MODULE 2

SOURCE CODING

Structure:

1. Introduction- Encoding of the source output
2. Shannon's encoding algorithm
3. Kraft McMillan Inequality property – KMI
4. Source coding theorem
5. Outcome

Objective:

1. To study the different encoding algorithms
2. To understand the concept of channels.

2.1 Kraft McMillan Inequality Property

Encoding' or 'Enciphering' is a procedure for asso ciating words constructed from a finite alphabet of a language with given words of another language in a one-to- one manner.
Let the source be characterized by the set of symbols

S=
$\{s_1, s_2 .. s_q\}$

We shall call ' S' as the " Source alphabet". Consider another set, X, comprising of ' r' symbols.

$X=\{x_1, x_2 \ldots x_r\}$

We shall call ' X' as the " code alphabet". We define " coding" as the mapping of all possible sequences of symbols of S into sequences of symbol of X. In other words " coding means representing each and every symbol of S by a sequence of symbols of X such that there shall be a one- to-one relationship" Any finite sequence of symbols from an alphabet will be called a " Word". Thus any sequence from the alphabet ' X' forms a " code word". The total number of symbols contained in the ' word' will be called " word length". For example the sequences { x_1 ; $x_1x_3x_4$; $x_3x_5x_7x_9$; $x_1x_1x_2x_2x_2$} form code words. Their word lengths are respectively1; 3; 4; and 5.The sequences {100001001100011000} and {1100111100001111000111000} are binary code words with word lengths 18 and 25 respectively.

Basic properties of codes:

The definition of codes given above is very broad and includes many undesirable properties. In order that the definition is useful in code synthesis, we require the codes to satisfy certain properties. We shall intentionally take trivial examples in order to get a better understanding of the desired properties.

1. **Block codes**:

A block code is one in which a particular message of the source is always encoded into the same "**fixed sequence**" of the code symbol. Although, in general, block m eans ' a group having identical property' we shall use the word here to mean a ' fixed sequence' only. Accordingly, the code can be a '**fixed length code**' or a " **variable length code**" and we shall be concentrating on the latter type in this chapter. To be more specific as to what we mean by a block code, consider a communication system with one transmitter and one receiver. Information is transmitted using certain set of code words. If the transmitter wants to change the code set, first thing to be done is to inform the receiver. Otherwise the receiver will never be able to understand what is being transmitted. Thus, until and unless the receiver is informed about the changes made you are not permitted to change the code set. In this sense the code words we are seeking shall be always **finite sequences** of the code alphabet-they are **fixed sequence codes**.

Example 2.1: Source alphabet is **S = {s1, s2, s3, s4}**, Code alphabet is **X = {0, 1}** and The Code words are: **C = {0, 11, 10, 11}**

2. **Non – singularcodes:**

A block code is said to be nonsingular if all the words of the code set X1, are "distinct". The Codes given in Example 6.1 do not satisfy this property as the codes for s2 and s4 are not different. We cannot distinguish the code words. If the codes are not distinguishable on a simple inspection we say the code set is " **singular in the small**". We modify the code as below.

Example 2.2: S = {s1, s2, s3, s4}, **X = {0, 1}**; Codes, **C = {0, 11, 10, 01}**

However, the codes given in Example 6.2 although appear to be non-singular, upon transmission would pose problems in decoding. For, if the transmitted sequence is **0011**, it might be interpreted as **s1 s1 s4** or **s2 s4**. Thus there is an ambiguity about the code. No doubt, the code is non-singular in the small, but becomes "Singular **in the large**".

3. **Uniquely decodable codes:**

A non-singular code is uniquely decipherable, if every word immersed in a sequence of words can be uniquely identified. The **n**$^{\mathbf{th}}$ extension of a code, that maps each message into the code words **C**, is defined as a code which maps the sequence of messages into a sequence of code words. This is also a block code, as illustrated in the following example.

Example 2.3: Second extension of the code set given in Example 6.2.

S^2**={s1s1,s1s2,s1s3,s1s4; s2s1,s2s2,s2s3,s2s4,s3s1,s3s2,s3s3,s3s4,s4s1,s4s2,s4s3,s4s4}**

Source Symbols	Codes	Source Symbols	Codes	Source Symbols	Codes	Source Symbols	Codes
s1s1	0 0	s2s1	1 1 0	s3s1	1 0 0	s4s1	0 1 0
s1s2	0 1 1	s2s2	1 1 1 1	s3s2	1 0 1 1	s4s2	0 1 1 1
s1s3	0 1 0	s2s3	1 1 1 0	s3s3	1 0 1 0	s4s3	0 1 1 0
s1s4	0 0 1	s2s4	1 1 0 1	s3s4	1 0 0 1	s4s4	0 1 0 1

