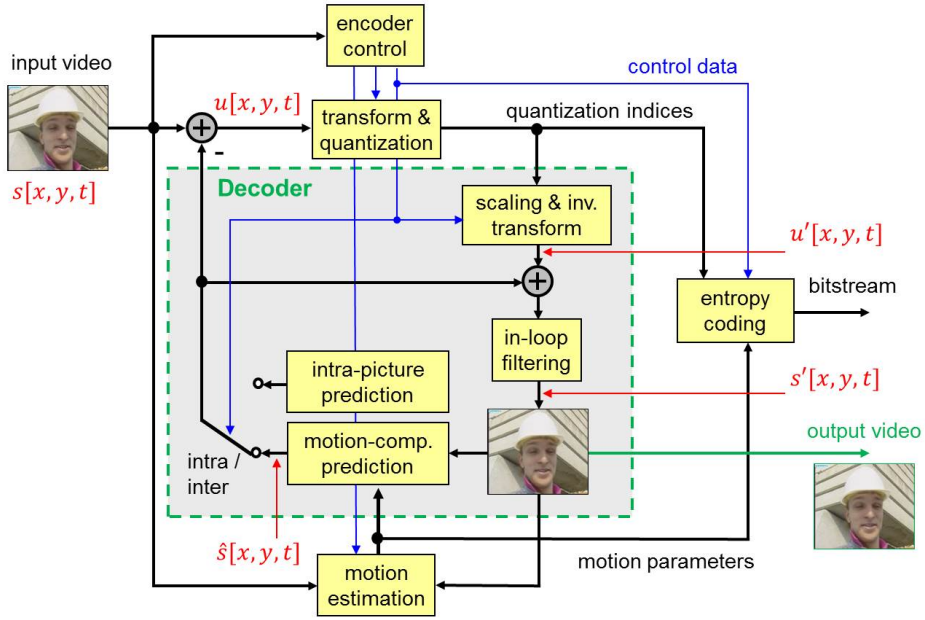
Source Coding and Compression

Heiko Schwarz



Contact:
Dr.-Ing. Heiko Schwarz
heiko.schwarz@hhi.fraunhofer.de

Outline

Part I: Source Coding Fundamentals

- Probability, Random Variables and Random Processes
- Lossless Source Coding
  - Introduction
  - Variable-Length Coding for Scalars
  - Variable-Length Coding for Vectors
  - **Elias and Arithmetic Coding**
- Rate-Distortion Theory
- Quantization
- Predictive Coding
- Transform Coding

Part II: Application in Image and Video Coding

- Still Image Coding / Intra-Picture Coding
- Hybrid Video Coding (From MPEG-2 Video to H.265/HEVC)

Elias Coding and Arithmetic Coding

- Scalar and conditional Huffman codes can be very inefficient
- Main drawback of block Huffman codes: Large table sizes
- Another class of uniquely decodable codes are **Elias and Arithmetic codes**
- Mapping of a string of N symbols $\boldsymbol{s} = \{s_0, s_1, ..., s_{N-1}\}$ onto a string of K bits $\boldsymbol{b} = \{b_0, b_1, ..., b_{K-1}\}$

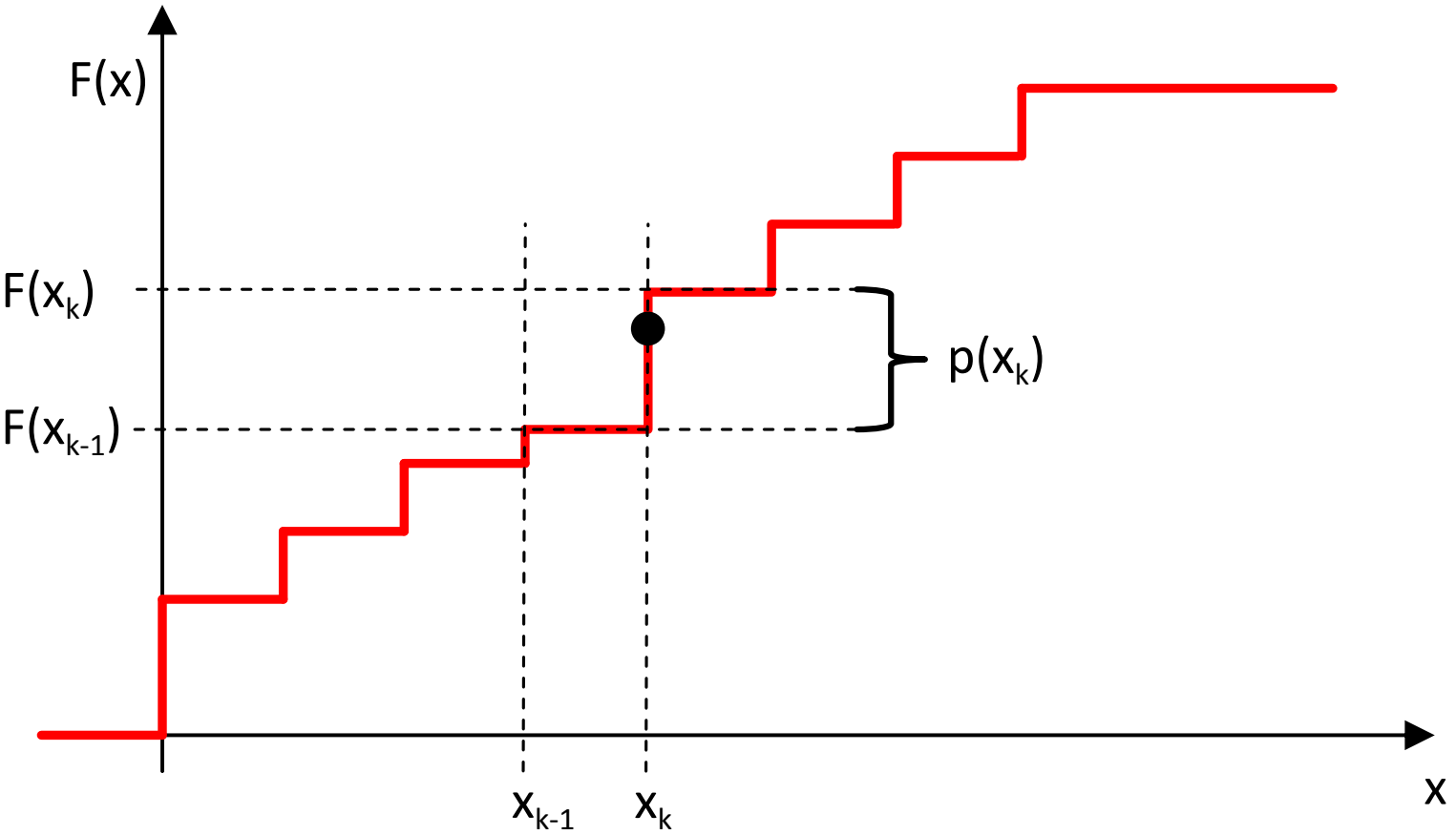
$$\gamma : \boldsymbol{s} \rightarrow \boldsymbol{b} \tag{119}$$

- Decoding or parsing maps the bit string onto the string of symbols

$$\gamma^{-1} : \boldsymbol{b} \rightarrow \boldsymbol{s} \tag{120}$$

- Complexity of code construction: Linear per symbol
- Suitable for adapting pmfs to instationary statistics

Idea of Elias Coding



- Order symbols or messages
- Transmit number in interval $[0, 1)$ which characterizes the symbol or message
- Number of transmitted bits depends on probability of the message

Define an Order of Symbol Sequences

- Consider coding of symbol sequences $\boldsymbol{s} = \{s_0, s_1, \ldots, s_{N-1}\}$
- Realization of sequence of random variables $\boldsymbol{S} = \{S_0, S_1, \ldots, S_{N-1}\}$
- Number N of symbols is known at encoder and decoder
- Each random variable S_n is characterized by an alphabet $\mathcal{A}_n$ of M_n symbols

- Statistical properties are characterized by joint pmf

$$p(\boldsymbol{s}) = P(\boldsymbol{S}=\boldsymbol{s}) = P(S_0=s_0, S_1=s_1, \cdots, S_{N-1}=s_{N-1}) \tag{121}$$

