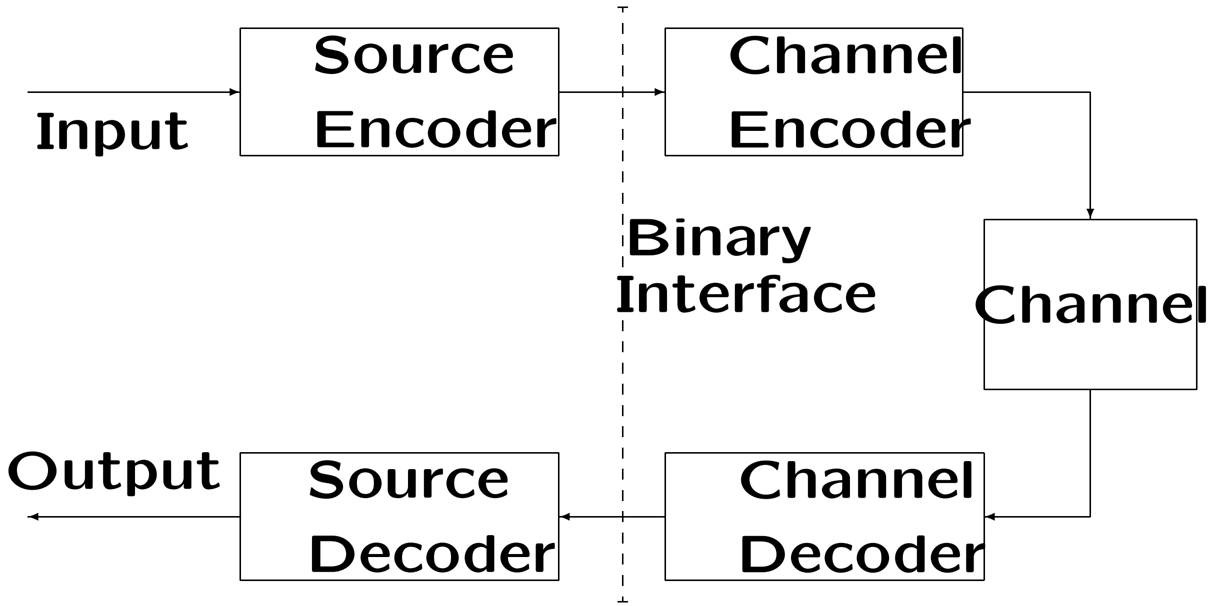
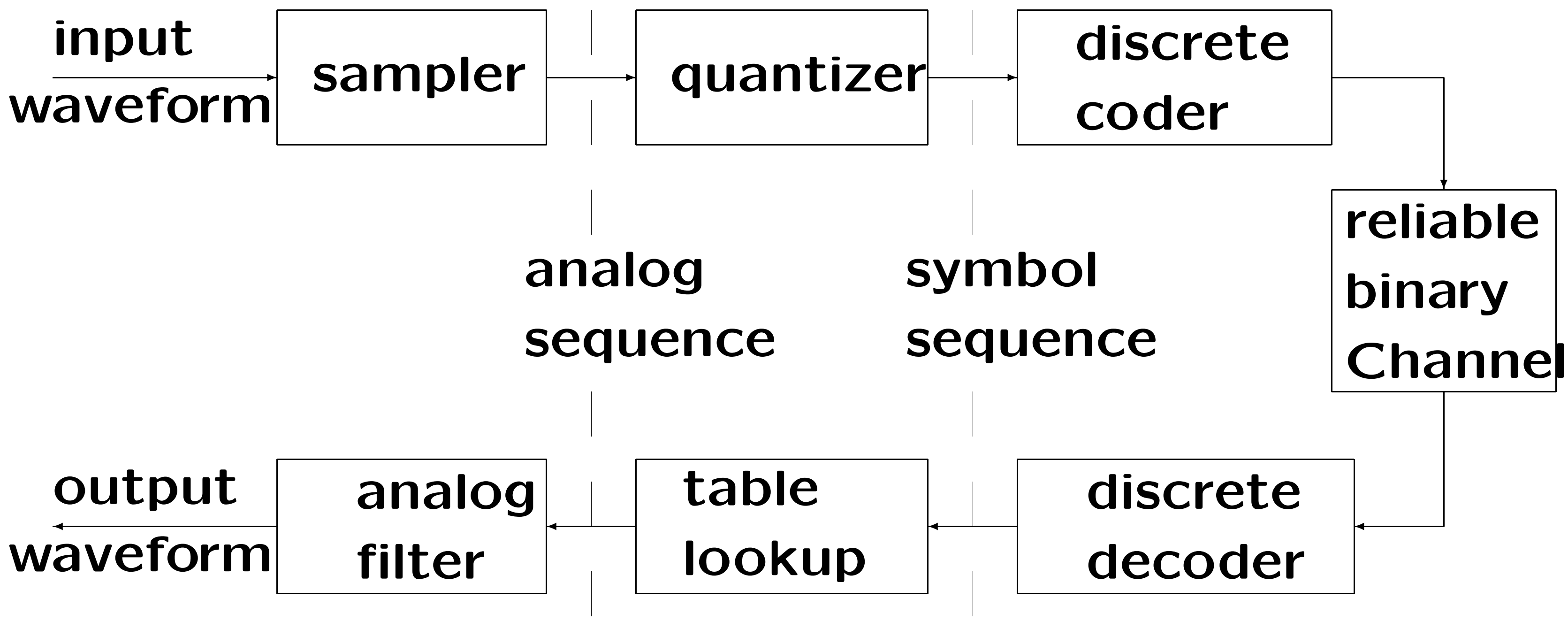
6.450, Lecture 2, 9/14/09; REVIEW



