{z_{n+1}-U \text{ bits}}\overbrace{00\cdots0}^{z_n-U \text{ bits}}\overbrace{1x\cdots x}^{\Delta z \text{ bits}}\underbrace{1x\cdots x}_{A_{n+1}}\overbrace{00\cdots0}^{U \text{ bits}}000\cdots$$

Refinement of Interval Width

Arithmetic Operations for Interval Width Update

$$W_n \cdot p(s_n) = 0.\underbrace{00000\cdots0}_{z_n - U \text{ bits}} \overbrace{\underbrace{00\cdots0}_{\Delta z \text{ bits}} \underbrace{1x\cdots x}_{U \text{ bits}} \underbrace{xx\cdots x}_{V-\Delta z \text{ bits}}}^{U+V \text{ bits: } A_n \cdot p_V(s_n)} 000\cdots$$

$$\downarrow$$

$$W_{n+1} = 0.\underbrace{00000\cdots0}_{z_{n+1} - U \text{ bits}} \underbrace{00\cdots0}_{\Delta z \text{ bits}} \underbrace{1x\cdots x}_{U \text{ bits}} 00\cdots0\ 000\cdots$$

Update of interval width

- Determine number Δz of leading zeros in $(U+V)$-bit integer $(A_n \cdot p_V(s_n))$
- Update interval width according to

$$A_{n+1} = \big(A_n \cdot p_V(s_n)\big) \gg (V - \Delta z) \qquad (\text{“}\gg\text{”} = \text{bit shift to the right})$$
$$z_{n+1} = z_n + \Delta z$$

➡ **Ensures unique decodability:** $0 < W_{n+1} \leq W_n \cdot p(s_n)$

Effect on Binary Representations of Lower Interval Boundary

Product of Interval Width and Modified Cmf Entry

- Remember:

$$L_{n+1} = L_n + W_n \cdot c(s_n)$$

- Binary representations of $W_n = A_n \cdot 2^{-z_n}$ and $c(s_n) = c_V(s_n) \cdot 2^{-V}$

$$W_n = 0.\overbrace{\underbrace{00000\cdots0}_{z_n - U \text{ bits}}\underbrace{1xx\cdots x}_{U \text{ bits}}}^{z_n \text{ bits}}000\cdots$$

$$c(s_n) = 0.\underbrace{xxx\cdots x}_{V \text{ bits}}000\cdots$$

- Binary representation of product $W_n \cdot c(s_n)$

$$W_n \cdot c(s_n) = 0.\overbrace{\underbrace{00000\cdots0}_{z_n - U \text{ bits}}\underbrace{xxx\cdots x}_{U+V \text{ bits}}}^{z_n + V \text{ bits}}000\cdots$$

Effect on Binary Representation of Lower Interval Boundary

Update of Lower Interval Boundary

$$W_n \cdot c(s_n) = 0.\overbrace{\underbrace{00000\cdots0}_{z_n - U \text{ bits}}\underbrace{xxx\cdots x}_{U+V \text{ bits}}}^{z_n+V \text{ bits}}000\cdots$$

- What is the effect on lower interval boundary?

$$L_n = 0.\overbrace{\underbrace{aaaaa\cdots a}_{\substack{z_n - c_n - U \\ \text{settled bits}}}\underbrace{0111111\cdots1}_{\substack{c_n \\ \text{outstanding bits}}}}^{z_n - U \text{ bits}}\underbrace{xxxxx\cdots x}_{\substack{U+V \\ \text{active bits}}}\underbrace{00000\cdots}_{\text{trailing bits}}$$

➡ **Trailing bits**: Equal to 0, but maybe changed later

➡ **Active bits**: Directly modified by the update $L_{n+1} = L_n + W_n\, c(s_n)$

➡ **Outstanding bits**: May be modified by a carry from the active bits

➡ **Settled bits**: Not modified in any following interval update

Arithmetic Coding: Lower Interval Boundary & Output

Representation of Lower Interval Boundary

$$L_n = 0.\overbrace{\underbrace{aaaaa\cdots a}_{z_n - c_n - U} \quad \underbrace{0111111\cdots 1}_{c_n}}^{z_n - U \text{ bits}} \quad \underbrace{xxxxx\cdots x}_{U+V} \quad \underbrace{00000\cdots}$$

settled bits | *outstanding bits* | *active bits* | *trailing bits*

- **Active bits**:
  - ➡ $(U+V)$-bit integer B_n
  - ➡ Intermediate value $B_n + A_n c_V(s_n)$ requires $(U+V+1)$-bit integer
- **Outstanding bits**:
  - ➡ Counter c_n (trailing $c_n - 1$ bits are equal to 1)
- **Settled bits**:
  - ➡ Output as soon as they become settled