Notice that, in the above example, the codes for the source sequences, **s1s3** and **s4s**1 are not distinct and hence the code is "Singular **in the Large**". Since such singularity properties introduce ambiguity in the decoding stage, we therefore require, in general, for unique decidability of our codes that " ***The n***th ***extension of the code be non-singular for every finite n."***

4. Instantaneous Codes:

A uniquely decodable code is said to be " **instantaneous**" if the end of any code word is recognizable with out the need of inspection of succeeding code symbols. That is **there is no time lag in the process of decoding**. To understand the concept, consider the following codes:

Example 2.4:

Source symbols	Code A	Code B	Code C
s 1	0 0	0	0
s 2	0 1	1 0	0 1
s 3	1 0	1 1 0	0 1 1
s 4	1 1	1 1 1 0	0 1 1 1

Code A undoubtedly is the simplest possible uniquely decipherable code. It is non- singular and all the code words have same length. The decoding can be done as soon as we receive two code symbols without any need to receive succeeding code symbols.

Code B is also uniquely decodable with a special feature that the **0**`s indicate the termination of a code word. It is called the "**comma code**". When scanning a sequence of code symbols, we may use the comma to determine the end of a code word and the beginning of the other. Accordingly, notice that the codes can be decoded as and when they are received and there is, once again, no time lag in the decoding process.

Whereas, although **Code C** is a non- singular and uniquely decodable code it cannot be decoded word by word as it is received. For example, if we receive '01', we cannot decode it as ' **s2**' until we receive the next code symbol. If the next code symbol is '0', indeed the previous word corresponds to s2, while if it is a '1' it may be the symbol **s3;** which can be concluded so if only if we receive a '**0**'in the fourth place. Thus, there is a definite 'time **lag**' before a word can be decoded. Such a 'time waste' is not there if we use either **Code A** or **Code B**. Further, what we are envisaging is the property by which a sequence of code words is uniquely and instantaneously decodable even if there is no spacing between successive words. The common English words do not posses this property. For example the words " **FOUND**", " **AT**" and " **ION**" when transmitted without spacing yield, at the receiver, an altogether new word" **FOUNDATION**"! A sufficient condition for such property is that

"No encoded word can be obtained from each other by the addition of more letters "This property is called " **prefix property**".

Let **Xk = xk1xk2….x km**, be a code word of some source symbol **sk**. Then the sequences of code symbols, (**xk1xk2….x k j**), **j ≤ m**, are called "prefixes" of the code word. Notice that a code word of length ' **m**' will have ' **m**' prefixes. For example, the code word **0111** has four prefixes, viz; **0**, **01**, **011** and **0111**.The complete code word is also regarded as a prefix.

Prefix property:"A necessary and sufficient condition for a code to be 'instantaneous' is that no complete code word be a prefix of some other code word".

The *sufficiency* condition follows immediately from the definition of the word "Instantaneous". If no word is a prefix of some other word, we can decode any received sequence of code symbols comprising of code words in a direct manner. We scan the received sequence until we come to subsequence which corresponds to a complete code word. Since by assumption it is not a prefix of any other code word, the decoding is unique and there will be no time wasted in the process of decoding. The "*necessary*" condition can be verified by assuming the contrary and deriving its "contradiction". That is, assume that there exists some word of our code, say ***xi***, which is a prefix of some other code word ***xj***. If we scan a received sequence and arrive at a subsequence that corresponds to ***xi***, this subsequences may be a complete code word or it may just be the first part of code word ***xj.*** We cannot possibly tell which of these alternatives is true until we examine some more code symbols of the sequence. Accordingly, there is definite time wasted before a decision can be made and hence the code is not instantaneous.

5. **Optimal codes:**

An instantaneous code is said to be optimal if it has "***minimum average word length***", for a source with a given probability assignment for the source symbols. In such codes, source symbols with higher probabilities of occurrence are made to correspond to shorter code words. Suppose that a Source symbol s_i has a probability of occurrence P_i and has a code word of length l_i assigned to it, while a source symbol s_j with probability P_j has a code word of length l_j. If $P_i > P_j$ then let $l_i < l_j$. For the two code words considered, it then follows, that the average length L_1 is given by

$$L_1 = P_i l_i + P_j l_j$$

Now, suppose we interchange the code words so that the code word of length l_j corresponds to s_i and that of length l_i corresponds to s_j. Then, the average length becomes

$$L_2 = P_i l_j + P_j l_i$$ It then follows,

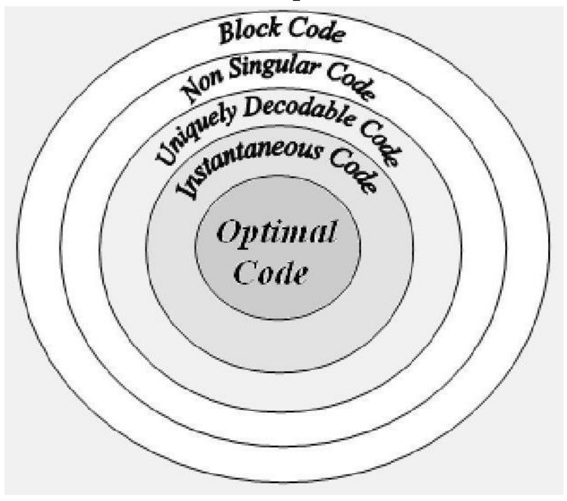
$$L_2 - L_1 = P_i(l_j - l_i) + P_j(l_i - l_j)$$