- Need to define an order for symbol sequences
- For example: Symbol sequence $\boldsymbol{s}_a = \{s_0^a, s_1^a, \cdots, s_{N-1}^a\}$ is less than another symbol sequence $\boldsymbol{s}_b = \{s_0^b, s_1^b, \cdots, s_{N-1}^b\}$ if and only if there exists an integer n, with $0 \leq n < N$ such that

$$s_k^a = s_k^b \quad \text{for } k = 0, \cdots, n-1 \quad \text{and} \quad s_n^a < s_n^b \tag{122}$$

Mapping of Symbol Sequences to Intervals

- Joint pmf

$$p(\boldsymbol{s}) = P(\boldsymbol{S}=\boldsymbol{s}) = P(S_0=s_0, S_1=s_1, \cdots, S_{N-1}=s_{N-1}) \tag{123}$$

- Using the defined order for symbol sequences, the pmf of $\boldsymbol{s}$ can be written

$$p(\boldsymbol{s}) = P(\boldsymbol{S}=\boldsymbol{s}) = P(\boldsymbol{S}\leq\boldsymbol{s}) - P(\boldsymbol{S}<\boldsymbol{s}) \tag{124}$$

- Mapping of $\boldsymbol{s} = \{s_0, s_1, \ldots, s_{N-1}\}$ to half-open interval $\mathcal{I}_N \subset [0,1)$

$$\boxed{\mathcal{I}_N(\boldsymbol{s}) = [L_N, L_N+W_N) = \big[P(\boldsymbol{S}<\boldsymbol{s}),\ P(\boldsymbol{S}\leq\boldsymbol{s})\big)} \tag{125}$$

with

$$L_N = P(\boldsymbol{S}<\boldsymbol{s}) \tag{126}$$

$$W_N = P(\boldsymbol{S}=\boldsymbol{s}) = p(\boldsymbol{s}) \tag{127}$$

Unique Identification: The Intervals are Disjoint

- Consider two symbol sequences $\boldsymbol{s}_a$ and $\boldsymbol{s}_b$, with $\boldsymbol{s}_a < \boldsymbol{s}_b$
- Intervals are disjoint if and only if $L_N^b \geq L_N^a + W_N^a$
- Proof:

$$\begin{aligned} L_N^b &= P(\boldsymbol{S} < \boldsymbol{s}_b) \\ &= P(\{\boldsymbol{S} \leq \boldsymbol{s}_a\} \cup \{\boldsymbol{s}_a < \boldsymbol{S} < \boldsymbol{s}_b\}) \\ &= P(\boldsymbol{S} \leq \boldsymbol{s}_a) + \underbrace{P(\boldsymbol{s}_a < \boldsymbol{S} < \boldsymbol{s}_b)}_{\geq 0} \\ &\geq P(\boldsymbol{S} \leq \boldsymbol{s}_a) \\ &= L_N^a + W_N^a \end{aligned} \tag{128}$$

$\Longrightarrow$ Intervals $\mathcal{I}_N^a$ and $\mathcal{I}_N^b$ do not overlap

$\Longrightarrow$ Any number v in the interval $\mathcal{I}_N(\boldsymbol{s})$ uniquely identifies $\boldsymbol{s}$

How Many Bits for Identifying an Interval?

- Identify an interval $\mathcal{I}_N(\boldsymbol{s})$ for a sequence $\boldsymbol{s}$ by a number v
- Number v can be represented as a binary fraction with K bits

$$v = \sum_{i=0}^{K-1} b_i \cdot 2^{i-1} = 0.b_0 b_1 \cdots b_{K-1} \in \mathcal{I}_N(\boldsymbol{s}) \tag{129}$$

- For identifying $\boldsymbol{s}$: Transmit bit sequence $\boldsymbol{b} = \{b_0, b_1, \cdots, b_{K-1}\}$
- Elias code: Assignment of bit sequences $\boldsymbol{b}$ to symbol sequences $\boldsymbol{s}$

- Question: How many bits do we need to uniquely identify an interval $\mathcal{I}_N(\boldsymbol{s})$?
- Intuitively: Size of interval, given by $p(\boldsymbol{s})$, governs number K of bits that are needed to identify the interval

$$\begin{aligned}
p(\boldsymbol{s}) = 1/2 \quad &\rightarrow \quad \mathcal{B} = \{.0, .1\} \\
p(\boldsymbol{s}) = 1/4 \quad &\rightarrow \quad \mathcal{B} = \{.00, .01, .10, .11\} \\
p(\boldsymbol{s}) = 1/8 \quad &\rightarrow \quad \mathcal{B} = \{.000, .001, .010, .011, .100, .101, .110, .111\}
\end{aligned}$$

How Many Bits for Identifying an Interval?

- Goal: **Choose real number $v \in \mathcal{I}_N$ that can be represented with the minimum amount of bits**
- Distance between successive binary fractions of K bits is 2^{-K}
- To be sure that a binary fraction of K bits falls inside an interval of width W_N, we need

$$\begin{aligned} 2^{-K} &\leq W_N \\ K &\geq -\log_2 W_N \end{aligned} \tag{130}$$

- Hence, we choose

$$K = K(\boldsymbol{s}) = \lceil -\log_2 W_N \rceil = \lceil -\log_2 p(\boldsymbol{s}) \rceil \tag{131}$$

- The binary number v identifying the interval $\mathcal{I}_N$ can be determined by

$$v = \lceil L_N \cdot 2^K \rceil \cdot 2^{-K} \tag{132}$$

Verification of Selection of Bits and Bitstring

- Binary number v identifying the interval $\mathcal{I}_n$

$$v = \lceil L_N \cdot 2^K \rceil \cdot 2^{-K} \qquad \text{with} \qquad K = \lceil -\log_2 W_N \rceil \tag{133}$$

- With $x \leq \lceil x \rceil$ and $\lceil x \rceil < x + 1$, we obtain

$$L_N \leq v < L_N + 2^{-K} \tag{134}$$

- Using the expression for the required number of bits

$$K = \lceil -\log_2 p(\boldsymbol{s}) \rceil \geq -\log_2 p(\boldsymbol{s}) \quad \Longrightarrow \quad 2^{-K} \leq p(\boldsymbol{s}) = W_N \tag{135}$$

yields

$$L_N \leq v < L_N + W_N \tag{136}$$

- The representative $v = 0.b_0 b_1 \ldots b_{K-1}$ always lies inside the interval $\mathcal{I}_N(\boldsymbol{s})$

$\Longrightarrow$ Message $\boldsymbol{s}$ can be uniquely decoded from the transmitted bit string $\boldsymbol{b} = \{b_0, b_1, \ldots, b_{K-1}\}$ of $K(\boldsymbol{s}) = \lceil -\log_2 p(\boldsymbol{s}) \rceil$ bits

Redundancy of Elias Coding

- Average codeword length per symbol

$$\bar{\ell} = \frac{E\{K(\boldsymbol{S})\}}{N} = \frac{E\{\lceil -\log_2 p(\boldsymbol{S}) \rceil\}}{N} \tag{137}$$

- Applying inequalities $x \leq \lceil x \rceil$ and $\lceil x \rceil < x + 1$, we obtain

$$\frac{E\{-\log_2 p(\boldsymbol{S})\}}{N} \leq \bar{\ell} < \frac{E\{1 - \log_2 p(\boldsymbol{S})\}}{N} \tag{138}$$

- Average codeword length is bounded

$$\boxed{\frac{H_N(\boldsymbol{S})}{N} \leq \bar{\ell} \leq \frac{H_N(\boldsymbol{S})}{N} + \frac{1}{N}} \tag{139}$$

- Note: Same bounds as for block Huffman codes
- For specific application: One additional bit required (see exercise)
- Question: What is the advantage?

