Modern Digital Communication Techniques
Prof. Suvra Sekhar Das
G. S. Sanyal School of Telecommunication
Indian Institute of Technology, Kharagpur

Lecture - 09
Source Coding (Contd.)

Welcome to the lectures on modern digital communication techniques, till the previous lecture we have been looking into source coding and what we have seen is fixed length coding where you have we have assigned a fixed number of binary strings for a particular symbol and of course, we do not need to mention we are looking at discrete sources.

So, for discrete sources what we mean is that we are having a source which has an alphabet, alphabet would mean a group of symbols which is fixed for that and then the source produces each letter or each symbol of that alphabet at a time and there is a certain probability for each of the symbols. So, when we do a fixed length coding what we said is we assign a fixed number of bits for each symbol, and we do it in such a way that each symbol gets a unique code; that means, there is no 2 symbol which are assigned the same code so, every one of them are different.

Now, when we did this we saw that there is a gap of approximately one bit that can happen and compared to what can be done in a better way. So, we did another method where we saw that instead of taking one symbol at a time, if the source outputs are grouped or if they are buffered and we take a n number of symbols. So, if we assume that there are m symbols in the alphabet, and now we say that we take n symbols and make a new symbol out of this which is a super symbol. So, then we can reduce the overload or the overhead and the gap reduces to one upon n, where n is the number of symbols we group together.

So, then we also discussed that instead of doing in a fixed length code, we can use some kind of a variable length code where we take into account the fact that the symbols do not occur with equal probability. For example, we have taken if we take the English alphabet that is probably e occurs with highest probability compared to z and then we could assign the lowest stream length to e, and the highest stream length to z. So, that overall when we take the average number of bits per symbol what we get to is lower than

what you used in a fixed length code and we had taken a particular example in the previous lectures and we had showed you that it is better to use variable length code.

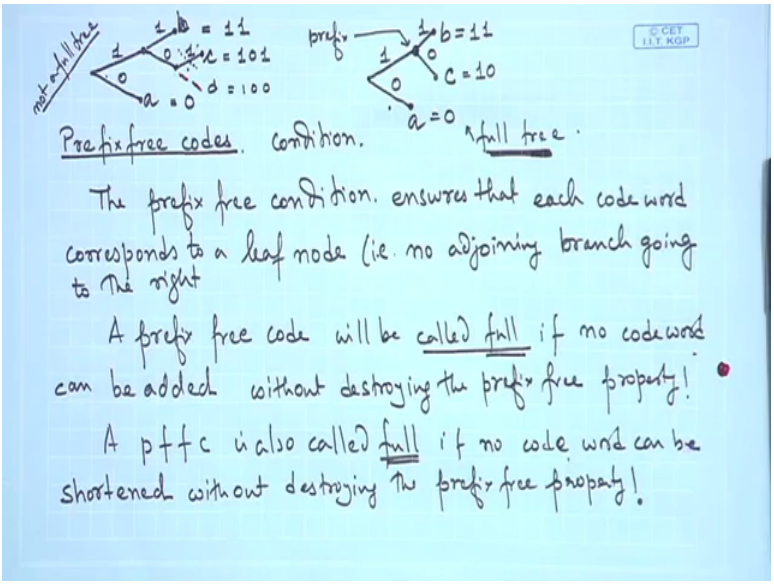
Now, when we did variable length code, what we saw is that if we are not wise enough then what could happen is that the receiver could get confused as to which particular code was sent. Although the receiver has a code book which is shared with the transmitter, but still the receiver make bit confused for example, if I use if I have three symbols a b and c and I use 0 for a one for b and 0 one for c in that case when a 0 one comes that the decoder does not know whether it is a or a or a b or a c. So, whereas, there was another example which was taken and a was given the code let us say 0, b was given 1 0 and c was given 1 1. In that particular case we found that each of the code were in such a manner that the receiver could parse it in such a way that it would not cause any confusion.

So, what we are hinting towards is that we get something known as unique decodability condition. Unique decodability condition what we said is that when the codes when the bits come in assuming there is a initial synchronisation it will read every bit go into the code book and check if that particular bit is in the code book. If it is not in the code book it comes back and records the next bit and now these 2 bits become a code word it goes back to the code book checks if that code book if that entry existed in the code book then it decodes the symbol if it does not exists it releases it, if it exists then it records that particular entry as the symbol that is been sent and clears the buffer. It is starts the reading or parts synch for in a new code word from the next bit onwards.

