

Information Theory and Coding
Prof. S. N. Merchant
Electrical Engineering
Indian Institute of Science, Bombay

Lecture - 19
Arithmetic Coding Part – II

In the previous class, we studied the procedure to generate a tag for a given sequence. However, the binary code for the sequence is what we really want to know. Therefore, we want to find a binary code that will represent the sequence in a unique and efficient manner. We have seen that tag forms a unique representation for the sequence. Therefore, it implies that binary representation of this tag forms a unique binary code for the sequence. However, we have placed no restriction on what values in the unit interval that tag can take.

Some of these values would be infinitely long, in which case although the code word is unique it may not be efficient. To make the code efficient, the binary representation for that tag is to be truncated, but if this code word for that tag is truncated then the question is the resulting code unique? The next question is that if the resulting code is unique, then is that code efficient? What we mean by that is how close or how far is the average length of the code away from the entropy of the source. So, let us examine these questions. We have seen that for a dyadic source that is if the probabilities of the source symbols are of the form.

(Refer Slide Time: 02:53)

$$P_i = \left(\frac{1}{2}\right)^{\alpha_i} \quad \text{where } \alpha_i \in \mathbb{I}$$

$$L_i = \log \frac{1}{P_i}$$

$$L_i = \left\lceil \log \frac{1}{P_i} \right\rceil + 1$$

$$\tilde{F}_x(x) \rightarrow [0, 1)$$

$$L(x) = \left\lceil \log \frac{1}{P(x)} \right\rceil + 1 \leftarrow$$

P_i equal to 1 by 2^{α_i} where α_i is an integer then the length of the code word is given by \log of 1 by P_i , but if the source is not dyadic then we have also seen that a reasonable choice for this length would be L_i is equal to upper flow of \log of 1 by P_i plus 1 . Now, in arithmetic coding each sequence is uniquely identified by the tag \tilde{F}_x . We know that this tag is a number in the interval 0 to 1 . So, a binary code word for this tag can be obtained by taking the binary representation of this number and then truncating it to L bits given by these expressions. So, let us look at the example which we had studied earlier.

(Refer Slide Time: 05:03)

Ex: $S = \{s_1, s_2, s_3, s_4\}$ $P: P(s_1) = 1/2$
 $X(s_i) = i \quad s_i \in S$ $P(s_2) = 1/4$
 $P(s_3) = 1/8$
 $P(s_4) = 1/8$

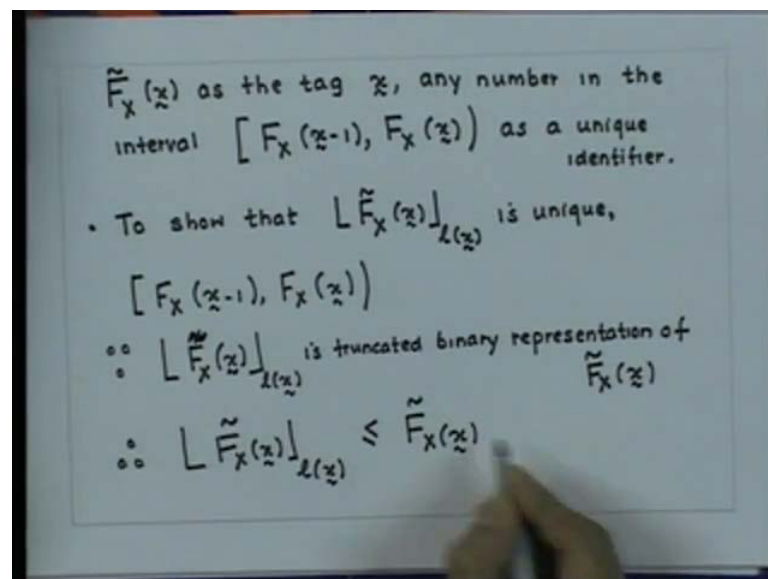
Symbol	F_x	\tilde{F}_x	In Binary	$\left\lceil \log \frac{1}{P(x)} \right\rceil + 1$	Codeword
1	0.5	0.25	.010	2	01
2	0.75	0.625	.101	3	101
3	0.875	0.8125	.1101	4	1101
4	1.0	0.9375	.1111	4	1111

A binary code for a 4-letter alphabet

So, we have a source consisting of four symbols and we use the mapping, which we had studied earlier in the Shannon Fano Elias coding to convert this symbols to integer random variables. And for this source with this probability model, we can generate the binary code based on the concept just discuss we have four symbols out here. So, we have the cumulative distribution function for each of the symbol.

