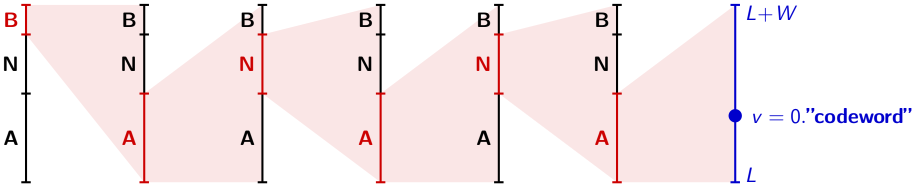
# Arithmetic Coding



# Last Lecture: Variable-Length Codes

## Fundamental Lossless Coding Theorem

- Entropy rate as greatest lower bound for lossless coding

$$\bar{\ell} \geq \bar{H}(\boldsymbol{X}) = \lim_{N\to\infty} \frac{H_N(\boldsymbol{X})}{N} \qquad \text{with} \qquad H_N(\boldsymbol{X}) = \mathrm{E}\Big\{ -\log_2 p(X_k, X_{k+1}, \cdots, X_{k+N-1}) \Big\}$$

## Variable-Length Codes

- Scalar codes: Individual codeword for each alphabet letter
- Conditional codes: Switching of codeword tables depending on a condition
- Block codes: One codeword for block of $N > 1$ symbols
- V2V codes: Codewords for variable-length symbol sequences

➡ Huffman algorithm yields optimal prefix code for each type of codes

!!! **Block Huffman codes for large $N$ yield coding efficiency very close to entropy rate**
(but cannot implemented due to extreme memory requirements for storing codeword table)

# Last Lecture: Shannon-Fano-Elias Coding

## Motivation

- Block codes for large $N$ stay efficient even if they are slightly suboptimal

$$\left(\frac{1}{N}H_N\right) + \left(\frac{A}{N}\right) \le \bar{\ell} < \left(\frac{1}{N}H_N\right) + \left(\frac{1+A}{N}\right) \quad \text{with} \quad A \ll N$$

➡ Realize encoding and decoding without storing a codeword table

## Shannon-Fano-Elias Coding

- Idea: Code message by transmitting one value inside a probability interval of the cdf
- Require $K = \lceil -\log_2 p(\cdots) \rceil$ bits (same as for Shannon code)

## Iterative Shannon-Fano-Elias Coding

- Probability interval can be determined by iterative refinement using simple models for conditional probabilities $p(a \mid \cdots)$ (e.g., iid model or Markov model)

➡ Enables iterative encoding and decoding with reasonably small pmfs

# Review: Iterative Shannon-Fano-Elias Encoding Algorithm

Given: Sequence $\boldsymbol{s} = \{s_1, s_2, s_3, \cdots, s_N\}$ of $N$ symbols

1. Initialization of probability interval:

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

2. Iterative refinement ( for $n = 1$ to $N$ ):

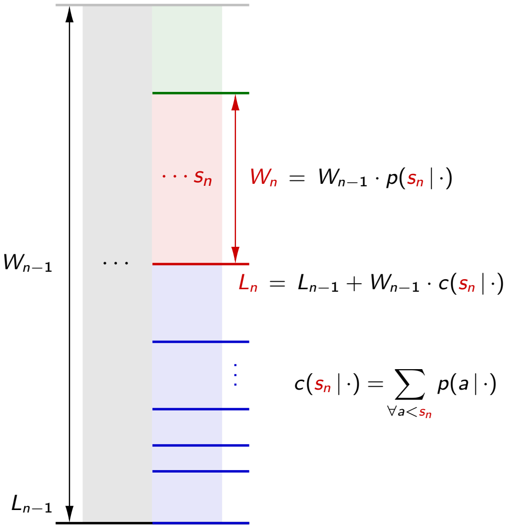
$$W_n = W_{n-1} \cdot p(\, s_n \mid \cdots )$$
$$L_n = L_{n-1} + W_{n-1} \cdot c(\, s_n \mid \cdots )$$

3. Determine codeword length and codeword value:

$$K = \lceil A - \log_2 W_N \rceil \qquad ( A = 0 \text{ or } 1 )$$
$$z = \lceil L_N \cdot 2^K \rceil$$

4. Transmit codeword:

➡ Binary representation of $z$ with $K$ bits



# Review: Iterative Shannon-Fano-Elias Decoding Algorithm

Given:
- Codeword: integer $z$ with $K$ bits
- Number $N$ of symbols to be decoded
- Ordered alphabet $\mathcal{A} = \{a_1, a_2 \cdots\}$

**1** Initialization: $W_0 = 1, \quad L_0 = 0, \quad v = z \cdot 2^{-K}$

**2** Iterative decoding ( for $n = 1$ to $N$ ):

**a** Upper boundary $U_1$ for first symbol $a_1$ of $\mathcal{A}$

$$k = 1, \qquad U_k = L_{n-1} + W_{n-1} \cdot p(\, a_k \,|\, \cdots )$$

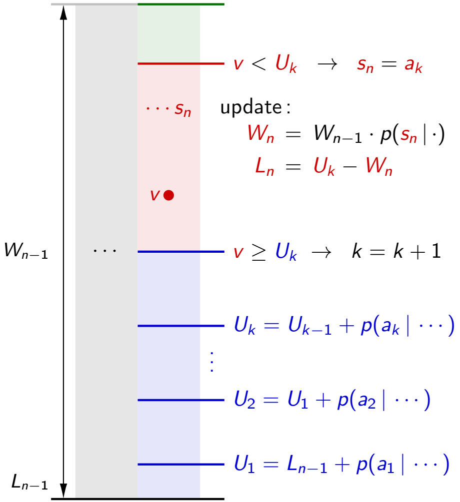
**b** While ( $v \geq U_k$ ), proceed to next symbol of $\mathcal{A}$

$$k = k + 1, \qquad U_k = U_{k-1} + W_{n-1} \cdot p(\, a_k \,|\, \cdots )$$

**c** Output decoded symbol $s_n = a_k$

**d** Update interval: $W_n = W_{n-1} \cdot p(\, s_n \,|\, \cdots )$

$$L_n = U_k - W_n$$



# Implementation Aspects of Shannon-Fano-Elias Coding

## Iterative Shannon-Fano-Elias Coding

- Iterative interval refinement
  - ➡ Very simple if we use simple models for conditional pmfs $p(a \,|\, \cdots)$
  - ➡ In practice: IID model or conditional model with small number of conditions

# Implementation Aspects of Shannon-Fano-Elias Coding

## Iterative Shannon-Fano-Elias Coding

- Iterative interval refinement
  - ➡ Very simple if we use simple models for conditional pmfs $p(a\,|\,\cdots)$
  - ➡ In practice: IID model or conditional model with small number of conditions
- Simple codeword construction
  - ➡ Straightforward concept for known interval boundaries

# Implementation Aspects of Shannon-Fano-Elias Coding

## Iterative Shannon-Fano-Elias Coding

- Iterative interval refinement
  - ➡ Very simple if we use simple models for conditional pmfs $p(a \mid \cdots)$
  - ➡ In practice: IID model or conditional model with small number of conditions
- Simple codeword construction
  - ➡ Straightforward concept for known interval boundaries

## Can we implement it ?

# Implementation Aspects of Shannon-Fano-Elias Coding

## Iterative Shannon-Fano-Elias Coding

- Iterative interval refinement
  - ➡ Very simple if we use simple models for conditional pmfs $p(a \,|\, \cdots)$
  - ➡ In practice: IID model or conditional model with small number of conditions
- Simple codeword construction
  - ➡ Straightforward concept for known interval boundaries

## Can we implement it ?

➡ **No, not really** (at least not for large $N$)

- Require arbitrarily high precision for real values $W_n$, $L_n$, and $v$
- ➡ Standard floating-point values are not sufficient

# Implementation Aspects of Shannon-Fano-Elias Coding

## Iterative Shannon-Fano-Elias Coding

- Iterative interval refinement
  - ➡ Very simple if we use simple models for conditional pmfs $p(a \,|\, \cdots)$
  - ➡ In practice: IID model or conditional model with small number of conditions
- Simple codeword construction
  - ➡ Straightforward concept for known interval boundaries

## Can we implement it ?

➡ **No, not really** (at least not for large $N$)

- Require arbitrarily high precision for real values $W_n$, $L_n$, and $v$
- ➡ Standard floating-point values are not sufficient

**Question :** Can we design a practical coding method based on Shannon-Fano-Elias coding ?

# Arithmetic Coding

## Idea of Arithmetic Coding

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

# Arithmetic Coding

## Idea of Arithmetic Coding

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

➡ Represent pmf with standard integers (e.g., of 8, 16, or 32 bits)

# Arithmetic Coding

## Idea of Arithmetic Coding

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

➡ Represent pmf with standard integers (e.g., of 8, 16, or 32 bits)

➡ Represent interval width and lower boundary with standard integers

# Arithmetic Coding

## Idea of Arithmetic Coding

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

➡ Represent pmf with standard integers (e.g., of 8, 16, or 32 bits)

➡ Represent interval width and lower boundary with standard integers

➡ Output bits of codeword as soon as possible

# Arithmetic Coding

## Idea of Arithmetic Coding

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

➡ Represent pmf with standard integers (e.g., of 8, 16, or 32 bits)

➡ Represent interval width and lower boundary with standard integers

➡ Output bits of codeword as soon as possible

## Realizations of Arithmetic Coding

- There are different variants

# Arithmetic Coding

## Idea of Arithmetic Coding

- Fixed-precision approximation of Shannon-Fano-Elias coding
- ➡ Represent pmf with standard integers (e.g., of 8, 16, or 32 bits)
- ➡ Represent interval width and lower boundary with standard integers
- ➡ Output bits of codeword as soon as possible

## Realizations of Arithmetic Coding

- There are different variants
- Will discuss original approach by R. Pasco (1976, PhD thesis, Stanford)

# Arithmetic Coding

## Idea of Arithmetic Coding

- Fixed-precision approximation of Shannon-Fano-Elias coding
- ➡ Represent pmf with standard integers (e.g., of 8, 16, or 32 bits)
- ➡ Represent interval width and lower boundary with standard integers
- ➡ Output bits of codeword as soon as possible

## Realizations of Arithmetic Coding

- There are different variants
- Will discuss original approach by R. Pasco (1976, PhD thesis, Stanford)
- Other popular realizations
  - Rissanen, "Generalized Kraft inequality and arithmetic coding," IBM Journal of Res. Develop., 1976
  - Witten, Neal, Cleary, "Arithmetic coding for data compression," Comm. of ACM, 1987.
  - Rissanen, Mohiuddin, "A multiplication-free multialphabet arithmetic code," IEEE Tr. Comm., 1989.