Arithmetic Coding: Interval Refinement

Update of Probability Interval

$$
\begin{aligned}
W_n &= 0.\overbrace{000000000000\cdots0}^{z_n - U}\ \overbrace{1xx\cdots x}^{A_n\ (U)}\ 000000000\cdots \\
L_n &= 0.\underbrace{aaaaa\cdots a}_{z_n - c_n - U}\ \underbrace{011\cdots1}_{c_n}\ \underbrace{xxx\cdots xxxxxx}_{B_n\ (U+V)}\ 00000\cdots
\end{aligned}
$$

$$
\begin{aligned}
W_{n+1} &= 0.\overbrace{000000000000\cdots0}^{z_n - U}\ \overbrace{0\cdots0}^{\Delta z}\ \overbrace{1xx\cdots x}^{A_{n+1}\ (U)}\ 000000000\cdots \\
L_{n+1} &= 0.\underbrace{aaaaa\cdots a}_{z_n - c_n - U}\ \underbrace{xxxxxxxx\cdots xxx}_{c_n + \Delta z}\ \underbrace{xxx\cdots xxxxxx}_{B_{n+1}\ (U+V)}\ 00000\cdots
\end{aligned}
$$

Interval update

$$
\begin{aligned}
\Delta z &= \text{Number of trailing zeros in } (U+V)\text{-bit integer } \big(A_n \cdot p_V(s_n)\big) \\
\text{mask} &= \big(1 \ll (U+V-\Delta z)\big) - 1 \\
A_{n+1} &= \big(A_n \cdot p_V(s_n)\big) \gg (V - \Delta z) \\
B_{n+1} &= \big(\big(B_n + A_n \cdot c_V(s_n)\big)\ \&\ \text{mask}\big) \ll \Delta z
\end{aligned}
$$

Arithmetic Coding: Output of Settled Bits

Output of Bits

$$L_{n+1} = 0.\underbrace{aaaaa\cdots a}_{z_n - c_n - U}\ \underbrace{xxxxxxxx\cdots xxx}_{c_n + \Delta z}\ \underbrace{xxx\cdots xxxxxx}_{B_{n+1}\ (U+V)}\ 00000\cdots$$

- Investigate modified $(c_n + \Delta z)$ bits

➡ Update counter c_{n+1} and output new settled bits

Total Number of Bits for Arithmetic Codeword

- Total number of bits to output (note: $W_N = A_N\, 2^{-z_n}$)

$$K = \lceil -\log_2 W_N \rceil = z_n - \lfloor \log_2 A_N \rfloor = z_N - U + 1 \qquad (47)$$

  ➡ Prefix-free version: $K = z_N - U + 2$

- Note: $z_N - U$ is the sum of settled and outstanding bits

  ➡ Output c_N outstanding bits

  ➡ Output one bit of B_N (prefix-free: two bits)

Termination of Arithmetic Codeword

Required Output

- Required output after last symbol was coded
  - c_N outstanding bits
  - most significant bit of B_N (prefix-free: two most significant bits)
- Note: Lower boundary must be rounded up to next multiple of 2^{-K}

Codeword Termination

- Set $n = 1$ (non-prefix free) or $n = 2$ (prefix-free)
- If the any of the last $(U + V - n)$ bits in B_N is equal to 1, do
  - Set $B_N = B_N + (1 \ll (U + V - n))$ (rounding up)
  - If $B_N \geq (1 \ll (U + V))$ (carry condition)
    - Invert outstanding bits, output new settled bits, set $c_N = 1$
    - Remove carry: $B_N = B_N - (1 \ll (U + V))$
- Output outstanding bits (one '0' and $(c_N - 1)$ times '1')
- Output n most significant bits of B_N

Overview of Arithmetic Encoding Process

Arithmetic Encoding

1. Initialization: $A_0 = 2^U - 1,\ B_0 = 0,\ c_0 = 0$
2. Iteration: For $n = 0, 1, \cdots, N-1$, do
   a. Calculate:
   $$A^* = A_n \cdot p_V(s_n) \qquad (U+V \text{ bits})$$
   $$B^* = B_n + A_n \cdot c_V(s_n) \qquad (U+V+1 \text{ bits})$$
   b. Determine number Δz of trailing zeros in $(U+V)$-bit integer A^*
   c. Update:
   $$\text{mask} = \big(1 \ll (U+V-\Delta z)\big) - 1$$
   $$A_{n+1} = A^* \gg (V - \Delta z)$$
   $$B_{n+1} = \big(B^* \,\&\, \text{mask}\big) \ll \Delta z$$
   d. Determine outstanding bits counter c_{n+1} (based on c_n and B^*)
   e. Output new settled bits ($c_n + \Delta z - c_{n+1}$ bits)
3. Termination:
   - Round up B_N
   - Output c_N outstanding bits + one/two most significant bit(s) of B_N