$$= (P_i - P_j)(l_j - l_i)$$

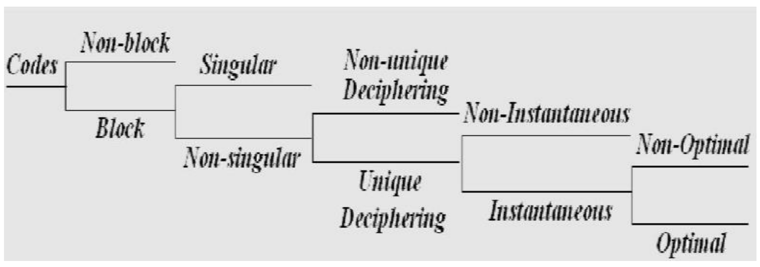
Since by assumption $P_i > P_j$ and $l_i < l_j$, it is clear that ($L_2 - L_1$) is positive. That is assignment of source symbols and code word length corresponding to the average length L_1 is shorter, which is the requirement for optimal codes.

A code that satisfies all the five properties is called an "**irreducible code**".

All the above properties can be arranged as shown in Fig 2.1 which serves as a quick reference of the basic requirements of a code. Fig 2.2 gives the requirements in the form of a 'Tree' diagram. Notice that both sketches illustrate one and the same concept.



2.1 Codes Sub grouping



2.2 Code Tree diagram

2.2 Construction of Instantaneous Codes:

Consider encoding of a 5 symbol source into Binary instantaneous codes i.e.

$S = \{s_1, s_2, s_3, s_4, s_5\}$; $X = \{0, 1\}$

We may start by assigning '*0*' to s_1
i.e. $s_1 \rightarrow 0$

If this is the case, to have prefix property, all other source symbols must correspond to code words beginning with ***1***. If we let s_2 correspond to ' ***1***', we would be left with no code symbol for encoding the remaining three source symbols. We might have

$s_2 \rightarrow 10$

This in turn would require the remaining code words to start with ***11***. If

$s_3 \rightarrow 110$;

Then the only 3 bit prefix unused is ***111*** and we might set

$s_4 \rightarrow 1110$

$s_5 \rightarrow 1111$

In the above code, notice that the starting of the code by letting s_1 correspond '0' has cut down the number of possible code words. Once we have taken this step, we are restricted to code words starting with '1'. Hence, we might expect to have more freedom if we select a ***2-binit*** code word for s_1. We now have four prefixes possible ***00, 01, 10*** and ***11***; the first three can be directly assigned to s_1, s_2 and s_3. With the last one we construct code words of length ***3***. Thus the possible instantaneous code is

$s_1 \rightarrow 00$

$s_2 \rightarrow 01$

$s_3 \rightarrow 10$

$s_4 \rightarrow 110$

$s_5 \rightarrow 111$

Thus, observe that shorter we make the first few code words, the longer we will have to make the later code words.

One may wish to construct an instantaneous code by pre-specifying the word lengths. The necessary and sufficient conditions for the existence of such a code are provided by the **'Kraft Inequality'.**

Kraft Inequality:

Given a source S = $\{s_1, s_2 \ldots s_q\}$.Let the word lengths of the codes corresponding to these symbols be $l_1, l_2 \ldots\ldots l_q$ and let the code alphabet be X = $\{x_1, x_2 \ldots x_r\}$. Then, an instantaneous code for the source exists iffy is called ***Kraft Inequality.***

$$\sum_{k=1}^{q} r^{-l_k} \leq 1$$

Example 2.5:

A six symbol source is encoded into Binary codes shown below. Which of these codes are instantaneous?

Source symbol	Code A	Code B	Code C	Code D	Code E
s1	0 0	0	0	0	(
s2	0 1	1 0 0 0	1 0	1 0 0 0	1 0
s3	1 0	1 1 0 0	1 1 0	1 1 1 0	1 1)
s4	1 1 0	1 1 1 0	1 1 1 0	1 1 1	1 1 1 0
s5	1 1 1 0	1 1 0 1	1 1 1 1 0	1 0 1 1	1 1 1 1 0
s6	1 1 1 1	1 1 1 1	1 1 1 1 1	1 1 0 0	1 1 1 1
$\sum_{k=1}^{6} 2^{-l_k}$	1	$\frac{13}{16} < 1$	1	$\frac{7}{8} < 1$	$1\frac{1}{32} > 1$

As a first test we apply the Kraft Inequality and the result is accordingly tabulated. ***Code E does not satisfy Kraft Inequality and it is not an instantaneous code***.