So, in that manners it can decode every symbol almost instantaneously. So, also what we will see is this types of codes are also in the instantaneous codes and what we have assigned are what we came up to was the term known as prefix free code. What it meant is that every symbol when it maps to a particular code is unique and not only that it will never be a prefix of any other code. It can be a unique, but if it prefix then it is it does not serve the purpose for example, if I take a 1 0 as one of my codes and another code is 1 0 1; in that case since 1 0 is a prefix of 1 0 one while decoding the moment the receiver finds 1 0 it will decode it as the previous symbol compared to the other symbol. So, what we landed up into was the definition of prefix free code.

So, when we discuss prefix free code we said that we can construct a binary tree; a binary tree which goes from the left and goes towards the right, and it starts at the node and at every node that we generate in the binary tree would either be a code which is the leaf node or if it is an intermediately node it will be a prefix of some other code, and only the leaf nodes will be assigned as code words because the leaf nodes would be unique.

(Refer Slide Time: 06:05)



And we have also taken one example where we had seen it is the binary code starts from here and it grows from the left to the right. So, this could be assigned as 1, this could be assigned as 0, there is an intermediately node and the tree keeps on growing 1 and a 0 and the tree keeps on going right and you could assign this particular leaf node as a, this particular node sorry this particular leaf node as a, this particular leaf node as b, this particular leaf node as c and then b would get a code word of 1 1, a could get a code word of 0, and c could get a code word of 1 0 1.

In this case what you will find is that this is unique there is no prefix, but still we have some more things to take care of and those few definitions which are very important for designing an efficient prefix free code is what we are going to start off with today. So, if we look at prefix free codes, what we can say is that we are going to look at the prefix free condition that we have mentioned in the previous lecture. So, what we will say is that the prefix free condition ensures that each code word corresponds to a leaf node that is what we have just mentioned, that is no adjoining branch going to the right.

So; that means, this is absolutely these three that we have identified are codes nothing which goes to the right of each one of them. So, absolutely perfect there is no problem. But we have another interesting point which is very important a prefix free code will be called full; now this is important will be called full if no code word can be added without destroying the prefix free property. So, this is one part so; that means, I cannot add another code word, if I can add another code word to this without destroying the prefix free property; that means, this is not full. So, we will see something more and I will write in short a prefix free code is also called full; if no code word can be shortened without destroying the prefix free property right

So, if we look at these 2, the first thing that we have here that a prefix free code will be called full if no code word can be added without destroying the prefix free property. Now if we look at this tree we can easily add one more branch over here without destroying the property right, but we cannot add anything here because then b will become a prefix and if we look at the previous definitions previous steps in the definition what we said is any node that you see the nodes that you see over here, these nodes are either intermediate nodes; that means, they are prefix or they are leaf nodes; that means, they are code words.

So, now in this case I can add one branch to this, and it does not destroy the prefix yet. So, if I call it something it is namely 1 0 0. So, this will be 1 0 0 let us say I put it as d right. So, this is not a prefix of anything and hence it can be easily added without destroying the prefix free. So, that would clearly mean that this is not a full tree the second definition the second statement here as a prefix free code is also called full, if no code word can be shortened without destroying the prefix free property. Now you should always note that this is a prefix free code there is no problem with it, but if I re draw it and I say that this is part right and I have this as leaf nodes call this a, I call this b, I call this c see in other words I have shorten this code I have removed this particular branch this is also feasible. So, if I do that I get one 0. So, this is how I get.

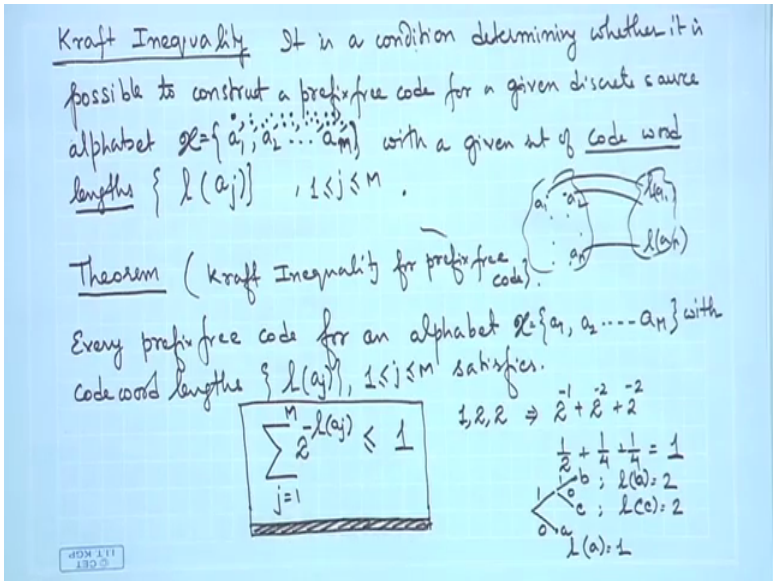
So, a remains 0, b remains 1 1, and c becomes 1 0. See if we compare these 2 what we get a is as it, b is as it, is only c is 1 0 1 e of the one is cropped so; that means, c is shortened from a 1 0 1 to a 1 0 without destroying the prefix free property right. So, this is very important note. So, this particular tree we can call a full prefix tree full tree right.

