<table>
<thead>
<tr>
<th>a_k</th>
<th>p_k</th>
<th>b_k</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0.16</td>
<td>111</td>
</tr>
<tr>
<td>b</td>
<td>0.04</td>
<td>0001</td>
</tr>
<tr>
<td>c</td>
<td>0.04</td>
<td>0000</td>
</tr>
<tr>
<td>d</td>
<td>0.16</td>
<td>110</td>
</tr>
<tr>
<td>e</td>
<td>0.23</td>
<td>01</td>
</tr>
<tr>
<td>f</td>
<td>0.07</td>
<td>1001</td>
</tr>
<tr>
<td>g</td>
<td>0.06</td>
<td>1000</td>
</tr>
<tr>
<td>h</td>
<td>0.09</td>
<td>001</td>
</tr>
<tr>
<td>i</td>
<td>0.15</td>
<td>101</td>
</tr>
</tbody>
</table>
Summary of Last Lecture

Prefix Codes
- Binary code trees / simple decoding process
- Instantaneous codes (suitable for complicated syntax)

Unique Decodability
- Necessary condition: Kraft inequality: \( \sum_k 2^{-\ell_k} \leq 1 \)
- Fulfilled Kraft inequality: Can always construct prefix code
- All uniquely decodable codes used in practice are prefix codes

Entropy
- Lower bound for average codeword length: \( H(p) = -\sum_k p_k \log_2 p_k \)
- Only achievable if all probability masses are negative integer powers of two
- Can always construct code with \( H(p) \leq \bar{\ell} < H(p) + 1 \)

Huffman Algorithm / Huffman Codes
- Algorithm for construction of optimal prefix codes
Conditional Codes

Random Processes with Memory

Example: Stationary Markov Process

- Statistical properties are given by conditional pmf $p(a_n|a_{n-1})$

| $a$  | $p(a|a_0)$ | $p(a|a_1)$ | $p(a|a_2)$ | $p(a)$  | Huffman code |
|------|------------|------------|------------|---------|--------------|
| $a_0$ | 0.90       | 0.15       | 0.05       | 29/45   | 1            |
| $a_1$ | 0.05       | 0.80       | 0.05       | 11/45   | 01           |
| $a_2$ | 0.05       | 0.05       | 0.60       | 1/9     | 00           |

- Average codeword length for conventional Huffman code

$$\ell_{SH} = \frac{61}{45} \approx 1.3556$$

$$H(S) \approx 1.2575$$

- Can we exploit the dependencies between successive symbols?
  - Design a Huffman code for each condition
  - Switch code table after each symbol
### Conditional Codes / Conditional Huffman Codes

### Conditional Variable-Length Codes

#### Example: Stationary Markov Process

- **Conditional code**

<table>
<thead>
<tr>
<th>$a_k$</th>
<th>$S_{n-1} = a_0$</th>
<th>$S_{n-1} = a_1$</th>
<th>$S_{n-1} = a_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_k$</td>
<td>code</td>
<td>$p_k$</td>
<td>code</td>
</tr>
<tr>
<td>$a_0$</td>
<td>0.90</td>
<td>0.15</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>$a_1$</td>
<td>0.05</td>
<td>0.80</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>01</td>
<td>1</td>
<td>00</td>
</tr>
<tr>
<td>$a_2$</td>
<td>0.05</td>
<td>0.05</td>
<td>0.60</td>
</tr>
<tr>
<td></td>
<td>00</td>
<td>00</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
p(a_0) = \frac{29}{45}, \quad p(a_1) = \frac{11}{45}, \quad p(a_2) = \frac{1}{9}
\]

- **Average codeword length for conditional code**

\[
\bar{\ell} = \sum_{\forall i} p(a_i) \cdot \bar{\ell}_i = \frac{521}{450} \approx 1.1578
\]

\[
\bar{\ell} < \bar{\ell}_{SH} \approx 1.3556
\]

\[
\bar{\ell} < H(S) \approx 1.2575
\]
Conditional Codes / Conditional Entropy

Average Codeword Length for Conditional Codes

Bounds on Minimum Average Codeword Length

- For each condition \( \{S_{n-1} = a_i\} \), we have the same relationship as in conventional variable-length coding

\[
H(S_n | a_i) \leq \bar{\ell}_i < H(S_n | a_i) + 1
\]  

(1)

where \( H(S_n | a_i) \) is the conditional entropy given the event \( \{S_{n-1} = a_i\} \),

\[
H(S_n | a_i) = H(S_n | S_{n-1} = a_i) = - \sum_{\forall k} p(a_k | a_i) \log_2 p(a_k | a_i)
\]  

(2)

- Each condition \( \{S_{n-1} = a_i\} \) occurs with the probability \( p_i = P(S_{n-1} = a_i) \)

- Hence, the resulting bounds on \( \bar{\ell} \) are given by

\[
\left( \sum_{\forall i} p_i H(S_n | a_i) \right) \leq \left( \sum_{\forall i} p_i \bar{\ell}_i \right) < \left( \sum_{\forall i} p_i H(S_n | a_i) \right) + 1
\]

(3)
Conditional Entropy

Lower bound for conditional coding

- Conditional entropy of $S_n$ given $S_{n-1}$

\[ H(S_n \mid S_{n-1}) = \sum_{\forall i} p(a_i) H(S_n \mid a_i) \]

\[ = \sum_{\forall i} p(a_i) \left( -\sum_{\forall k} p(a_k \mid a_i) \log_2 p(a_k \mid a_i) \right) \]

\[ = -\sum_{\forall i, k} p(a_k, a_i) \log_2 p(a_k \mid a_i) \quad (4) \]

\[ = \mathbb{E}\left\{ -\log_2 p(S_n \mid S_{n-1}) \right\} \quad (5) \]

- Minimum average codeword length of a conditional code is bounded by

\[ H(S_n \mid S_{n-1}) \leq \bar{\ell}_{\text{min}} < H(S_n \mid S_{n-1}) + 1 \quad (6) \]
General Conditional Coding

Arbitrary Condition
- Can use arbitrary random variable $X$ as condition
- Example: Any function of already coded symbols $X = f(S_{n-1}, S_{n-2}, \cdots)$
- Design a code (codeword table) for each possible value of $X$