Arithmetic Coding Example

Example: Preparation

- Coding example
  - IID source with symbol alphabet $\{\mathrm{A}, \mathrm{N}, \mathrm{B}\}$
  - Pmf is given by $\{\, 1/2,\, 1/3,\, 1/6 \,\}$
  - Consider arithmetic coding with $V = 4$ and $U = 4$
  - Symbol sequence "BANANA"

- Preparation: Quantization of pmf (and cmf) with $V = 4$ bits

a	$p(a)$	$p(a) \cdot 2^4$	$p_V(a)$	$c_V(a)$
A	$1/2$	$16/2 = 8.00$	8	0
N	$1/3$	$16/3 \approx 5.33$	5	8
B	$1/6$	$16/6 \approx 2.66$	3	13

- Note: Quantized pmf $p_V(a)$ fulfills the requirement $\sum p_V(a) \leq 2^V$

Arithmetic Coding Example

Example: Step 1

s_n	p_V	c_V	parameter updates & output
initialization			$A_0 = 15 = \text{'}\mathbf{1111}\text{'}$ $c_0 = 0 \quad (\text{''})$ $B_0 = 0 = \text{'}\mathbf{0000\ 0000}\text{'}$ bitstream = **“ ”**
“B”	3	13	$A_0 \cdot p_V = 15 \cdot 3 = 45 = \text{'}\mathbf{0010\ 1101}\text{'}$ $B_0 + A_0 \cdot c_V = 0 + 15 \cdot 13 = 195 = \text{'}\mathbf{0\ 1100\ 0011}\text{'}$ $\Delta z = 2$
			$A_1 = \text{'}\mathbf{1011}\text{'} = 11$ $c_1 = 0 \quad (\text{''})$ $B_1 = \text{'}\mathbf{0000\ 1100}\text{'} = 12$ **output** = **“11”** bitstream = **“11”**

Arithmetic Coding Example

Example: Step 2

s_n	p_V	c_V	parameter updates & output
after step 1			$A_1 = 11 = \text{'}\mathbf{1011}\text{'}$ $c_1 = 0 \quad (\text{''})$ $B_1 = 12 = \text{'}\mathbf{0000\ 1100}\text{'}$ bitstream = **"11"**
"A"	8	0	$A_1 \cdot p_V = 11 \cdot 8 = 88 = \text{'}\mathbf{0101\ 1000}\text{'}$ $B_1 + A_1 \cdot c_V = 12 + 11 \cdot 0 = 12 = \text{'}\mathbf{0\ 0000\ 1100}\text{'}$ $\Delta z = 1$
			$A_2 = \text{'}\mathbf{1011}\text{'} = 11$ $c_2 = 1 \quad (\text{'}\mathbf{0}\text{'})$ $B_2 = \text{'}\mathbf{0001\ 1000}\text{'} = 24$ **output** = **" "** bitstream = **"11"**

Arithmetic Coding Example

Example: Step 3

s_n	p_V	c_V	parameter updates & output
after step 2			$A_2 = 11 = \text{'}\mathbf{1011}\text{'}$ $c_2 = 1 \quad (\text{'}\mathbf{0}\text{'})$ $B_2 = 24 = \text{'}\mathbf{0001\ 1000}\text{'}$ bitstream = **"11"**
"N"	5	8	$A_2 \cdot p_V = 11 \cdot 5 = 55 = \text{'}\mathbf{0011\ 0111}\text{'}$ $B_2 + A_2 \cdot c_V = 24 + 11 \cdot 8 = 112 = \text{'}\mathbf{0\ 0111\ 0000}\text{'}$ $\Delta z = 2$
			$A_3 = \text{'}\mathbf{1101}\text{'} = 13$ $c_3 = 2 \quad (\text{'}\mathbf{01}\text{'})$ $B_3 = \text{'}\mathbf{1100\ 0000}\text{'} = 192$ **output** = **"0"** bitstream = **"110"**