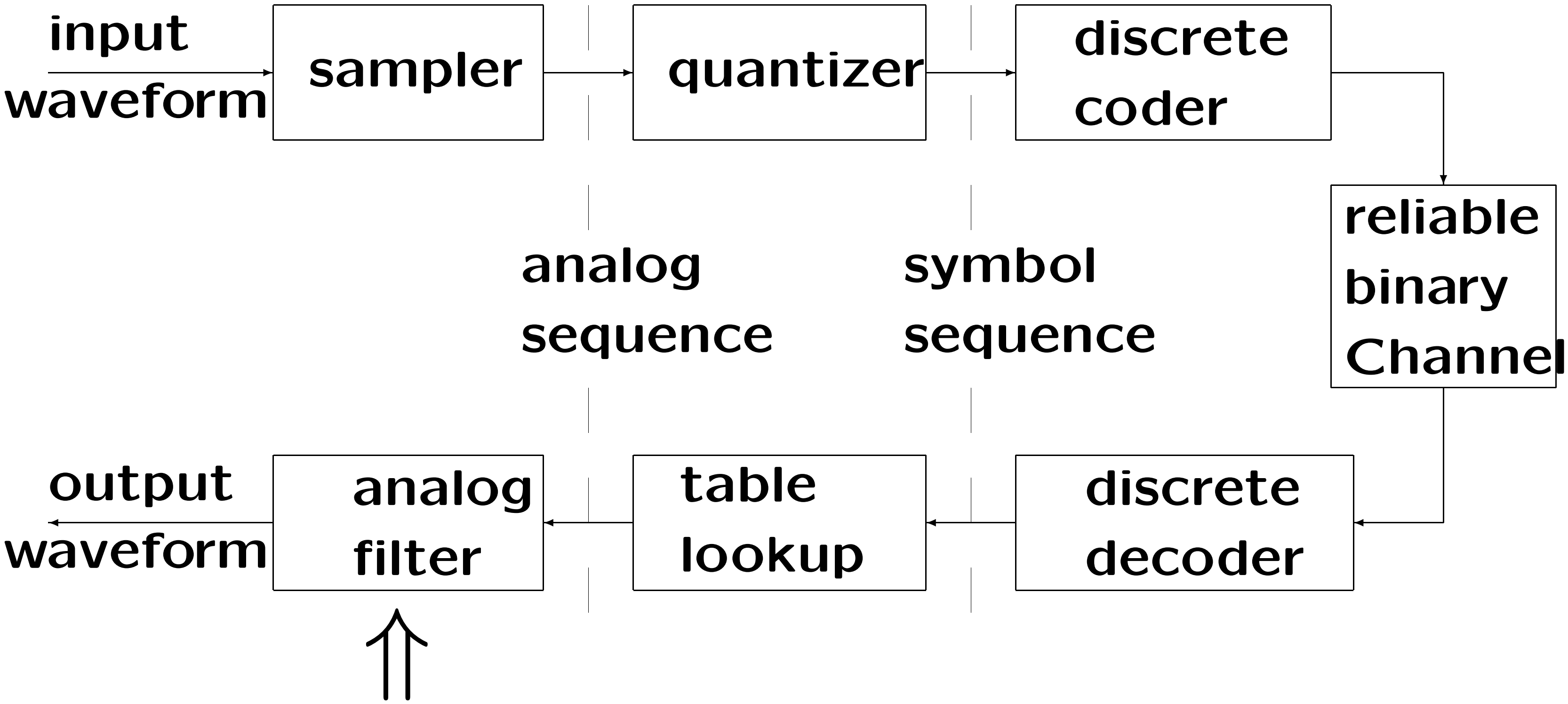
Separation of source and channel coding.