Conditional Entropy
- Conditional entropy of a random variable $S$ given a random variable $X$

$$H(S \mid X) = \mathbb{E}\left\{ - \log_2 p_{S \mid X}(S \mid X) \right\} = - \sum_{s, x} p_{S \mid X}(s, x) \log_2 p_{S \mid X}(s \mid x) \quad (7)$$

Bounds on Minimum Average Codeword Length
- Minimum average codeword length for conditional coding is bounded by

$$H(S \mid X) \leq \bar{\ell}_{S \mid X} < H(S \mid X) + 1 \quad (8)$$
By using the divergence inequality, we obtain

\[ H(S \mid X) = - \sum_{s, x} p_{SX}(s, x) \log_2 p_{S \mid X}(s \mid x) \text{ with } p_{S \mid X}(s \mid x) = \frac{p_{SX}(s, x)}{p_X(x)} \]

\[ = - \sum_{s, x} p_{SX}(s, x) \log_2 \left( \frac{p_{SX}(s, x) p_S(s)}{p_X(x) p_S(s)} \right) \]

\[ = - \sum_s p_S(s) \log_2 p_S(s) - \sum_{s, x} p_{SX}(s, x) \log_2 \left( \frac{p_{SX}(s, x)}{p_X(x) p_S(s)} \right) \]

\[ = H(S) - D(p_{SX} \mid\mid p_S p_X) \]

\[ \leq H(S) \] (9)

Equality is only obtained if the random variables \( S \) and \( X \) are independent

\[ D(p_{SX} \mid\mid p_S p_X) = 0 \iff p_{SX}(s, x) = p_S(s) \cdot p_X(x) \]
### Conditional Codes / Conditional Huffman Examples

**Example: Stationary Markov Process**

#### Summary on Markov Process Example

<table>
<thead>
<tr>
<th>Conditional coding</th>
<th>Conventional coding</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_{n-1} = a_0$</td>
<td>$S_{n-1} = a_1$</td>
</tr>
<tr>
<td>$a_k$</td>
<td>$p_k$</td>
</tr>
<tr>
<td>$a_0$</td>
<td>0.90</td>
</tr>
<tr>
<td>$a_1$</td>
<td>0.05</td>
</tr>
<tr>
<td>$a_2$</td>
<td>0.05</td>
</tr>
</tbody>
</table>

$H_0 = 0.5690$, $\bar{\ell}_0 = 1.1$

$H_1 = 0.8842$, $\bar{\ell}_1 = 1.2$

$H_2 = 1.3527$, $\bar{\ell}_2 = 1.4$

$H(S_n | S_{n-1}) = 0.7331$, $\bar{\ell}_{\text{cond}} = 1.1578$

$H(S) = 1.2575$, $\bar{\ell}_{\text{conv}} = 1.3556$

→ Conditioning reduces entropy from 1.2575 to 0.7331

→ Conditioning reduces average codeword length from 1.3556 to 1.1578
Table 9-5 - coeff_token mapping to TotalCoeff(coeff_token) and TrailingOnes(coeff_token)

<table>
<thead>
<tr>
<th>TrailingOnes(coeff_token)</th>
<th>TotalCoeff(coeff_token)</th>
<th>0 &lt;= nC &lt; 2</th>
<th>2 &lt;= nC &lt; 4</th>
<th>4 &lt;= nC &lt; 8</th>
<th>8 &lt;= nC</th>
<th>nC == -1</th>
<th>nC == -2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>11</td>
<td>1111</td>
<td>0000 11</td>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0001 01</td>
<td>0010 11</td>
<td>0011 11</td>
<td>0000 00</td>
<td>0001 11</td>
<td>0001 11</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>01</td>
<td>10</td>
<td>1110</td>
<td>0000 01</td>
<td>1</td>
<td>01</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0000 0111</td>
<td>0001 11</td>
<td>0010 11</td>
<td>0001 00</td>
<td>0001 00</td>
<td>0001 110</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0001 00</td>
<td>0011 1</td>
<td>0111 1</td>
<td>0001 01</td>
<td>0001 10</td>
<td>0001 101</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>001</td>
<td>011</td>
<td>1101</td>
<td>0001 10</td>
<td>001</td>
<td>001</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>0000 0111</td>
<td>0000 111</td>
<td>0010 00</td>
<td>0010 00</td>
<td>0000 11</td>
<td>0000 0111</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0000 0110</td>
<td>0010 10</td>
<td>0110 0</td>
<td>0010 01</td>
<td>0000 011</td>
<td>0001 100</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>0000 101</td>
<td>0010 01</td>
<td>0111 0</td>
<td>0010 10</td>
<td>0000 010</td>
<td>0001 011</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0001 1</td>
<td>0101</td>
<td>1100</td>
<td>0010 11</td>
<td>0001 01</td>
<td>0000 1</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>0000 0001 11</td>
<td>0000 0111</td>
<td>0001 111</td>
<td>0011 00</td>
<td>0000 10</td>
<td>0000 0110</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>0000 0011 0</td>
<td>0001 10</td>
<td>0101 0</td>
<td>0011 01</td>
<td>0000 011</td>
<td>0000 0101</td>
</tr>
</tbody>
</table>

(continued)
Block Codes / Problem with Scalar Codes

Example: Binary Markov Process

- Binary random process $S = \{S\}$:
  
  $S = 0 \rightarrow$ white sample  
  $S = 1 \rightarrow$ black sample

- Statistics measured over a large set of examples documents:
  
  $p(0) = 0.8$  
  $p(0|0) = 0.9$

- Model: Stationary Markov process
- Determine remaining probabilities:
  
  $p(0) = 0.2 = 1 - p(0)$  
  $p(1|0) = 0.1 = 1 - p(0|0)$  
  $p(0|1) = 0.4 = p(1|0) p(0)/p(1)$  
  $p(1|1) = 0.6 = 1 - p(0|1)$

---

David A. Huffman, Associate, IEEE

**A Method for the Construction of Minimum-Redundancy Codes**

**Introduction**

One important method of transmitting messages is to transmit them in their place sequences of symbols. If there are more messages which might be sent than there are symbols available, then some of the messages must use more than one symbol. If it is assumed that each symbol requires the same time for transmission, then the time for transmission (length) of a message is directly proportional to the number of symbols associated with it. In this paper, the symbol or sequence of symbols associated with a given message will be called the "message code." The total number of messages which might be transmitted will be called the "message ensemble." The mutual agreement between the transmitter and the receiver about the meaning of the code for each message of the ensemble will be called the "ensemble code."