Arithmetic Coding Example

Example: Step 4

s_n	p_V	c_V	parameter updates & output
after step 3			$A_3 = 13 = \text{'}\mathbf{1101}\text{'}$ $c_3 = 2 \quad (\text{'}\mathbf{01}\text{'})$ $B_3 = 192 = \text{'}\mathbf{1100\ 0000}\text{'}$ bitstream = **"110"**
"A"	8	0	$A_3 \cdot p_V = 13 \cdot 8 = 104 = \text{'}\mathbf{0110\ 1000}\text{'}$ $B_3 + A_3 \cdot c_V = 192 + 13 \cdot 0 = 192 = \text{'}\mathbf{0\ 1100\ 0000}\text{'}$ $\Delta z = 1$
			$A_4 = \text{'}\mathbf{1101}\text{'} = 13$ $c_4 = 3 \quad (\text{'}\mathbf{011}\text{'})$ $B_4 = \text{'}\mathbf{1000\ 0000}\text{'} = 128$ **output** = **" "** bitstream = **"110"**

Arithmetic Coding Example

Example: Step 5

s_n	p_V	c_V	parameter updates & output
after step 4			$A_4 = 13 = \text{'}\mathbf{1101}\text{'}$ $c_4 = 3 \quad (\text{'}\mathbf{011}\text{'})$ $B_4 = 128 = \text{'}\mathbf{1000\ 0000}\text{'}$ bitstream = **"110"**
"N"	5	8	$A_4 \cdot p_V = 13 \cdot 5 = 65 = \text{'}\mathbf{0100\ 0001}\text{'}$ $B_4 + A_4 \cdot c_V = 128 + 13 \cdot 8 = 232 = \text{'}\mathbf{0\ 1110\ 1000}\text{'}$ $\Delta z = 1$
			$A_5 = \text{'}\mathbf{1000}\text{'} = 8$ $c_5 = 4 \quad (\text{'}\mathbf{0111}\text{'})$ $B_5 = \text{'}\mathbf{1101\ 0000}\text{'} = 208$ **output** = **" "** bitstream = **"110"**

Arithmetic Coding Example

Example: Step 6

s_n	p_V	c_V	parameter updates & output
after step 5			$A_5 = 8 = \text{'}\mathbf{1000}\text{'}$ $c_5 = 4 \quad (\text{'}\mathbf{0111}\text{'})$ $B_5 = 208 = \text{'}\mathbf{1101\ 0000}\text{'}$ bitstream = **"110"**
"A"	8	0	$A_5 \cdot p_V = 8 \cdot 8 = 64 = \text{'}\mathbf{0100\ 0000}\text{'}$ $B_5 + A_5 \cdot c_V = 208 + 8 \cdot 0 = 208 = \text{'}\mathbf{0\ 1101\ 0000}\text{'}$ $\Delta z = 1$
			$A_6 = \text{'}\mathbf{1000}\text{'} = 8$ $c_6 = 5 \quad (\text{'}\mathbf{01111}\text{'})$ $B_6 = \text{'}\mathbf{1010\ 0000}\text{'} = 160$ **output** = **" "** bitstream = **"110"**

Arithmetic Coding Example

Example: Codeword Termination

s_n	p_V c_V	parameter updates & output
after step 6		$A_6 = 8 = \text{'}\mathbf{1000}\text{'}$ $c_6 = 5 \quad (\text{'}\mathbf{01111}\text{'})$ $B_6 = 160 = \text{'}\mathbf{1010\ 0000}\text{'}$ bitstream = **"110"**
final rounding		$B^* =$ **"1 0010 0000"** (rounding up B_6) bitstream = **"1101 000"** ($c_6 - 1$ inverted bits) $c = 1 \quad (\text{'}\mathbf{0}\text{'})$ $B =$ **"0010 0000"**
termination		final bitstream = **"1101 0000 0"** ($c + 1$ bits added)

- Bitstream $\boldsymbol{b}$ = "**1101 0000 0**" (for sequence $\boldsymbol{s}$ = "**BANANA**")
- Same number of bits ($K = 9$) as for Shannon-Fano-Elias coding

Decoding With Finite Precision

Identification of Intervals

- Important: Same rounding of interval width as in encoder ($A_n \mapsto A_{n+1}$)
- Arithmetic codeword "$b_0 b_1 b_2 b_3 b_4 b_5 b_6 \cdots$" represents binary fraction

$$v = (0.b_0 b_1 b_2 b_3 b_4 b_5 b_6 \cdots)_{\mathrm{b}}$$

- Iterative decoding: Output symbol s_n which fulfills inequality

$$L_n + W_n \cdot c(s_n) \leq v < L_n + W_n \cdot c(s_n) + W_n \cdot p(s_n)$$

Observation:

