Ch. 2 Math Preliminaries for Lossless Compression

Section 2.4 Coding
Some General Considerations

Definition: An *Instantaneous Code* maps each symbol into a codeword

Notation: \( a_i \rightarrow \phi (a_i) \)

For Ex. 1: \( \phi (a_3) = 00 \)

Ex. 1:

\[
\begin{align*}
    a_1 & \rightarrow 0 \\
    a_2 & \rightarrow 1 \\
    a_3 & \rightarrow 00 \\
    a_4 & \rightarrow 11
\end{align*}
\]

Ex. 2:

\[
\begin{align*}
    a_1 & \rightarrow 0 \\
    a_2 & \rightarrow 10 \\
    a_3 & \rightarrow 110 \\
    a_4 & \rightarrow 111
\end{align*}
\]

This code has a tree structure:
What characteristics must a code $\phi$ have?

**Unambiguous (UA):** For $a_i \neq a_j$, $\phi(a_i) \neq \phi(a_j)$

The codes in Ex. 1 and Ex.2 each are UA

**Is UA enough??** No! Consider Ex. 1 coding two different source sequences:

- $a_1 \ a_2 \ a_1 \ a_1 \ a_2 \ a_2$
- $a_1 \ a_2 \ a_3 \ a_4$
- $a_1 \ a_2 \ a_1 \ a_1 \ a_2 \ a_2$
- $a_1 \ a_2 \ a_3 \ a_4$

They each get coded to the bit stream: 0 1 0 0 1 1

Can’t uniquely decode this bit sequence!!

So… UA guarantees that can decode each symbol by itself but not necessarily a stream of coded symbols!!
Define mapping of sequences under code \( \phi \)
\[
\Phi(a_{i_1} a_{i_2} a_{i_3} a_{i_4} \ldots a_{i_N}) = \phi(a_{i_1}) \phi(a_{i_2}) \phi(a_{i_3}) \phi(a_{i_4}) \ldots \phi(a_{i_N})
\]

Concatenation of code words

Don’t want two sequences of symbols to map to the same bit stream:

Don’t want two sequences of symbols to map to the same bit stream:

Leads to need for…

**Uniquely Decodable (UD):** Let \( S_i \) & \( S_j \) be two sequences from the same source (not necessarily of the same length).

Then code \( \phi \) is UD if the only way that \( \Phi(S_i) = \Phi(S_j) \) is for \( S_i \neq S_j \).
Does UD → UA???

YES!

Then UD is enough???

YES!

But in practice it is helpful to restrict to a subset of UD codes called “Prefix Codes”.

UA Codes

UD Codes
**Prefix Code**: A UD code in which no codeword may be the prefix of another codeword.

Ex. 2 above is a prefix code:

<table>
<thead>
<tr>
<th></th>
<th>( a_1 )</th>
<th>( a_2 )</th>
<th>( a_3 )</th>
<th>( a_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_1 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( a_3 )</td>
<td>110</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( a_4 )</td>
<td>111</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

This code is not prefix

Ex. 3:

<table>
<thead>
<tr>
<th></th>
<th>( a_1 )</th>
<th>( a_2 )</th>
<th>( a_3 )</th>
<th>( a_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_1 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>01</td>
<td>01</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>( a_3 )</td>
<td>011</td>
<td>011</td>
<td>011</td>
<td>011</td>
</tr>
<tr>
<td>( a_4 )</td>
<td>0111</td>
<td>0111</td>
<td>0111</td>
<td>0111</td>
</tr>
</tbody>
</table>

Does Prefix \( \Rightarrow \) UD??? YES!

Do we lose anything by restricting to prefix codes?

No... as we’ll see later!
How do we compare various UD codes???
(i.e., What is our measure of performance?)

**Average Code Length**: Info theory says to use average code length per symbol… For a source with symbols $a_1, a_2, \ldots a_N$ and a code $\phi$ the average code length is defined by

$$\bar{l}(\phi) = \sum_{i=1}^{N} P(a_i) n(\phi(a_i))$$

Optimum Code: The UD code with the smallest average code length
**Example:** For \( P(a_1) = \frac{1}{2} \) \( P(a_2) = \frac{1}{4} \) \( P(a_3) = P(a_3) = \frac{1}{8} \)

This source has an entropy of 1.75 bits

Here are three possible codes and their average lengths:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>UA Non-UD Code (Ex. 1)</th>
<th>Prefix Code (Ex. 2)</th>
<th>UD Non-Prefix (Ex. 3)</th>
<th>Info of symbol (-\log_2[P(a_i)])</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_1 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>1</td>
<td>10</td>
<td>01</td>
<td>2</td>
</tr>
<tr>
<td>( a_3 )</td>
<td>00</td>
<td>110</td>
<td>011</td>
<td>3</td>
</tr>
<tr>
<td>( a_4 )</td>
<td>11</td>
<td>111</td>
<td>0111</td>
<td>3</td>
</tr>
</tbody>
</table>

**Avg. Length:** 1.25 bits 1.75 bits 1.875 bits \( H(S) = 1.75 \) bits

- Length better than \( H(S) \)!!
- Length equals \( H(S) \)
- Length larger than \( H(S) \)

**BUT**… not usable because it is Non-UD

Prefix Code gives smallest usable code!!
Info Theory Says: Optimum Code is always a prefix code!!