Probability of the most familiar ensemble code was stated in the phrase "one if by land and two if by sea." In this case, the message ensemble consisted of the two individual messages "by land" and "by sea," and the message code was "one" and "two."

In order to formalize the requirements of an ensemble code, the coding symbols will be represented by numbers. Thus, if there are $D$ different types of symbols to be used in coding, they will be represented by the digits $0, 1, 2, \ldots, (D - 1).$ For example, a ternary code will be constructed using the three digits 0, 1, and 2 as coding symbols.

The number of messages in the ensemble will be called $N$. Let $P(0)$ be the probability of the $i$th message. Then

$$
p(i) = 1 - P(0) = 1 - \frac{i}{N}
$$

The length of a message, $L(i)$, is the number of coding digits assigned to it. Therefore, the average message length is

$$
L = \frac{\sum P(i) L(i)}{N}
$$

The term "redundancy" has been defined by Shannon$^3$ as a property of codes. A "minimum-redundancy code" will be defined here as an ensemble code which, for a message ensemble consisting of a finite number of members, $N$, and for a given number of coding digits, $D$, yields the lowest possible average message length.

---

**Notes:**

1. C. E. Shannon and R. M. Fano have developed ensemble coding procedures for the purpose of proving that the average number of binary digits required per message approaches from above the average amount of information per message. Their coding procedures are not optimum, but approach the optimum behavior when $N$ approaches infinity. Some work has been done by T. A. Storer deriving a coding method which gives an average code length as close as possible to the ideal when the ensemble contains a finite number of members. However, up to the present time, no definite procedure has been suggested for the construction of such a code.
Example: Conventional and Conditional Coding

Black and White Document Scan (continued)

<table>
<thead>
<tr>
<th>( S_{n-1} = 0 )</th>
<th>( S_{n-1} = 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_k )</td>
<td>( p_k )</td>
</tr>
<tr>
<td>0</td>
<td>0.9</td>
</tr>
<tr>
<td>1</td>
<td>0.1</td>
</tr>
</tbody>
</table>

\[ H_0 = 0.4690 \]
\[ \bar{\ell}_0 = 1 \]

\[ H_1 = 0.9710 \]
\[ \bar{\ell}_1 = 1 \]

\[ H(S_n | S_{n-1}) = 0.5694 \]
\[ \bar{\ell}_{\text{cond}} = 1 \]

<table>
<thead>
<tr>
<th>( p_k )</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.8</td>
<td>0</td>
</tr>
<tr>
<td>0.2</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ H(S) = 0.7219 \]
\[ \bar{\ell}_{\text{conv}} = 1 \]

- Conditioning does not improve coding efficiency in our case
- No codeword can be shorter than one bit, hence \( \bar{\ell} \geq 1 \)
- Problem for sources with probability masses \( \gg 0.5 \)
- **How can we increase coding efficiency?**
Variable-Length Coding For Blocks of Symbols

Block Codes

- Design variable length code for blocks of \( N > 2 \) symbols
- The \( N \) symbols of a block are jointly coded
- Optimal block code: Huffman algorithm for \( N \)-dimensional joint pmf

\[
p(s_0, s_1, \ldots, s_{N-1}) = P(S_0 = s_0, S_1 = s_1, \ldots, S_{N-1} = s_{N-1})
\]

Block Huffman Coding for Black and White Document Scans

\( N = 2 \) symbols

\[
\begin{array}{lll}
 s_0s_1 & p(s_0, s_1) & \text{codewords} \\
 00 & 0.72 & 1 \\
 01 & 0.08 & 010 \\
 10 & 0.08 & 011 \\
 11 & 0.12 & 00 \\
\end{array}
\]

\[
\bar{\ell}_2 = 1.44 \\
\bar{\ell} = \bar{\ell}_2/2 = 0.72
\]

\( N = 3 \) symbols

\[
\begin{array}{lll}
 s_0s_1s_2 & p(s_0, s_1, s_2) & \text{codewords} \\
 000 & 0.648 & 1 \\
 001 & 0.072 & 000 \\
 010 & 0.032 & 01000 \\
 011 & 0.048 & 0101 \\
 100 & 0.072 & 001 \\
 101 & 0.008 & 01001 \\
 110 & 0.048 & 0110 \\
 111 & 0.072 & 0111 \\
\end{array}
\]

\[
\bar{\ell} = 0.65
\]
**Block Entropy**

### Bounds on minimum average codeword length

- Let $\bar{\ell}_N$ denote the average codeword length for joint coding of $N$ symbols.
- For combined alphabet: Same relationship as in scalar case.
- Minimum average codeword length is bounded by

$$E\{-\log_2 p(S_0, \cdots, S_{N-1})\} \leq \bar{\ell}_N < E\{-\log_2 p(S_0, \cdots, S_{N-1})\} + 1 \quad (11)$$

### Block Entropy

- Lower bound for average codeword length for $N$ symbols

$$H_N(S) = H(S_0, S_1, \cdots, S_{N-1})$$

$$= E\{-\log_2 p(S_0, \cdots, S_{N-1})\}$$

$$= - \sum_{a_0, a_1, \cdots, a_{N-1}} p(a_0, a_1, \cdots, a_{N-1}) \log_2 p(a_0, a_1, \cdots, a_{N-1}) \quad (12)$$
Bounds on Minimum Average Codeword Length

Variable Length Coding of Fixed-Length Symbol Blocks

- Bounds for minimum average codeword length per $N$ symbols
  \[ H_N(S) \leq \bar{\ell}_N < H_N(S) + 1 \]  
  (13)

- Bounds for minimum average codeword length per symbol
  \[ \frac{H_N(S)}{N} \leq \bar{\ell} < \frac{H_N(S)}{N} + \frac{1}{N} \]  
  (14)