- Lower interval boundary L_n cannot be represented with reasonable precision

➡ Idea: Subtract L_n from the inequality

➡ Symbol s_n is identified by

$$W_n \cdot c(s_n) \leq v - L_n < W_n \cdot c(s_n) + W_n \cdot p(s_n)$$

➡ The value $u_n = v - L_n$ used in comparisons can be stored with $(U + V)$ bits, but needs to be updated after a symbol s_n is decoded

Arithmetic Decoding: Binary Representations

Analyse Binary Representations

$$W_n = 0.\overbrace{0000000000000\cdots0}^{z_n-U}\ \overbrace{1xx\cdots x}^{U}\ 00000000000000\cdots$$

$$W_n \cdot (c_V + p_V) = 0.\overbrace{0000000000000\cdots0}^{z_n-U}\ \overbrace{xxx\cdots xxxxxx}^{U+V}\ 000000000\cdots$$

$$v - L_n = 0.\ 0000000000000\cdots0\ \ xxx\cdots xxxxxx\ \ xxxxxxxxx\ \cdots$$

$$W_n \cdot c_V = 0.\underbrace{0000000000000\cdots0}_{z_n-U}\ \underbrace{xxx\cdots xxxxxx}_{U+V}\ 000000000\cdots$$

- ➡ Use $(U+V)$-bit integer u_n in comparisons (down-rounded value of $v - L_n$)
  - Initialization: $u_n =$ (first $U+V$ bits from bitstream)
  - Update $u_n \mapsto u_{n+1}$
    - ➡ Subtract lower boundary: $u_n = u_n - W_n \cdot c_V(s_n)$
    - ➡ Align with interval width: $u_n = u_n \ll \Delta z$ (leading zeros in $A_n \cdot p_V(s_n)$)
    - ➡ Fill least significant bits with next Δz bits from bitstream

Overview of Arithmetic Decoding Process

Arithmetic Decoding

1. Initialization: $A_0 = 2^U - 1$,
   $u_0 = (\text{first } U+V \text{ bits from bitstream})$
2. Iteration: For $n = 0, 1, \cdots, N-1$, do

   a. **Identify next symbol**: For $k = 0, 1, \cdots,$ do *(loop over alphabet)*
   - Calculate upper boundary $U(a_k) = A_n \cdot (c_V(a_k) + p_V(a_k))$
   - If $u_n < U(a_k)$, then
     - ➡ Output next symbol $s_n = a_k$
     - ➡ break loop over k

   b. **Update parameters**:
   - Calculate intermediate value: $A^* = A_n \cdot p_V(s_n)$
   - Determine number Δz of trailing zeros in $(U+V)$-bit integer A^*

   ➡ $A_{n+1} = A^* \gg (V - \Delta z)$

   ➡ $u_{n+1} = (u_n \ll \Delta z) + (\text{next } \Delta z \text{ bits from bitstream})$

Arithmetic Decoding Example

Decode Bitstream obtained in Encoding Example

- Coding example (see encoding example)
  - IID source with symbol alphabet $\{\mathrm{A}, \mathrm{N}, \mathrm{B}\}$
  - Pmf is given by $\{\, 1/2,\, 1/3,\, 1/6 \,\}$
  - Arithmetic coding with $V = 4$ and $U = 4$
- Quantized pmf (and cmf) with $V = 4$ bits

a	$p(a)$	$p(a) \cdot 2^4$	$p_V(a)$	$c_V(a)$
A	$1/2$	$16/2 = 8.00$	8	0
N	$1/3$	$16/3 \approx 5.33$	5	8
B	$1/6$	$16/6 \approx 2.66$	3	13

- Bitstream $\boldsymbol{b} =$ "1101 0000 0"

Arithmetic Decoding Example

Example: Step 1

a	c_V	p_V	decoding & update	output
initialization			bitstream = "1101 0000 **0(000 0000 0 $\cdots$)**" $A_0 = 15 =$ '**1111**' $u_0 = 208 =$ '**1101 0000**'	
A	0	8	$U(\mathrm{A}) = 15 \cdot (0+8) = 120 \rightarrow U(\mathrm{A}) \leq u_0$	
N	8	5	$U(\mathrm{N}) = 15 \cdot (8+5) = 195 \rightarrow U(\mathrm{N}) \leq u_0$	
B	13	3	$U(\mathrm{B}) = 15 \cdot (13+3) = 240 \rightarrow U(\mathrm{B}) > u_0$	**B**
			$A^* = 15 \cdot 3 = 45 =$ '**0010 1101**' $u^* = 208 - 15 \cdot 13 = 13 =$ '**0000 1101**' $\Delta z = 2$ $A_1 =$ '**1011**' $= 11$ $u_1 =$ '**0011 0100**' $= 52$ bitstream = "1101 0000 **0(000 0000 0 $\cdots$)**"	