Derivation of Iterative Algorithm for Elias Coding

Iterative construction of codewords

- Consider sub-sequences $\boldsymbol{s}^{(n)} = \{s_0, s_1, \cdots, s_{n-1}\}$ with $1 \leq n \leq N$
- Interval width W_{n+1} for the sub-sequence $\boldsymbol{s}^{(n+1)} = \{\boldsymbol{s}^{(n)}, s_n\}$

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

- Iteration rule for interval width

$$\boxed{W_{n+1} = W_n \cdot p(s_n \,|\, s_0, s_1, \ldots, s_{n-1})} \tag{140}$$

- Since $p(s_n \,|\, s_0, s_1, \ldots, s_{n-1}) \leq 1$, it follows

$$W_{n+1} \leq W_n \tag{141}$$

Derivation of Iterative Algorithm for Elias Coding

- Derivation for lower interval border L_{n+1} for the sub-sequence $\boldsymbol{s}^{(n+1)} = \{\boldsymbol{s}^{(n)}, s_n\}$

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

- Iteration rule of lower interval boundary

$$\boxed{L_{n+1} = L_n + W_n \cdot c(s_n \,|\, s_0, s_1, \ldots, s_{n-1})} \tag{142}$$

with the cmf $c(\cdot)$ being defined as

$$c(s_n \,|\, s_0, s_1, \ldots, s_{n-1}) = \sum_{\forall a \in \mathcal{A}_n:\, a < s_n} p(a \,|\, s_0, s_1, \ldots, s_{n-1}) \tag{143}$$

- Note: The function $c(\cdot)$ excludes the current symbol
- Since $W_n \cdot c(s_n \,|\, s_0, s_1, \ldots, s_{n-1}) \geq 0$, it follows

$$L_{n+1} \geq L_n \tag{144}$$

Intervals Are Nested

- Iteration rules:

$$\begin{aligned} W_{n+1} &= W_n \cdot P\big(S_n = s_n \,\big|\, \boldsymbol{S}^{(n)} = \boldsymbol{s}^{(n)}\big) \\ L_{n+1} &= L_n + W_n \cdot P\big(S_n < s_n \,\big|\, \boldsymbol{S}^{(n)} = \boldsymbol{s}^{(n)}\big) \end{aligned}$$

- We have already shown: $L_{n+1} \geq L_n$
- Now, we consider upper interval boundary $L_{n+1} + W_{n+1}$

$$\begin{aligned} L_{n+1} + W_{n+1} &= L_n + W_n \cdot P\big(S_n < s_n \,\big|\, \boldsymbol{S}^{(n)} = \boldsymbol{s}^{(n)}\big) \\ &\quad + W_n \cdot P\big(S_n = s_n \,\big|\, \boldsymbol{S}^{(n)} = \boldsymbol{s}^{(n)}\big) \\ &= L_n + W_n \cdot P\big(S_n \leq s_n \,\big|\, \boldsymbol{S}^{(n)} = \boldsymbol{s}^{(n)}\big) \\ &= L_n + W_n - \underbrace{W_n \cdot P\big(S_n > s_n \,\big|\, \boldsymbol{S}^{(n)} = \boldsymbol{s}^{(n)}\big)}_{\geq 0} \\ &\leq L_n + W_n \end{aligned} \tag{145}$$

$\Longrightarrow$ Intervals are nested: $\mathcal{I}_{n+1} \subset \mathcal{I}_n$

Iterative Algorithm for IID and Markov Sources

- Derivation above for general case of dependent and differently distributed random variables (may even have different alphabets)
- Initialization

$$W_0 = 1 \tag{146}$$
$$L_0 = 0 \tag{147}$$

- For **iid sources**, interval refinement can be simplified

$$W_{n+1} = W_n \cdot p(s_n) \tag{148}$$
$$L_{n+1} = L_n + W_n \cdot c(s_n) \tag{149}$$

- For **Markov sources**: Conditional pmf $p(s_n|s_{n-1})$ and cmf $c(s_n|s_{n-1})$

$$W_{n+1} = W_n \cdot p(s_n|s_{n-1}) \tag{150}$$
$$L_{n+1} = L_n + W_n \cdot c(s_n|s_{n-1}) \tag{151}$$

- Non-stationary sources: Probabilities $p(\cdot)$ can be adapted during coding

Elias Coding Example: IID Source

- Example for an iid source for which an optimum Huffman code exists

symbol a_k	pmf $p(a_k)$	Huffman code	cmf $c(a_k)$
a_0='A'	$0.25 = 2^{-2}$	00	$0.00 = 0$
a_1='B'	$0.25 = 2^{-2}$	01	$0.25 = 2^{-2}$
a_2='C'	$0.50 = 2^{-1}$	1	$0.50 = 2^{-1}$

- Suppose we intend to send the symbol string $\boldsymbol{s}$ = "CABAC"
- Using the Huffman code, the bit string would be $\boldsymbol{b} = 10001001$ (8 bits)
- An alternative to Huffman coding is Elias coding
- Probability of the symbol string "CABAC" is given by

$$p(\boldsymbol{s}) = p('C') \cdot p('A') \cdot p('B') \cdot p('A') \cdot p('C') = \frac{1}{2}\frac{1}{4}\frac{1}{4}\frac{1}{4}\frac{1}{2} = \frac{1}{256}$$

- The size of the bit string is

$$K = \lceil -\log_2 p(\boldsymbol{s}) \rceil = 8 \text{ bits}$$

Encoding Algorithm for Elias Codes

Encoding algorithm:

1. Given is a sequence $\{s_0, \cdots, s_{N-1}\}$ of N symbols
2. Initialization of the iterative process by $W_0 = 1$, $L_0 = 0$
3. For each $n = 0, 1, \cdots, N-1$, determine the interval $\mathcal{I}_{n+1}$ by

$$\begin{aligned} W_{n+1} &= W_n \cdot p(s_n | s_0, \cdots, s_{n-1}) \\ L_{n+1} &= L_n + W_n \cdot c(s_n | s_0, \cdots, s_{n-1}) \end{aligned}$$

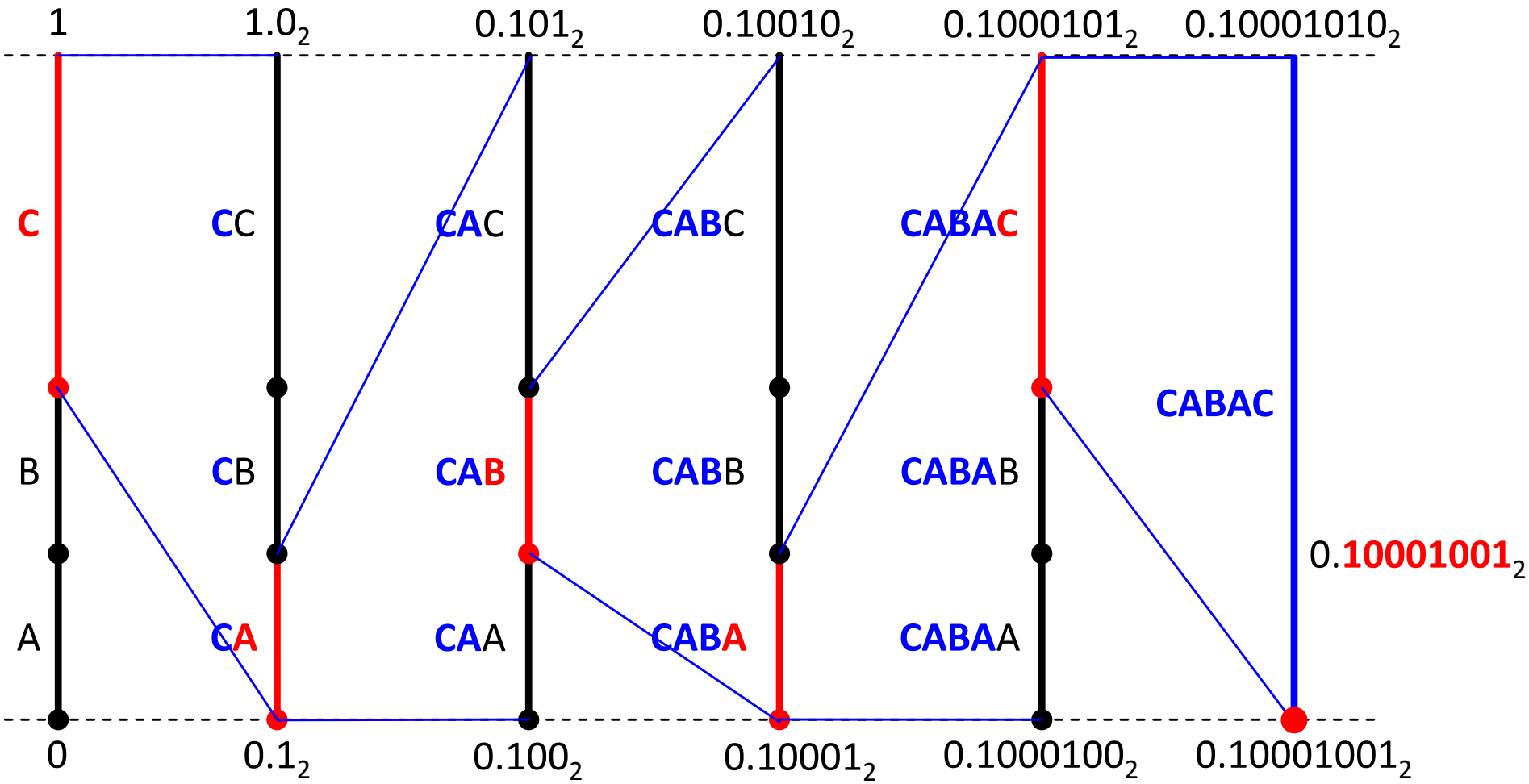
4. Determine the codeword length by $K = \lceil -\log_2 W_N \rceil$
5. Transmit the codeword $\boldsymbol{b}^{(K)}$ of K bits that represents the fractional part of $v = \lceil L_N \, 2^K \rceil \, 2^{-K}$