2.3 Source coding theorem:

Compact code: Huffman's Minimum Redundancy code:

The Huffman code was created by American, D. A. Huffman, in 1952. Huffman`s procedure is applicable for both Binary and Non- Binary encoding. It is clear that a code with minimum average length, L, would be more efficient and hence would have minimum redundancy associated with it. A compact code is one which achieves this objective. Thus for an optimum coding we require:

(1) Longer code word should correspond to message lowest probability.

(2) $l_k \geq l_{k-1}\ \forall\ k = 1,2,\ldots\ldots q^{-r+1}$

(3) $l_{p-r} = l_{q-r-1} = l_{q-r-2} = \ldots\ldots = l_q$

(4)The codes must satisfy the prefix property.

Huffman has suggested a simple method that guarantees an optimal code even is not satisfied. The procedure consists of step- by- step reduction of the original source followed by a code construction, starting with the final reduced source and working backwards to the original source. The procedure requires α steps, where

$$q = r + \alpha(r-1)$$

Notice that α is an integer and if Eq.(6.24) is not satisfied one has to add few dummy symbols with zero probability of occurrence and proceed with the procedure or the first step is performed by setting $r_1 = q-\alpha(r-1)$ while the remaining steps involve clubbing of the last r messages of the respective stages. The procedure is as follows:

List the source symbols in the decreasing order of probabilities

Check if $q = r + \alpha(r-1)$ is satisfied and find the integer ' α' . Otherwise add suitable number of dummy symbols of zero probability of occurrence to satisfy the equation. ***This step is not required if we are to determine binary codes.***

1. Repeat steps ***1*** and ***3*** respectively on the resulting set of symbols until in the final step exactly ***r***- symbols are left.
2. Assign codes freely to the last ***r***-composite symbols and work backwards to the original source to arrive at the optimum code
3. Alternatively, following the steps carefully a tree diagram can be constructed starting from the final step and codes read off directly.
4. Discard the codes of the dummy symbols.

Before we present an example, it is in order to discuss the steps involved. In the first step, after arranging the symbols in the decreasing order of probabilities; we club the last ***r***-symbols into a composite symbol, say σ_1 whose probability equals the sum of the last ***r***-probabilities. Now we are left with ***q-r+1*** symbols .In the second step, we again club the last ***r***-symbols and the second reduced source will now have $(q-r+1)-r+1= q-2r+2$ symbols .Continuing in this way we find the ***k***-th reduced source will have $q- kr + k = q - k(r - 1)$ symbols. Accordingly, if α -steps are required and the final reduced source should have exactly ***r***-symbols, then we must have $r = q - \alpha(r - 1)$ last $r_1=q-\alpha(r - 1)$ symbols while the second and subsequent reductions involve last ***r***-symbols only. However, if the reader has any confusion, he can add the dummy messages as indicated and continue with the procedure and the final result is no different at all.

Let us understand the meaning of " *working backwards*". Suppose α_k is the composite symbol obtained in the k^{th} step by clubbing the last ***r***-Symbols of the (***k***-1) ***th*** reduced source. ***Then whatever code is assigned to α_k will form the starting code sequence for the code words of its constituents in the (k-1) th reduction.***

Example 2.5: (***Binary Encoding***)

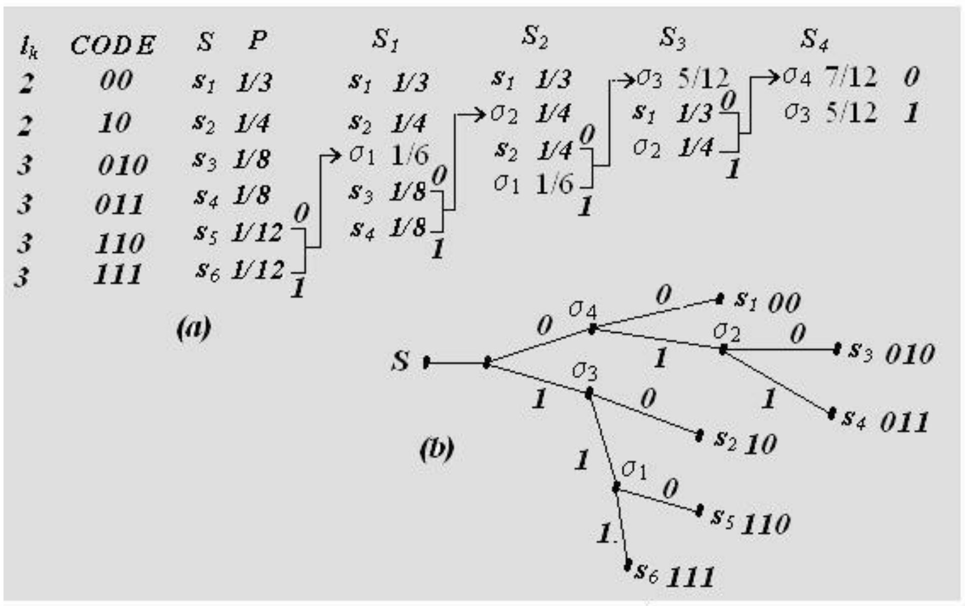
$S = \{s_1, s_2, s_3, s_4,$

$s_5, s_6\}, X = \{0, 1\};$

$P = \left\{\frac{1}{3}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}, \frac{1}{12}, \frac{1}{12}\right\}$