Arithmetic Decoding Example

Example: Step 2

a	c_V	p_V	decoding & update	output
after step 1			bitstream = "1101 0000 0(000 0000 0 $\cdots$)" $A_1 = 11 = $ '**1011**' $u_1 = 52 = $ '**0011 0100**'	
A	0	8	$U(\mathrm{A}) = 11 \cdot (0 + 8) = 88 \quad \rightarrow U(\mathrm{A}) > u_1$	**A**
N	8	5	$U(\mathrm{N}) = 11 \cdot (8 + 5) = 143$	
B	13	3	$U(\mathrm{B}) = 11 \cdot (13 + 3) = 176$	
			$A^* = 11 \cdot 8 = 88 = $ '**0101 1000**' $u^* = 52 - 11 \cdot 0 = 52 = $ '**0011 0100**' $\Delta z = 1$ $A_2 = $ '**1011**' $= 11$ $u_2 = $ '**0110 1000**' $= 104$ bitstream = "1101 0000 0(000 0000 0 $\cdots$)"	

Arithmetic Decoding Example

Example: Step 3

a	c_V	p_V	decoding & update	output
after step 2			bitstream = "1101 0000 0(000 0000 0 $\cdots$)" $A_2 = 11 =$ '**1011**' $u_2 = 104 =$ '**0110 1000**'	
A	0	8	$U(\mathrm{A}) = 11 \cdot (0+8) = 88 \quad \rightarrow U(\mathrm{A}) \leq u_2$	
N	8	5	$U(\mathrm{N}) = 11 \cdot (8+5) = 143 \quad \rightarrow U(\mathrm{N}) > u_2$	**N**
B	13	3	$U(\mathrm{B}) = 11 \cdot (13+3) = 176$	
			$A^* = 11 \cdot 5 = 55 =$ '**0011 0111**' $u^* = 104 - 11 \cdot 8 = 16 =$ '**0001 0000**' $\Delta z = 2$ $A_3 =$ '**1101**' $= 13$ $u_3 =$ '**0100 0000**' $= 64$ bitstream = "1101 0000 0(000 0000 0 $\cdots$)"	

Arithmetic Decoding Example

Example: Step 4

a	c_V	p_V	decoding & update	output
after step 3			bitstream = "1101 0000 0(000 0000 0 $\cdots$)" $A_3 = 13 =$ '1101' $u_3 = 64 =$ '0100 0000'	
A	0	8	$U(\mathrm{A}) = 13 \cdot (0+8) = 104 \quad \rightarrow U(\mathrm{A}) > u_3$	A
N	8	5	$U(\mathrm{N}) = 13 \cdot (8+5) = 169$	
B	13	3	$U(\mathrm{B}) = 13 \cdot (13+3) = 208$	
			$A^* = 13 \cdot 8 = 104 =$ '0110 1000' $u^* = 64 - 13 \cdot 0 = 64 =$ '0100 0000' $\Delta z = 1$ $A_4 =$ '1101' $= 13$ $u_4 =$ '1000 0000' $= 128$ bitstream = "1101 0000 0(000 0000 0 $\cdots$)"	

Arithmetic Decoding Example

Example: Step 5

a	c_V	p_V	decoding & update	output
after step 4			bitstream = "1101 0000 0(000 0000 0 $\cdots$)" $A_4 = 13 =$ '1101' $u_4 = 128 =$ '1000 0000'	
A	0	8	$U(\mathrm{A}) = 13 \cdot (0+8) = 104 \quad \rightarrow U(\mathrm{A}) \leq u_4$	
N	8	5	$U(\mathrm{N}) = 13 \cdot (8+5) = 169 \quad \rightarrow U(\mathrm{N}) > u_4$	N
B	13	3	$U(\mathrm{B}) = 13 \cdot (13+3) = 208$	
			$A^* = 13 \cdot 5 = 65 =$ '0100 0001' $u^* = 128 - 13 \cdot 8 = 24 =$ '0001 1000' $\Delta z = 1$ $A_5 =$ '1000' $= 8$ $u_5 =$ '0011 0000' $= 48$ bitstream = "1101 0000 0(000 0000 0 $\cdots$)"	