# Quantization of PMF

## Fixed-Precision Approximation of Probability Masses

- Represent probability masses $p(a)$ by $V$-bit integers $p_V(a)$

$$p(a) = p_V(a) \cdot 2^{-V} = 0.\overbrace{\underbrace{xxx\cdots x}_{p_V(a)}}^{V\text{ bits}}000\cdots$$

# Quantization of PMF

## Fixed-Precision Approximation of Probability Masses

- Represent probability masses $p(a)$ by $V$-bit integers $p_V(a)$

$$p(a) = p_V(a) \cdot 2^{-V} = 0.\underbrace{\overbrace{xxx\cdots x}^{V \text{ bits}}}_{p_V(a)}000\cdots$$

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

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

# Quantization of PMF

## Fixed-Precision Approximation of Probability Masses

- Represent probability masses $p(a)$ by $V$-bit integers $p_V(a)$

$$p(a) = p_V(a) \cdot 2^{-V} = 0.\underbrace{\overbrace{xxx\cdots x}^{V \text{ bits}}}_{p_V(a)}000\cdots$$

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

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

## Requirements on Pmf Approximation

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

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

# Quantization of Interval Width

## Fixed-Precision Approximation of Interval Width

- Represent interval width $W_n$ by $U$-bit integers $A_n$ and counter $z_n$

$$W_n = A_n \cdot 2^{-z_n} = 0.\overbrace{\underbrace{00000\cdots0}_{z_n - U \text{ bits}}\underbrace{1xxxx\cdots x}_{A_n \;(U \text{ bits})}}^{z_n \text{ bits}}000\cdots$$

$\Big\{$ very similar to representation of floating-point numbers in a computer

# Quantization of Interval Width

## Fixed-Precision Approximation of Interval Width

- Represent interval width $W_n$ by $U$-bit integers $A_n$ and counter $z_n$

$$W_n = A_n \cdot 2^{-z_n} = 0.\overbrace{\underbrace{00000\cdots0}_{z_n - U \text{ bits}}\underbrace{1xxxx\cdots x}_{A_n\ (U \text{ bits})}}^{z_n \text{ bits}}000\cdots$$

{ very similar to representation of floating-point numbers in a computer

- Use maximum possible precision for $A_n$

➡ $A_n$ should always have the form : $A_n = \underbrace{1xxx\cdots x}_{U \text{ bits}}$ (binary representation)

# Quantization of Interval Width

## Fixed-Precision Approximation of Interval Width

- Represent interval width $W_n$ by $U$-bit integers $A_n$ and counter $z_n$

$$W_n = A_n \cdot 2^{-z_n} = 0.\overbrace{\underbrace{00000\cdots0}_{z_n - U \text{ bits}}\underbrace{1xxxx\cdots x}_{A_n\ (U \text{ bits})}}^{z_n \text{ bits}}000\cdots$$

$\Big\{$ very similar to representation of floating-point numbers in a computer

- Use maximum possible precision for $A_n$

  ➡ $A_n$ should always have the form : $A_n = \underbrace{1xxx\cdots x}_{U \text{ bits}}$ (binary representation)

  ➡ That means, $A_n$ is restricted to : $2^{U-1} \le A_n < 2^U$

# Quantization of Interval Width

## Fixed-Precision Approximation of Interval Width

- Represent interval width $W_n$ by $U$-bit integers $A_n$ and counter $z_n$

$$W_n = A_n \cdot 2^{-z_n} = 0.\overbrace{\underbrace{00000\cdots0}_{z_n - U \text{ bits}}\underbrace{1xxxx\cdots x}_{A_n\ (U \text{ bits})}}^{z_n \text{ bits}}000\cdots$$

$\left\{\right.$ very similar to representation of floating-point numbers in a computer

- Use maximum possible precision for $A_n$

  ➡ $A_n$ should always have the form: $\quad A_n = \underbrace{1xxx\cdots x}_{U \text{ bits}}$ (binary representation)

  ➡ That means, $A_n$ is restricted to: $\quad 2^{U-1} \le A_n < 2^U$

  ➡ Use following initialization:

$$\begin{aligned} A_0 &= 2^U - 1 \\ z_0 &= U \end{aligned} \quad \Longleftrightarrow \quad W_0 = 1 - 2^U = 0.\underbrace{1111\cdots1}_{U \text{ bits}}$$

# Refinement of Interval Width

## Conventional Refinement of Interval Width

$$
\begin{aligned}
W_n &= W_{n-1} \cdot p(s_n) \\
A_n \cdot 2^{-z_n} &= \underbrace{\Big( A_{n-1} \cdot p_V(s_n) \Big)}_{(U+V)\text{-bit integer}} \cdot 2^{-(z_{n-1}+V)}
\end{aligned}
$$

# Refinement of Interval Width

## Conventional Refinement of Interval Width

$$W_n = W_{n-1} \cdot p(s_n)$$

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

- In general: $W_{n-1} \cdot p(s_n)$ cannot be represented using a $U$-bit integer

# Refinement of Interval Width

## Conventional Refinement of Interval Width

$$\begin{aligned} W_n &= W_{n-1} \cdot p(s_n) \\ A_n \cdot 2^{-z_n} &= \underbrace{\Big( A_{n-1} \cdot p_V(s_n) \Big)}_{(U+V)\text{-bit integer}} \cdot 2^{-(z_{n-1}+V)} \end{aligned}$$

- In general: $W_{n-1} \cdot p(s_n)$ cannot be represented using a $U$-bit integer

➡ **What can we do?**

# Refinement of Interval Width

## Conventional Refinement of Interval Width

$$W_n = W_{n-1} \cdot p(s_n)$$

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

- In general: $W_{n-1} \cdot p(s_n)$ cannot be represented using a $U$-bit integer

➞ **What can we do?**

## Requirement for Unique Decodability

- Code remains uniquely decodable if we ensure : $0 < W_n \le W_{n-1} \cdot p(s_n)$ (nested intervals)

# Refinement of Interval Width

## Conventional Refinement of Interval Width

$$\begin{aligned} W_n &= W_{n-1} \cdot p(s_n) \\ A_n \cdot 2^{-z_n} &= \underbrace{\Big( A_{n-1} \cdot p_V(s_n) \Big)}_{(U+V)\text{-bit integer}} \cdot 2^{-(z_{n-1}+V)} \end{aligned}$$

- In general: $W_{n-1} \cdot p(s_n)$ cannot be represented using a $U$-bit integer

➡ **What can we do?**

## Requirement for Unique Decodability

- Code remains uniquely decodable if we ensure : $0 < W_n \leq W_{n-1} \cdot p(s_n)$ (nested intervals)

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

# Rounding of Interval Width in Iterative Interval Refinement

- Binary representations of interval width $W_{n-1}$ and probability mass $p(s_n)$

$$
\begin{aligned}
W_{n-1} &= A_{n-1} \cdot 2^{-z_{n-1}} &\quad\rightarrow\quad W_{n-1} &= 0.\overbrace{00000\cdots0}^{z_{n-1}-U \text{ bits}}\overbrace{1xx\cdots x}^{U \text{ bits}}000\cdots \\
p(s_n) &= p_V(s_n) \cdot 2^{-V} &\quad\rightarrow\quad p(s_n) &= 0.\underbrace{xxx\cdots x}_{V \text{ bits}}000\cdots
\end{aligned}
$$

# Rounding of Interval Width in Iterative Interval Refinement

- Binary representations of interval width $W_{n-1}$ and probability mass $p(s_n)$

$$W_{n-1} = A_{n-1} \cdot 2^{-z_{n-1}} \quad \rightarrow \quad W_{n-1} = 0.\overbrace{00000\cdots0}^{z_{n-1}-U \text{ bits}}\overbrace{1xx\cdots x}^{U \text{ bits}}000\cdots$$

$$p(s_n) = p_V(s_n) \cdot 2^{-V} \quad \rightarrow \quad p(s_n) = 0.\underbrace{xxx\cdots x}_{V \text{ bits}}000\cdots$$

➡ Rounding in interval refinement $W_{n-1} \cdot p(s_n) \mapsto W_n$

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

# Rounding of Interval Width in Iterative Interval Refinement

- Binary representations of interval width $W_{n-1}$ and probability mass $p(s_n)$

$$W_{n-1} = A_{n-1} \cdot 2^{-z_{n-1}} \quad \rightarrow \quad W_{n-1} = 0.\overbrace{00000\cdots0}^{z_{n-1}-U \text{ bits}}\overbrace{1xx\cdots x}^{U \text{ bits}}000\cdots$$

$$p(s_n) = p_V(s_n) \cdot 2^{-V} \quad \rightarrow \quad p(s_n) = 0.\underbrace{xxx\cdots x}_{V \text{ bits}}000\cdots$$

➡ Rounding in interval refinement $W_{n-1} \cdot p(s_n) \mapsto W_n$

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

$$\Big\downarrow$$

$$W_n = 0.\underbrace{\overbrace{00000\cdots0}^{z_{n-1}-U \text{ bits}}\overbrace{00\cdots0}^{\Delta z \text{ bits}}}_{z_n-U \text{ bits}}\underbrace{\overbrace{1x\cdots x}^{U \text{ bits}}}_{A_n}00\cdots0\,000\cdots$$