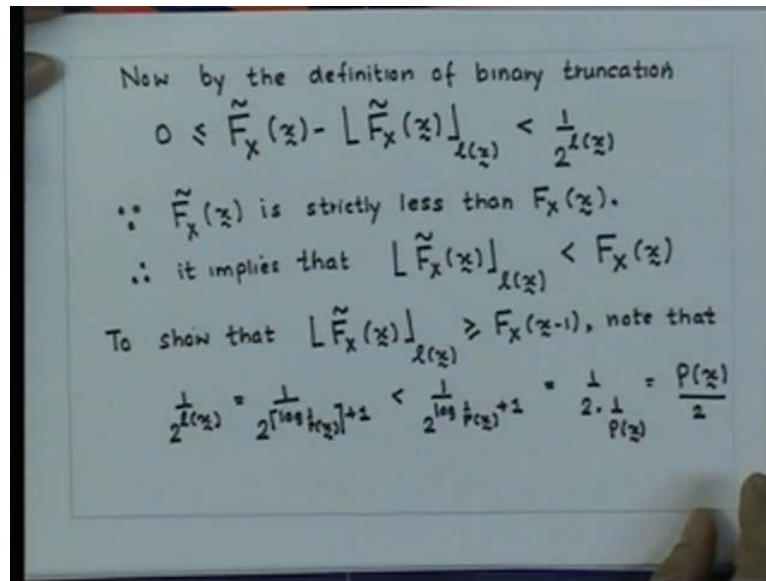
We can get a modified cumulative distribution function for the symbol as shown here, then the representation of this cumulative distribution function in binary is given as shown here. Then we truncate this binary representation by the length, which is calculated as shown here and using that we get the code word as shown in the last column. Now, we will show that the code obtained in this fashion is a uniquely decodable code. We first show that this code is unique and then we will show that it is uniquely decodable.

(Refer Slide Time: 06:23)



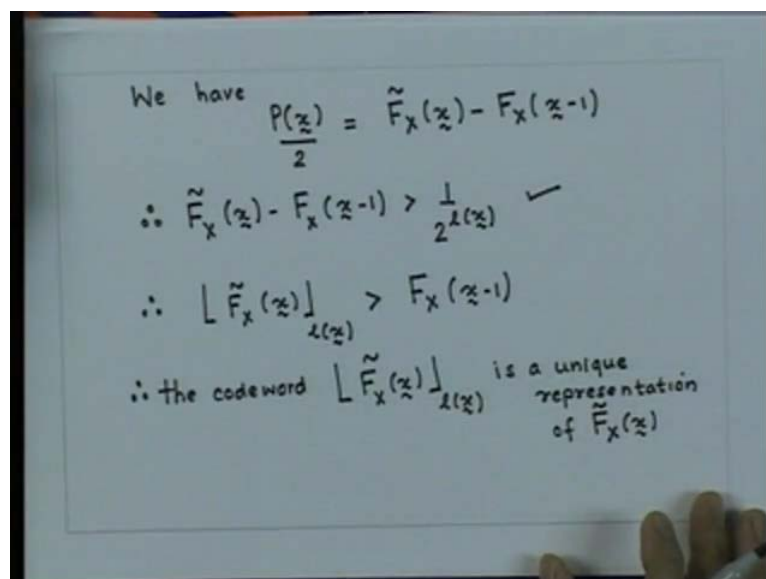
Recall that we have the tag for using $\tilde{F}_X(x)$ as the tag for a sequence x any number in the interval given by this as a unique identifier. Now, to show that when we truncate this tag to the length $l(x)$, which is indicated by this is to show that this is unique. We need to only show that it is contained in the interval that means this term is contained in the interval given by this. Now, because lower floor is truncated binary representation of $\tilde{F}_X(x)$ this implies therefore, $\tilde{F}_X(x)$ truncated value is less than or equal to $\tilde{F}_X(x)$.