.Notice that the tree diagram can be easily constructed from the final step of the source reduction and decomposing the composite symbols towards the original symbols. Further, observe that the codes are originating from the same source and diverge out into different tree branches thus ensuring prefix property to the resultant code. Finally, notice that there is no restriction in the allocation of codes in each step and accordingly, the order of the assignment can be changed in any or all steps. Thus for the problem illustrated there can be as many as ***2. (2.2+2.2) = 16*** possible instantaneous code patterns. For example we can take the compliments of First column, Second column, or Third column and combinations there off as illustrated below.

Code	***I***	***II***	***III***
s_1.........*00*	*1 0*	*1 1*	*1 1*
s_2........ *1 0*	*0 0*	*0 1*	*0 1*
s_3....... *0 1 0*	*1 1 0*	*1 0 0*	*1 0 1*
s_4....... *0 1 1*	*1 1 1*	*1 0 1*	*1 0 0*
s_5 ...... *1 1 0*	*0 1 0 0 0 0*		*0 0 1*
s_6....... *1 1 1*	*0 1 1*	*0 0 1*	*0 0 0*



l_k CODE S P
2 00 s_1 1/3
2 10 s_2 1/4
3 010 s_3 1/8
3 011 s_4 1/8
3 110 s_5 1/12
3 111 s_6 1/12
S_1: s_1 1/3, s_2 1/4, σ_1 1/6, s_3 1/8, s_4 1/8
S_2: s_1 1/3, σ_2 1/4, s_2 1/4, σ_1 1/6
S_3: σ_3 5/12, s_1 1/3, σ_2 1/4
S_4: σ_4 7/12 0, σ_3 5/12 1
(a)
S
s_1 00
s_3 010
s_4 011
s_2 10
s_5 110
s_6 111
(b)

Fig 5.7 (a) Reduction Diagram (b) Tree Diagram For Binary Huffman Coding of Example 5.12

Code I is obtained by taking complement of the first column of the original code. ***Code II*** is obtained by taking complements of second column of ***Code I***. ***Code III*** is obtained by taking complements of third column of ***Code II***. However, notice that, l_k, the word length of the code word for s_k is a constant for all possible codes.

For the binary code generated, we have:

$$L=\sum_{k=1}^{6} p_k l_k = \frac{1}{3}\times 2+\frac{1}{4}\times 2+\frac{1}{8}\times 3+\frac{1}{8}\times 3+\frac{1}{12}\times 3+\frac{1}{12}\times 3=\frac{29}{12}\text{ binits/sym}=2.4167\text{ binits/sym}$$

$$H(S)=\frac{1}{3}\log 3+\frac{1}{4}\log 4+2\times\frac{1}{8}\log 8+2\times\frac{1}{12}\log 12$$

$$=\frac{1}{12}(6\log 3+19)\text{ bits/sym}=2.3758\text{ bits/ sym}$$

$$\therefore\ \eta_c=\frac{6\log 3+19}{29}=98.31\%\ ;\ E_c=1.69\%$$

Example 2.6(***Trinary Coding***)

We shall consider the source of Example 6.12. For Trinary codes ***r = 3, [X = (0, 1, 2)]***

Since $q = 6$, we have from

$$q=r+\alpha(r-1)$$

$$\alpha=\frac{q-r}{r-1}=\frac{6-3}{2}=\frac{3}{2}=1.5$$

Thus α is not an integer and hence *we require one dummy message which makes* α = 2.