# Arithmetic Operations for Update of Interval Width

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

$$\downarrow$$

$$W_n \;=\; 0.\underbrace{\overbrace{00000\cdots0}^{z_{n-1}-U\text{ bits}}\;\overbrace{00\cdots0}^{\Delta z\text{ bits}}}_{z_n-U\text{ bits}}\;\underbrace{\overbrace{1x\cdots x}^{U\text{ bits}}}_{A_n}\,00\cdots0\;000\cdots$$

## Update of interval width

1. Calculate intermediate $(U+V)$-bit integer : $\quad A_n^* = A_{n-1} \cdot p_V(s_n)$

# Arithmetic Operations for Update of Interval Width

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

$$\downarrow$$

$$W_n \;=\; 0.\underbrace{\overbrace{00000\cdots0}^{z_{n-1}-U \text{ bits}}\;\overbrace{00\cdots0}^{\Delta z \text{ bits}}}_{z_n-U \text{ bits}}\;\underbrace{\overbrace{1x\cdots x}^{U \text{ bits}}}_{A_n}00\cdots0\;000\cdots$$

## Update of interval width

1. Calculate intermediate $(U+V)$-bit integer : $\quad A_n^* = A_{n-1} \cdot p_V(s_n)$
2. Determine number $\Delta z$ of leading zeros in $(U+V)$-bit integer $A_n^*$ $\quad$ (check at most $V$ bits)

# Arithmetic Operations for Update of Interval Width

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

$$\downarrow$$

$$W_n \;=\; 0.\underbrace{\overbrace{00000\cdots0}^{z_{n-1}-U\text{ bits}}\;\overbrace{00\cdots0}^{\Delta z\text{ bits}}}_{z_n-U\text{ bits}}\;\underbrace{\overbrace{1x\cdots x}^{U\text{ bits}}}_{A_n}00\cdots0\;000\cdots$$

## Update of interval width

1. Calculate intermediate $(U+V)$-bit integer: $\quad A_n^* = A_{n-1} \cdot p_V(s_n)$
2. Determine number $\Delta z$ of leading zeros in $(U+V)$-bit integer $A_n^*$ (check at most $V$ bits)
3. Update interval width according to:

$$A_n = (A_n^* \ll \Delta z) \gg V \qquad \text{(“}\ll\text{” and “}\gg\text{” = bit shifts to the left and to the right)}$$

$$z_n = z_{n-1} + \Delta z \qquad \text{(note: } z_n \text{ is not required for updating the interval width)}$$

# Intermediate Summary: Calculation of Interval Width

## Representation of Interval Width and Probability Masses

$$W_n = A_n \cdot 2^{-z_n} \qquad \text{with } A_n \text{ being an } U\text{-bit integer with } 2^{U-1} \le A_n < 2^U$$

$$p(s_n) = p_V(s_n) \cdot 2^{-V} \qquad \text{with } p_V(s_n) \text{ being a } V\text{-bit integer with } p_V(s_n) > 0 \text{ and } \sum_{\forall a} p_V(a) \le 2^V$$

## Arithmetic Calculation of Interval Width

1. Initialization

$$A_0 = (1 \ll U) - 1$$

2. Update in each iteration

$$A_n^* = A_{n-1} \cdot p_V(s_n)$$

$$\Delta z = \text{number of leading zeros in } (U+V)\text{-bit integer } A_n^*$$

$$A_n = (A_n^* \ll \Delta z) \gg V$$

# Update of Lower Interval Boundary

- Remember: Conventional update

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

# Update of Lower Interval Boundary

- Remember: Conventional update

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

- Binary representations of interval width $W_{n-1}$ and modified cmf $c(s_n)$

$$\begin{aligned} W_{n-1} &= A_{n-1} \cdot 2^{-z_{n-1}} & \rightarrow \quad W_{n-1} &= 0.\overbrace{0000000\cdots0}^{z_{n-1}-U \text{ bits}}\overbrace{1xxxx\cdots x}^{A_n\ (U \text{ bits})}000\cdots \\ c(s_n) &= c_V(s_n) \cdot 2^{-V} & \rightarrow \quad c(s_n) &= 0.\underbrace{xxxxxx\cdots x}_{c_V(s_n)\ (V \text{ bits})}000\cdots \end{aligned}$$

# Update of Lower Interval Boundary

- Remember: Conventional update

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

- Binary representations of interval width $W_{n-1}$ and modified cmf $c(s_n)$

$$\begin{aligned} W_{n-1} &= A_{n-1} \cdot 2^{-z_{n-1}} & \rightarrow \quad W_{n-1} &= 0.\overbrace{0000000\cdots0}^{z_{n-1}-U \text{ bits}}\overbrace{1xxxx\cdots x}^{A_n\ (U \text{ bits})}000\cdots \\ c(s_n) &= c_V(s_n) \cdot 2^{-V} & \rightarrow \quad c(s_n) &= 0.\underbrace{xxxxxx\cdots x}_{c_V(s_n)\ (V \text{ bits})}000\cdots \end{aligned}$$

➡ Binary representation of product $W_{n-1} \cdot c(s_n)$

$$W_{n-1} \cdot c(s_n) = 0.\overbrace{\underbrace{00000\cdots0}_{z_{n-1}-U \text{ bits}}\underbrace{xxxxxxx\cdots x}_{\substack{A_{n-1}\cdot c_V(s_n) \\ (U+V \text{ bits})}}}^{z_{n-1}+V \text{ bits}}000\cdots$$

# Effect on Binary Representation of Lower Interval Boundary

$$W_{n-1} \cdot c(s_n) = 0.\overbrace{\underbrace{00000\cdots0}_{z_{n-1}-U \text{ bits}}\underbrace{xxxxxxx\cdots x}_{\substack{A_{n-1}\cdot c_V(s_n) \\ (U+V \text{ bits})}}}^{z_{n-1}+V \text{ bits}}000\cdots$$

What is the effect of an update $L_n = L_{n-1} + W_{n-1} \cdot c_V(s_n)$ on lower interval boundary?

# Effect on Binary Representation of Lower Interval Boundary

$$W_{n-1} \cdot c(s_n) = 0.\overbrace{\underbrace{00000\cdots0}_{z_{n-1}-U \text{ bits}}\underbrace{xxxxxxx\cdots x}_{\substack{A_{n-1}\cdot c_V(s_n) \\ (U+V \text{ bits})}}}^{z_{n-1}+V \text{ bits}}000\cdots$$

What is the effect of an update $L_n = L_{n-1} + W_{n-1} \cdot c_V(s_n)$ on lower interval boundary?

$$L_{n-1} = 0.\overbrace{\underbrace{aaaaa\cdots a}_{z_{n-1}-c_{n-1}-U}\underbrace{0111111\cdots1}_{c_{n-1}}}^{z_{n-1}-U \text{ bits}}\underbrace{xxxxx\cdots x}_{U+V}\underbrace{00000\cdots}$$

# Effect on Binary Representation of Lower Interval Boundary

$$W_{n-1} \cdot c(s_n) = 0.\overbrace{\underbrace{00000\cdots0}_{z_{n-1}-U \text{ bits}}\,\underbrace{xxxxxxx\cdots x}_{\substack{A_{n-1}\cdot c_V(s_n) \\ (U+V \text{ bits})}}}^{z_{n-1}+V \text{ bits}}000\cdots$$

What is the effect of an update $L_n = L_{n-1} + W_{n-1} \cdot c_V(s_n)$ on lower interval boundary?

$$L_{n-1} = 0.\overbrace{\underbrace{aaaaa\cdots a}_{z_{n-1}-c_{n-1}-U}\,\underbrace{0111111\cdots1}_{c_{n-1}}}^{z_{n-1}-U \text{ bits}}\,\underbrace{xxxxx\cdots x}_{U+V}\,\underbrace{00000\cdots}_{\textit{trailing bits}}$$

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

# Effect on Binary Representation of Lower Interval Boundary

$$W_{n-1} \cdot c(s_n) = 0.\overbrace{\underbrace{00000\cdots0}_{z_{n-1}-U \text{ bits}}\,\underbrace{xxxxxxx\cdots x}_{\substack{A_{n-1}\cdot c_V(s_n)\\(U+V \text{ bits})}}}^{z_{n-1}+V \text{ bits}}\,000\cdots$$

What is the effect of an update $L_n = L_{n-1} + W_{n-1} \cdot c_V(s_n)$ on lower interval boundary?

$$L_{n-1} = 0.\overbrace{\underbrace{aaaaa\cdots a}_{z_{n-1}-c_{n-1}-U}\,\underbrace{0111111\cdots1}_{c_{n-1}}}^{z_{n-1}-U \text{ bits}}\,\underbrace{xxxxx\cdots x}_{\substack{U+V\\ \textit{active bits}}}\,\underbrace{00000\cdots}_{\textit{trailing bits}}$$

- Trailing bits: Equal to 0, but maybe changed later
- Active bits: Directly modified by the update $L_n = L_{n-1} + W_{n-1} \cdot c(s_n)$

# Effect on Binary Representation of Lower Interval Boundary

$$W_{n-1} \cdot c(s_n) = 0.\overbrace{\underbrace{00000\cdots0}_{z_{n-1}-U \text{ bits}}\,\underbrace{xxxxxxx\cdots x}_{\substack{A_{n-1}\cdot c_V(s_n) \\ (U+V \text{ bits})}}}^{z_{n-1}+V \text{ bits}}000\cdots$$

What is the effect of an update $L_n = L_{n-1} + W_{n-1} \cdot c_V(s_n)$ on lower interval boundary?