(Refer Slide Time: 09:58)



Now, by the definition of Binary truncation, we can write the following expression. If you recall, this procedure is similar to the one which we had adopted for the Shannon Fano Elias coding. Now, because \tilde{F}_x is strictly less than cumulative distribution function. Therefore, it implies that truncated version of \tilde{F}_x is less than F_x . Now, to show that truncated version of \tilde{F}_x is greater than equal to $F_x(x-1)$, note that $\frac{1}{2^{l(x)}}$ is equal to $\frac{1}{2^{\lceil \log_2 \frac{1}{P(x)} \rceil + 1}}$, this is less than $\frac{1}{2^{\log_2 \frac{1}{P(x)} + 1}}$. And this quantity is equal to $\frac{1}{2} \cdot \frac{1}{P(x)}$. So, this is equal to $\frac{P(x)}{2}$. So, we will use this relationship.

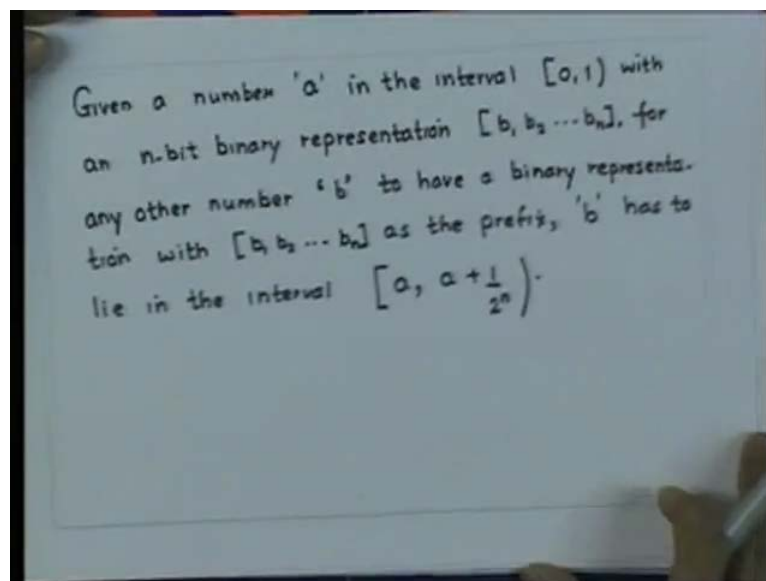
(Refer Slide Time: 13:45)



So, we have from the definition of the above modified C D F, P_x by 2 is equal to $F_{\tilde{x}}$ minus cumulative distribution function of the sequence x minus 1. So, from this we get therefore, $F_{\tilde{x}}$ minus f_1 is greater than 1 by $2^{-1} x$. So, from this relationship and this relationship which we have derived 1 by $2^{-1} x$ is less than P_x by 2 we get this relationship. Therefore, what it implies that truncated version of $F_{\tilde{x}}$ is greater than F of x minus 1. Therefore, it implies that the code word obtained by truncating the binary representation of $F_{\tilde{x}}$ is a unique representation of $F_{\tilde{x}}$.

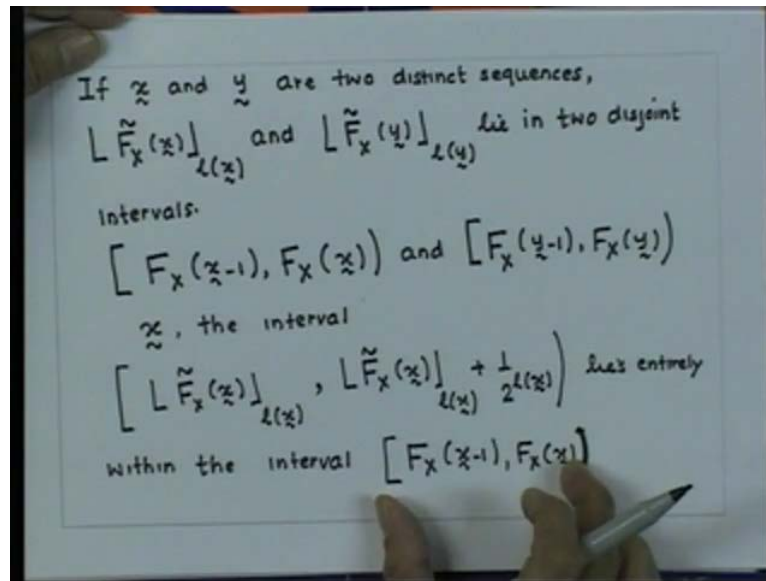
Now, to show that this code is uniquely decodable, we will show that the code is a prefix code that is no code word is a prefix of another code word because a prefix code is always a uniquely decodable code by showing that arithmetic code is a prefix code, we automatically show that it is uniquely decodable. Now, to go ahead with the proof of that let us look at one fact that if I have a number.