REASONS FOR BINARY INTERFACE

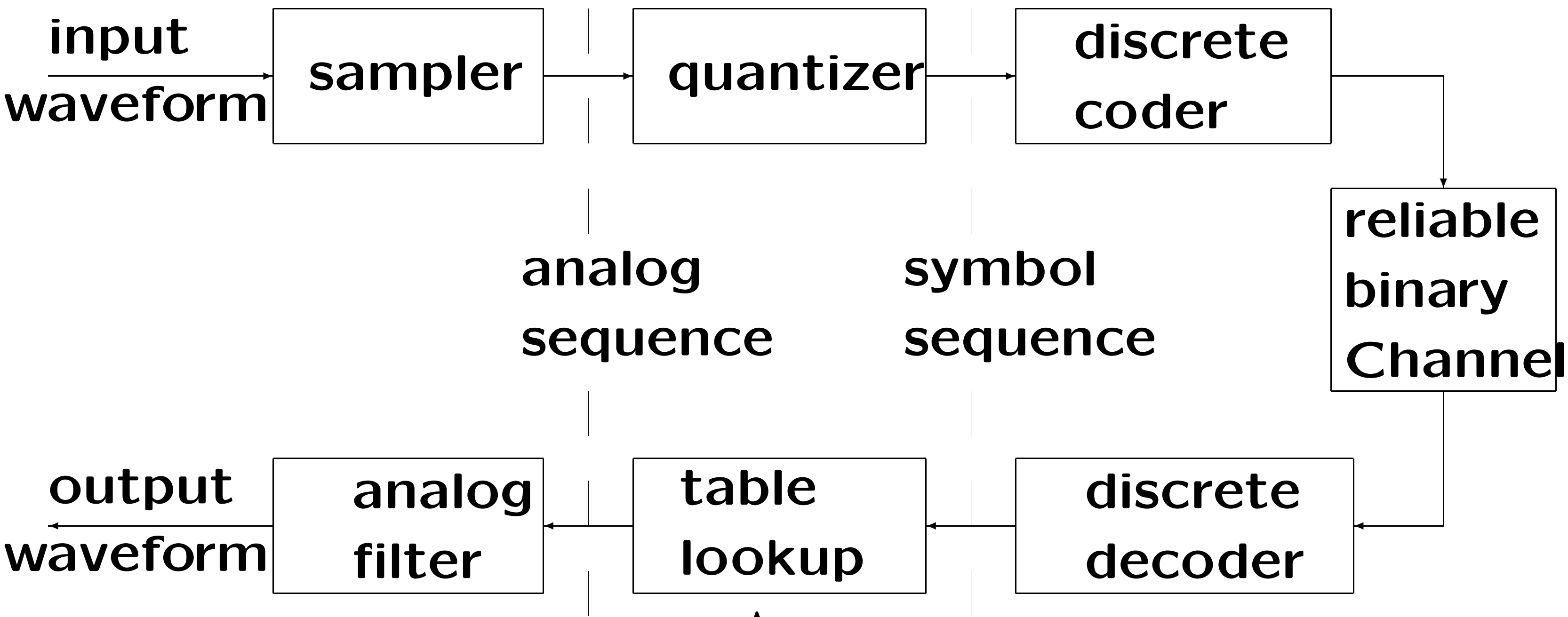
- Standardization (Simplifies implementation)
- Layering (Simplifies conceptualization)
- Loses nothing in performance (Shannon says)