So, this is an example of a full tree and this is you can say non full tree or not a full tree right. So, both are prefix free there is no problem with this.

If I now apply these 2 to this tree, what we get is that I cannot add anything. So, if I look at this if I add I will be growing the tree further right. So, it says you cannot add to the existing tree, if the tree was here we could say that yes this is something which I could have added while I if I add this I am not destroying the prefix free property, but still I am able to remain within the tree structure, but here what is happening is I have if I have to add a tree I am changing the tree this will be different tree if I add it into this, and if I shorten c suppose I short this tree I want to make it one instead of c as 1 0 I want to make it as 1 . So, when I make it one it is an intermediary node and every intermediately node is a prefix. So, this point this is a prefix only leaf nodes so; that means, I cannot add this condition and this condition I cannot shorten. So, b if I shorten it becomes a intermediately node c if I shorten it, it become an intermediately node. So, if I do any of these things then it does not the prefix free property does not hold, and then I can say that the tree is full.

Now, this is very important now, what is the importance of a full tree is what we are going to see very shortly because the importance that you see if it is not a full tree. The important thing to note is that I can add a code to it so; that means, the code is somewhat not efficient you can say or in some manner I mean it is not correct to say it is not a efficient you can say that there is some provision to take a few more symbols and if you can shorten; that means, we have been unnecessarily using a longer code length right. So, which we do not want to do because if I am using a longer code word which is not necessary, then I am basically unnecessarily creating more bits which is going to inner lies my bandwidth that means, called uses extra bits extras bits per second or extra capacity of my communication links. So, I would like to happy to as less as possible. So, full tree is is very very important.

(Refer Slide Time: 15:14)



So, in this context we move forward and we have something known as the Kraft inequality. So, Kraft inequality is something which tells us that whether it is possible to construct a prefix free code with the set of code word lengths. For example, I give you a set of code word lengths let us say one to 2 so; that means, there will be one code word with the length of one bit and there will be 2 code words with lengths 2 bits age. So, now, the question is I am not given you code words I have given you lengths.

Now, what we should understand is that lengths are very important for us, once you have the lengths then you can conduct then you can construct prefix free codes. Now there is this connection is very very important, because if you start constructing the code well one can approach that, but rather it is quite interesting that what would be my average code word length of this, this is because this finally, decides the bit rate of the source. So, Kraft inequality something which will tell you that given a set of lengths whether prefix free codes are possible or not this is something critical.

So, we can write down that it is condition determining whether it is possible to construct a prefix free code for a given discrete source alphabet x would symbols let us say a one a 2 a m. Now it is not important to what we put we just put there are m alphabets with a given set of code word lengths marked by l of a j where j is less than or equal to m starting from 1. So, this if you note the way we are changing from the initial statements what we are saying is that let there will be a discrete source with this alphabet; that

means, x defined by this set of symbols. So, with which goes with our previous example you can take the dye this is one example you can take you can take the heads and tails of the coin you can also take letters of English alphabet as we have been always saying.

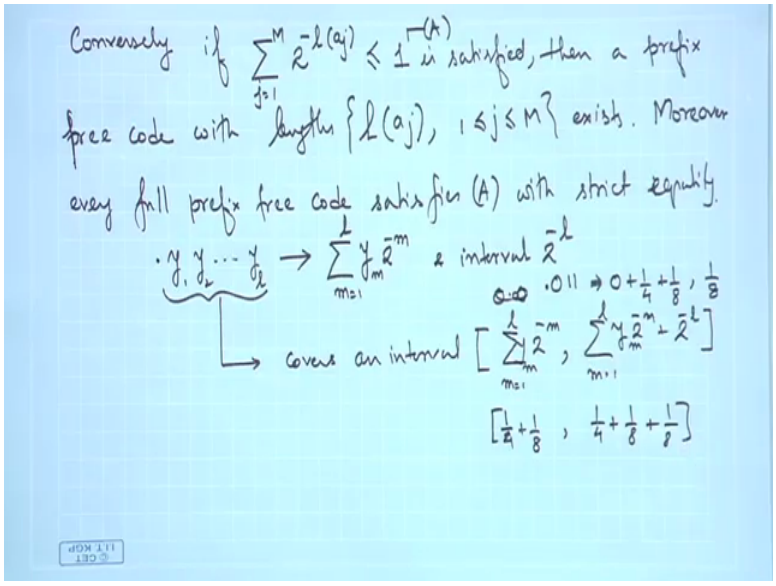
So, if we say that this alphabet is given and now we say instead of probabilities what we are talking about is suppose few lengths are given this is important we are not talking about, the code we are saying code word lengths we have not said about the code word only the lengths now given this lengths code word could be any combination that is a different story. So, now, given this lengths whether from these lengths you can construct codes look at the construct, I have given you a symbol and a set of symbols which is an alphabet now with these symbols I have also given you a few set of lengths.