$$L_{n-1} = 0.\overbrace{\underbrace{aaaaa\cdots a}_{z_{n-1}-c_{n-1}-U}\,\underbrace{0111111\cdots1}_{\substack{c_{n-1} \\ \textit{outstanding bits}}}}^{z_{n-1}-U \text{ bits}}\,\underbrace{xxxxx\cdots x}_{\substack{U+V \\ \textit{active bits}}}\,\underbrace{00000\cdots}_{\textit{trailing bits}}$$

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

➡ Active bits: Directly modified by the update $L_n = L_{n-1} + W_{n-1} \cdot c(s_n)$

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

# Effect on Binary Representation of Lower Interval Boundary

$$W_{n-1} \cdot c(s_n) = 0.\overbrace{\underbrace{00000\cdots0}_{z_{n-1}-U \text{ bits}}\underbrace{xxxxxxx\cdots x}_{\substack{A_{n-1}\cdot c_V(s_n) \\ (U+V \text{ bits})}}}^{z_{n-1}+V \text{ bits}}000\cdots$$

What is the effect of an update $L_n = L_{n-1} + W_{n-1} \cdot c_V(s_n)$ on lower interval boundary?

$$L_{n-1} = 0.\overbrace{\underbrace{aaaaa\cdots a}_{\substack{z_{n-1}-c_{n-1}-U \\ \text{settled bits}}}\underbrace{0111111\cdots1}_{\substack{c_{n-1} \\ \text{outstanding bits}}}}^{z_{n-1}-U \text{ bits}}\underbrace{xxxxx\cdots x}_{\substack{U+V \\ \text{active bits}}}\underbrace{00000\cdots}_{\text{trailing bits}}$$

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

➡ Active bits: Directly modified by the update $L_n = L_{n-1} + W_{n-1} \cdot c(s_n)$

➡ 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

$$L_n = 0.\overbrace{\underbrace{aaaaa\cdots a}_{\substack{z_n - c_n - U \\ \textit{settled bits}}}\;\underbrace{0111111\cdots 1}_{\substack{c_n \\ \textit{outstanding bits}}}}^{z_n - U \text{ bits}}\;\underbrace{xxxxx\cdots x}_{\substack{U+V \\ \textit{active bits}}}\;\underbrace{00000\cdots}_{\textit{trailing bits}}$$

# Representation of Lower Interval Boundary

$$L_n = 0.\overbrace{\underbrace{aaaaa\cdots a}_{\substack{z_n-c_n-U \\ \textit{settled bits}}}\;\underbrace{0111111\cdots 1}_{\substack{c_n \\ \textit{outstanding bits}}}}^{z_n-U\text{ bits}}\;\underbrace{xxxxx\cdots x}_{\substack{U+V \\ \textit{active bits}}}\;\underbrace{00000\cdots}_{\textit{trailing bits}}$$

- **Active bits**:
  - ➡ Represent as $(U+V)$-bit integer $B_n$
  - ➡ Intermediate values $B_{n-1} + A_{n-1} \cdot c_V(s_n)$ require $(U+V+1)$-bit integer

# Representation of Lower Interval Boundary

$$L_n = 0.\overbrace{\underbrace{aaaaa\cdots a}_{\substack{z_n - c_n - U \\ \textit{settled bits}}}\,\underbrace{0111111\cdots 1}_{\substack{c_n \\ \textit{outstanding bits}}}}^{z_n - U \text{ bits}}\,\underbrace{xxxxx\cdots x}_{\substack{U+V \\ \textit{active bits}}}\,\underbrace{00000\cdots}_{\textit{trailing bits}}$$

- **Active bits**:
  - ➡ Represent as $(U+V)$-bit integer $B_n$
  - ➡ Intermediate values $B_{n-1} + A_{n-1} \cdot c_V(s_n)$ require $(U+V+1)$-bit integer
- **Outstanding bits**:
  - ➡ Represent as integer counter $c_n$ (trailing $c_n - 1$ bits are equal to 1)

# Representation of Lower Interval Boundary

$$L_n = 0.\overbrace{\underbrace{aaaaa\cdots a}_{\substack{z_n - c_n - U \\ \textit{settled bits}}}\;\underbrace{0111111\cdots 1}_{\substack{c_n \\ \textit{outstanding bits}}}}^{z_n - U \text{ bits}}\;\underbrace{xxxxx\cdots x}_{\substack{U+V \\ \textit{active bits}}}\;\underbrace{00000\cdots}_{\textit{trailing bits}}$$

- **Active bits**:
  - ➡ Represent as $(U+V)$-bit integer $B_n$
  - ➡ Intermediate values $B_{n-1} + A_{n-1} \cdot c_V(s_n)$ require $(U+V+1)$-bit integer
- **Outstanding bits**:
  - ➡ Represent as integer counter $c_n$ (trailing $c_n - 1$ bits are equal to 1)
- **Settled bits**:
  - ➡ Output as soon as they become settled

# Update of Probability Interval

$$
\begin{aligned}
W_{n-1} &= 0.\overbrace{0000000000000000\cdots0}^{z_{n-1}-U}\;\overbrace{1xx\cdots x}^{A_{n-1}\,(U)}\;000000000\cdots \\
L_{n-1} &= 0.\underbrace{aaaaaaaaa\cdots a}_{z_{n-1}-c_{n-1}-U}\;\underbrace{011\cdots1}_{c_{n-1}}\;\underbrace{xxx\cdots xxxxxx}_{B_{n-1}\,(U+V)}\;00000\cdots
\end{aligned}
$$

$$
\begin{aligned}
W_{n} &= 0.\overbrace{0000000000000000\cdots0}^{z_{n-1}-U}\;\overbrace{00\cdots0}^{\Delta z}\;\overbrace{1xx\cdots x}^{A_{n}\,(U)}\;000000000\cdots \\
L_{n} &= 0.\underbrace{aaaaaaaaa\cdots a}_{z_{n-1}-c_{n-1}-U}\;\underbrace{xxxxxxxxx\cdots xxx}_{c_{n-1}+\Delta z}\;\underbrace{xxx\cdots xxxxxx}_{B_{n}\,(U+V)}\;00000\cdots
\end{aligned}
$$

# Update of Probability Interval

$$
\begin{aligned}
W_{n-1} &= 0.\overbrace{00000000000000000\cdots0}^{z_{n-1}-U}\;\overbrace{1xx\cdots x}^{A_{n-1}\;(U)}\;000000000\cdots \\
L_{n-1} &= 0.\underbrace{aaaaaaaaa\cdots a}_{z_{n-1}-c_{n-1}-U}\;\underbrace{011\cdots1}_{c_{n-1}}\;\underbrace{xxx\cdots xxxxxx}_{B_{n-1}\;(U+V)}\;00000\cdots
\end{aligned}
$$

$$
\begin{aligned}
W_{n} &= 0.\overbrace{00000000000000000\cdots0}^{z_{n-1}-U}\;\overbrace{00\cdots0}^{\Delta z}\;\overbrace{1xx\cdots x}^{A_{n}\;(U)}\;000000000\cdots \\
L_{n} &= 0.\underbrace{aaaaaaaaa\cdots a}_{z_{n-1}-c_{n-1}-U}\;\underbrace{xxxxxxxxx\cdots xxx}_{c_{n-1}+\Delta z}\;\underbrace{xxx\cdots xxxxxx}_{B_{n}\;(U+V)}\;00000\cdots
\end{aligned}
$$

**Interval update :**

$$A_n^* = A_{n-1}\cdot p_V(s_n) \qquad (A_n^* \text{ is an } (U{+}V)\text{-bit integer})$$

$$B_n^* = B_{n-1} + A_{n-1}\cdot c_V(s_n) \qquad (B_n^* \text{ is an } (U{+}V{+}1)\text{-bit integer})$$

$$\Delta z = \text{number of leading zeros in } (U{+}V)\text{-bit integer } A_n^*$$

$$A_n = (A_n^* \ll \Delta z) \gg V$$

$$B_n = (B_n^* \ll \Delta z)\ \&\ ((1 \ll (U+V)) - 1) \qquad (\text{mask} = U{+}V \text{ bits equal to 1})$$

# Output of Settled Bits and Update of Outstanding Counter

Investigate intermediate $(U+V+1)$-bit integer $B_n^*$ and outstanding counter $c_{n-1}$

$$B_n^* : \quad c \; \overbrace{zz\cdots zz}^{\Delta z \text{ bits}} \; \overbrace{xxx\cdots xxx}^{U+V-\Delta z \text{ bits}}$$

(trailing $U+V-\Delta z$ bits are first bits of new $B_n$)

$$c_{n-1} : \; (\,\textit{settled bits}\,) \; \underbrace{0111\cdots1}_{c_{n-1} \text{ bits}} \; 0000000000\cdots$$

(outstanding bits can be inverted by carry $c$)

# Output of Settled Bits and Update of Outstanding Counter

Investigate intermediate $(U+V+1)$-bit integer $B_n^*$ and outstanding counter $c_{n-1}$

$$B_n^* : \quad c \quad \overbrace{zz \cdots zz}^{\Delta z \text{ bits}} \quad \overbrace{xxx \cdots xxx}^{U+V-\Delta z \text{ bits}}$$

(trailing $U+V-\Delta z$ bits are first bits of new $B_n$)

$$c_{n-1} : \; (\,\textit{settled bits}\,) \; \underbrace{0111 \cdots 1}_{c_{n-1} \text{ bits}} \; 0000000000 \cdots$$

(outstanding bits can be inverted by carry $c$)

**1** Check for carry

- ➡ If $(c = 1)$, output one '1' and $(c_{n-1} - 2)$ '0's, remove carry (set $c = 0$), and set $c_{n-1} = 1$
- ➡ From now on, $B_n^*$ is considered a $(U+V)$-bit integer