Layering of source coding

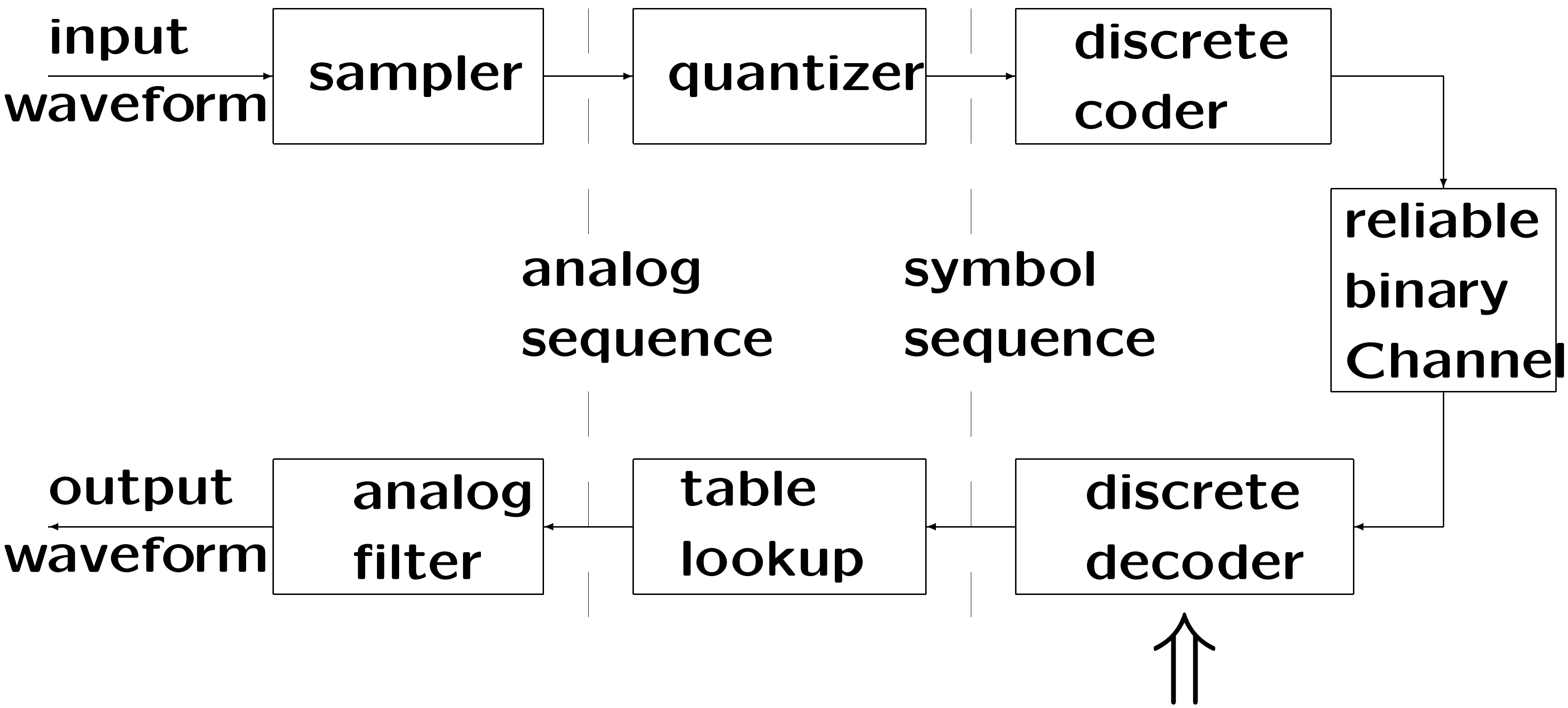


A waveform source is usually sampled or expanded into a series, producing a sequence of real or complex numbers.



$\Uparrow$

The analog sequence is encoded by quantization into sequence of symbols.



Both analog and discrete sources then require
binary encoding of sequence of symbols.

DISCRETE SOURCE CODING

OBJECTIVE: Map sequence of symbols into binary sequence with unique decodability.

SIMPLEST APPROACH: Map each source symbol into an L-tuple of binary digits.

Choose L as smallest integer satisfying $2^L \geq M$, i.e.,

$$\log_2 M \leq L < \log_2 M + 1; \qquad L = \lceil \log_2 M \rceil$$

Example (for alphabet $\{$red, blue, green, yellow, purple, magenta$\}$):

$$\begin{array}{ll} \text{red} \rightarrow & 000 \\ \text{blue} \rightarrow & 001 \\ \text{green} \rightarrow & 010 \\ \text{yellow} \rightarrow & 011 \\ \text{purple} \rightarrow & 100 \\ \text{magenta} \rightarrow & 101 \end{array}$$

This can be easily decoded.

Example: the ASCII code maps letters, numbers, etc. into bytes.

These are called fixed length codes.

FIXED-TO-FIXED LENGTH SOURCE CODES

Segment source symbols into n-tuples.

Map each n-tuple into binary L-tuple where

$$\log_2 M^n \le L < \log_2 M^n + 1; \qquad L = \lceil n \log_2 M \rceil$$

Let $\overline{L} = \frac{L}{n}$ be number of bits per source symbol

$$\log_2 M \le \overline{L} < \log_2 M + \frac{1}{n}$$

VARIABLE LENGTH SOURCE CODES

Motivation: Probable symbols should have shorter codewords than improbable to reduce bpss.

A variable-length source code $\mathcal{C}$ encodes each symbol x in source alphabet $\mathcal{X}$ to a binary codeword $\mathcal{C}(x)$ of length $l(x)$.

For example, for $\mathcal{X} = \{a, b, c\}$

$$\begin{aligned} \mathcal{C}(a) &= \ 0 \\ \mathcal{C}(b) &= \ 10 \\ \mathcal{C}(c) &= \ 11 \end{aligned}$$

Decoder must parse the received sequence.

Requires unique decodability: For every string of source letters $\{x_1, x_2, \ldots, x_n\}$, the encoded output $\{\mathcal{C}(x_1)\mathcal{C}(x_2), \ldots, \mathcal{C}(x_n)\}$ must be distinct, i.e., must differ from $\{\mathcal{C}(x'_1)\mathcal{C}(x'_2), \ldots, \mathcal{C}(x'_m)\}$ for any other source string $\{x'_1, \ldots, x'_m\}$.

If $\mathcal{C}(x_1)\cdots\mathcal{C}(x_n) = \mathcal{C}(x'_1)\cdots\mathcal{C}(x'_m)$, decoder must fail on one of these inputs.

We will show that prefix-free codes are uniquely decodable.