(Refer Slide Time: 17:04)



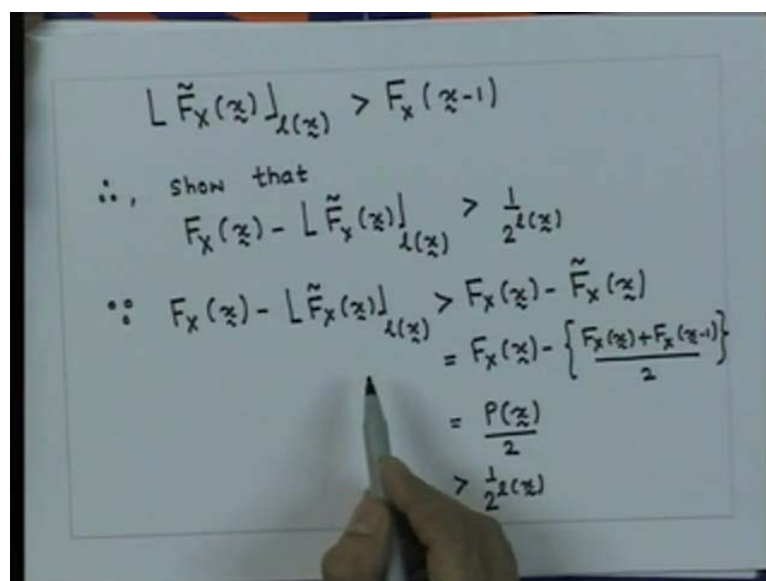
Let us say a number in the interval 0 to 1 with an n bit binary representation b_1, b_2 up to b_n . Then for any other number b to have a binary representation with this as a prefix with b_1, b_2 up to the b_n as the prefix b has to lie in the interval a, a plus 1 by 2 raise to n . So, using this fact will try to prove that arithmetic code is a instantaneous or prefix code. Now, what this implies is that.

(Refer Slide Time: 19:32)



If x and y are two distinct sequences, then we know that F tilde truncated value of it given by this expression and truncated value for the representation of the sequence y lie in two disjoint intervals. The intervals are given by f of x y minus 1. Therefore, if we can show that for any sequence x the interval given by truncated value of F x . So, if we can show that this interval lies entirely within the interval F x x minus 1, F x is a open interval. So, if we can show that this lies entirely with within this interval this will mean that the code for one sequence cannot be the prefix for the code of another sequence.

(Refer Slide Time: 23:05)



Now, we have already shown that, truncated value of F tilde is greater than $F \times x$ minus 1, therefore all we need to do is show that $F \times x$ minus truncated version of $F \times$ is greater than $1 - 2^{-l(x)}$. Now, this is true because F of x minus truncated version of F tilde is greater than $F \times$ minus F tilde \times because this quantity is less than this quantity and this is can be written as minus this by definition is by 2.

So, this is equal to probability of the sequence x by $2^{-l(x)}$ and this we have shown is greater than $1 - 2^{-l(x)}$. Therefore, this code is prefix free and by taking the binary representation of F tilde \times and truncating it to $l(x)$ given by upper flow of \log of $1 - P(x)$ plus 1 bits, we obtain a uniquely decodable code. Now, although the code is uniquely decodable, the next question is how efficient is it? So, let us answer that question, we have shown that the number of bits.

(Refer Slide Time: 26:03)

$l(x)$ required to represent $\tilde{F}_x(x)$ with enough accuracy
 $l(x) = \lceil \log \frac{1}{P(x)} \rceil + 1$
 $l_{S^m} = \sum_x P(x) l(x) = \sum_x P(x) \left\{ \lceil \log \frac{1}{P(x)} \rceil + 1 \right\}$
 $\ll \sum_x P(x) \left\{ \log \frac{1}{P(x)} + 1 + 1 \right\}$
 $= -\sum_x P(x) \log P(x) + 2 \sum_x P(x)$
 $= H(S^m) + 2$

$l(x)$ required to represent F tilde x with enough accuracy, such that the code for different values of sequence x are distinct is $l(x)$ is equal to \log of $1 - P(x)$ plus 1. So $l(x)$ is the number of bits required to encode the entire sequence x . So, the average length of an Arithmetic code for the sequence of length n is given by $\sum_x P(x) l(x)$ summation over all x now this is equal to summation of $\sum_x P(x) l(x)$ is chosen by this expression.