# Output of Settled Bits and Update of Outstanding Counter

Investigate intermediate $(U+V+1)$-bit integer $B_n^*$ and outstanding counter $c_{n-1}$

$$
\begin{array}{lllll}
B_n^* : & & c \;\; \overbrace{zz\cdots zz}^{\Delta z \text{ bits}} \;\; \overbrace{xxx\cdots xxx}^{U+V-\Delta z \text{ bits}} & & \text{(trailing } U+V-\Delta z \text{ bits are first bits of new } B_n\text{)} \\
c_{n-1} : & (\,\textit{settled bits}\,) & \underbrace{0111\cdots 1}_{c_{n-1} \text{ bits}} \;\; 0000000000\cdots & & \text{(outstanding bits can be inverted by carry } c\text{)}
\end{array}
$$

**1** Check for carry

- ➡ If $(c = 1)$, output one '1' and $(c_{n-1} - 2)$ '0's, remove carry (set $c = 0$), and set $c_{n-1} = 1$
- ➡ From now on, $B_n^*$ is considered a $(U+V)$-bit integer

**2** If $(\Delta z > 0)$, determine number $n_1$ of trailing ones in $\Delta z$ leading bits of $(U+V)$-bit integer $B_n^*$

# Output of Settled Bits and Update of Outstanding Counter

Investigate intermediate $(U+V+1)$-bit integer $B_n^*$ and outstanding counter $c_{n-1}$

$$B_n^* : \quad c \quad \overbrace{zz\cdots zz}^{\Delta z \text{ bits}} \quad \overbrace{xxx\cdots xxx}^{U+V-\Delta z \text{ bits}}$$

(trailing $U+V-\Delta z$ bits are first bits of new $B_n$)

$$c_{n-1} : (\textit{settled bits}) \quad \underbrace{0111\cdots 1}_{c_{n-1} \text{ bits}} \quad 0000000000\cdots$$

(outstanding bits can be inverted by carry $c$)

**1** Check for carry

- ➡ If $(c=1)$, output one '1' and $(c_{n-1}-2)$ '0's, remove carry (set $c=0$), and set $c_{n-1}=1$
- ➡ From now on, $B_n^*$ is considered a $(U+V)$-bit integer

**2** If $(\Delta z > 0)$, determine number $n_1$ of trailing ones in $\Delta z$ leading bits of $(U+V)$-bit integer $B_n^*$

- **a** If $(n_1 < \Delta z)$, output all $c_{n-1}$ outstanding bits, output first $(\Delta z - n_1 - 1)$ bits of $B_n^*$, set $c_n = n_1 + 1$

# Output of Settled Bits and Update of Outstanding Counter

Investigate intermediate $(U+V+1)$-bit integer $B_n^*$ and outstanding counter $c_{n-1}$

$$
\begin{array}{rlllll}
B_n^* : & & c & \overbrace{zz\cdots zz}^{\Delta z \text{ bits}} & \overbrace{xxx\cdots xxx}^{U+V-\Delta z \text{ bits}} & \text{(trailing } U+V-\Delta z \text{ bits are first bits of new } B_n) \\
c_{n-1} : & (\,\textit{settled bits}\,) & \underbrace{0111\cdots1}_{c_{n-1} \text{ bits}} & 0000000000\cdots & & \text{(outstanding bits can be inverted by carry } c)
\end{array}
$$

**1** Check for carry

- ➡ If $(c = 1)$, output one '1' and $(c_{n-1} - 2)$ '0's, remove carry (set $c = 0$), and set $c_{n-1} = 1$
- ➡ From now on, $B_n^*$ is considered a $(U+V)$-bit integer

**2** If $(\Delta z > 0)$, determine number $n_1$ of trailing ones in $\Delta z$ leading bits of $(U+V)$-bit integer $B_n^*$

- **a** If $(n_1 < \Delta z)$, output all $c_{n-1}$ outstanding bits, output first $(\Delta z - n_1 - 1)$ bits of $B_n^*$, set $c_n = n_1 + 1$
- **b** If $(n_1 = \Delta z \;\; \&\& \;\; c_{n-1} > 0)$, increase outstanding counter $c_n = c_{n-1} + n_1$

# Output of Settled Bits and Update of Outstanding Counter

Investigate intermediate $(U+V+1)$-bit integer $B_n^*$ and outstanding counter $c_{n-1}$

$$B_n^* : \quad {\color{blue} c} \; \overbrace{{\color{red} zz \cdots zz}}^{\Delta z \text{ bits}} \; \overbrace{{\color{green} xxx \cdots xxx}}^{U+V-\Delta z \text{ bits}} \qquad (\text{trailing } U+V-\Delta z \text{ bits are first bits of new } B_n)$$

$$c_{n-1} : (\textit{settled bits}) \; \underbrace{{\color{purple} 0111 \cdots 1}}_{c_{n-1} \text{ bits}} \; 0000000000 \cdots \qquad (\text{outstanding bits can be inverted by carry } c)$$

**1** Check for carry

- ➡ If $(c = 1)$, output one '1' and $(c_{n-1} - 2)$ '0's, remove carry (set $c = 0$), and set $c_{n-1} = 1$
- ➡ From now on, $B_n^*$ is considered a $(U+V)$-bit integer

**2** If $(\Delta z > 0)$, determine number $n_1$ of trailing ones in $\Delta z$ leading bits of $(U+V)$-bit integer $B_n^*$

- **a** If $(n_1 < \Delta z)$, output all $c_{n-1}$ outstanding bits, output first $(\Delta z - n_1 - 1)$ bits of $B_n^*$, set $c_n = n_1 + 1$
- **b** If $(n_1 = \Delta z \;\&\&\; c_{n-1} > 0)$, increase outstanding counter $c_n = c_{n-1} + n_1$
- **c** If $(n_1 = \Delta z \;\&\&\; c_{n-1} = 0)$, output first $\Delta z$ bits of $B_n^*$, set $c_n = 0$ (rare case)

# Termination of Arithmetic Codeword

**Total Number of Bits for Arithmetic Codeword** (with $a = 1$ for prefix-free, and $a = 0$ otherwise)

$$K = \left\lceil a - \log_2 W_N \right\rceil$$

# Termination of Arithmetic Codeword

**Total Number of Bits for Arithmetic Codeword** (with $a = 1$ for prefix-free, and $a = 0$ otherwise)

$$K = \left\lceil a - \log_2 W_N \right\rceil = \left\lceil a - \log_2 \left( A_N \cdot 2^{-z_n} \right) \right\rceil$$

# Termination of Arithmetic Codeword

**Total Number of Bits for Arithmetic Codeword** (with $a = 1$ for prefix-free, and $a = 0$ otherwise)

$$K = \left\lceil a - \log_2 W_N \right\rceil = \left\lceil a - \log_2 \left( A_N \cdot 2^{-z_n} \right) \right\rceil = a + z_n - \left\lfloor \log_2 A_N \right\rfloor$$

# Termination of Arithmetic Codeword

**Total Number of Bits for Arithmetic Codeword** (with $a = 1$ for prefix-free, and $a = 0$ otherwise)

$$K = \left\lceil a - \log_2 W_N \right\rceil = \left\lceil a - \log_2 \left( A_N \cdot 2^{-z_n} \right) \right\rceil = a + z_n - \left\lfloor \log_2 A_N \right\rfloor = a + z_N - U + 1$$

# Termination of Arithmetic Codeword

**Total Number of Bits for Arithmetic Codeword** (with $a = 1$ for prefix-free, and $a = 0$ otherwise)

$$K = \left\lceil a - \log_2 W_N \right\rceil = \left\lceil a - \log_2 \left( A_N \cdot 2^{-z_n} \right) \right\rceil = a + z_n - \left\lfloor \log_2 A_N \right\rfloor = a + z_N - U + 1$$

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

# Termination of Arithmetic Codeword

**Total Number of Bits for Arithmetic Codeword** (with $a = 1$ for prefix-free, and $a = 0$ otherwise)

$$K = \left\lceil a - \log_2 W_N \right\rceil = \left\lceil a - \log_2 \left( A_N \cdot 2^{-z_n} \right) \right\rceil = a + z_n - \left\lfloor \log_2 A_N \right\rfloor = a + z_N - U + 1$$

- ➡ Note: The sum of settled and outstanding bits is $z_N - U$
- ➡ Need to output all $c_N$ outstanding bits and first $(1+a)$ bits of $B_N$

# Termination of Arithmetic Codeword

## Total Number of Bits for Arithmetic Codeword (with $a = 1$ for prefix-free, and $a = 0$ otherwise)

$$K = \left\lceil a - \log_2 W_N \right\rceil = \left\lceil a - \log_2 \left( A_N \cdot 2^{-z_n} \right) \right\rceil = a + z_n - \left\lfloor \log_2 A_N \right\rfloor = a + z_N - U + 1$$

- ➡ Note: The sum of settled and outstanding bits is $z_N - U$
- ➡ Need to output all $c_N$ outstanding bits and first $(1{+}a)$ bits of $B_N$

## Codeword Termination

1. Rounding up lower interval boundary

# Termination of Arithmetic Codeword

## Total Number of Bits for Arithmetic Codeword (with $a = 1$ for prefix-free, and $a = 0$ otherwise)

$$K = \left\lceil a - \log_2 W_N \right\rceil = \left\lceil a - \log_2 \left( A_N \cdot 2^{-z_n} \right) \right\rceil = a + z_n - \left\lfloor \log_2 A_N \right\rfloor = a + z_N - U + 1$$

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

➡ Need to output all $c_N$ outstanding bits and first $(1+a)$ bits of $B_N$

## Codeword Termination

1. Rounding up lower interval boundary
   - If any of the last $X = (U + V - a - 1)$ bits in $B_N$ is equal to 1, then
     - Set $B_N = B_N + (1 \ll X)$ (rounding up lower interval boundary to required precision)
     - If carry bit is set in $B_N$, handle carry as in normal interval update

# Termination of Arithmetic Codeword

## Total Number of Bits for Arithmetic Codeword (with $a = 1$ for prefix-free, and $a = 0$ otherwise)

$$K = \left\lceil a - \log_2 W_N \right\rceil = \left\lceil a - \log_2 \left( A_N \cdot 2^{-z_n} \right) \right\rceil = a + z_n - \left\lfloor \log_2 A_N \right\rfloor = a + z_N - U + 1$$