Arithmetic Decoding Example

Example: Step 6 (last symbol)

a	c_V	p_V	decoding & update	output
after step 5			bitstream = "1101 0000 0(000 0000 0 0 $\cdots$)" $A_5 = 8 =$ '**1000**' $u_5 = 48 =$ '**0011 0000**'	
A	0	8	$U(\mathrm{A}) = 8 \cdot (0 + 8) = 64 \quad \rightarrow U(\mathrm{A}) > u_4$	**A**
N	8	5	$U(\mathrm{N}) = 8 \cdot (8 + 5) = 104$	
B	13	3	$U(\mathrm{B}) = 8 \cdot (13 + 3) = 128$	

bitstream "**1101 0000 0**" $\Longleftrightarrow$ symbol sequence "**BANANA**"

Note: Required some bits after end of the bitstream

- For non-prefix variant: Use bits equal to 0
- For prefix-free variant: Any bit values (0 or 1) can be used

Efficiency of Arithmetic Coding

Increase in Codeword Length relative to Shannon-Fano-Elias Coding

- Excess rate due to rounding of interval width

$$\Delta\ell = \lceil -\log_2 W_N \rceil - \lceil -\log_2 p(\boldsymbol{s}) \rceil < 1 + \log_2 \frac{p(\boldsymbol{s})}{W_N} \tag{48}$$

- Upper bound for increase in codeword length per symbol relative to infinite-precision Shannon-Fano-Elias coding

$$\Delta\bar{\ell} < \frac{1}{N} + \log_2 \left(1 + 2^{1-U}\right) - \log_2 \left(1 - \frac{2^{-V}}{p_{\min}}\right) \tag{49}$$

  (for a derivation see Wiegand, Schwarz, page 51-52)

- Example:
  - Number of coded symbols $N = 1000$,
  - Arithmetic precision: $V = 16$ and $U = 12$,
  - Minimum probablity mass $p_{\min} = 0.02$

  ➡ Increase in codeword length is less than 0.003 bit per symbol

Complexity Reduction: Binary Arithmetic Coding

Binary Arithmetic Coding

- Most popular type of arithmetic coding: JPEG 2000, H.264, H.265
- Binarization of $S \in \{a_0, a_1, \ldots, a_{M-1}\}$ produces $C \in \{0, 1\}$
  - ➡ Any prefix code can be used for binarization
  - ➡ Example: Truncated unary binarization

S_n	number of bins B	C_0	C_1	C_2	$\cdots$	C_{M-2}	C_{M-1}
a_0	1	1					
a_1	2	0	1				
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$			
a_{M-2}	$M-2$	0	0	0	$\cdots$	0	1
a_{M-1}	$M-2$	0	0	0	$\cdots$	0	0

➡ Entropy unchanged due to binarization $\boldsymbol{S} \mapsto \boldsymbol{C}$

$$H(\boldsymbol{S}) = \mathrm{E}\{-\log_2 p(\boldsymbol{S})\} = \mathrm{E}\{-\log_2 p(\boldsymbol{C})\} = H(\boldsymbol{C})$$

Practical Arithmetic Coding

Complexity Reduction

- Binary arithmetic coding
- Multiplication-free implementations
- Bypass mode: Low-complexity coding of bins with $p = 0.5$

Practical Design Aspects

1. Context selection
   - Use reasonable context variables $X = f(S_{n-1}, S_{n-2}, \cdots)$ for switching probability tables $p(a \mid X)$
   - Use context switching only when useful (certain bins)
2. Estimate probabilities during coding
   - Choose appropriate "window sizes" for estimation
3. Suitably combine context selection and probability estimation

Experimental Comparison of Lossless Coding Techniques

Example: Markov Source

- Stationary Markov source given by conditional pmf

a	$p(a\|a_0)$	$p(a\|a_1)$	$p(a\|a_2)$
a_0	0.90	0.15	0.05
a_1	0.05	0.80	0.05
a_2	0.05	0.05	0.60