So, this can be simplified as less than equal to summation of $\sum_x P(x) \log$ of $1 - P(x)$ plus 1, like this is strictly less than and this can be simplified as equal to $\sum_x P(x) \log P(x)$ plus 2 summation over all the sequences. So, we get this is by definition then entropy of the

emit extension of the source is to. Now, given that the average length of a code is always greater than the entropy the bounds on 1 average length of emit extension will be as follows.

(Refer Slide Time: 29:01)

$$H(S^m) \leq l_{S^m} \leq H(S^m) + 2$$

$$(L_{av})_{AC} \therefore \frac{H(S^m)}{m} \leq (L_{av})_{AC} \leq \frac{H(S^m)}{m} + \frac{2}{m}$$

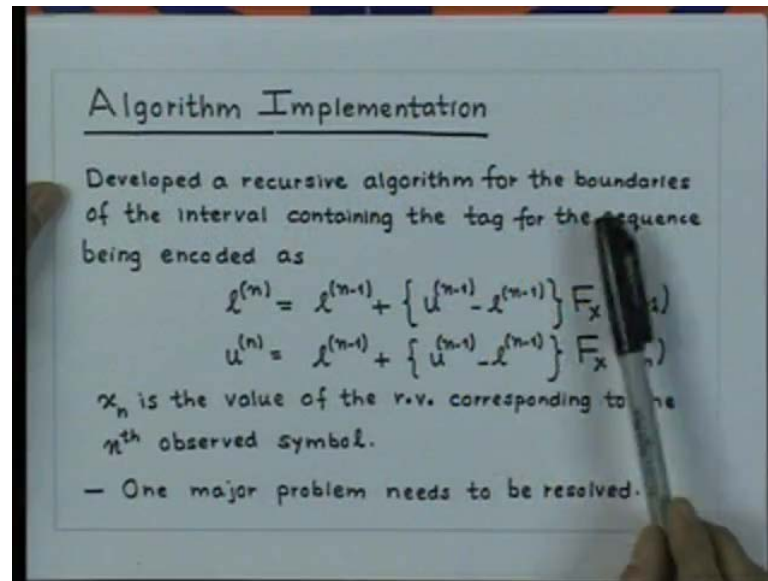
$$\text{iid } S \quad H(S^m) = mH(S)$$

$$\therefore H(S) \leq (L_{av})_{AC} \leq H(S) + \frac{2}{m}$$

So, this is the length we get for emit extension, so the length per symbol we denote by 1 average and since, we are talking about the arithmetic code we will denote it by AC for the arithmetic code will be given by dividing this quantity by m. Therefore, we get H of S m over m is less than equal to 1 average for arithmetic code less than equal to H of S m divided by m plus 2 m. Now, for the iid source we know H of S m is equal to m times H S.

Therefore, from this we get and therefore, entropy of the source 1 average for the arithmetic code is less than equal to the entropy of the source plus 2 by m. So, by increasing the length of the sequence, we can guarantee an average length of the code as close to the entropy as we desire. Now, after having studied the mathematical aspect of generation of the tag for the arithmetic code, and representation binary representation of the tag. And also having studied procedure to decipher this tag, let us look at some of the aspects of the implementation of the arithmetic code. So, let us look at the algorithmic implementation for the arithmetic code, we have developed a recursive algorithm.

(Refer Slide Time: 31:42)



For the boundaries of the interval containing the tag for the sequence being encoded as given by these expressions shown here, where x_n is the value of the random variable corresponding to the n^{th} observation symbol. Now, one major problem needs to be resolved while implementing this algorithm. Recall that the rationale for using numbers in the interval 0 to 1 as a tag was that there are infinite number of numbers in this interval. However, in practice the number of numbers that can be uniquely represented on a machine is limited by the maximum number of bits we can use for representing the number. Consider values of l_n and u_n in the example which we had discussed earlier, for the generation of the tag.

(Refer Slide Time: 32:57)

Ex: 1 3 2 1

$l^{(1)} = 0$	$u^{(1)} = 0.8$
$l^{(2)} = 0.656$	$u^{(2)} = 0.8$
$l^{(3)} = 0.7712$	$u^{(3)} = 0.77408$
$l^{(4)} = 0.7712$	$u^{(4)} = 0.773504$

Consider the values of $l^{(n)}$ and $u^{(n)}$

- As n gets larger, these values come closer and closer together

⇒ In order to represent all the subintervals uniquely we need increasing precision as the length of the sequences increases.