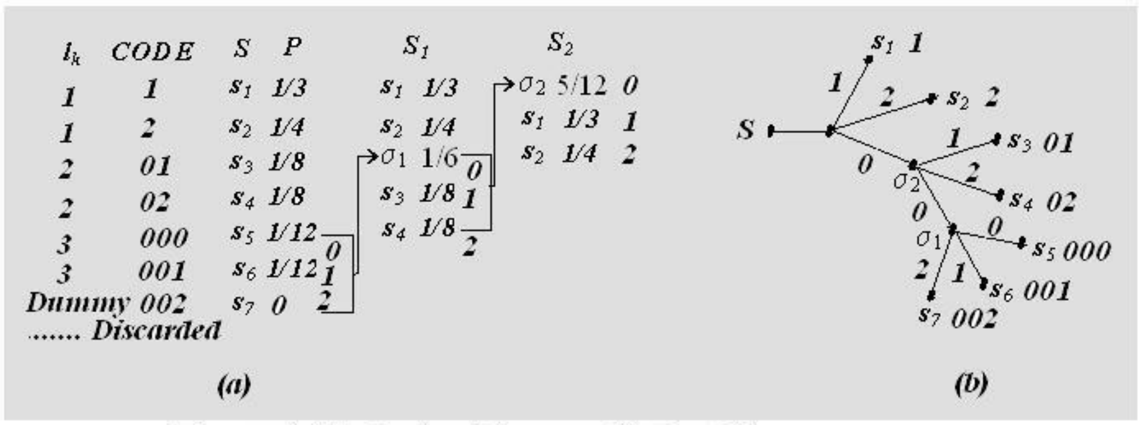


Fig 5.8 (a) Reduction Diagram (b) Tree Diagram For Trinary Huffman Coding of Example 5.13

For this code, we have

$$L = 1 \times \frac{1}{3} + 1 \times \frac{1}{4} + 2 \times \frac{1}{8} + 2 \times \frac{1}{8} + 3 \times \frac{1}{12} + 3 \times \frac{1}{12} = \frac{19}{12} \text{ Trinits / sym .}$$

And $\eta_c = \dfrac{6 \log 3 + 19}{19} = 94.672\%$, $E_c = 5.328\%$

Example 2.7:

We conclude this section with an example illustrating Shannon's noiseless coding theorem. Consider a source $S = \{s_1, s, s_3\}$ with $P = \{1/2, 1/3, 1/6\}$

A compact code for this source is: $s_1 \to 0$, $s_2 \to 10$, $s_3 \to 11$

Hence we have

$$L = \frac{1}{2} + \frac{2}{3} + \frac{2}{6} = 1.5$$

$$H(S) = \frac{1}{2}\log 2 + \frac{1}{3}\log 3 + \frac{1}{6}\log 6$$

$$= 1.459147917 \text{ bits/sym}$$

$\therefore\ \eta_c = 97.28\%$

The second extension of this source will have $3^2 = 9$ symbols and the corresponding probabilities are computed by multiplying the constituent probabilities as shown below

$s_1 s_1$	$\frac{1}{4}$	$s_2 s_1$	$\frac{1}{6}$	$s_3 s_1$	$\frac{1}{12}$
$s_1 s_2$	$\frac{1}{6}$	$s_2 s_2$	$\frac{1}{9}$	$s_3 s_2$	$\frac{1}{18}$
$s_1 s_3$	$\frac{1}{12}$	$s_2 s_3$	$\frac{1}{18}$	$s_3 s_3$	$\frac{1}{36}$

These messages are now labeled 'm_k' and are arranged in the decreasing order of probability.

$M = \{m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8, m_9\}$

$$P = \frac{1}{4}, \frac{1}{6}, \frac{1}{6}, \frac{1}{9}, \frac{1}{12}, \frac{1}{12}, \frac{1}{18}, \frac{1}{18}, \frac{1}{36}$$

The Reduction diagram and tree diagram for code construction of the second extended source is shown in Fig 5.9.