Chain Rule for Entropies

- Remember: Conditional probabilities
  \[ p(s_0, s_1, \ldots, s_{N-1}) = p(s_0) p(s_1 | s_0) p(s_2 | s_0, s_1) \cdots p(s_{N-1} | s_0, \ldots, s_{N-2}) \]

- Consequence: Chain rule for entropies
  \[ H(S_0, S_1, \ldots, S_{N-1}) = H(S_0) + H(S_1 | S_0) + H(S_2 | S_0, S_1) + \cdots \]
  \[ \cdots + H(S_{N-1} | S_0, \ldots, S_{N-2}) \]  
  (15)
Properties of Block Entropy

Block Entropy vs Conditional Entropy

- Remember: Chain rule

\[ H(S_0, S_1, \cdots, S_{N-1}) = H(S_0) + H(S_1 | S_0) + H(S_2 | S_0, S_1) + \cdots + H(S_{N-1} | S_0, \cdots, S_{N-2}) \]

- Remember: Conditioning never increases entropy

\[ H(S_0, S_1, \cdots, S_{N-1}) \geq N \cdot H(S_{N-1} | S_0, \cdots, S_{N-2}) \] (16)

Block Entropy for Different Block Sizes

- Remember: Conditional probabilities

\[ p(s_0, s_1, \cdots, s_{N-1}) = p(s_0, \cdots, s_{N-2}) p(s_{N-1} | s_0, \cdots, s_{N-2}) \]

- Consequence for block entropy

\[ H(S_0, \cdots, S_{N-1}) = H(S_0, \cdots, S_{N-2}) + H(S_{N-1} | S_0, \cdots, S_{N-2}) \] (17)
Increasing Block Size Never Increases Lower Bound

Effect of Increasing Block Size

- For stationary random processes, we have shown

\[ H_N(S) = H_{N-1}(S) + H(S_n \mid S_{n-1}, \cdots, S_{n-N+1}) \]

\[ H_N(S) \geq N \cdot H(S_n \mid S_{n-1}, \cdots, S_{n-N+1}) \]

- Combining these relationships yields

\[ N \cdot H_N(S) \leq N \cdot H_{N-1}(S) + H_N(S) \]

\[ (N - 1) \cdot H_N(S) \leq N \cdot H_{N-1} \]

⇒ Hence: Increasing block size never increases lower bound

\[ \frac{H_N(S)}{N} \leq \frac{H_{N-1}(S)}{N - 1} \]  \hspace{1cm} (18)

⇒ Equality if \( H(S_n \mid S_{n-1}, \cdots) = H(S_n) \) (iid processes)
Fundamental Lossless Source Coding Theorem

**Entropy Rate**

- **Definition of entropy rate**

  \[
  \bar{H}(\mathbf{S}) = \lim_{N \to \infty} \frac{H(S_0, \cdots, S_{N-1})}{N} = \lim_{N \to \infty} \frac{H_N(\mathbf{S})}{N} \tag{19}
  \]

  - Limit (for \( N \to \infty \)) of lower bound for block coding
  - The limit always exists for stationary random sources

**Fundamental Lossless Source Coding Theorem**

- Average codeword length for all lossless codes is bounded by

  \[
  \bar{\ell} \geq \bar{H}(\mathbf{S}) = \lim_{N \to \infty} \frac{H_N(\mathbf{S})}{N} \tag{20}
  \]

  - Asymptotically achievable with block Huffman coding for \( N \to \infty \)
  - Size of codeword tables exponentially increases with \( N \)
Entropy Rate for Special Sources

IID Processes

- Use chain rule for entropies

\[
\bar{H}(S) = \lim_{N \to \infty} \frac{1}{N} H(S_0, S_1, \ldots, S_{N-1})
\]

\[
= \lim_{N \to \infty} \frac{1}{N} \left( H(S_0) + H(S_1 | S_0) + H(S_2 | S_0, S_1) + \cdots \right)
\]

\[
= \lim_{N \to \infty} \frac{1}{N} \sum_{n=0}^{N-1} H(S_n)
\]

\[
= \lim_{N \to \infty} \frac{1}{N} \left( N \cdot H(S) \right)
\]

\[
= H(S)
\]

Note: Block Huffman coding may still improve coding efficiency compared to scalar Huffman coding
Entropy Rate for Special Sources

Stationary Markov Processes

- Use chain rule for entropies

\[
\bar{H}(S) = \lim_{N \to \infty} \frac{1}{N} H(S_0, S_1, \cdots, S_{N-1})
\]

\[
= \lim_{N \to \infty} \frac{1}{N} \left( H(S_0) + H(S_1 | S_0) + H(S_2 | S_0, S_1) + \cdots \right)
\]

\[
= \lim_{N \to \infty} \frac{1}{N} \left( H(S) + (N - 1) \cdot H(S_n | S_{n-1}) \right)
\]

\[
= \lim_{N \to \infty} \frac{H(S)}{N} + \lim_{N \to \infty} \frac{N - 1}{N} H(S_n | S_{n-1})
\]

\[
= H(S_n | S_{n-1}) \quad (22)
\]

- Note: Can also use conditional block Huffman coding
- Switch block code based on conditional variable
Example: Stationary Markov Process

## Markov Source

| $a$   | $p(a|a_0)$ | $p(a|a_1)$ | $p(a|a_2)$ | $H(S)$ | $\bar{H}(S)$ |
|-------|------------|------------|------------|--------|--------------|
| $a_0$ | 0.90       | 0.15       | 0.05       | 1.2575 | 0.7331       |
| $a_1$ | 0.05       | 0.80       | 0.05       |        |              |
| $a_2$ | 0.05       | 0.05       | 0.60       |        |              |