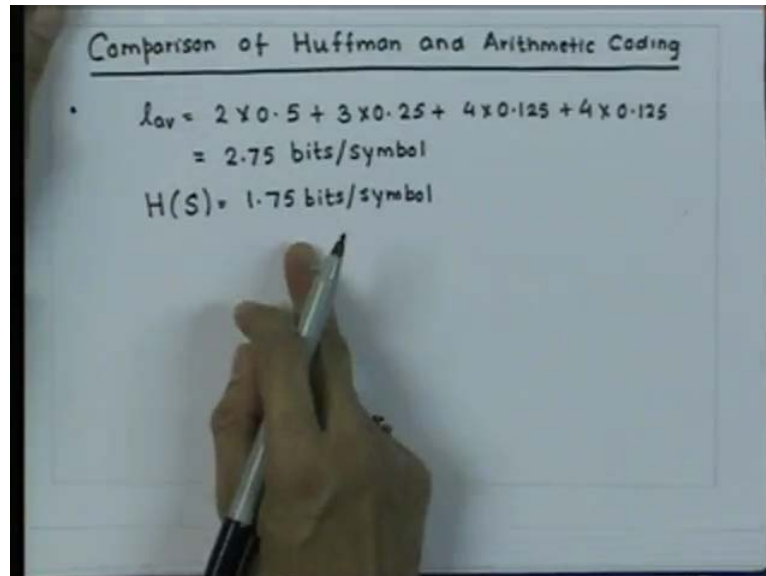
So, if we examine the values, which we had obtained then are shown here, so at the step number 1 when we have received the sequence number the first symbol, in the sequence that is 1 the lower limits are shown here that 0 an upper limit is 0.8. After this, we have received the symbol 3 the limits are shown here. So, from this result it is clear that as n gets larger and larger this values of l_n and u_n would come closer and closer together. So, the difference this 0.8 difference is less than 0.8 and this difference is still less than this and this difference is still less than this.

So, in order to represent all the sub intervals uniquely we need increasing precision as the length of the sequences increases. Now, in a system with finite precision the two values are bound to converge, and if these two values converge we will lose all information about the sequence from the point at which the two values converge. Now, to avoid this situation in a practical scenario rescaling of the interval is done.

This is done in a manner that will preserve the information that is being transmitted and also it is done in a manner so that the encoding is incremental. What you mean by that is that, using incremental encoding procedure we can transmit portions of the code word as the sequences being observed, rather than wait until the entire sequence has been observed before transmitting the first bit. Now, in the literature integer implementation of this algorithm has also been discussed. We will not go into the details for as far as this course is concerned. Now, after having studied arithmetic coding and Huffman code the

question is which is the efficient coding scheme? So, let us look at some of the points or advantages and disadvantages related to both Huffman and arithmetic coding.

(Refer Slide Time: 36:04)



Comparison of Huffman and Arithmetic Coding

$$\lambda_{av} = 2 \times 0.5 + 3 \times 0.25 + 4 \times 0.125 + 4 \times 0.125$$
$$= 2.75 \text{ bits/symbol}$$
$$H(S) = 1.75 \text{ bits/symbol}$$

Comparison of Huffman and arithmetic coding. Arithmetic coding is more complicated than the Huffman coding, but arithmetic coding allows us to code sequences of symbols. Now, how well this coding scheme works depends on how it is used. Let us try to use this code for encoding sources for which we know the Huffman code. So, let us go back to the example which we had discussed earlier in today's class. For the example which we had discussed earlier if we look at the arithmetic code, which we obtained for this source is given here. If we calculate the average length for this code then it turns out to be 2×0.5 plus 3×0.25 plus 4×0.125 plus 4×0.125 , and this is equal to 2.575 bits per symbol.