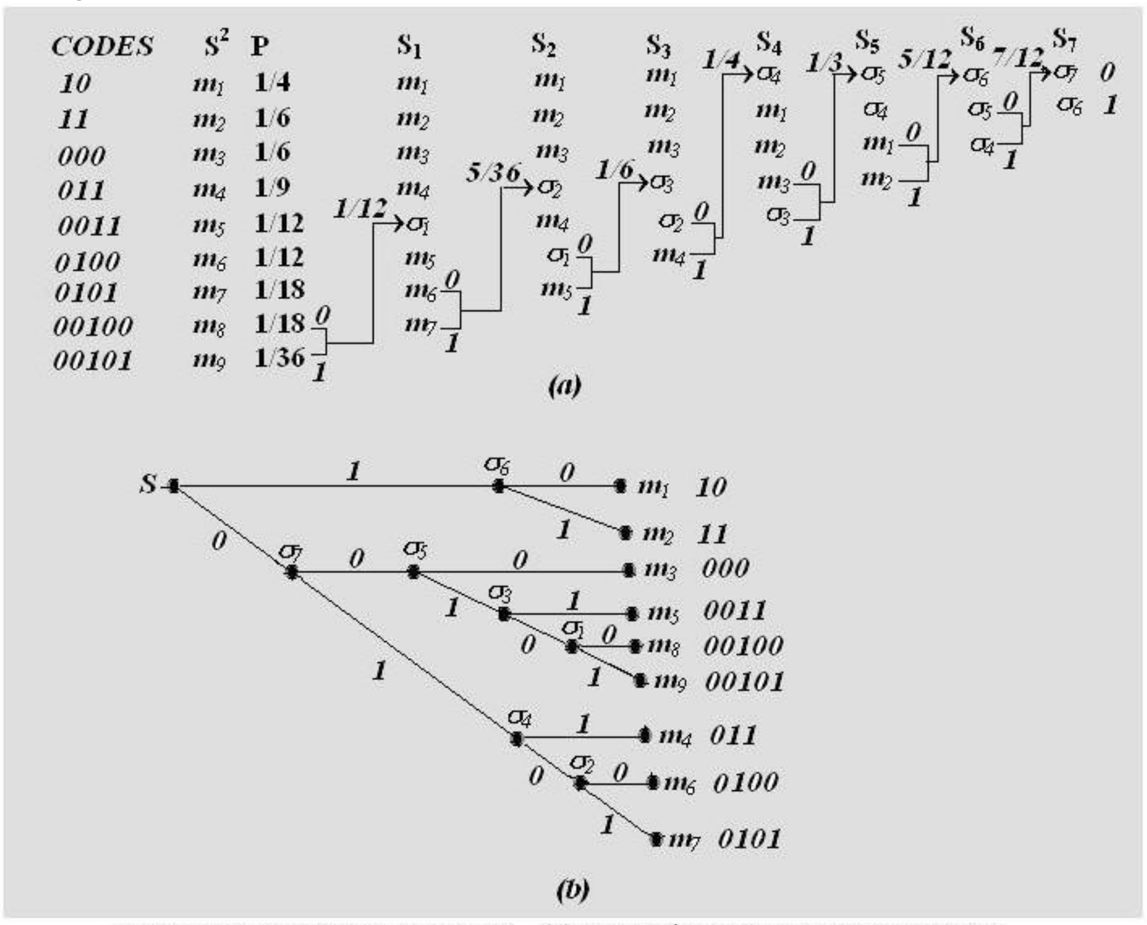


Fig 5.9 (a) Reduction Diagram (b) Tree Diagram for Example 5.14

For the codes of second extension, we have the following:

H (S²) = 2 H(S)

$$L = 2\times\frac{1}{4} + 2\times\frac{1}{6} + 3\times\frac{1}{6} + 3\times\frac{1}{9} + 4\times\frac{1}{12} + 4\times\frac{1}{12} + 4\times\frac{1}{18} + 5\times\frac{1}{18} + 5\times\frac{1}{36}$$

$$= \frac{107}{36} \text{ binits/symbol} = 2.97222222 \text{ binits/sym}$$

$$\eta = \frac{H(S^2)}{L \log 2} = \frac{2 \text{ x } 1.459147917}{2.97222222} = 98.186\ \% \quad E_c = 1.814\ \%$$

An increase in efficiency of ***0.909 %*** (absolute) is achieved.

This problem illustrates how encoding of extensions increase the efficiency of coding in accordance with Shannon’s noiseless coding theorem.

One non- uniqueness in Huffman coding arises in making decisions as to where to move a composite symbol when you come across identical probabilities. In Shannon- Fano binary encoding you came across a situation where you are required to make a logical reasoning in deciding the partitioning. To illustrate this point, consider the following example.

Example 2.8:

Consider a zero memory source with

*S= {**s1, s2, s3, s4, s5**}; P= {0.55, 0.15, 0.15, 0.10, 0.05}; X= {0, 1}*

Construct two different Huffman binary codes as directed below:

(a) Move the composite symbol as 'high' as possible.
(b) Move the composite symbol as 'low' as possible
(c) In each case compute the variance of the word lengths and comment on the results.

(a)We shall ***place*** the composite symbol as ' ***high***' as possible. The source reduction and the corresponding tree diagram are shown in Fig 6.10