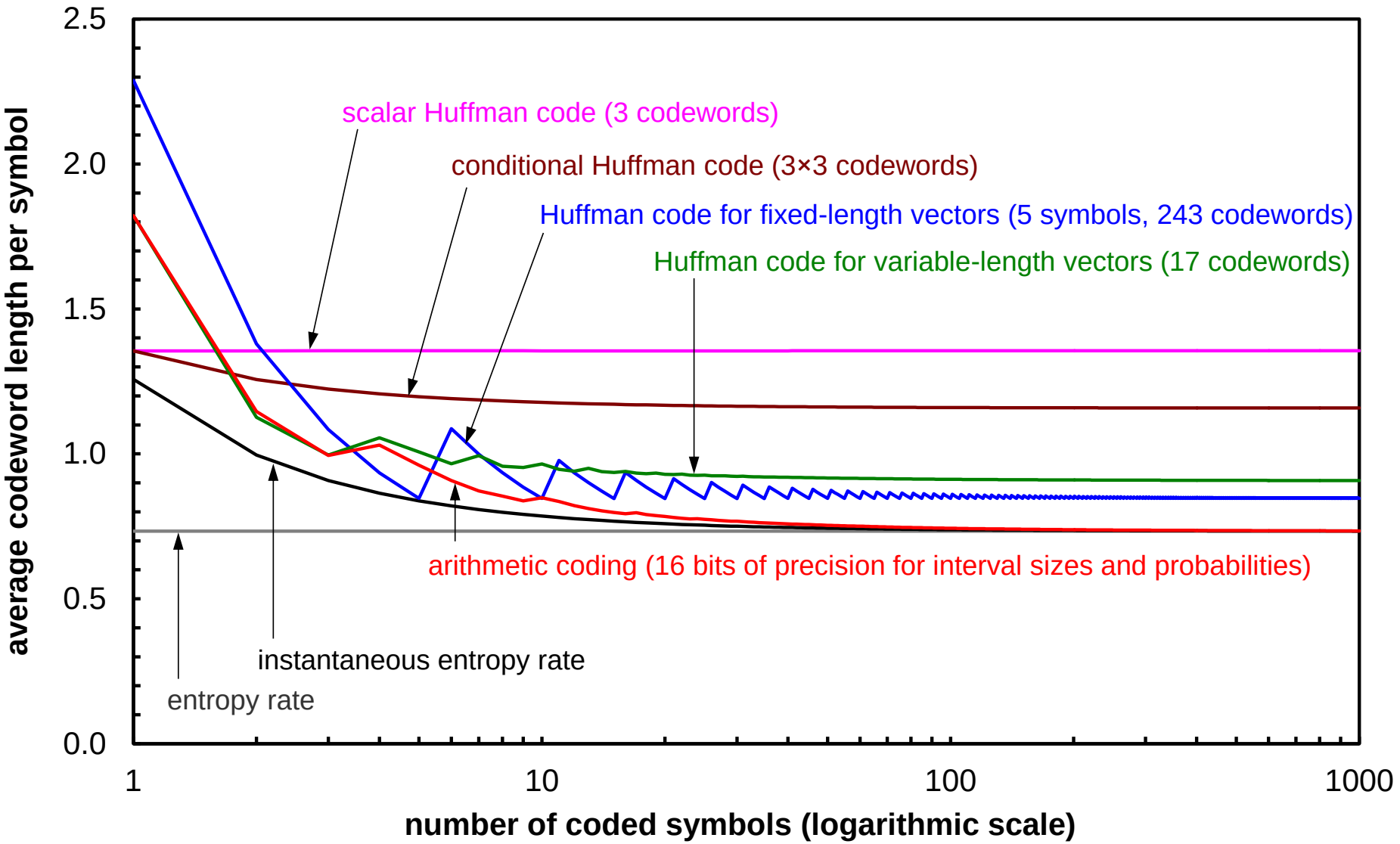
$$H(S) = 1.2575$$
$$\bar{H}(\boldsymbol{S}) = 0.7331$$

- Bounds for lossless coding
  - Entropy rate $\bar{H}(\boldsymbol{S})$ for coding of infinitely many symbols
  - Instantaneous entropy rate $\bar{H}_{\text{inst}}(\boldsymbol{S}, L)$ for coding L symbols

$$\bar{H}_{\text{inst}}(\boldsymbol{S}, L) = \frac{1}{L} H(S_0, S_1, \cdots, S_{L-1}) \tag{50}$$

- Coding experiment
  - Coding of 1 000 000 realizations of example stationary Markov source
  - Calculate average codeword length for sequences of 1 to 1000 symbols

Experimental Results



Part Summary

Uniquely decodable codes & bounds for lossless coding

- Kraft inequality, prefix codes
- Scalar entropy, conditional entropy, block entropy
- Entropy rate, instantaneous entropy rate

Variable-length codes for scalars and vectors

- Optimal code for given pmf: Huffman code
- Scalar, conditional codes: Inefficient for pmfs with $p(a) \gg 0.5$
- Block codes and V2V codes: Code tables can become extremely large
- Difficult adaptation to instationary sources

Arithmetic coding

- No codeword table: Iterative construction of codeword
- Close to entropy bound for $N \gg 1$
- Well suited for exploiting statistical dependencies
- Well suited for adapting probabilities during coding