(Refer Slide Time: 38:41)

Ex: $S = \{s_1, s_2, s_3, s_4\}$ $P: P(s_1) = 1/2$
 $X(s_i) = i$ $s_i \in S$ $P(s_2) = 1/4$
 $P(s_3) = 1/8$
 $P(s_4) = 1/8$

Symbol	F_x	\tilde{F}_x	In Binary	$\lceil \log_{\frac{1}{P(x)}} \rceil + 1$	Codeword
1	0.5	0.25	.010	2	01
2	0.75	0.625	.101	3	101
3	0.875	0.8125	.1101	4	1101
4	1.0	0.9375	.1111	4	1111

A binary code for a 4 alphabet

And entropy for this source is 1.75 bits per symbol and because this is a dyadic source. Huffman code is able to achieve this entropy. So, obviously arithmetic coding is not a good idea if you are going to encode your message one symbol at a time. So, let us repeat the example with messages consisting of 2 symbols. Now, it is important to note that we are only doing this to demonstrate a point, in practice we would not code symbols this shot using an Arithmetic code. So, if we encode two symbols at a time then that arithmetic code for that can be obtained as shown here.

(Refer Slide Time: 39:28)

Ex: Encoding two symbols at a time

Message	$P(x)$	$\tilde{F}_x(x)$	$\tilde{F}_x(x)$ in binary	$\lceil \log_{\frac{1}{P(x)}} \rceil + 1$	Codeword
11	0.25	0.125	.001	3	001
12	0.125	0.3125	.0101	4	0101
13	0.0625	0.40625	.01101	5	01101
14	0.0625	0.46875	.01111	5	01111
21	0.125	0.5625	.1001	4	1001
22	0.0625	0.65625	.10101	5	10101
23	0.03125	0.703125	.101101	6	101101
24	0.03125	0.734375	.101111	6	101111
31	0.0625	0.78125	.11001	5	11001
32	0.03125	0.828125	.110101	6	110101
33	0.015625	0.8515625	.1101101	7	1101101
34	0.015625	0.8671875	.1101111	7	1101111
41	0.0625	0.90625	.11101	5	11101
42	0.03125	0.93125	.111101	6	111101
43	0.015625	0.946875	.1111101	7	1111101
44	0.015625	0.9625	.1111111	7	1111111

So, these are the other different messages which can be generated for this different messages. We can find out what the probabilities of the sequences of then two we can find out what is the modified C D F. We can represent this modified C D F in Binary and then we can truncate this modified C D F in binary by using the lengths calculated as shown in this column and finally, we get the code words I have shown here.

Now, the average length per message for this coding scheme turns out to be 4.5 bits therefore, using two symbols at a time we get. The average code word length of 2.25 bits per symbol which is certainly less than 2.75 bits per symbol, which we obtained earlier, but still this is not as good as the best average code length, which can be obtained with the Huffman code and that is 1.75 bits per symbol.

However, we see that as we increase the number of symbols per message our results get better and better. Now, the question is how many samples or how many symbols do we have to group together to make the arithmetic coding, perform better than the Huffman coding scheme. We can get some idea by looking at the bounds on the average length for the code.

(Refer Slide Time: 41:10)

$$H(S) \leq (L_{av})_{Ac} \leq H(S) + \frac{2}{m} \leftarrow AC$$

$$H(S) \leq (L_{av})_{Hc} \leq H(S) + \frac{1}{m} \leftarrow Hc$$

$$q \quad q^m \quad q=16 \text{ and } m=20$$

$$q^{20} = \underline{16^{20}!}$$

For the arithmetic code the bound, which we had obtained is given by this inequality. Now, it does not take many symbols in a sequence before the average length of the code arithmetic code becomes quite close to the entropy. However, for the Huffman code if

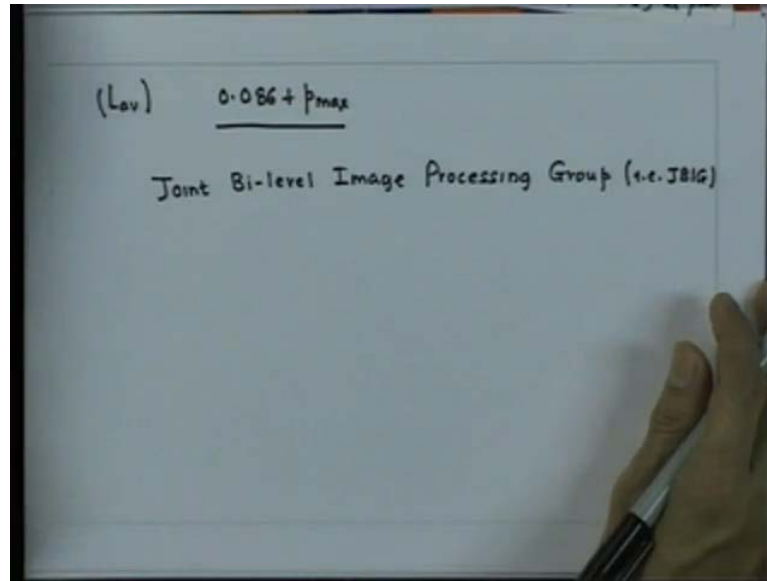
we block m symbols together then the average length of the code is given by this inequality which we had proved earlier.