Example for Elias Encoding

s_0='C'	s_1='A'	s_2='B'
$W_1 = W_0 \cdot p(\text{'C'}) = 1 \cdot 2^{-1} = 2^{-1} = (0.1)_2$	$W_2 = W_1 \cdot p(\text{'A'}) = 2^{-1} \cdot 2^{-2} = 2^{-3} = (0.001)_2$	$W_3 = W_2 \cdot p(\text{'B'}) = 2^{-3} \cdot 2^{-2} = 2^{-5} = (0.00001)_2$
$L_1 = L_0 + W_0 \cdot c(\text{'C'}) = 0 + 1 \cdot 2^{-1} = 2^{-1} = (0.1)_2$	$L_2 = L_1 + W_1 \cdot c(\text{'A'}) = 2^{-1} + 2^{-1} \cdot 0 = 2^{-1} = (0.100)_2$	$L_3 = L_2 + W_2 \cdot c(\text{'B'}) = 2^{-1} + 2^{-3} \cdot 2^{-2} = 2^{-1} + 2^{-5} = (0.10001)_2$

s_3='A'	s_4='C'	termination
$W_4 = W_3 \cdot p(\text{'A'}) = 2^{-5} \cdot 2^{-2} = 2^{-7} = (0.0000001)_2$	$W_5 = W_4 \cdot p(\text{'C'}) = 2^{-7} \cdot 2^{-1} = 2^{-8} = (0.00000001)_2$	$K = \lceil -\log_2 W_5 \rceil = 8$ $v = \lceil L_5 \, 2^K \rceil \, 2^{-K} = 2^{-1} + 2^{-5} + 2^{-8}$ $\boldsymbol{b} = \text{'}\mathbf{10001001}\text{'}$
$L_4 = L_3 + W_3 \cdot c(\text{'A'}) = (2^{-1} + 2^{-5}) + 2^{-5} \cdot 0 = 2^{-1} + 2^{-5} = (0.1000100)_2$	$L_5 = L_4 + W_4 \cdot c(\text{'C'}) = (2^{-1} + 2^{-5}) + 2^{-7} \cdot 2^{-1} = 2^{-1} + 2^{-5} + 2^{-8} = (0.10001001)_2$	

Illustration of Iteration



Decoding Algorithm for Elias Codes

Decoding algorithm:

1. Given is the number N of symbols to be decoded and a codeword $\boldsymbol{b}^{(K)} = \{b_0, \cdots, b_{K-1}\}$ of K_N bits
2. Determine the interval representative v according to

$$v = \sum_{i=0}^{K-1} b_i \, 2^{-i}$$

3. Initialization of the iterative process by $W_0 = 1$, $L_0 = 0$
4. For each $n = 0, 1, \cdots, N-1$, do the following:
   1. For each $a_i \in \mathcal{A}_n$, determine the interval $\mathcal{I}_{n+1}(a_i)$ by

$$\begin{aligned} W_{n+1}(a_i) &= W_n \cdot p(a_i|s_0, \ldots, s_{n-1}) \\ L_{n+1}(a_i) &= L_n + W_n \cdot c(a_i|s_0, \ldots, s_{n-1}) \end{aligned}$$

   2. Select the letter $a_i \in \mathcal{A}_n$ for which $v \in \mathcal{I}_{n+1}(a_i)$ and set $s_n = a_i$, $W_{n+1} = W_{n+1}(a_i)$, $L_{n+1} = L_{n+1}(a_i)$

Properties of Elias Codes

Efficiency

- Bounds on average codeword length are the same as for block Huffman codes
- Concatenations cannot always be decoded using iterative procedure
- Issue can be solved by adding one bit per message (see exercise)
- Block Huffman codes of same size N are optimal

$\Longrightarrow$ **No higher efficiency than block Huffman codes of same size**

Code construction

- Can iteratively construct the codeword for a given message
- Don't need to construct and store the entire code table
- Easy to incorporate adaptation of probabilities to source statistics

$\Longrightarrow$ **That's the advantage of Elias codes**

Are there any issues?

- Require extremely high precision arithmetic for calculating interval size and lower interval boundary for long messages
- Solution: **Approximation using fixed-precision integer arithmetic**

Arithmetic Coding

Elias coding

- Very efficient for long symbol sequences (if suitable probabilities are used)
- Simple codeword construction
- Do not need to store codeword tables
- Problem: Precision requirement for W_n and L_n

Arithmetic coding

- Fixed-precision variant of Elias coding
- Can be realized with standard precision integer arithmetic
- Loss in efficiency due to fixed-precision arithmetic is negligible
- Following approximations are used

  $\Longrightarrow$ Represent probabilities with fixed-precision integers

  $\Longrightarrow$ Represent interval width with fixed-precision integers

  $\Longrightarrow$ Output bits as soon as they cannot be modified in following steps

Quantization of Pmf and Cmf

- Represent pmfs $p(a)$ and cmfs $c(a)$ by V-bit integers $p_V(a)$ and $c_V(a)$

$$p(a) = p_V(a) \cdot 2^{-V} \tag{152}$$

$$c(a) = c_V(a) \cdot 2^{-V} = \sum_{a_i < a} p_V(a_i) \cdot 2^{-V} \tag{153}$$

- Following condition has to be fulfilled

$$\left(\sum_{\forall a_i} p_V(a_i) \right) \cdot 2^{-V} \leq 1 \tag{154}$$

- Simple (but not necessarily best) approach

$$p_V(a_i) = \left\lfloor p(a_i) \cdot 2^V \right\rfloor \tag{155}$$

Note: V must be so large that $p_V(a_i) > 0$ for all a_i

Quantization of Interval Width

- Observation: Elias code remains decodable if intervals are always nested

$$0 < W_{n+1} \leq W_n \cdot p(s_n) \tag{156}$$

$\Longrightarrow$ Rounding down of $W_n \cdot p(s_n)$ at each iteration (for fixed-precision rep.)