The proof of this uses the Kraft-McMillan Inequality which we’ll discuss next.

How do we find the optimum prefix code?
(Note: not just any prefix code will be optimum!)
We’ll discuss this later….
2.4.3 Kraft-McMillan Inequality

This result tells us that an optimal code can always be chosen to be a prefix code!!! The Theorem has 2 parts….

**Theorem Part #1:** Let $C$ be a code having $N$ codewords… with codeword lengths of $l_1, l_2, l_3, \ldots, l_N$

If $C$ is uniquely decodable, then

$$\sum_{i=1}^{N} 2^{-l_i} \leq 1$$

For notation: $K(C) \triangleq \sum_{i=1}^{N} 2^{-l_i}$

**Proof:** Here is the main idea used in the proof…

If $K(C) > 1$, then $[K(C)]^n$ grows exponentially w.r.t. $n$

So… if we can show that $[K(C)]^n$ grows, say, no more than linearly we have our proof. Thus we need to show that

$$[K(C)]^n \leq \alpha n + \beta$$

Some constants
For arbitrary integer \( n \):

\[
[K(C)]^n = \left[ \sum_{i=1}^{N} 2^{-l_i} \right]^n = \left[ \sum_{i_1=1}^{N} 2^{-l_{i_1}} \right] \left[ \sum_{i_2=1}^{N} 2^{-l_{i_2}} \right] \cdots \left[ \sum_{i_n=1}^{N} 2^{-l_{i_n}} \right]
\]

\[
= \sum_{i_1=1}^{N} \sum_{i_2=1}^{N} \cdots \sum_{i_n=1}^{N} 2^{-(l_{i_1}+l_{i_2}+\cdots+l_{i_n})}
\]

Note that this exponent is nothing more than the length of a sequence of selected codewords of code \( C \)… Let this be \( L(i_1, i_2, i_3, \ldots, i_n) \) and we can re-write (★) as

\[
[K(C)]^n = \sum_{i_1=1}^{N} \sum_{i_2=1}^{N} \cdots \sum_{i_n=1}^{N} 2^{-L(i_1,i_2,\ldots,i_n)} = 2^{-L(1,1,\ldots,1)} + 2^{-L(1,1,\ldots,2)} + \cdots + 2^{-L(N,N,\ldots,N)}
\]

The smallest \( L(i_1, i_2, i_3, \ldots, i_n) \) can be is \( n \) (when each codeword in the sequence is 1 bit long)

The longest \( L(i_1, i_2, i_3, \ldots, i_n) \) can be is \( nl \) where \( l \) is the longest codeword in \( C \).

So then:

\[
[K(C)]^n = 2^{-L(1,1,\ldots,1)} + 2^{-L(1,1,\ldots,2)} + \cdots + 2^{-L(N,N,\ldots,N)}
\]

\[
= A_n 2^{-n} + A_{n+1} 2^{-(n+1)} + \cdots + A_{nl} 2^{-(nl)}
\]

(★ ★ ★)  \[
[K(C)]^n = \sum_{k=n}^{nl} A_k 2^{-k}
\]

\( A_k = \) # times \( L(i_1, i_2, i_3, \ldots, i_n) = n \)
Remember that we are trying to establish this bound: \[ [K(C)]^n \leq \alpha n + \beta \]

we don’t need the \( A_k \) values exactly… just need a good upper bound on them!

**First**: The \# of \( k \)-bit binary sequences = \( 2^k \)

**Second**: If our code is uniquely decodable, then each of these can represent one and only one sequence of codewords whose total length = \( k \) bits

\[ A_k \leq 2^k \]

We can now use this bound in ( ★ ★ ) to get a bound on \([K(C)]^n\):

\[ [K(C)]^n = \sum_{k=n}^{nl} A_k 2^{-k} \leq \sum_{k=n}^{nl} 2^k 2^{-k} = nl - n + 1 \]

Thus… \([K(C)]^n\) grows slower than exponentially

Hence… \( K(C) \leq 1 \quad <\text{End of Proof}> \)
**Part #1 says:** If code with lengths \( \{l_1, l_2, \ldots, l_N\} \) is uniquely decodable, then the lengths satisfy the inequality

**Part #2 says:** Given lengths \( \{l_1, l_2, \ldots, l_N\} \) that satisfy the inequality, then we can always find a prefix code w/ these lengths