Now, looking at these two inequalities this is for arithmetic code this is for Huffman code, the advantage seems to lie with the Huffman code all though the advantage decreases with increasing m . However, it is important to remember that to generate a code word for a sequence of length n using the Huffman procedure requires building the entire code, for all possible sequences of length m . So, if the only alphabet size was q and we are coding sequences of length m then the size of code book would be q raised to m . Now, taking reasonable values of q equal to 16 and m equal to 20, this would give us a code book size of the value which is equal to 16 raised to 20.

Now, this is obviously not a viable option for the arithmetic coding procedure we do not need to build the entire code book. Instead we simply obtain the code word for the tag corresponding to a given sequence. Therefore, it is entirely feasible to code sequences of length 20 or much more using arithmetic coding procedure. So, in practice we can make m large for the arithmetic coder, but we cannot do the same way for the Huffman coder. This means, that for most of the sources we can get the average length of the code closer to the entropy using arithmetic coding procedure than by using Huffman coding procedure.

The exceptions are sources whose probabilities are powers of 2. In this case the single letter Huffman coding achieves the entropy and we cannot do any better with arithmetic coding, no matter how long a sequence we pick. The amount of gain also depends on the source recall that for the Huffman codes, we are guaranteed to obtain the average length of the codes.

(Refer Slide Time: 44:44)



Within $0.086 + p_{\max}$ of the entropy, where p_{\max} is the probability of the most probable letter in the alphabet. Now, if the alphabet size is relatively large and the probabilities are not too unbalanced the maximum probability p_{\max} is generally small. In this cases, the advantage of arithmetic coding over Huffman coding is small and it may not be worth the extra complexity to use arithmetic coding rather than Huffman coding. However, there are many sources like alphabet size is small and the probabilities are highly unbalanced in this cases the use of arithmetic coding is generally worth that is complexity.

Another major advantage of arithmetic coding is that it is easy to implement a system with multiple arithmetic codes. Now, this may seem a bit contradictory as we have claimed that arithmetic coding is more complex than Huffman coding. However, it is the computational complexity of the machinery that causes the increase in the complexity. Once we have the computational machinery to implement one arithmetic code all we need to implement more than single arithmetic code is the availability of more probability tables.

So, if the alphabet size of the source small as in the case of binary source there is very little added complexity. In fact it is possible to develop multiplication free arithmetic coders that are quite simple to implement. Finally, it is much easier to adapt arithmetic codes to changing input statistics. All we need to do is estimate the probabilities of the

input alphabet. Now, this can be done by keeping a count of the letters as they are encoded. There is no need to preserve a tree as with adaptive Huffman codes. Furthermore there is no need to generate a code upright as in the case of Huffman coding.

Now, this property allows us to separate the modelling and coding procedures in a manner that is not very feasible in Huffman coding. This separation permits greater flexibility in the design of compression systems which can be used to great advantage. Arithmetic coding is being used in a variety of lossless and lossy compression application. Arithmetic coding is the coding scheme recommended by the joint bi-level image processing group that is J B I G as part of the standard for coding binary images. So, far in our course we were concerned with properties of information sources and their transformations of sequences of source symbols into sequences of code symbols.

We were able to relate our information measure to properties of information source. In particular, we showed that the entropy of our source could be use to obtain lower bound to the average number of code symbols needed to encode each source symbol. We also use this bound to define the redundancy and efficiency of a code. We devoted our study to synthesise of course, containing as little redundancy as possible.

Now, we will shift our study from the information source to the information channel that is from information generation to information transmission. We shall see that it is not always desirable to use codes with little or no redundancy. We will be concerned primarily with methods of putting redundancy back into codes. We will also use the entropy idea to describe, how we may utilise an unreliable information channel to transmit reliable information. So, from the next class onwards we will shift our attention to Information channels and the concepts related to it.