<table>
<thead>
<tr>
<th>$N$</th>
<th>$\frac{H_N(S)}{N}$</th>
<th>$\bar{\ell} = \frac{\bar{\ell}_N}{N}$</th>
<th>number of codewords</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.2575</td>
<td>1.3556</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>0.9953</td>
<td>1.0094</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>0.9079</td>
<td>0.9150</td>
<td>27</td>
</tr>
<tr>
<td>4</td>
<td>0.8642</td>
<td>0.8690</td>
<td>81</td>
</tr>
<tr>
<td>5</td>
<td>0.8380</td>
<td>0.8462</td>
<td>243</td>
</tr>
<tr>
<td>6</td>
<td>0.8205</td>
<td>0.8299</td>
<td>729</td>
</tr>
<tr>
<td>7</td>
<td>0.8080</td>
<td>0.8153</td>
<td>2187</td>
</tr>
<tr>
<td>8</td>
<td>0.7987</td>
<td>0.8027</td>
<td>6561</td>
</tr>
<tr>
<td>9</td>
<td>0.7914</td>
<td>0.7940</td>
<td>19683</td>
</tr>
</tbody>
</table>

Scalar Huffman code:  
\[ \bar{\ell} = 1.3556 \]

Conditional Huffman code:  
\[ \bar{\ell} = 1.1578 \]
Example for Block Huffman Code: CBF in MPEG-2

\[
\begin{array}{cccc}
Y & 0/1 & 0/1 & 0/1 \\
   & 0/1 & 0/1 & 0/1 \\
\end{array}
\]

\[
\begin{array}{cccc}
Cb & 0/1 & 0/1 & \\
Cr & 0/1 & & \\
\end{array}
\]

coded_block_pattern = xxxxxx (bit mask)  
(values: 0..63)

Table B.9 – Variable length codes for coded_block_pattern

<table>
<thead>
<tr>
<th>coded_block_pattern VLC code</th>
<th>cbp</th>
<th>coded_block_pattern VLC code</th>
<th>cbp</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>60</td>
<td>0001 1100</td>
<td>35</td>
</tr>
<tr>
<td>1101</td>
<td>4</td>
<td>0001 1011</td>
<td>13</td>
</tr>
<tr>
<td>1100</td>
<td>8</td>
<td>0001 1010</td>
<td>49</td>
</tr>
<tr>
<td>1011</td>
<td>16</td>
<td>0001 1001</td>
<td>21</td>
</tr>
<tr>
<td>1010</td>
<td>32</td>
<td>0001 1000</td>
<td>41</td>
</tr>
<tr>
<td>1001 1</td>
<td>12</td>
<td>0001 0111</td>
<td>14</td>
</tr>
<tr>
<td>1001 0</td>
<td>48</td>
<td>0001 0110</td>
<td>50</td>
</tr>
<tr>
<td>1000 1</td>
<td>20</td>
<td>0001 0101</td>
<td>22</td>
</tr>
<tr>
<td>1000 0</td>
<td>40</td>
<td>0001 0100</td>
<td>42</td>
</tr>
<tr>
<td>0111 1</td>
<td>28</td>
<td>0001 0011</td>
<td>15</td>
</tr>
<tr>
<td>0111 0</td>
<td>44</td>
<td>0001 0010</td>
<td>51</td>
</tr>
</tbody>
</table>

(continued)
V2V Codes

Generalization of Block Codes

- Assign codewords to **symbol sequences of variable length**
- How to select symbol sequences?
  - All messages must be representable by symbol sequences
  - Desirable: Redundancy-free set of symbol sequences

Examples

- Consider binary symbol alphabet \( A = \{a, b\} \)

<table>
<thead>
<tr>
<th>code A</th>
<th>code B</th>
<th>code C</th>
</tr>
</thead>
<tbody>
<tr>
<td>aaaaa 0</td>
<td>aaa 000</td>
<td>aaaaa 0</td>
</tr>
<tr>
<td>aaab 10</td>
<td>aa 01</td>
<td>aaab 10</td>
</tr>
<tr>
<td>aab 110</td>
<td>a 1</td>
<td>aab 110</td>
</tr>
<tr>
<td>bba 1110</td>
<td>b 0010</td>
<td>ab 1110</td>
</tr>
<tr>
<td>ba 1111</td>
<td>bb 0011</td>
<td>b 1111</td>
</tr>
<tr>
<td>abbbb... ?</td>
<td>redundant!</td>
<td>suitable</td>
</tr>
</tbody>
</table>
Suitable Set of Variable-Length Symbol Sequences

Finite alphabet of $M$ letters

- Select symbol sequences that are representable by full $M$-ary tree

→ All messages are representable by a concatenation of symbol sequences
→ Redundancy-free set of symbol sequences
→ Instantaneous encodable codes
### V2V Codes as Double Tree

#### IID Source

<table>
<thead>
<tr>
<th>symbol</th>
<th>probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0.80</td>
</tr>
<tr>
<td>b</td>
<td>0.15</td>
</tr>
<tr>
<td>c</td>
<td>0.05</td>
</tr>
</tbody>
</table>

#### Entropy Rate
- \( \tilde{H}(S) = 0.88418 \) (marginal entropy)
- Scalar Huffman: \( \bar{\ell} = 1.2 \) (3 codewords)
- 2-symbol blocks: \( \bar{\ell} = 0.93375 \) (9 codewords)
- V2V code: \( \bar{\ell} = 0.88934 \) (7 codewords)
- Redundancy: \( \varrho = 0.00516 \) (0.58\%)
V2V Codes / Average Codeword Length

**Average Codeword Length**

**V2V Code Design**

- Choose set of symbol sequences that are representable by full $M$-ary tree
- Determine pmf of symbol sequences (leaf nodes of the $M$-ary tree)
- Design Huffman code for pmf of variable-length symbol sequences

**Average Codeword Length of V2V Codes**

Given the pmf for the leaf nodes, the average codeword length is given by

$$\bar{\ell} = \frac{\sum_{k=0}^{L-1} p_k \ell_k}{\sum_{k=0}^{L-1} p_k n_k} = \frac{\text{average codeword length per sequence}}{\text{average number of symbols per sequence}}$$

(23)

with

- $L$: number of symbols sequences (leaf nodes)
- $p_k$: probability of $k$-th symbol sequence
- $\ell_k$: length of codeword assigned to $k$-th symbol sequence
- $n_k$: number of symbols in $k$-th symbol sequence
**PMF for Symbol Sequences**