---

**Theorem Part #2:** Given integers \( \{l_1, l_2, \ldots, l_N\} \) such that \( \sum_{i=1}^{N} 2^{-l_i} \leq 1 \)

We can always find a prefix code with lengths \( \{l_1, l_2, \ldots, l_N\} \)

**Proof:** This is a “Proof by Construction”: we will show how to construct the desired prefix code. “WLOG”…. Assume that \( l_1 \leq l_2 \leq \ldots \leq l_N \)

Define the numbers \( w_1, w_2, \ldots, w_N \) using

\[
\begin{align*}
    w_1 &= 0 \\
    w_j &= \sum_{i=1}^{j-1} 2^{l_j-l_i}, \quad j > 1
\end{align*}
\]

Think of this in terms of a binary representation (see next slide for an example)
Example of Creating the $w_j$

\[ l_1 = 1 \quad l_2 = 3 \quad l_3 = 3 \quad l_4 = 5 \quad l_5 = 5 \quad \sum_{i=1}^{5} 2^{-l_i} = 0.8125 < 1 \]

$w_1 = 0$

$w_2 = \sum_{i=1}^{1} 2^{l_2-l_i} = 2^{3-1} = 4 = 100_2$

$w_3 = \sum_{i=1}^{2} 2^{l_3-l_i} = 2^{3-1} + 2^{3-3} = 5 = 101_2$

$w_4 = \sum_{i=1}^{3} 2^{l_4-l_i} = 2^{5-1} + 2^{5-3} + 2^{5-3} = 24 = 11000_2$

$w_5 = \sum_{i=1}^{4} 2^{l_5-l_i} = 2^{5-1} + 2^{5-3} + 2^{5-3} + 2^{5-5} = 25 = 11001_2$

For $j > 1$, the binary representation of $w_j$ uses $\lceil \log_2 w_j \rceil$ bits

Easy to show (see textbook) that: \# bits in $w_j$ \leq l_j \Rightarrow \lceil \log_2 w_j \rceil \leq l_j, \: for \: j \geq 1

This is where we use that $\sum_{i=1}^{N} 2^{-l_i} \leq 1$
Now use the binary reps of the $w_j$ to construct the prefix codewords having lengths $\{l_1, l_2, \ldots, l_N\}$

$$\left\lfloor \log_2 w_j \right\rfloor = l_j \quad \text{then set } j^{\text{th}} \text{ codeword} = \text{binary } w_j$$

If

$$\left\lfloor \log_2 w_j \right\rfloor < l_j \quad \text{then set } j^{\text{th}} \text{ codeword} = \text{[binary } w_j \ 0 \ldots 0]$$

So now we’ve constructed a code with the desired lengths… Is it a prefix code???

Show it is by using contradiction… Assume that it is NOT a prefix code and show that it leads to something that contradicts a known condition…
Suppose that the constructed code is not prefix... thus, for some \( j < k \) the codeword \( C_j \) is a prefix of codeword \( C_k \...\)

\[
(l_j \text{ MSBs of } w_k) = w_j
\]

\[
\text{Right-Shift} \& \text{Chop } w_k = w_j
\]

\[
\left\lfloor \frac{w_k}{2^{l_k - l_j}} \right\rfloor = w_j \quad (\star)
\]

But... by “design”: \( w_k = \sum_{i=1}^{k-1} 2^{l_k - l_i} \)

So see if (\( \star \)) contradicts this required condition:

Put this \( w_k \) into (\( \star \)) and show that something goes wrong
\[
\frac{w_k}{2^{l_k-l_j}} = \sum_{i=1}^{k-1} 2^{l_j-l_i} \\
= \sum_{i=1}^{j-1} 2^{l_j-l_i} + \sum_{i=j}^{k-1} 2^{l_j-l_i} \\
= w_j + 2^0 + \sum_{i=j+1}^{k-1} 2^{l_j-l_i} \geq w_j + 1
\]

Thus, \( \frac{w_k}{2^{l_k-l_j}} \geq w_j + 1 > w_j \)

\[ \left\lfloor \frac{w_k}{2^{l_k-l_j}} \right\rfloor \geq w_j + 1 \]

…which contradicts \((\star)\)

So code is prefix!

<End of Proof>
Meaning of Kraft-McMillan Theorem

**Question**: So what do these two parts of the theorem tell us???

**Answer**:

- We are looking for the optimal UD code.
- Once we find it we know its codeword lengths satisfy the K-M inequality
  - Part #1 of the theorem tells us that!!!
- Once we have such lengths (that satisfy the K-M ineq.) we can *construct* a prefix code having those *optimal* lengths…
  - This is guaranteed by Part #2 of the theorem
  - This gives us a prefix code that is optimal!!!

So… everytime we find the optimal code, if it isn’t already prefix we can replace it with a prefix code that is just as optimal!

Can focus on finding optimal prefix codes… w/o worrying that we could find a better code that is not prefix!