- Represent W_n by U-bit integer A_n and integer $z_n \geq U$

$$W_n = A_n \cdot 2^{-z_n} \tag{157}$$

- Initialization: Approximate $W_0 = 1$ by

$$A_0 = 2^U - 1 \qquad \text{and} \qquad z_0 = U \tag{158}$$

- Restriction for A_n

$$\underbrace{2^{U-1}}_{\text{max. precision}} \leq A_n < \underbrace{2^U}_{U\text{-bit integer}} \tag{159}$$

Rounding in Interval Refinement

- Representation of interval width

$$W_n = A_n \cdot 2^{-z_n} \qquad \text{with} \qquad 2^{U-1} \leq A_n < 2^U \tag{160}$$

- Interval refinement for arbitrary precision

$$\begin{aligned} W_{n+1} &= W_n \cdot p(s_n) \\ A_{n+1} \cdot 2^{-z_{n+1}} &= A_n \cdot 2^{-z_n} \cdot p_V(s_n) \cdot 2^{-V} \\ &= \left(A_n \cdot p_V(s_n) \cdot 2^{-y_{n+1}}\right) \cdot 2^{-z_n - V + y_{n+1}} \end{aligned} \tag{161}$$

- Interval refinement for fixed-precision arithmetic

$$A_{n+1} = \left\lfloor A_n \cdot p_V(s_n) \cdot 2^{-y_{n+1}} \right\rfloor \qquad \text{(simple right shift by } y_{n+1} \text{ bits)} \tag{162}$$

$$z_{n+1} = z_n + V - y_{n+1} \tag{163}$$

- Choose y_{n+1} so that $2^{U-1} \leq A_{n+1} < 2^U$ (some comparison operations)

$$y_{n+1} = \underbrace{\left\lceil \log_2(A_n \cdot p_V(s_n) + 1) \right\rceil}_{\text{pos. of most-sign. bit in } A_n p_V(s_n)} - U \tag{164}$$

Analysis of Binary Representations

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

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

- Binary representation of cmf $c(s_n) = c_V(s_n) \cdot 2^{-V}$:

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

- Binary representation of product $W_n \cdot c(s_n)$ (added to L_n in update):

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

Effect on Lower Interval Boundary

- Remember: Update of lower interval boundary $L_{n+1} = L_n + W_n \cdot c(s_n)$
- Binary representation of the product $W_n \cdot c(s_n)$:

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

- What is the effect on lower interval boundary

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

  - Trailing bits: Equal to 0, but maybe changed later
  - Active bits: Directly modified by the update $L_{n+1} = L_n + W_n \cdot c(s_n)$
  - Outstanding bits: May be modified by a carry from the active bits
  - Settled bits: Not modified in any following interval update

Representation of Lower Interval Boundary

- Lower interval boundary L_n

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

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

- Active bits can be represented by an $(U+V)$-bit integer B_n
- Outstanding bits can be represented by a counter c_n
- Settled bits are output as soon as they become settled
- Total number of bits to output is

$$K = \lceil -\log_2 W_N \rceil = z_n - \lfloor \log_2 A_N \rfloor = z_n - U + 1 \qquad (165)$$

- Termination of arithmetic coding
  $\Longrightarrow$ Output all outstanding bits
  $\Longrightarrow$ Output most significant bit of $(U+V)$-integer B_n

Overview of Process of Arithmetic Coding

Initialization:

- Initialize integer representation of interval width: $A_0 = 2^U - 1$
- Initialize $U + V$ active bits: $B_0 = 0$
- Initialize number of outstandig bits: $c_0 = 0$

Iterative coding (for $n = 0$ to $n = N - 1$):

- Calculate product $A^*_{n+1} = A_n \cdot p_V(s_n)$
- Determine bit shift parameter y_{n+1} (check first bit equal to "1" in A^*_{n+1})
- Update interval width: $A_{n+1} = A^*_{n+1} \gg y_{n+1}$
- Output settled bits
- Update active bits B_{n+1} and counter c_{n+1} for outstanding bits

Termination

- Output outstanding bits
- Output most significant bit of $(U + V)$-integer B_N

Example for Arithmetic Coding

- Consider iid process with symbol alphabet $\mathcal{A} = \{\mathsf{M}, \mathsf{I}, \mathsf{S}, \mathsf{P}\}$
- Marginal pmf given by $p(a_i) = \{1/11, 4/11, 4/11, 2/11\}$
- Consider arithmetic coding with $V = 4$ and $U = 4$
- Consider coding of symbol sequence "MISSISSIPPI"

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

a_i	$p(a_i)$	$p(a_i) \cdot 2^4$	$p_V(a_i)$	$c_V(a_i)$
M	$1/11$	$16/11 \approx 1.45$	1	0
I	$4/11$	$64/11 \approx 5.82$	6	1
S	$4/11$	$64/11 \approx 5.82$	6	7
P	$2/11$	$32/11 \approx 2.91$	3	13

- Note: Quantized pmf $p_Q(a_n)$ fulfills the requirement $\sum p_Q(a_n) \leq 1$

$$\sum_{\forall a_i} p_Q(a_i) = \sum_{\forall a_i} p_V(a_i) \cdot 2^{-4} = 16 \cdot 2^{-4} = 1$$

Example for Arithmetic Coding – 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 = **“”**
“M”	1	0	$A_0 \cdot p_V = 15 \cdot 1 = 15 = \text{'}\mathbf{0000\ 1111}\text{'}$ $B_0 + A_0 \cdot c_V = 0 + 15 \cdot 0 = 0 = \text{'}\mathbf{0\ 0000\ 0000}\text{'}$ $y_1 = 0$
			$A_1 = \text{'}\mathbf{1111}\text{'} = 15$ $c_1 = 1 \quad (\text{'}\mathbf{0}\text{'})$ $B_1 = \text{'}\mathbf{0000\ 0000}\text{'} = 0$ **output** = **“000”** bitstream = **“000”**

Example for Arithmetic Coding – Step 2

s_n	p_V	c_V	parameter updates & output
after step 1			$A_1 = 15 = \text{'}\mathbf{1111}\text{'}$ $c_1 = 1 \quad (\text{'}\mathbf{0}\text{'})$ $B_1 = 0 = \text{'}\mathbf{0000\ 0000}\text{'}$ bitstream = **"000"**
"l"	6	1	$A_1 \cdot p_V = 15 \cdot 6 = 90 = \text{'}\mathbf{0101\ 1010}\text{'}$ $B_1 + A_1 \cdot c_V = 0 + 15 \cdot 1 = 15 = \text{'}\mathbf{0\ 0000\ 1111}\text{'}$ $y_2 = 3$
			$A_2 = \text{'}\mathbf{1011}\text{'} = 11$ $c_2 = 1 \quad (\text{'}\mathbf{0}\text{'})$ $B_2 = \text{'}\mathbf{0001\ 1110}\text{'} = 30$ **output** = **"0"** bitstream = **"0000"**

Example for Arithmetic Coding – Step 3

s_n	p_V c_V	parameter updates & output
after step 2		$A_2 = 11 =$ '**1011**' $c_2 = 1$ ('**0**') $B_2 = 30 =$ '**0001 1110**' bitstream = **"0000"**
"S"	6 7	$A_2 \cdot p_V = 11 \cdot 6 = 66 =$ '**0100 0010**' $B_2 + A_2 \cdot c_V = 30 + 11 \cdot 7 = 107 =$ '**0 0110 1011**' $y_3 = 3$
		$A_3 =$ '**1000**' $= 8$ $c_3 = 1$ ('**0**') $B_3 =$ '**1101 0110**' $= 214$ **output** = **"0"** bitstream = **"0000 0"**

Example for Arithmetic Coding – Step 4

s_n	p_V c_V	parameter updates & output
after step 3		$A_3 = 8 =$ '**1000**' $c_3 = 1$ ('**0**') $B_3 = 214 =$ '**1101 0110**' bitstream = **"0000 0"**
"S"	6 7	$A_3 \cdot p_V = 8 \cdot 6 = 48 =$ '**0011 0000**' $B_3 + A_3 \cdot c_V = 214 + 8 \cdot 7 = 270 =$ '**1 0000 1110**' $y_4 = 2$
		$A_4 =$ '**1100**' $= 12$ $c_4 = 1$ ('**0**') $B_4 =$ '**0011 1000**' $= 56$ **output** = **"10"** (invert outstanding bits) bitstream = **"0000 010"**

Example for Arithmetic Coding – Step 5

s_n	p_V c_V	parameter updates & output
after step 4		$A_4 = 12 =$ '**1100**' $c_4 = 1$ ('**0**') $B_4 = 56 =$ '**0011 1000**' bitstream = **"0000 010"**
"l"	6 1	$A_4 \cdot p_V = 12 \cdot 6 = 72 =$ '**0100 1000**' $B_4 + A_4 \cdot c_V = 56 + 12 \cdot 1 = 68 =$ '**0 0100 0100**' $y_5 = 3$
		$A_5 =$ '**1001**' $= 9$ $c_5 = 1$ ('**0**') $B_5 =$ '**1000 1000**' $= 136$ **output** = **"0"** bitstream = **"0000 0100"**