**How to determine pmf for variable-length sequences?**

- In general: Probability that a new symbol sequence starts depends on previous symbols in message

\[
p(a) = p(a_0 | B) p(a_1 | a_0, B) \cdots p(a_{K-1} | a_0, \cdots, a_{K-2}, B)
\] (24)

with \( B \) being the event that the preceding message symbols were coded using a complete symbol sequence of the given set (V2V code tree)

- Conditional pmfs \( p(a_m | a_0, \cdots, a_{m-1}, B) \) are given by
  - Conditional pmfs \( p(a_m | a_0, \cdots, a_{m-1}) \) of the random process
  - Structure of the V2V code (\( M \)-ary symbol tree)

**IID Processes**

- No dependencies on previous symbols

\[
p(a) = p(a_0) p(a_1) \cdots p(a_{K-1})
\] (25)
PMF for Symbol Sequences

Markov Processes

- Probability of a symbol sequence \( \mathbf{a} = (a_0, a_1, \cdots, a_{K-1}) \) is given by
  \[
  p(\mathbf{a}) = p(a_0 | \mathcal{B})p(a_1 | a_0)p(a_2 | a_1) \cdots p(a_{K-1} | a_{K-2})
  \]  
  (26)

- Probability that a new symbol sequence starts with letter \( a_m \) is given by
  \[
  p(a_m | \mathcal{B}) = \sum_{k=0}^{L-1} p(a_m | a_{nk-1}^k) \cdot p(a_{nk-1}^k | a_{nk-2}^k) \cdots p(a_1^k | a_0^k) \cdot p(a_0^k | \mathcal{B})
  \]  
  (27)

  with
  - \( L \) : number of symbol sequences
  - \( n_k \) : number of symbols in \( k \)-th symbol sequence
  - \( a_n^k \) : \( n \)-th symbol in \( k \)-th symbol sequence

- Together with the condition \( \sum_m p(a_m | \mathcal{B}) = 1 \) we have a linear equation system with a unique solution

\[\rightarrow\] Can calculate all unknown probabilities \( p(a_n | \mathcal{B}) \) and thus also the pmf for the symbol sequences (leaf nodes)
Example for Binary Markov Source

|       | a | p(a|0) | p(a|1) | p(a) |
|-------|---|-------|-------|------|
| 0     |   | 0.9   | 0.4   | 0.8  |
| 1     |   | 0.1   | 0.6   | 0.2  |

sequences

\[
p(0|B) = p(0|B) \cdot p(0|0) \cdot p(0|0) \cdot p(0|0) \cdot p(0|0) \cdot p(0|0) + \]
\[
p(0|B) \cdot p(0|0) \cdot p(0|0) \cdot p(0|0) \cdot p(1|0) \cdot p(0|1) + \]
\[
p(0|B) \cdot p(0|0) \cdot p(0|0) \cdot p(1|0) \cdot p(0|1) + \]
\[
p(0|B) \cdot p(0|0) \cdot p(1|0) \cdot p(0|1) + \]
\[
p(1|B) \cdot p(0|1) \cdot p(0|1) + \]
\[
p(1|B) \cdot p(1|1) \cdot p(0|1) \cdot p(0|0) + \]
\[
p(1|B) \cdot p(1|1) \cdot p(1|1) \cdot \]

\[
p(0|B) = 0.72805 \cdot p(0|B) + 0.72 \cdot p(1|B) \]
\[
= 0.72805 \cdot p(0|B) + 0.72 \cdot (1 - p(0|B)) \]
\[
\implies p(0|B) = 0.725842 \quad \implies p(1|B) = 0.274157 \]
Example for Binary Markov Source

Black and White Document Scan (continued)

- Calculate pmf for symbol sequences and design Huffman code

\[ p(a_0, a_1, \cdots, a_{k-1}) = p(a_0 | B) \cdot p(a_1 | a_0) \cdots p(a_{k-1} | a_{k-2}) \]

<table>
<thead>
<tr>
<th>sequences</th>
<th>probabilities</th>
<th>Huffman code</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>0.476226</td>
<td>1</td>
</tr>
<tr>
<td>00001</td>
<td>0.052914</td>
<td>0100</td>
</tr>
<tr>
<td>0001</td>
<td>0.058793</td>
<td>0101</td>
</tr>
<tr>
<td>001</td>
<td>0.065326</td>
<td>0110</td>
</tr>
<tr>
<td>01</td>
<td>0.072584</td>
<td>0111</td>
</tr>
<tr>
<td>10</td>
<td>0.109663</td>
<td>001</td>
</tr>
<tr>
<td>110</td>
<td>0.065798</td>
<td>0000</td>
</tr>
<tr>
<td>111</td>
<td>0.098697</td>
<td>0001</td>
</tr>
</tbody>
</table>

\[ \bar{H}(S) = 0.59049 \]
\[ \bar{\ell} = 0.62561 \]
\[ \varrho = 0.03512 \ (5.9\%) \]

⇒ More efficient than block Huffman code with the same number of codewords

\[ \bar{\ell} = 0.65 \quad \Rightarrow \quad \varrho = 0.05951 \ (10.1\%) \]
Optimal V2V Codes?

Optimization Problem

- Best V2V code for given maximum number of codewords?
- No known design algorithm
- Exhaustive search over all possible symbol trees

Exhaustive V2V code design

- Investigate all symbol trees with a maximum number of leaf nodes
  - Create symbol tree
  - Calculate probabilities for leaf nodes (symbol sequences)
  - Determine codewords using Huffman algorithm
  - Calculate average codeword length
- Choose symbol tree (and code) that minimizes average codeword length
- Extremely complex
### Example for Optimal V2V codes

**Markov Source**

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

\[ H(S) = 1.2575 \]
\[ \bar{H}(S) = 0.7331 \]

**V2V:**

<table>
<thead>
<tr>
<th>$N_C$</th>
<th>$\bar{\ell}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1.1784</td>
</tr>
<tr>
<td>7</td>
<td>1.0551</td>
</tr>
<tr>
<td>9</td>
<td>1.0049</td>
</tr>
<tr>
<td>11</td>
<td>0.9733</td>
</tr>
<tr>
<td>13</td>
<td>0.9412</td>
</tr>
<tr>
<td>15</td>
<td>0.9293</td>
</tr>
<tr>
<td>17</td>
<td>0.9074</td>
</tr>
<tr>
<td>19</td>
<td>0.8980</td>
</tr>
<tr>
<td>21</td>
<td><strong>0.8891</strong></td>
</tr>
</tbody>
</table>