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

➡ Need to output all $c_N$ outstanding bits and first $(1+a)$ bits of $B_N$

## Codeword Termination

1. Rounding up lower interval boundary
   - If any of the last $X = (U+V-a-1)$ bits in $B_N$ is equal to 1, then
     - Set $B_N = B_N + (1 \ll X)$ (rounding up lower interval boundary to required precision)
     - If carry bit is set in $B_N$, handle carry as in normal interval update
2. Output all outstanding bits (one '0' and $(c_N - 1)$ times '1')

# Termination of Arithmetic Codeword

## Total Number of Bits for Arithmetic Codeword (with $a = 1$ for prefix-free, and $a = 0$ otherwise)

$$K = \left\lceil a - \log_2 W_N \right\rceil = \left\lceil a - \log_2 \left( A_N \cdot 2^{-z_n} \right) \right\rceil = a + z_n - \left\lfloor \log_2 A_N \right\rfloor = a + z_N - U + 1$$

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

➡ Need to output all $c_N$ outstanding bits and first $(1{+}a)$ bits of $B_N$

## Codeword Termination

1. Rounding up lower interval boundary
   - If any of the last $X = (U{+}V{-}a{-}1)$ bits in $B_N$ is equal to 1, then
     - Set $B_N = B_N + (1 \ll X)$ (rounding up lower interval boundary to required precision)
     - If carry bit is set in $B_N$, handle carry as in normal interval update
2. Output all outstanding bits (one '0' and $(c_N - 1)$ times '1')
3. Output $(a + 1)$ most significant bits of $B_N$

# Summary: Arithmetic Encoding

1. Initialization: $A_0 = 2^U - 1,\;\; B_0 = 0,\;\; c_0 = 0,\;\; \text{mask} = (1 \ll (U+V)) - 1$

2. Iterative encoding ( for $n = 1$ to $N$ )

   a. Calculate: $A_n^* = A_{n-1} \cdot p_V(s_n)$ ($U + V$ bits)
   
      $B_n^* = B_{n-1} + A_{n-1} \cdot c_V(s_n)$ ($U + V + 1$ bits)

   b. Determine number $\Delta z$ of leading zeros in $(U+V)$-bit integer $A_n^*$

   c. Check for carry bit in $B_n^*$ (and update $c_{n-1}$ and settled bits accordingly)

   d. Investigate $\Delta z$ leading bits in $(U+V)$ bits of $B_n^*$
   
      ➡ Output new settled bits and update counter $c_n$ for new outstanding bits

   e. Update: $A_n = (\, A_n^* \ll \Delta z \,) \gg V$
   
      $B_n = (\, B_n^* \ll \Delta z \,)\; \&\; \text{mask}$

3. Codeword Termination:

   ➡ Round up $B_N$ (check for carry as in iterations)

   ➡ Output $c_N$ outstanding bits

   ➡ Output two most significant bits of $B_N$ (only 1 bit for non-prefix-free variant)

# Arithmetic Encoding Example

## Example: Preparation

- **Encoding 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 Encoding Example (continued)

## Example: Step 1

| $s_n$ | $p_V$ | $c_V$ | parameter updates & output |
|---|---|---|---|
| initialization | | | $A_0 = 15 = \text{'}\mathbf{1111}\text{'} \quad (2^4 - 1)$<br>$c_0 = 0 \quad (\text{''})$<br>$B_0 = 0 = \text{'}\mathbf{0000\ 0000}\text{'}$<br>bitstream = “ ” |
| “B” | 3 | 13 | $A_1^* = A_0 \cdot p_V = 15 \cdot 3 = 45 = \text{'}\mathbf{0010\ 1101}\text{'}$<br>$B_1^* = B_0 + A_0 \cdot c_V = 0 + 15 \cdot 13 = 195 = \text{'}\mathbf{0\ 1100\ 0011}\text{'}$<br>$\Delta z = 2$ |
| | | | **output** = “**11**” (*cannot be outstanding bits*)<br>$c_1 = 0 \quad (\text{''})$<br>$A_1 = \text{'}\mathbf{1011}\text{'} = 11$<br>$B_1 = \text{'}\mathbf{0000\ 1100}\text{'} = 12$<br>bitstream = “**11**” |

# Arithmetic Encoding Example (continued)

## Example: Step 2

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

# Arithmetic Encoding Example (continued)

## Example: Step 3

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

# Arithmetic Encoding Example (continued)

## Example: Step 4

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

# Arithmetic Encoding Example (continued)

## Example: Step 5

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

# Arithmetic Encoding Example (continued)

## Example: Step 6

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

# Arithmetic Encoding Example (final step)

## Example: Codeword Termination

| $s_n$ | $p_V$ $c_V$ | parameter updates & output |
|---|---|---|
| after step 6 | | $A_6 = 8 =$ '**1000**'<br>$c_6 = 5$ ('**01111**')<br>$B_6 = 160 =$ '**1010 0000**'<br>bitstream = "**110**" |
| final rounding | | $B^* =$ "**1 0010 0000**" (rounding up $B_6$)<br>bitstream = "**1101 000**" ($c_6 - 1$ inverted bits)<br>$c = 1$ ('**0**')<br>$B =$ "**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

# Arithmetic Decoding

## Identification of Intervals

- Important: Same rounding of interval width as in encoder ($A_{n-1} \mapsto A_n$)

# Arithmetic Decoding

## Identification of Intervals

- Important: Same rounding of interval width as in encoder $(A_{n-1} \mapsto A_n)$
- Arithmetic codeword "$b_0 b_1 b_2 \cdots$" represents binary fraction $v = (0.b_0 b_1 b_2 \cdots)_\mathrm{b}$

# Arithmetic Decoding

## Identification of Intervals

- Important: Same rounding of interval width as in encoder ($A_{n-1} \mapsto A_n$)
- Arithmetic codeword "$b_0 b_1 b_2 \cdots$" represents binary fraction $v = (0.b_0 b_1 b_2 \cdots)_\mathrm{b}$
- Iterative decoding: Output symbol $s_n$ which fulfills inequality

$$L_{n-1} + W_{n-1} \cdot c(s_n) \;\leq\; v \;<\; L_{n-1} + W_{n-1} \cdot \big(c(s_n) + p(s_n)\big)$$

# Arithmetic Decoding

## Identification of Intervals

- Important: Same rounding of interval width as in encoder ($A_{n-1} \mapsto A_n$)
- Arithmetic codeword "$b_0 b_1 b_2 \cdots$" represents binary fraction $v = (0.b_0 b_1 b_2 \cdots)_\mathrm{b}$
- Iterative decoding: Output symbol $s_n$ which fulfills inequality

$$L_{n-1} + W_{n-1} \cdot c(s_n) \;\leq\; v \;<\; L_{n-1} + W_{n-1} \cdot \big(c(s_n) + p(s_n)\big)$$

## Observation:

- Lower interval boundary cannot be represented with reasonable precision

# Arithmetic Decoding

## Identification of Intervals

- Important: Same rounding of interval width as in encoder ($A_{n-1} \mapsto A_n$)
- Arithmetic codeword "$b_0 b_1 b_2 \cdots$" represents binary fraction $v = (0.b_0 b_1 b_2 \cdots)_\mathrm{b}$
- Iterative decoding: Output symbol $s_n$ which fulfills inequality

$$L_{n-1} + W_{n-1} \cdot c(s_n) \;\leq\; v \;<\; L_{n-1} + W_{n-1} \cdot \big(c(s_n) + p(s_n)\big)$$

## Observation:

- Lower interval boundary cannot be represented with reasonable precision

➡ Idea: Subtract $L_{n-1}$ from the inequality

# Arithmetic Decoding

## Identification of Intervals

- Important: Same rounding of interval width as in encoder ($A_{n-1} \mapsto A_n$)
- Arithmetic codeword "$b_0 b_1 b_2 \cdots$" represents binary fraction $v = (0.b_0 b_1 b_2 \cdots)_\mathrm{b}$
- Iterative decoding: Output symbol $s_n$ which fulfills inequality

$$L_{n-1} + W_{n-1} \cdot c(s_n) \;\leq\; v \;<\; L_{n-1} + W_{n-1} \cdot \big( c(s_n) + p(s_n) \big)$$

## Observation:

- Lower interval boundary cannot be represented with reasonable precision

➡ Idea: Subtract $L_{n-1}$ from the inequality

➡ Symbol $s_n$ is identified by

$$W_{n-1} \cdot c(s_n) \;\leq\; \big( v - L_{n-1} \big) \;<\; W_{n-1} \cdot \big( c(s_n) + p(s_n) \big)$$

# Arithmetic Decoding

## Identification of Intervals

- Important: Same rounding of interval width as in encoder ($A_{n-1} \mapsto A_n$)
- Arithmetic codeword "$b_0 b_1 b_2 \cdots$" represents binary fraction $v = (0.b_0 b_1 b_2 \cdots)_\mathrm{b}$
- Iterative decoding: Output symbol $s_n$ which fulfills inequality

$$L_{n-1} + W_{n-1} \cdot c(s_n) \;\leq\; v \;<\; L_{n-1} + W_{n-1} \cdot \big( c(s_n) + p(s_n) \big)$$

## Observation:

- Lower interval boundary cannot be represented with reasonable precision

➡ Idea: Subtract $L_{n-1}$ from the inequality

➡ Symbol $s_n$ is identified by

$$W_{n-1} \cdot c(s_n) \;\leq\; \big( v - L_{n-1} \big) \;<\; W_{n-1} \cdot \big( c(s_n) + p(s_n) \big)$$

➡ The value $u_{n-1} = v - L_{n-1}$ used in comparisons can be stored with $(U + V)$ bits, but needs to be updated after a symbol $s_n$ is decoded