Example for Arithmetic Coding – Step 6

s_n	p_V c_V	parameter updates & output
after step 5		$A_5 = 9 =$ '**1001**' $c_5 = 1$ ('**0**') $B_5 = 136 =$ '**1000 1000**' bitstream = **"0000 0100"**
"S"	6 7	$A_5 \cdot p_V = 9 \cdot 6 = 54 =$ '**0011 0110**' $B_5 + A_5 \cdot c_V = 136 + 9 \cdot 7 = 68 =$ '**0 1100 0111**' $y_6 = 2$
		$A_6 =$ '**1101**' $= 13$ $c_6 = 3$ ('**011**') $B_6 =$ '**0001 1100**' $= 28$ **output** = **""** bitstream = **"0000 0100"**

Example for Arithmetic Coding – Step 7

s_n	p_V c_V	parameter updates & output
after step 6		$A_6 = 13 =$ '**1101**' $c_6 = 3$ ('**011**') $B_6 = 28 =$ '**0001 1100**' bitstream = **"0000 0100"**
"S"	6 7	$A_6 \cdot p_V = 13 \cdot 6 = 78 =$ '**0100 1110**' $B_6 + A_6 \cdot c_V = 28 + 13 \cdot 7 = 119 =$ '**0 0111 0111**' $y_7 = 3$
		$A_7 =$ '**1001**' $= 9$ $c_7 = 1$ ('**0**') $B_7 =$ '**1110 1110**' $= 238$ **output** = **"011"** bitstream = **"0000 0100 011"**

Example for Arithmetic Coding – Step 8

s_n	p_V	c_V	parameter updates & output
after step 7			$A_7 = 9 = \text{'}\mathbf{1001}\text{'}$ $c_7 = 1 \quad (\text{'}\mathbf{0}\text{'})$ $B_7 = 238 = \text{'}\mathbf{1110\ 1110}\text{'}$ bitstream = **"0000 0100 011"**
"l"	6	1	$A_7 \cdot p_V = 9 \cdot 6 = 54 = \text{'}\mathbf{0011\ 0110}\text{'}$ $B_7 + A_7 \cdot c_V = 238 + 9 \cdot 1 = 247 = \text{'}\mathbf{0\ 1111\ 0111}\text{'}$ $y_8 = 2$
			$A_8 = \text{'}\mathbf{1101}\text{'} = 13$ $c_8 = 3 \quad (\text{'}\mathbf{011}\text{'})$ $B_8 = \text{'}\mathbf{1101\ 1100}\text{'} = 220$ **output** = **""** bitstream = **"0000 0100 011"**

Example for Arithmetic Coding – Step 9

s_n	p_V	c_V	parameter updates & output
after step 8			$A_8 = 13 = \text{'}\mathbf{1101}\text{'}$ $c_8 = 3 \quad (\text{'}\mathbf{011}\text{'})$ $B_8 = 220 = \text{'}\mathbf{1101\ 1100}\text{'}$ bitstream = **"0000 0100 011"**
"P"	3	13	$A_8 \cdot p_V = 13 \cdot 3 = 39 = \text{'}\mathbf{0010\ 0111}\text{'}$ $B_8 + A_8 \cdot c_V = 220 + 13 \cdot 13 = 389 = \text{'}\mathbf{1\ 1000\ 0101}\text{'}$ $y_9 = 2$
			$A_9 = \text{'}\mathbf{1001}\text{'} = 9$ $c_9 = 1 \quad (\text{'}\mathbf{0}\text{'})$ $B_9 = \text{'}\mathbf{0001\ 0100}\text{'} = 20$ **output** = **"1001"** (invert outstanding bits) bitstream = **"0000 0100 0111 001"**

Example for Arithmetic Coding – Step 10

s_n	p_V c_V	parameter updates & output
after step 9		$A_9 = 9 = \text{'}\mathbf{1001}\text{'}$ $c_9 = 1 \quad (\text{'}\mathbf{0}\text{'})$ $B_9 = 20 = \text{'}\mathbf{0001\ 0100}\text{'}$ bitstream = **"0000 0100 0111 001"**
"P"	3 13	$A_9 \cdot p_V = 9 \cdot 3 = 27 = \text{'}\mathbf{0001\ 1011}\text{'}$ $B_9 + A_9 \cdot c_V = 20 + 9 \cdot 13 = 137 = \text{'}\mathbf{0\ 1000\ 1001}\text{'}$ $y_{10} = 1$
		$A_{10} = \text{'}\mathbf{1101}\text{'} = 13$ $c_{10} = 1 \quad (\text{'}\mathbf{0}\text{'})$ $B_{10} = \text{'}\mathbf{0100\ 1000}\text{'} = 72$ **output** = **"010"** bitstream = **"0000 0100 0111 0010 10"**

Example for Arithmetic Coding – Step 11

s_n	p_V c_V	parameter updates & output
after step 10		$A_{10} = 13 = \text{'}\mathbf{1101}\text{'}$ $c_{10} = 1 \quad (\text{'}\mathbf{0}\text{'})$ $B_{10} = 72 = \text{'}\mathbf{0100\ 1000}\text{'}$ bitstream = **"0000 0100 0111 0010 10"**
"l"	6 1	$A_{10} \cdot p_V = 13 \cdot 6 = 78 = \text{'}\mathbf{0100\ 1110}\text{'}$ $B_{10} + A_{10} \cdot c_V = 72 + 13 \cdot 1 = 85 = \text{'}\mathbf{0\ 0101\ 0101}\text{'}$ $y_{11} = 3$
		$A_{11} = \text{'}\mathbf{1001}\text{'} = 9$ $c_{11} = 1 \quad (\text{'}\mathbf{0}\text{'})$ $B_{11} = \text{'}\mathbf{1010\ 1010}\text{'} = 170$ **output = "0"** bitstream = **"0000 0100 0111 0010 100"**

Example for Arithmetic Coding – Termination

s_n	p_V c_V	parameter updates & output
after step 11		$A_{11} = 9 =$ '**1001**' $c_{11} = 1$ ('**0**') $B_{11} = 170 =$ '**1010 1010**' bitstream = **"0000 0100 0111 0010 100"**
termination		**output** = **"01"** (outstanding + first bit of B_{11}) bitstream = **"0000 0100 0111 0010 1000 1"**

- Transmitted bitstream: "0000 0100 0111 0010 1000 1"
- Number of transmitted bits: $K = 21$

Example for Arithmetic Coding – Efficiency

- Number of written bits using arithmetic coding: $K_{AC} = 21$

- Number of bits for Elias coding

$$\begin{aligned} K_{EC} &= \left\lceil -\log_2 p(\text{“MISSISSIPPI”}) \right\rceil \\ &= \left\lceil -\log_2 \left(\frac{1}{11} \cdot \frac{4}{11} \cdot \frac{4}{11} \cdot \frac{4}{11} \cdot \frac{4}{11} \cdot \frac{4}{11} \cdot \frac{4}{11} \cdot \frac{4}{11} \cdot \frac{2}{11} \cdot \frac{2}{11} \cdot \frac{4}{11} \right) \right\rceil \\ &= \left\lceil -\log_2 (0.000\ 000\ 918\ \cdots) \right\rceil = \left\lceil 20.053 \cdots \right\rceil = 21 \end{aligned}$$

- Scalar Huffman coding

a_i	$p(a_i)$	codeword
M	1/11	**000**
I	4/11	**01**
S	4/11	**1**
P	2/11	**001**

- Bitstream:
  “**0000 1110 1110 1001 0010 1**”

- Number of written bits: $K_{HC} = 21$

Efficiency of Arithmetic Coding

- Excess rate due to down rounding of interval width

$$\Delta\ell = \left\lceil -\log_2 W_N \right\rceil - \left\lceil -\log_2 p(\boldsymbol{s}) \right\rceil < 1 + \log_2 \frac{p(\boldsymbol{s})}{W_N} \tag{166}$$

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

$$\Delta\bar{\ell} < \frac{1}{N} + \log_2 \left(1 + 2^{1-U}\right) - \log_2 \left(1 - \frac{2^{-V}}{p_{\min}}\right) \tag{167}$$