Unique Decodability: For every string of source letters $\{x_1, x_2, \ldots, x_n\}$, the encoded output $\{\mathcal{C}(x_1)\mathcal{C}(x_2), \ldots, \mathcal{C}(x_n)\}$ must be distinct, i.e., must differ from $\{\mathcal{C}(x_1')\mathcal{C}(x_2'), \ldots, \mathcal{C}(x_m')\}$ for any other source string $\{x_1', \ldots, x_m'\}$.

If $\mathcal{C}(x_1)\cdots\mathcal{C}(x_n) = \mathcal{C}(x_1')\cdots\mathcal{C}(x_m')$, decoder must fail on one of these inputs.

Example: Consider $a \to 0, b \to 01, c \to 10$

Then $ac \to 010$ and $ba \to 010$,

Not uniquely decodable.

Unique Decodability: For every string of source letters $\{x_1, x_2, \ldots, x_n\}$, the encoded output $\{\mathcal{C}(x_1)\mathcal{C}(x_2)\cdots\mathcal{C}(x_n)\}$ must be distinct, i.e., must differ from $\{\mathcal{C}(x'_1)\mathcal{C}(x'_2)\cdots\mathcal{C}(x'_m)\}$ for any other source string $\{x'_1, \ldots, x'_m\}$.

If $\mathcal{C}(x_1)\cdots\mathcal{C}(x_n) = \mathcal{C}(x'_1)\cdots\mathcal{C}(x'_m)$, decoder must fail on one of these inputs.

Example: Consider $a \to 0, b \to 01, c \to 11$

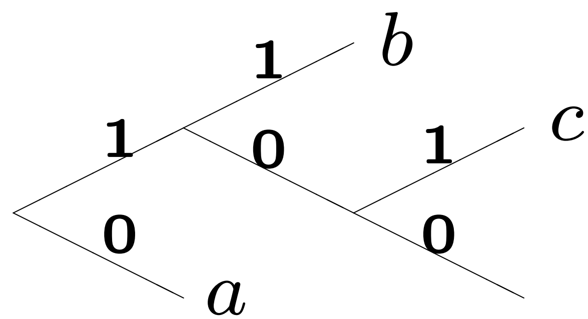
Then $accc \to 0111111{=}01^6$; $\quad bccc \to 01111111{=}01^7$.

This can be shown to be uniquely decodable.

PREFIX-FREE CODES

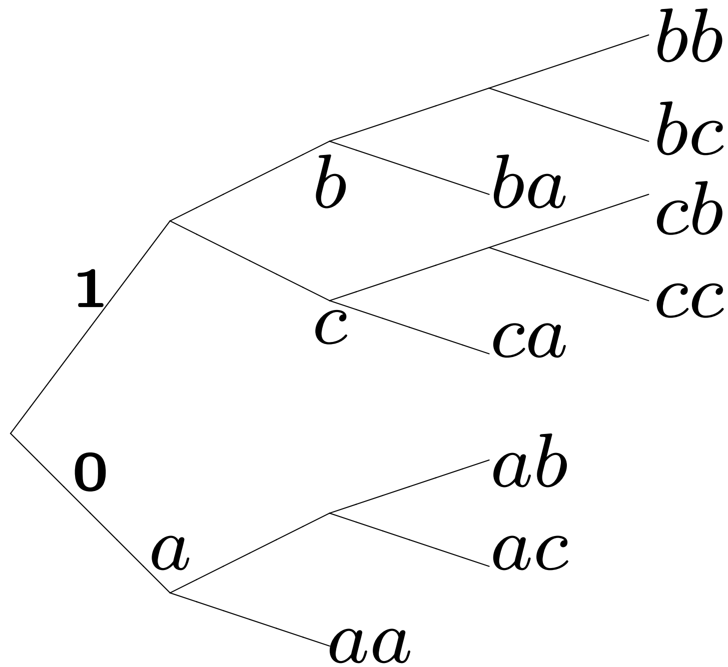
A code is <u>prefix-free</u> if no codeword is a prefix of any other codeword. A prefix of a string $y_1, \ldots, y_k$ is $y_1, \ldots, y_i$ for any $i \leq k$.

A prefix-free code can be represented by a binary tree which grows from left to right; leaves represent codewords.



$a \rightarrow 0$
$b \rightarrow 11$
$c \rightarrow 101$

Every codeword is at a leaf, but not all leaves are codewords. Empty leaves can be shortened. A <u>full code tree</u> has no empty leaves.



Prefix-free codes are uniquely decodable:

Construct a tree for a concatenation of code-words.

To decode, start at the left, and parse whenever a leaf in the tree is reached.

THE KRAFT INEQUALITY

The Kraft inequality is a test on the existence of prefix-free codes with a given set of codeword lengths $\{l(x), x \in \mathcal{X}\}$.