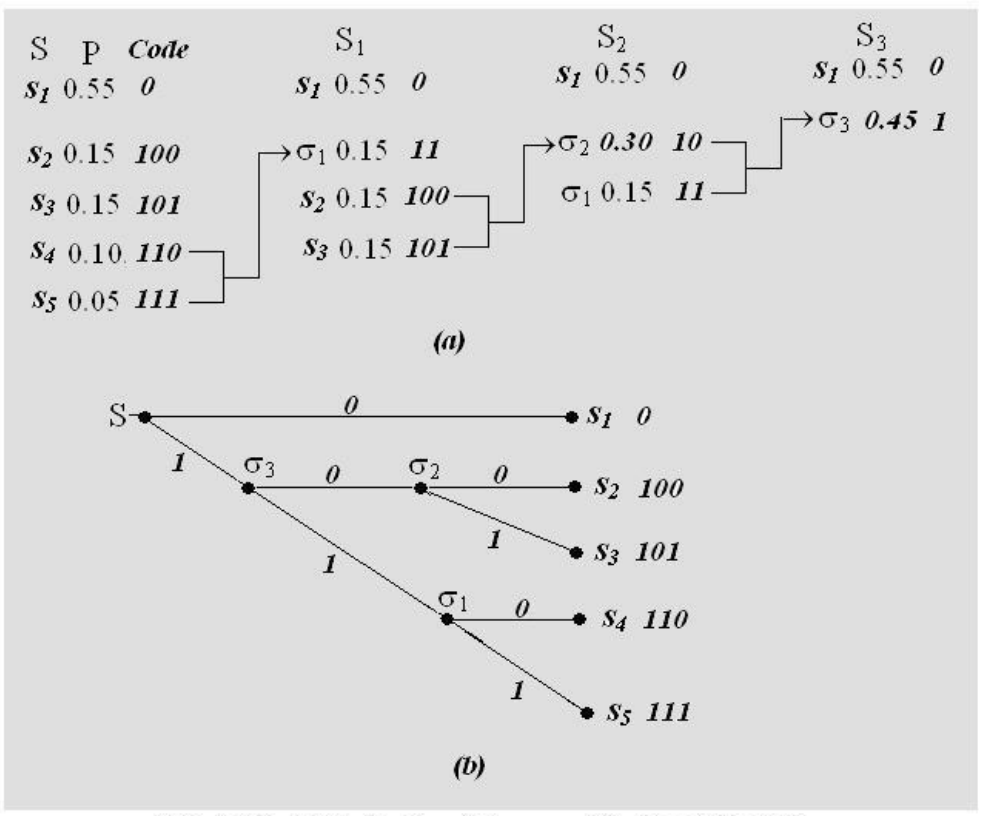


Fig 5.10 (a) Reduction Diagram (b) Tree Diagram

Symbols	s1	s2	s3	s4	s5
Codes	0	100	101	110	111
lk	1	3	3	3	3

We compute the average word length and variance of the word lengths as below:

L=0.55+3(0.15+0.15+0.10+0.05) =1.90 binits/symbol

σ^2_l **= 0.55(1-1.90)^2 + 0.45 (3-19)^2 = 0.99** is the variance of the word length.

(a) We shall ***move*** the composite symbol as ' ***low***' as possible. The source reduction and the corresponding tree diagram are shown in Fig 5.11.We get yet another code, completely different in structure to the previous one.

Symbols	s1	s2	s3	s4	s5

Codes	0	11	100	10 10	10 11
lk	1	2	3	4	4

For this case we have: ***L = 0.55 + 0.30 + 0.45 + 0.20= 1.90 binits/symbol***

Notice that the average length of the codes is same.

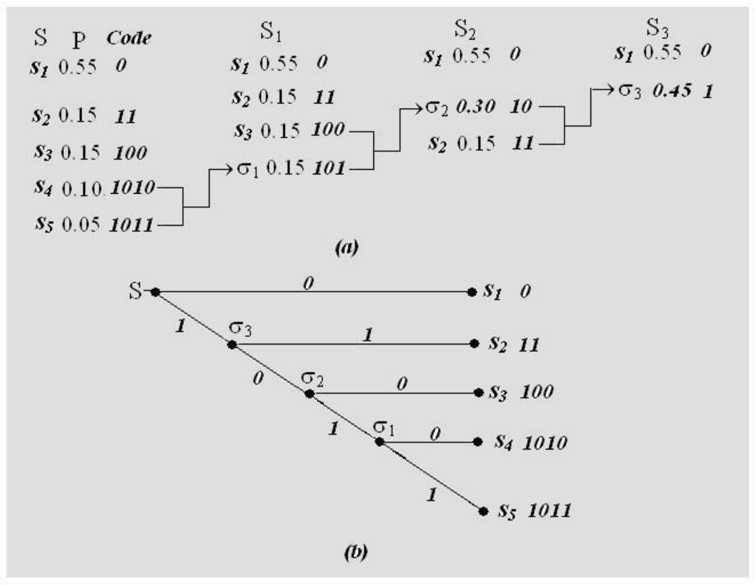


Fig 5.11 (a) Reduction Diagram (b) Tree Diagram

$\sigma_2^2 = 0.55 (1 -1.9)^2 + 0.15 (2 -1.9)^2 + 0.15(3 – 1.9)^2 + 0.10(4 -1.9)^2 + 0.05(4 -1.9)^2$

= 1.29 is the variance of the word lengths.

Thus, if the composite symbol is moved as high as possible, the variance of the average code word length over the ensemble of source symbols would become smaller, which, indeed, is desirable. Larger variance implies larger buffer requirement for storage purposes. Further, if the variance is large, there is always a possibility of data overflow and the time required to transmit information would be larger. We must avoid such a situation. Hence we always look for codes that have minimum possible variance of the word lengths. Intuitively " avoid reducing a reduced symbol in the immediate next step as far as possible moving the composite symbol as high as possible"

Outcome:

Able to understand the concept of Kraft McMillan Inequality property.

Able to understand and apply Shannon's encoding algorithm steps and procedure.

Able to solve problem related to binary coding.