# Binary Representation of Representative Value $u_n = v - L_n$

$$W_{n-1} \;=\; 0.\overbrace{0000000000000\cdots0}^{z_{n-1}-U}\;\overbrace{1xx\cdots x}^{U}\;0000000000000\cdots$$

# Binary Representation of Representative Value $u_n = v - L_n$

$$W_{n-1} \;=\; 0.\overbrace{0000000000000\cdots0}^{z_{n-1}-U}\;\overbrace{1xx\cdots x}^{U}\;00000000000000\cdots$$

$$W_{n-1}\cdot(c(s_n)+p(s_n)) \;=\; 0.\overbrace{0000000000000\cdots0}^{z_{n-1}-U}\;\overbrace{xxx\cdots xxxxxx}^{U+V}\;000000000\cdots$$

$$W_{n-1}\cdot c(s_n) \;=\; 0.\underbrace{0000000000000\cdots0}_{z_{n-1}-U}\;\underbrace{xxx\cdots xxxxxx}_{U+V}\;000000000\cdots$$

# Binary Representation of Representative Value $u_n = v - L_n$

$$W_{n-1} = 0.\overbrace{0000000000000\cdots0}^{z_{n-1}-U}\;\overbrace{1xx\cdots x}^{U}\;00000000000000\cdots$$

$$\begin{aligned}
W_{n-1}\cdot\big(c(s_n)+p(s_n)\big) &= 0.\overbrace{0000000000000\cdots0}^{z_{n-1}-U}\;\overbrace{xxx\cdots xxxxxx}^{U+V}\;000000000\cdots \\
{\color{blue}v - L_{n-1}} &= {\color{blue}0.\,0000000000000\cdots0\;\;xxx\cdots xxxxxx\;\;xxxxxxxxx\;\cdots} \\
W_{n-1}\cdot c(s_n) &= 0.\underbrace{0000000000000\cdots0}_{z_{n-1}-U}\;\underbrace{xxx\cdots xxxxxx}_{U+V}\;000000000\cdots
\end{aligned}$$

# Binary Representation of Representative Value $u_n = v - L_n$

$$W_{n-1} = 0.\overbrace{0000000000000\cdots0}^{z_{n-1}-U}\;\overbrace{1xx\cdots x}^{U}\;00000000000000\cdots$$

$$\begin{aligned}
W_{n-1}\cdot(c(s_n)+p(s_n)) &= 0.\overbrace{0000000000000\cdots0}^{z_{n-1}-U}\;\overbrace{xxx\cdots xxxxxx}^{U+V}\;000000000\cdots \\
v - L_{n-1} &= 0.\,0000000000000\cdots0\;\;xxx\cdots xxxxxx\;\;xxxxxxxxx\;\cdots \\
W_{n-1}\cdot c(s_n) &= 0.\underbrace{0000000000000\cdots0}_{z_{n-1}-U}\;\underbrace{xxx\cdots xxxxxx}_{U+V}\;000000000\cdots
\end{aligned}$$

Use $(U+V)$-bit integer $u_n$ in comparisons (down-rounded value of $v - L_n$)

# Binary Representation of Representative Value $u_n = v - L_n$

$$W_{n-1} = 0.\overbrace{0000000000000\cdots0}^{z_{n-1}-U}\ \overbrace{1xx\cdots x}^{U}\ 00000000000000\cdots$$

$$\begin{aligned}
W_{n-1}\cdot(c(s_n)+p(s_n)) &= 0.\overbrace{0000000000000\cdots0}^{z_{n-1}-U}\ \overbrace{xxx\cdots xxxxxx}^{U+V}\ 000000000\cdots \\
v - L_{n-1} &= 0.\ 0000000000000\cdots0\ \ xxx\cdots xxxxxx\ \ xxxxxxxxx\ \cdots \\
W_{n-1}\cdot c(s_n) &= 0.\underbrace{0000000000000\cdots0}_{z_{n-1}-U}\ \underbrace{xxx\cdots xxxxxx}_{U+V}\ 000000000\cdots
\end{aligned}$$

Use $(U+V)$-bit integer $u_n$ in comparisons (down-rounded value of $v - L_n$)

1. Initialization: $u_0 = $ (first $U+V$ bits from bitstream)

# Binary Representation of Representative Value $u_n = v - L_n$

$$W_{n-1} = 0.\overbrace{0000000000000\cdots0}^{z_{n-1}-U}\;\overbrace{1xx\cdots x}^{U}\;0000000000000\cdots$$

$$\begin{aligned}
W_{n-1}\cdot(c(s_n)+p(s_n)) &= 0.\overbrace{0000000000000\cdots0}^{z_{n-1}-U}\;\overbrace{xxx\cdots xxxxxx}^{U+V}\;000000000\cdots \\
v - L_{n-1} &= 0.\,0000000000000\cdots0\;xxx\cdots xxxxxx\;xxxxxxxxx\;\cdots \\
W_{n-1}\cdot c(s_n) &= 0.\underbrace{0000000000000\cdots0}_{z_{n-1}-U}\;\underbrace{xxx\cdots xxxxxx}_{U+V}\;000000000\cdots
\end{aligned}$$

Use $(U+V)$-bit integer $u_n$ in comparisons (down-rounded value of $v - L_n$)

1. Initialization: $u_0 = $ (first $U+V$ bits from bitstream)
2. Update $u_{n-1} \mapsto u_n$

# Binary Representation of Representative Value $u_n = v - L_n$

$$W_{n-1} = 0.\overbrace{0000000000000\cdots0}^{z_{n-1}-U}\;\overbrace{1xx\cdots x}^{U}\;0000000000000\cdots$$

$$\begin{aligned}
W_{n-1}\cdot(c(s_n)+p(s_n)) &= 0.\overbrace{0000000000000\cdots0}^{z_{n-1}-U}\;\overbrace{xxx\cdots xxxxxx}^{U+V}\;000000000\cdots \\
v - L_{n-1} &= 0.\,0000000000000\cdots0\;\;xxx\cdots xxxxxx\;\;xxxxxxxxx\;\cdots \\
W_{n-1}\cdot c(s_n) &= 0.\underbrace{0000000000000\cdots0}_{z_{n-1}-U}\;\underbrace{xxx\cdots xxxxxx}_{U+V}\;000000000\cdots
\end{aligned}$$

Use $(U+V)$-bit integer $u_n$ in comparisons (down-rounded value of $v - L_n$)

1. Initialization: $u_0 = (\text{first } U+V \text{ bits from bitstream})$
2. Update $u_{n-1} \mapsto u_n$
   - ➡ Subtract lower boundary: $u_n^* = u_{n-1} - A_{n-1}\cdot c_V(s_n)$

# Binary Representation of Representative Value $u_n = v - L_n$

$$W_{n-1} = 0.\overbrace{0000000000000\cdots0}^{z_{n-1}-U}\ \overbrace{1xx\cdots x}^{U}\ 0000000000000\cdots$$

$$\begin{aligned}
W_{n-1}\cdot(c(s_n)+p(s_n)) &= 0.\overbrace{0000000000000\cdots0}^{z_{n-1}-U}\ \overbrace{xxx\cdots xxxxxx}^{U+V}\ 000000000\cdots \\
{\color{blue}v - L_{n-1}} &= {\color{blue}0.\ 0000000000000\cdots0\ \ xxx\cdots xxxxxx\ \ xxxxxxxxx\ \cdots} \\
W_{n-1}\cdot c(s_n) &= 0.\underbrace{0000000000000\cdots0}_{z_{n-1}-U}\ \underbrace{xxx\cdots xxxxxx}_{U+V}\ 000000000\cdots
\end{aligned}$$

Use $(U+V)$-bit integer $u_n$ in comparisons (down-rounded value of $v - L_n$)

1. Initialization: $u_0 =$ (first $U+V$ bits from bitstream)
2. Update $u_{n-1} \mapsto u_n$
   - ➡ Subtract lower boundary: $u_n^* = u_{n-1} - A_{n-1}\cdot c_V(s_n)$
   - ➡ Align with interval width: $u_n^{**} = u_n^* \ll \Delta z$ $\quad$ ($\Delta z =$ leading zeros in $A_n \cdot p_V(s_n)$)

# Binary Representation of Representative Value $u_n = v - L_n$

$$W_{n-1} = 0.\overbrace{0000000000000\cdots0}^{z_{n-1}-U}\;\overbrace{1xx\cdots x}^{U}\;00000000000000\cdots$$

$$\begin{aligned}
W_{n-1}\cdot(c(s_n)+p(s_n)) &= 0.\overbrace{0000000000000\cdots0}^{z_{n-1}-U}\;\overbrace{xxx\cdots xxxxxx}^{U+V}\;000000000\cdots\\
{\color{blue}v - L_{n-1}} &= {\color{blue}0.\,0000000000000\cdots0\;\; xxx\cdots xxxxxx\;\; xxxxxxxxx\;\cdots}\\
W_{n-1}\cdot c(s_n) &= 0.\underbrace{0000000000000\cdots0}_{z_{n-1}-U}\;\underbrace{xxx\cdots xxxxxx}_{U+V}\;000000000\cdots
\end{aligned}$$

Use $(U+V)$-bit integer $u_n$ in comparisons (down-rounded value of $v - L_n$)

1. Initialization: $u_0 =$ (first $U+V$ bits from bitstream)

2. Update $u_{n-1} \mapsto u_n$
   - ➡ Subtract lower boundary: $u_n^* = u_{n-1} - A_{n-1} \cdot c_V(s_n)$
   - ➡ Align with interval width: $u_n^{**} = u_n^* \ll \Delta z$ $\quad$ ($\Delta z =$ leading zeros in $A_n \cdot p_V(s_n)$)
   - ➡ $u_n^{**} \mapsto u_n$: Fill least significant bits with next $\Delta z$ bits from bitstream