Theorem (Kraft): Every prefix-free code for an alphabet $\mathcal{X}$ with codeword lengths $\{l(x), x \in \mathcal{X}\}$ satisfies

$$\sum_{x \in \mathcal{X}} 2^{-l(x)} \leq 1 \tag{1}$$

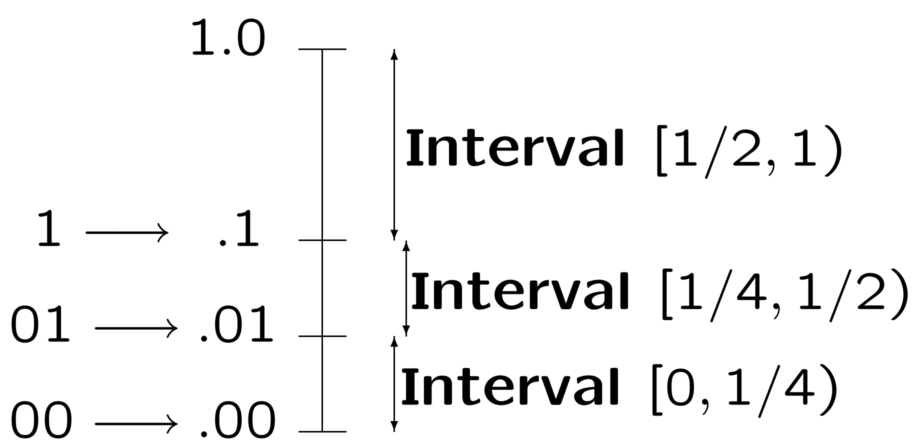
Conversely, if (1), then a prefix-free code with lengths $\{l(x)\}$ exists.

Moreover, a prefix-free code is full iff (1) is satisfied with equality.

We prove this by associating codewords with base 2 expansions i.e., 'decimals' in base 2.

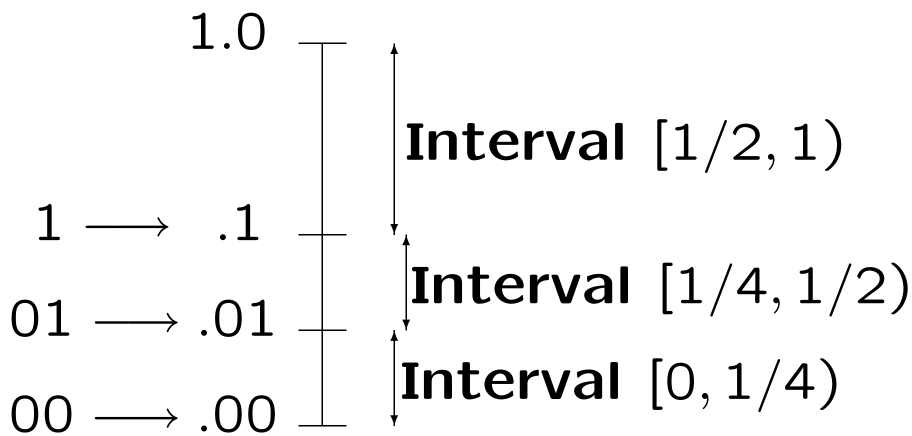
Represent binary codeword $y_1, y_2, \ldots, y_m$ **as**