$N_C$: number of codewords

**Scalar Huffman code:**

\[ \bar{\ell} = 1.3556 \]

**Conditional Huffman code:**

\[ \bar{\ell} = 1.1578 \]

**Block Huffman code:**

\[ N_C = 9 : \quad \bar{\ell} = 1.0094 \]
\[ N_C = 27 : \quad \bar{\ell} = 0.9150 \]
In Practice

- Typically: Only structured V2V codes
- Set of symbol sequences follows a certain structure

Well-Known Example: Run-Level Coding

- Often: Long sequences of symbols equal to zero
- Map sequence a symbols (transform coefficients) into (run,level) pairs, including a special end-of-block (eob) symbol
  - **level**: value of next non-zero symbol
  - **run**: number of zero symbols that precede next non-zero symbol
  - **eob**: all following symbols are equal to zero (end-of-block)

- Assign codewords to (run,level) pairs (including eob symbol)

- **Example**:
  64 symbols: 5 3 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 ...
  (run,level) pairs: (0,5) (0,3) (3,1) (1,1) (2,1) (eob)
## Run-Level Coding in MPEG-2 Video

### Table B.14 – DCT coefficients Table zero

<table>
<thead>
<tr>
<th>Variable length code (Note 1)</th>
<th>Run</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 (Note 2)</td>
<td>End of Block</td>
<td></td>
</tr>
<tr>
<td>1 s (Note 3)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>11 s (Note 4)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>011 s</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0100 s</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0101 s</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0010 1 s</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>0011 1 s</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>0011 0 s</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>0001 10 s</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0001 11 s</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>0001 01 s</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>0001 00 s</td>
<td>7</td>
<td>1</td>
</tr>
</tbody>
</table>

*(continued)*
Adaptive VLC codes

In Practice

- Messages with different properties
  - Classical music has other properties than heavy metal
  - MRT images have other properties than natural images
  - Cartoon movies have other properties than documentaries
- Messages itself have typically instationary statistical properties
  - Optimize code tables for actual data

Adaptation of Variable-Length Codes

Two basic approaches

1. Forward adaptation
   - Transmit adaptation signal (pmf, code, ...) as side information
2. Backward adaptation
   - Adapt code simultaneously at encoder and decoder

Possible to combine forward and backward adaptation
Forward Adaptation

Principle

- Gather statistics for large enough block of source symbols
- Transmit adaptation signal to decoder (e.g., at start of message)
  - Probability mass function
  - Codeword table
- Disadvantage: Increased bit rate due to side information

Example: JPEG: Transmission of codeword table
**Backward Adaptation**

**Principle**

- Gather statistics at both encoder and decoder
- Adapt code simultaneously at encoder and decoder during coding
  - Probability mass functions
  - Codeword tables
- Disadvantage: Decreased error robustness

Example: H.264/AVC and H.265/HEVC: Adaptive arithmetic coding
Variable-Length Codes

- Optimal code for given pmf: Huffman code
- Discussed: Scalar codes, conditional codes, block codes, V2V codes
- Scalar and conditional codes can be very inefficient
- Entropy rate can be asymptotically achieved using block codes ($N \rightarrow \infty$)
- Impractical: Codeword tables grow exponentially with $N$

Practical Block Codes

- **Shannon-Fano-Elias codes**
  - Sub-optimal block codes (still close to optimal for large $N$)
  - Iterative construction of codewords (no need to store codeword table)

- **Arithmetic codes**
  - Fixed-precision variant of Shannon-Fano-Elias codes
  - State-of-the art in lossless coding (very flexible)
Shannon-Fano-Elias Coding: Frameork

Random Process Model

- Assume stationary discrete random process $S = \{S_n\}$
- Statistical properties are given by $N$-th order joint pmf

$$p(s) = P(S = s) = P(S_0 = s_0, S_1 = s_1, \cdots, S_{N-1} = s_{N-1})$$ (28)

- Messages $s^{(L)} = \{s_0, s_1, s_2, \cdots, s_{L-1}\}$ of $L$ symbols are finite-length realizations of random process $S$

Coding Framework

- Code blocks of $N$ symbols ($N$ is known to encoder and decoder)
- Consider two configurations:
  1. $N < L$: Message is split into multiple blocks of symbols
     - Prefix code
  2. $N = L$: All symbols of a message are jointly coded
     - No prefix code required
Shannon-Fano-Elias Coding

**Basic Idea**

- Order all symbol sequences with \( N \) symbols: \( s_0, s_1, s_2, \cdots \)
- Each symbol sequence is associated with an interval of the cdf \( F(s) \)
- Transmit any number (as binary fraction) inside the corresponding interval
  - Required number of bits depends on probability of symbol sequence

Transmitted binary fraction: \( 0.011_b \) (codeword “011”)

\[
p(s_k) = F(s_k) - F(s_{k-1})
\]
Mapping of Symbol Sequences to Intervals

Order of Symbol Sequences

- Require a defined order of symbol sequences (with \( N \) symbols)
- Order \( \{s_0, s_1, s_2, \cdots \} \) must be known to encoder and decoder

Mapping to Intervals

- Each symbol sequence \( s_k \) is mapped to an **half-open interval** \( I(s_k) \subset [0, 1) \)
  \[
  I(s_k) = \left[ L(s_k), U(s_k) \right) = \left[ L(s_k), L(s_k) + W(s_k) \right)
  \]
  (29)

- Intervals \( I(s_k) \) can be characterized by
  - **lower interval boundary** \( L(s_k) \)
    \[
    L(s_k) = F(s_{k-1}) = P(S < s_k) = \sum_{\forall i < k} p(s_i)
    \]
    (30)
  - **interval width** \( W(s_k) \)
    \[
    W(s_k) = F(s_k) - F(s_{k-1}) = P(S = s_k) = p(s_k)
    \]
    (31)
Unique Identification of Intervals