# Summary: Arithmetic Decoding

**1** Initialization: $A_0 = 2^U - 1,$
$u_0 = (\text{first } U+V \text{ bits from bitstream})$

**2** Iterative decoding (for $n = 1$ to $N$)

**a** **Identify next symbol**: For $k = 0, 1, 2, \cdots$ (*loop over sorted symbol alphabet*)

- Calculate upper boundary $U(a_k) = A_{n-1} \cdot (c_V(a_k) + p_V(a_k))$
- If $(\, u_{n-1} < U(a_k) \,)$, then
  - ➡ Output next symbol $s_n = a_k$
  - ➡ break loop over $k$

**b** **Update parameters**:

- Calculate intermediate value: $A_n^* = A_{n-1} \cdot p_V(s_n)$
- Determine number $\Delta z$ of leading zeros in $(U+V)$-bit integer $A_n^*$

➡ $A_n = \left( A_n^* \ll \Delta z \right) \gg V$

➡ $u_n = \left( \left( u_{n-1} - A_{n-1} \cdot c_V(s_n) \right) \ll \Delta z \right) + (\text{next } \Delta z \text{ bits from bitstream})$

# Arithmetic Decoding Example

## Decode Bitstream obtained in Encoding Example

- **Decoding example** (see encoding example)
  - IID source with symbol alphabet $\{A, N, B\}$
  - 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 (continued)

## Example: Step 1

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

# Arithmetic Decoding Example (continued)

## Example: Step 2

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

# Arithmetic Decoding Example (continued)

## Example: Step 3

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

# Arithmetic Decoding Example (continued)

## Example: Step 4

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

# Arithmetic Decoding Example (continued)

## Example: Step 5

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

# Arithmetic Decoding Example (final step)

## Example: Step 6 (last symbol)

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

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

# Coding Efficiency of Arithmetic Coding

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

- Increase due to rounding of interval width

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

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

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

(for a derivation see [Wiegand, Schwarz, "Source Coding", 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

# Binary Arithmetic Coding

- Binarization of $S \in \{a_1, a_2, \ldots, a_M\}$ produces $C \in \{0, 1\}$
  - ➡ Any prefix code can be used for binarization
  - ➡ Example: Truncated unary binarization

| $a_k$ | number of bins $B$ | $C_1$ | $C_2$ | $C_3$ | $\cdots$ | $C_{M-1}$ | $C_M$ |
|---|---|---|---|---|---|---|---|
| $a_1$ | 1 | 1 | | | | | |
| $a_2$ | 2 | 0 | 1 | | | | |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\ddots$ | | | |
| $a_{M-2}$ | $M-2$ | 0 | 0 | 0 | $\cdots$ | 1 | |
| $a_{M-1}$ | $M-1$ | 0 | 0 | 0 | $\cdots$ | 0 | 1 |
| $a_M$ | $M-1$ | 0 | 0 | 0 | $\cdots$ | 0 | 0 |

# Binary Arithmetic Coding

- Binarization of $S \in \{a_1, a_2, \ldots, a_M\}$ produces $C \in \{0, 1\}$
  - ➡ Any prefix code can be used for binarization
  - ➡ Example: Truncated unary binarization

| $a_k$ | number of bins $B$ | $C_1$ | $C_2$ | $C_3$ | $\cdots$ | $C_{M-1}$ | $C_M$ |
|---|---|---|---|---|---|---|---|
| $a_1$ | 1 | 1 | | | | | |
| $a_2$ | 2 | 0 | 1 | | | | |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\ddots$ | | | |
| $a_{M-2}$ | $M-2$ | 0 | 0 | 0 | $\cdots$ | 1 | |
| $a_{M-1}$ | $M-1$ | 0 | 0 | 0 | $\cdots$ | 0 | 1 |
| $a_M$ | $M-1$ | 0 | 0 | 0 | $\cdots$ | 0 | 0 |

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

# Binary Arithmetic Coding

- Binarization of $S \in \{a_1, a_2, \ldots, a_M\}$ produces $C \in \{0, 1\}$
  - ➡ Any prefix code can be used for binarization
  - ➡ Example: Truncated unary binarization

| $a_k$ | number of bins $B$ | $C_1$ | $C_2$ | $C_3$ | $\cdots$ | $C_{M-1}$ | $C_M$ |
|---|---|---|---|---|---|---|---|
| $a_1$ | 1 | 1 | | | | | |
| $a_2$ | 2 | 0 | 1 | | | | |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\ddots$ | | | |
| $a_{M-2}$ | $M-2$ | 0 | 0 | 0 | $\cdots$ | 1 | |
| $a_{M-1}$ | $M-1$ | 0 | 0 | 0 | $\cdots$ | 0 | 1 |
| $a_M$ | $M-1$ | 0 | 0 | 0 | $\cdots$ | 0 | 0 |

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

➡ **Most popular form of arithmetic coding in practice**

- ➡ Reduced complexity, better adaptation capabilities
- ➡ Used in JPEG-2000, H.264/AVC, H.265/HEVC, VVC

# Arithmetic Coding in Practice

**Complexity Reduction** (example: CABAC in AVC and HEVC)

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

# Arithmetic Coding in Practice

**Complexity Reduction** (example: CABAC in AVC and HEVC)

- 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)

# Arithmetic Coding in Practice

## Complexity Reduction (example: CABAC in AVC and HEVC)

- 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 $p(a \,|\, X)$
   - Use context switching only when useful (certain bins)
2. Estimate probabilities during coding
   - Choose appropriate "window sizes" for estimation

# Arithmetic Coding in Practice

## Complexity Reduction (example: CABAC in AVC and HEVC)

- 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 $p(a \,|\, X)$
   - 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: Stationary 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 |

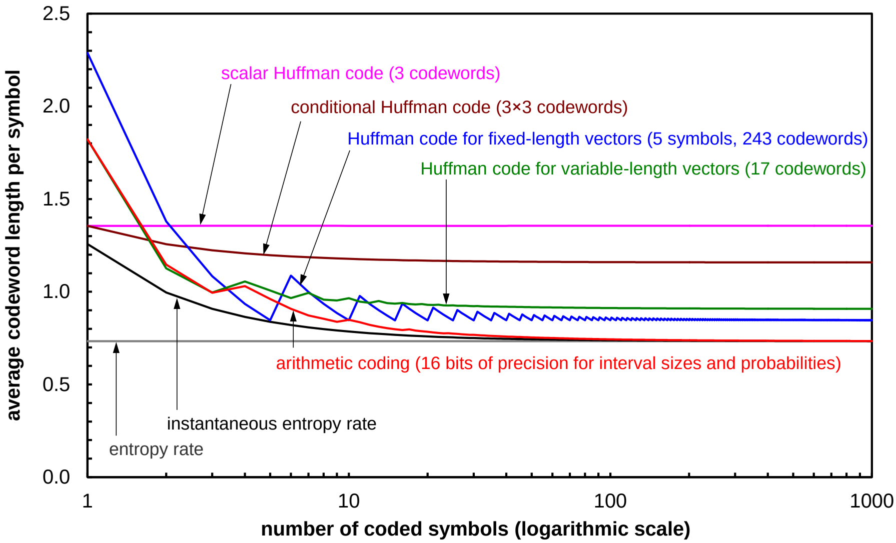
$$H(S) = 1.2575$$
$$\bar{H}(\boldsymbol{S}) = 0.7331$$

- 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

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

- 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 for Stationary Markov Source



# Summary of Lecture: Arithmetic Coding

## Arithmetic Coding

- Suboptimal block code (finite precision realization of Shannon-Fano-Elias coding)
- No codeword table required
- ➞ Iterative construction of codeword
- ➞ Very close to entropy bound for $N \gg 1$
- ➞ Well suited for exploiting statistical dependencies
- ➞ Well suited for adapting probabilites during coding

## Arithmetic Coding in Practice

- Typically only binary arithmetic coding (using simple structured codes for binarization)
- Low-complexity variants (multiplication-free, extra low-complex bypass mode)
- Adaptive probability models (updated after encoding/decoding each symbol)
- Often context-based selection of probability models

# Exercise 1: Study Arithmetic Codec

On the course web-site, you find an implementation of an arithmetic encoder and decoder in C++:

| | |
|---|---|
| bitstream.h | classes for input and output bitstreams (header only) |
| arithCoding.h | header for arithmetic encoder and decoder classes |
| arithCoding.cpp | implementation of arithmetic encoder and decoder classes |
| main.cpp | toy example for usage of arithmetic encoder and decoder |

1. Study the arithmetic encoder and decoder in detail (compare it with the lecture slides)
2. Play around with the implementation, try different value for $U$.
   Try other example messages, pmfs (including values of $V$), and alphabets.
3. If you don't want to use C++ for the following exercises, rewrite the implementation using a programming language of your choice

Note: We don't need to modify the bitstream classes and the actual implementation of the arithmetic codec in the following exercises.

We only need to modify the Pmf class (in main.cpp) and the usage of the arithmetic codec.

# Exercise 2: Arithmetic Coding of 8-bit Audio Data

In the following we want to efficiently code the 8-bit audio file "audioData.raw" from the course web site.

1. Write a first encoder and decoder that use a fixed pmf. The encoder should do the following:
   - Count the number of samples (bytes) in the input file and write it as 32-bit integer at the beginning of the bitstream (use function OBitstream::addFixed(.)).
   - Estimate the marginal pmf, quantize it to $V$-bits of precision (choose suitable $V$, check validity).
   - Write all 256 probability masses $p_V(x)$ to the bitstream (each using $V$-bits).
   - Encode all samples of the input file using arithmetic coding with the estimated pmf.

2. Think about how you can estimate the pmf during encoding and decoding.
   Implement a pmf class that estimates the pmf during encoding and decoding
   (there is already a pure virtual function IPmf::update(.) in the interface class IPmf).
   In principle, you have to count symbols during encoding and decoding.

3. Once you have a working adaptive pmf, try to improve the codec by using conditional coding. That means, use 256 different adaptive pmfs in your encoder and decoder. The pmf for a current sample has to be selected based on the value of the previous sample.