$$.y_1 y_2 \cdots y_m = y_1/2 + y_2/4 + \cdots + y_m 2^{-m}$$

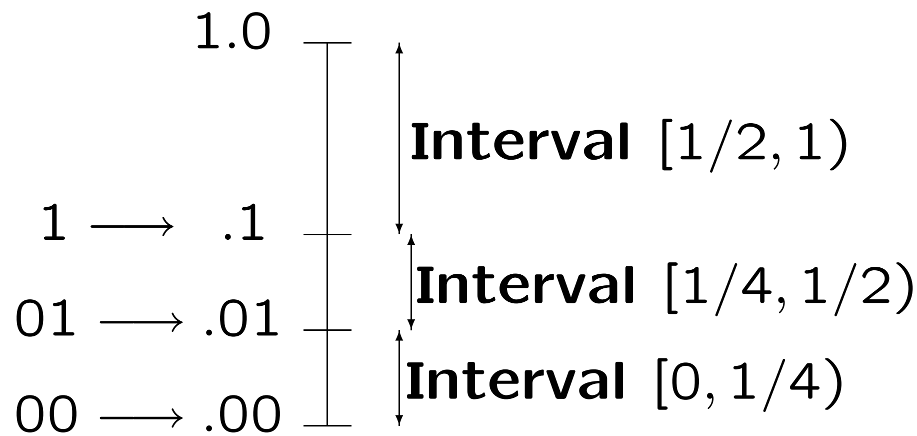


Represent binary codeword $y_1, y_2, \ldots, y_m$ as

$$.y_1 y_2 \cdots y_m = y_1/2 + y_2/4 + \cdots + y_m 2^{-m}$$

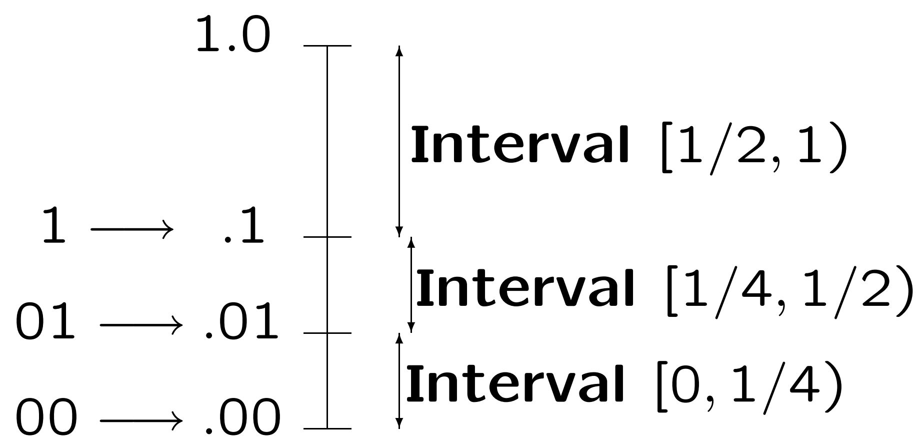


$\mathcal{C}(a_j)$ is a prefix of $\mathcal{C}(a_i)$ if and only if the expansion of $\mathcal{C}(a_j)$ contains the expansion of $\mathcal{C}(a_i)$ in its "approximation interval."



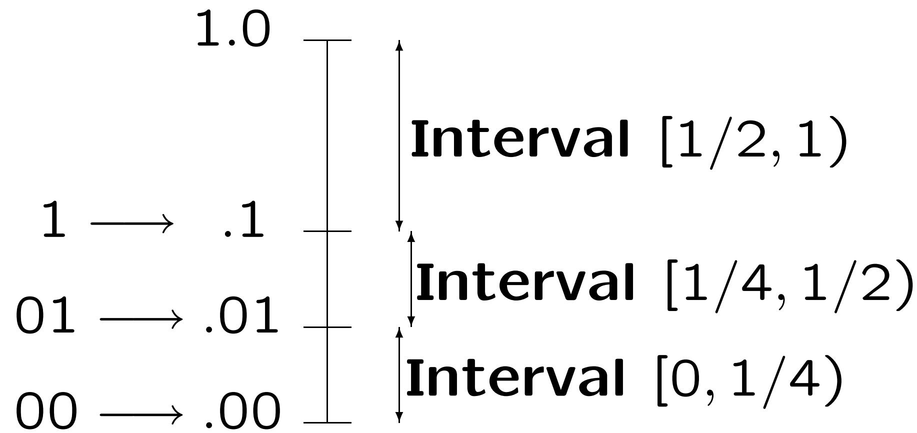
$\mathcal{C}(a_j)$ **is a prefix of** $\mathcal{C}(a_i)$ **if and only if the expansion of** $\mathcal{C}(a_j)$ **contains the expansion of** $\mathcal{C}(a_i)$ **in its "approximation interval."**

Thus a code is a prefix code iff the base 2 approximation intervals are disjoint.



A code is a prefix code iff the base 2 approximation intervals are disjoint.

But the sum of disjoint approximation intervals is at most 1.



The sum of disjoint approximation intervals is at most 1.

Code is full iff approximation intervals fill up [0, 1)

DISCRETE MEMORYLESS SOURCES

- **The source output is an unending sequence, $X_1, X_2, X_3, \ldots$, of randomly selected letters from a finite set $\mathcal{X}$, called the source alphabet.**

- **Each source output $X_1, X_2, \ldots$ is selected from $\mathcal{X}$ using a common probability measure.**

- **Each source output X_k is statistically independent of the other source outputs $X_1, \ldots$, $X_{k-1}, X_{k+1}, \ldots$.**

Probability Structure for Discrete Sources

English text: e, i, and o are far more probable than q, x, and z.

Successive letters are dependent; (th and qu).

Some letter strings are words, others are not.

Long term grammatical constraints.

The discrete memoryless source is a toy model that can be easily generalized after understanding it.

PREFIX-FREE CODES FOR DMS

Let $l(x)$ be the length of the codeword for letter $x \in \mathcal{X}$.

Then $L(X)$ is a random variable (rv) where $L(X) = l(x)$ for $X = x$.

Thus $L(X) = l(x)$ with probability $p_X(x)$.

$$E(L) = \overline{L} = \sum_x p_X(x) l(x)$$

Thus $\overline{L}$ is the number of encoder output bits per source symbol.

OBJECTIVE: choose integers $\{l(x)\}$ subject to Kraft to minimize $\overline{L}$.