  For a derivation see Wiegand, Schwarz, page 51-52

- Example:
  - number of coded symbols $N = 1000$,
  - precision for representing probabilities $V = 16$,
  - precision for representing intervals $U = 12$,
  - minimum probablity $p_{\min} = 0.02$

  $\Longrightarrow$ Increase in codeword length is less than 0.003 bit per symbol

Binary Arithmetic Coding

- Complexity reduction: Most popular type of arithmetic coding
  - MQ-coder in JPEG 2000
  - M-coder in H.264/AVC and H.265/HEVC
- 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 in table: Unary truncated 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 and efficiency of coding unchanged due to binarization $\boldsymbol{S} \mapsto \boldsymbol{C}$

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

Comparison of Lossless Coding Techniques

Experimental determination of average codeword length

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

Tested lossless coding techniques

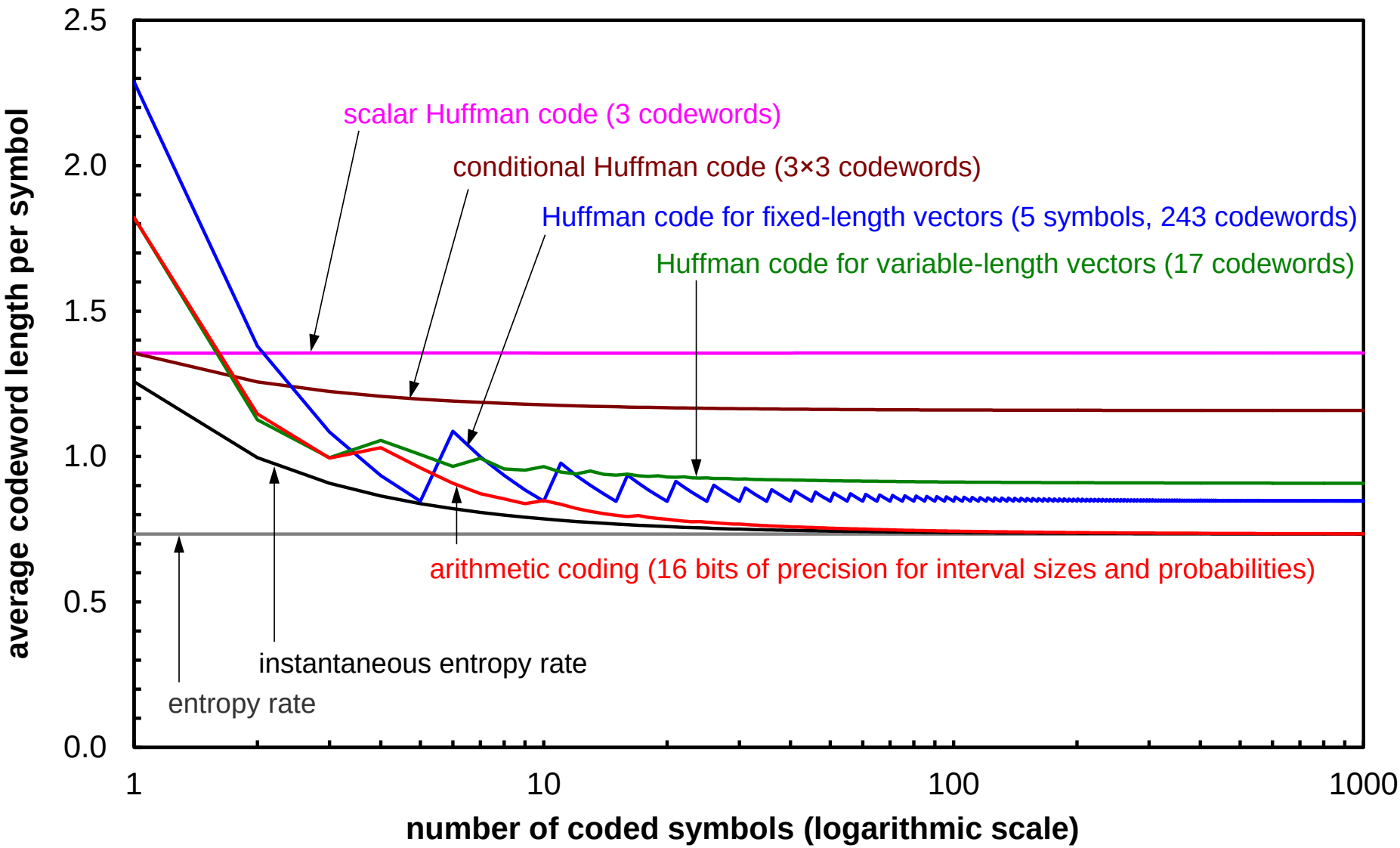
- Scalar Huffman coding (3 codewords)
- Conditional Huffman coding (9 codewords)
- Block Huffman coding of 5 symbols (243 codewords)
- Huffman coding for variable-length vectors (17 codewords)
- Arithmetic coding with $U = V = 16$

Bounds for lossless coding

- Marginal entropy $H(S)$ for coding of a single symbol
- Entropy rate $\bar{H}(\boldsymbol{S})$ for coding of infinite many symbols
- Instantaneous entropy rate $\bar{H}_{\text{inst}}(\boldsymbol{S}, L)$ for coding L symbols

$$\bar{H}_{\text{inst}}(\boldsymbol{S}, L) = \frac{1}{L} H(S_0, S_1, \ldots, S_{L-1}) \tag{168}$$

Comparison of Lossless Coding Techniques



Conditional and Adaptive Codes

- Question: How can we efficiently code sources with memory and/or with varying statistics
- Conditional Huffman coding (adaptive for instationary sources)
  - The resulting number of code tables is often too large in practice
  - Adaptation of Huffman code tables is often considered as to complex
- Block Huffman coding (adaptive for instationary sources)
  - Code tables are typically too large to be used in practice
  - Adaptation of Huffman code tables is often considered as to complex
- Conditional and adaptive arithmetic coding
  - Easy to incorporate conditional probabilities as well as varying probabilities
  - In adaptive arithmetic coding, probabilities $p(a_k)$ are estimated/adapted simultaneously at encoder and decoder
  - Statistical dependencies can be exploited using so-called context modeling techniques: Conditional probabilities $p(a_k|\boldsymbol{z}_k)$ with $\boldsymbol{z}_k$ being a context/state that is simultaneously computed at encoder and decoder based on already transmitted symbols

Forward and Backward Adaptation

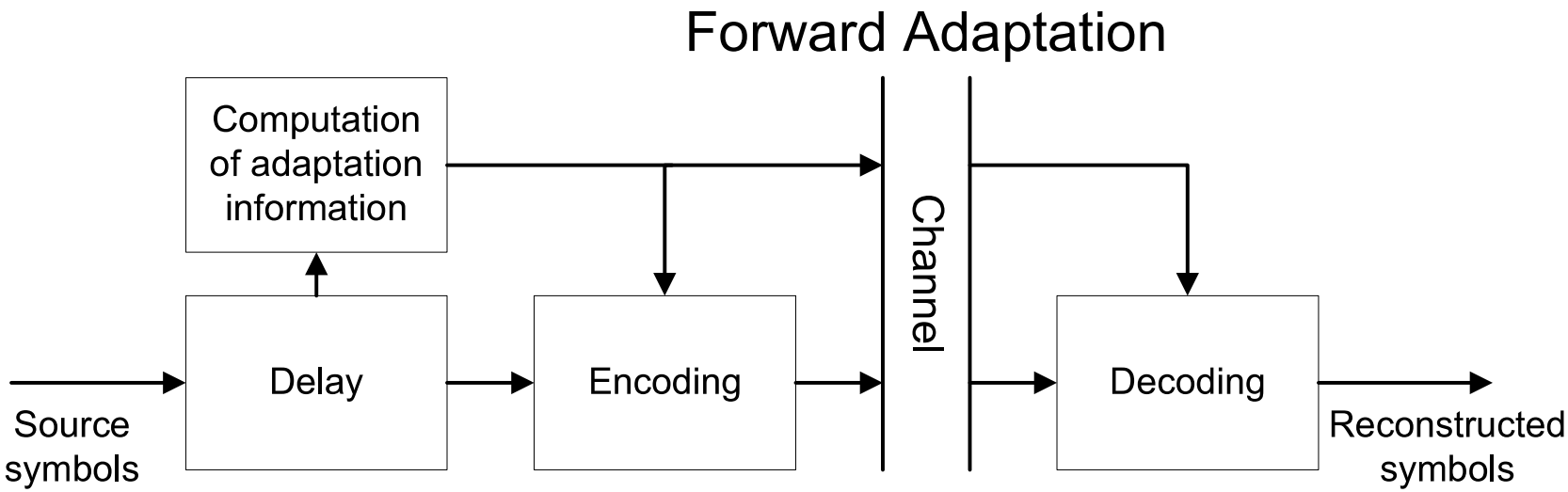
The two basic approaches for adaptation are