Disjoint Intervals

- Half-open intervals $\mathcal{I}(s_k)$ are disjoint by definition
- All real numbers $v \in [0, 1)$ belong to exactly one interval

Representative Number Inside Interval

- Transmit any number $v \in \mathcal{I}(s_k)$ for uniquely identifying
  - the interval $\mathcal{I}(s_k)$ and, thus,
  - the symbol sequence $s_k$
- Represent number $v \in \mathcal{I}(s_k)$ as binary fraction with $K$ bits of precision
  \[
  v = (0.b_0 b_1 b_2 \cdots b_{K-1})_b = \sum_{i=0}^{K-1} b_i \cdot 2^{-(i+1)}
  \]
  \[\text{(32)}\]
- Number $v$ is an integer multiple of $2^{-K}$
- Codeword is given by bit sequence $b = \{b_0, b_1, b_2, \cdots, b_{K-1}\}$ of $K$ bits
How Many Bits for Identifying an Interval?

**Required Number of Bits**

- Distance between successive binary fractions of $K$ bits is $2^{-K}$

- For guaranteeing that a binary fraction of $K$ bits falls inside an interval $I(s_k)$ of width $W(s_k)$, we require

$$W(s_k) \geq 2^{-K}$$

$$K \geq -\log_2 W(s_k)$$ (33)

- Hence, we choose

$$K = K(s_k) = \lceil -\log_2 W(s_k) \rceil = \lceil -\log_2 p(s_k) \rceil$$ (34)
How to Select Interval Representative and Codeword?

**Interval Representative**

- Round up lower interval boundary $L$ to next binary fraction of $K$ bits
- For interval $I = [L, L + W)$, choose binary number $v$ according to
  \[ v = \lceil L \cdot 2^K \rceil \cdot 2^{-K} \quad \text{with} \quad K = \lceil - \log_2 W \rceil \] (35)

**Codewords**

- $K$ fractional bits of interval representative $v = (0.b_0 b_1 b_2 \cdots b_{K-1})_b$
- Binary representation $[b_0 b_1 \cdots b_{K-1}]$ with $K$ bits of integer number
  \[ z = \lceil L \cdot 2^K \rceil = v \cdot 2^K \] (36)
Shannon-Fano-Elias Encoding: Illustration

- Codeword $b(s_k)$: binary representation of $z$ with $K$ bits
- Integer part: $z = \lceil L \cdot 2^K \rceil$
- Number of bits: $K = \lceil -\log_2 W \rceil$
- $v = \lceil L \cdot 2^K \rceil \cdot 2^{-K}$
- $\mathcal{I}(s_k) = [L, L+W)$
- $L = \sum_{i<k} p(s_i)$
- $W = p(s_k)$
Shannon-Fano-Elias Encoding: Summary

**Determination of Codewords**

- Given: Ordered set of symbol sequences \( \{s_k\} \) with associated pmf \( \{p_k\} \)
- Construct codeword \( b_k = b(s_k) \) for any particular sequence \( s_k \) by
  1. Determine interval width \( W_k \) and lower interval boundary \( L_k \)
     \[
     W_k = p_k \quad (37)
     \]
     \[
     L_k = \sum_{i<k} p_i \quad (38)
     \]
  2. Determine codeword length \( K_k \)
     \[
     K_k = \lceil -\log_2 W_k \rceil \quad (39)
     \]
  3. Determine representative integer \( z_k \)
     \[
     z_k = \lceil L_k \cdot 2^{K_k} \rceil \quad (40)
     \]
  4. Codeword \( b_k \): Binary representation of \( z_k \) with \( K_k \) bits
Shannon-Fano-Elias Decoding: Illustration

read codeword \( b \): binary representation of \( z \) with \( K \) bits

representative value:
\[ v = z \cdot 2^{-K} \]

decoding process: Compare \( v \) with upper interval boundaries \( U = L + W \) in increasing order

\[ U_0 \leq v \]
\[ U_{k-1} \leq v \]
\[ U_k > v \]

\( U_k = \sum_{i \leq k} p(s_i) \)

\( s_0 \) \( \cdots \) \( s_{k-1} \) \( s_k \) \( s_{k+1} \) \( \cdots \)
Shannon-Fano-Elias Decoding: Summary

Decoding of a Symbol Sequence

1. **Given:** Ordered set of symbol sequences \( \{s_k\} \) with associated pmf \( \{p_k\} \)

2. **Read codeword** \( b \): Binary representation of integer \( z \) with \( K \) bits

3. **Initialization of iterative decoding**
   - representative value: \( v = z \cdot 2^{-K} \)
   - iteration index: \( k = 0 \)
   - upper interval boundary: \( U_0 = L_0 + W_0 = p_0 \)

4. **Compare** \( v \) with \( U_k \)
   - If \( v < U_k \)
     - Output decoded symbol sequence \( s_k \)
     - Terminate decoding
   - Otherwise (\( v \geq U_k \))
     - Update iteration index: \( k = k + 1 \)
     - Update upper interval boundary: \( U_k = U_{k-1} + p_k \)
     - Goto step 3
Summary: Extended Variable-Length Coding

Conditional Codes
- Switching between codeword tables (depending on previous symbol(s))
- Bound for average codeword length: Conditional entropy
- Conditioning never increases average codeword length or entropy

Block Codes
- Variable-length code for blocks of $N$ symbols
- Bound for average codeword length: Block entropy divided by $N$
- Increasing block size never increases average codeword length or entropy
- Fundamental lossless coding theorem: $\bar{\ell} \geq \bar{H}$ (entropy rate)

V2V Codes
- Assign codewords to variable-length symbol sequences
- Important example: Run-level coding of transform coefficients

Adaptive Variable-Length Codes
- Adapt code during encoding/decoding (forward/backward adaptation)
Intermediate Summary: Shannon-Fano-Elias Coding

Shannon-Fano-Elias Codes

- Special Block Code
- In general: Worse than Huffman code for same block size
- On-the-fly construction of codewords (no need to store codeword table)
- On-the-fly decoding of messages

Next Lecture

- Iterative encoding and decoding for Shannon-Fano-Elias codes
- Arithmetic coding as practical implementation of Shannon-Fano-Elias codes