Let $\mathcal{X} = \{1, 2, \ldots, M\}$ with pmf $p_1, \ldots, p_M$. Denote the unknown lengths by $l_1, \ldots, l_M$.

$$\overline{L}_{min} = \min_{l_1,\ldots,l_M : \sum 2^{-l_i} \le 1} \left\{ \sum_{i=1}^{M} p_i l_i \right\}$$

Forget about the lengths being integer for now. Minimize Lagrangian: $\sum_i (p_i l_i + \lambda 2^{-l_i})$.

$$\frac{\partial \sum_i (p_i l_i + \lambda 2^{-l_i})}{\partial l_i} = p_i - \lambda (\ln 2) 2^{-l_i} = 0$$

Choose λ so that the optimizing $\{l_i\}$ satisfy $\sum_i 2^{-l_i} = 1$. Then any other choice of $\{l_i\}$ satisfying constraint will have a larger $\bar{L}$.

$$\frac{\partial \sum_i (p_i l_i + \lambda 2^{-l_i})}{\partial l_i} = p_i - \lambda (\ln 2) 2^{-l_i} = 0$$

If we choose $\lambda = 1/\ln 2$, then

$$p_i = 2^{-l_i}$$

$$l_i = -\log p_i$$

$$\overline{L}_{min}(\text{non}-\text{int.}) = \sum_i -p_i \log p_i = \mathbf{H}(X)$$

$\mathbf{H}(X)$ is called the entropy of the rv X. We will see that it is the minimum number of binary digits per symbol needed to represent the source.

For now, it is a lower bound for prefix-free codes.

Theorem: Entropy bounds

Let $\overline{L}_{min}$ **be the minimum expected codeword length over all prefix-free codes for** X**. Then**

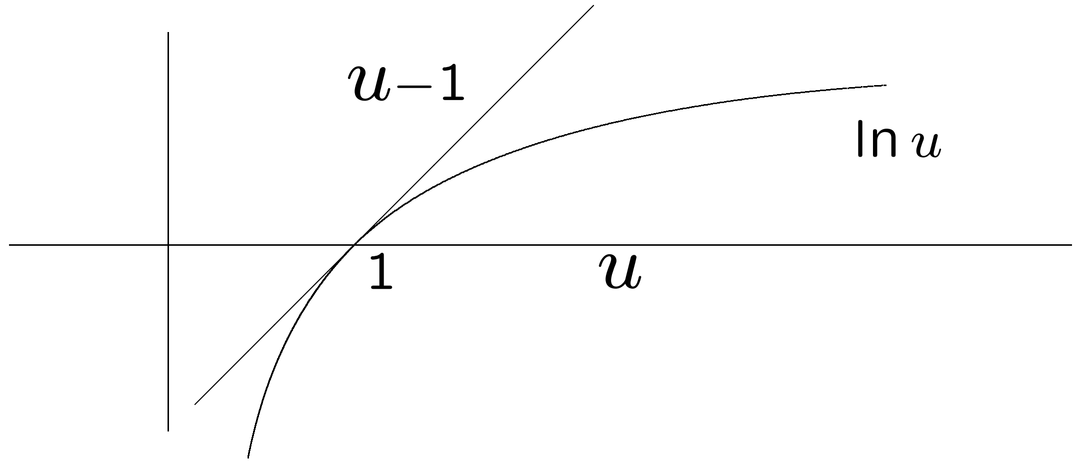
$$H(X) \le \overline{L}_{min} < H(X) + 1$$

$\overline{L}_{min} = H(X)$ **iff each** p_i **is integer power of 2.**

Proof of $H(X) \le \overline{L}$ **for prefix-free codes:**

Let $l_1, \ldots, l_M$ **be codeword lengths.**

$$\begin{aligned} H(X) - \overline{L} &= \sum_i p_i \log \frac{1}{p_i} - \sum_i p_i l_i \\ &= \sum_i p_i \log \frac{2^{-l_i}}{p_i}, \end{aligned}$$



The inequality $\ln u \le u-1$ **or** $\log u \le (\log e)(u-1)$**.**

This inequality is strict except at $u = 1$**.**

$$\begin{aligned}\mathbf{H}(X) - \overline{L} \;&=\; \textstyle\sum_i p_i \log \dfrac{2^{-l_i}}{p_i} \le \sum_i p_i \left[\dfrac{2^{-l_i}}{p_i} - 1\right] \log e \\ &=\; \textstyle\sum_i \left[2^{-l_i} - p_i\right] \log e \;\le\; 0\end{aligned}$$

Equality occurs iff $p_i = 2^{-l_i}$ **for each** i**.**

Theorem: Entropy bound for prefix-free codes:

$$\mathbf{H}(X) \le \overline{L}_{min} < \mathbf{H}(X) + 1$$

$\overline{L}_{\text{min}} = \mathbf{H}(X)$ **iff each** p_i **is integer power of 2.**

Proof that $\overline{L}_{\text{min}} < \mathbf{H}(X) + 1$**:**

Choose $l_i = \lceil -\log(p_i) \rceil$. **Then**

$$l_i < -\log(p_i) + 1 \quad \textbf{so} \quad \overline{L}_{\text{min}} \le \overline{L} < \mathbf{H}(X) + 1$$

$$l_i \ge \log(p_i) \quad \textbf{so} \quad \sum_i 2^{-l_i} \le \sum_i p_i = 1$$