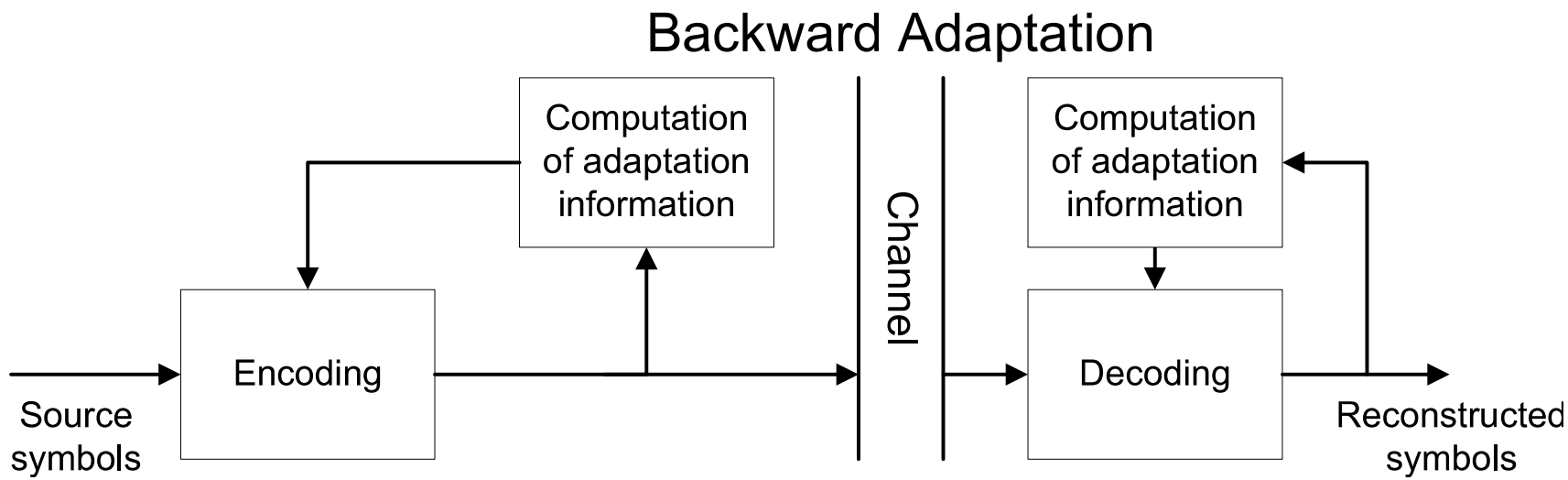
- **Forward adaptation**:
  - Gather statistics for a large enough block of source symbols
  - Transmit adaptation signal to decoder as side information
  - Disadvantage: Increased bit rate due to side information
- **Backward adaptation**:
  - Gather statistics simultaneously at coder and decoder
  - Drawback: Error resilience

With today's packet-switched transmission systems, an efficient combination of the two adaptation approaches can be achieved:

1. Gather statistics for the entire packet and provide initialization of entropy code at the beginning of the packet
2. Conduct backwards adaptation for each symbol inside the packet in order to minimize the size of the packet

Forward and Backward Adaptation





Chapter Summary

Uniquely decodable codes and bounds for lossless coding

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

Huffman codes for scalars and vectors (optimal prefix codes)

- Efficient and simple entropy coding method
- Needs a code table
- Can be inefficient for certain probabilities
- Difficult to use for sources with memory or time-varying statistics

Arithmetic coding (fixed-precision variant of Elias coding)

- Universal method for encoding strings of symbols
- Does not need a code table, but a table for storing probabilities
- Approaches entropy for long symbol sequences
- Well suited for exploiting statistical dependencies and coding of instationary sources (probability updates)

Exercise 9 – Part 1/2

Given is a Bernoulli process $\mathbf{X} = \{X_n\}$ with the alphabet $\mathcal{A} = \{a, b\}$ and the pmf $p_X(a) = 1/4$ and $p_X(b) = 3/4$.

(a) Consider Elias coding and derive the codeword for the symbol sequence "$abba$" using the iterative encoding procedure.

(b) Develop the complete code for an Elias coding of 3 symbols (i.e., determine the codewords for all symbol sequences that consist of 3 symbols).

Determine the average codeword length per symbol and compare it to the entropy rate and the average codeword length per symbol for a joint Huffman code for sequences of 3 symbols.

Is the Elias code more efficient than the Huffman code for the same number of jointly coded symbols?

(c) Decode the 3-symbol sequence $\{s_0, s_1, s_2\}$ represented by the bit string "100" using the iterative Elias decoding algorithm.

Exercise 9 – Part 2/2

(d) Consider the case in which the developed Elias code is used for coding multiple 3-symbol sequences. The codewords for the 3-symbol sequences are concatenated. Given is a bit string "10011100".

Decode the symbol sequence using the developed code table.

Decode the first three symbols (i.e., the first 3-symbol sequence) using the iterative decoding algorithm.

What do you observe?

(e) How many bits have to be used for a codeword in order to make an Elias code uniquely decodable using the iterative decoding algorithm for a sequence of codewords.

Derive a lower and upper bound for the average codeword length per symbol for the corresponding Elias code if a codeword is generated for a sequences of N symbols.

Exercise 10 – Part 1/3 (Optional Self Study)

Given is a discrete iid process $\mathbf{X} = \{X_n\}$ with the symbol alphabet $\mathcal{A} = \{\text{'M'}, \text{'I'}, \text{'S'}, \text{'P'}\}$ and the pmf

$$p_X(x) = \begin{cases} 0.1 & : \quad x = \text{'M'} \\ 0.3 & : \quad x = \text{'I'} \\ 0.4 & : \quad x = \text{'S'} \\ 0.2 & : \quad x = \text{'P'} \end{cases}$$

Consider binary arithmetic coding of the given source.

(a) Use a fixed-length code for binarizing the given source $\mathbf{X}$.

Verify on the given example that binarization does not have any impact on the coding efficiency (assuming a successive coding that achieves the entropy rate).

What binarization schemes can be used in the context of binary arithmetic coding?

Show that binarization does not change the lower bound for the average codeword length per symbol.

Exercise 10 – Part 2/3 (Optional Self Study)

(b) For arithmetic coding, the probability masses have to be represented with finite precision. Round the pmfs for the binary symbols to a precision of $V = 4$ bit.

What conditions need to be fulfilled for the rounded version of a pmf? Are these conditions fulfilled for the example?

Investigate the impact on the average codeword length per symbol of the given source $\mathbf{X}$ (assuming that the following arithmetic coding process does not have any negative impact on the coding efficiency).

(c) For arithmetic coding, the interval width has to be represented with finite precision.

Show that, for a coding of N symbols, the corresponding increase in average codeword length per arithmetically coded symbol is less than $1/N + \log_2(1 + 2^{1-U})$ bits, if U is the number of bits used for representing the interval width.

Determine the minimum precision U that is required to guarantee that the coding efficiency loss due to rounding the interval width is less than 0.1% when more than 10000 symbols X are transmitted.

Exercise 10 – Part 3/3 (Optional Self Study)

(d) Consider arithmetic coding that uses U bits for representing the interval width and V bits for representing the probability masses.

Show that the lower interval boundaries can be represented by a counter and an integer value of $U + V$ bits.

(e) Consider binary arithmetic coding for the given source $\mathbf{X}$ with fixed-length binarization, $V = 4$ bits for representing the probability masses, and $U = 4$ bits for representing the interval width.

Generate the arithmetic codeword for the symbol sequence "MISS".

Compare the length of the arithmetic codeword with the length of the codeword that would be generated by Elias coding.