So, as if there is a set of symbols a 1, a 2 up to let us say a m and I have told you that let there will be certain lengths. So, which maps l of a 1 and let us say this is length of am. So, let there be one is to one mapping and there could be one to many mapping also it does not matter there would be a few code words have the same lengths not a problem we have not talked about code words itself this is important. So, all we want to know is that if it is possible to construct a code word which is prefix free now once you know that it is possible to construct a prefix free code then life is easy what you can do is you can refer back to this binary tree and you can construct the code tree and then you can assign the leaf nodes and you know from our previous discussion that this tree forms a prefix free code right.

So, at this point we have this theorem known as the Kraft inequality. So, it also known as theorem for Kraft inequality for prefix free code. So, it states that every prefix free code for an alphabet let us say x which is equal to a 1 a 2 up to a m with code word lengths l of a j with j ranging from 1 to m satisfies this inequality sum of j equals 1 to m 2 to the power of minus l of a j is less than or equal to 1. So, if I give you lest say the lengths as 1 2 and 2 and ask you that whether I can construct a prefix free code with this or not what you are going to do is 2 to the power of minus 1 plus 2 to the power of minus 2 plus 2 to the power of minus 2 so; that means, a half plus 1 upon 4 plus 1 upon 4 which is equal to half plus half it is 1 so; that means, this sum which is the left hand side which is less than or equal to one. So, if since it is equal to 1 then we can say that yes this inequality is satisfied which means that a prefix free code can be constructed.

Now, what is the importance of this? The importance of this is that since I know of prefix free code can be constructed I can easily go head and construct the prefix free code now how do we construct the prefix free code get back to the tree and construct this tree which has code word length of one. So, length of if I call this a is equal to 1, I can call this as b and this is 11. So, length of the code word for b is 2 and this is c and I can say length of c is 2. So, clearly a is not a prefix of b or c and it is a full tree that I have conduct constructed and from this example you can also see that this if the tree is full this inequality is satisfied with the equality constraint. Now if it is not a full tree then that you can easily guess that it will be a less than a 1 this will not be equal to one. So, that is another criteria for other condition through which you can check whether the tree that you are going to construct will be a full tree or will it not be a full tree.

(Refer Slide Time: 24:27)



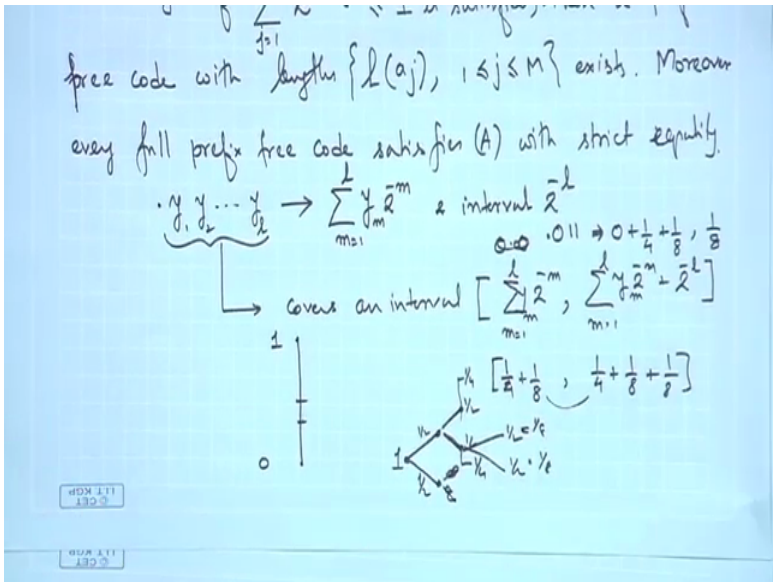
The advantage of the full tree is that full tree that you are using a full set of symbols as best as possible. So, we carry one with this and you can additionally say is that conversely if this summation that j equals one to capital M 2 to the power of minus the lengths of the code words are less than or equal to 1; that means, if this inequality is satisfied then a prefix free code with lengths length of a j 1 less than or equal to j is less than or equal to m exists more over every full this we have already said prefix free code satisfied if I say this is let us say A this is A with strict in with strict equality right.

So, what we mean by this is that you have this set of symbols, this symbols you are mapping to binary sequence. So, what we are asking the question before we map to the binary sequence whether at all prefix free codes can be formed with a certain set of lengths right and then you go ahead once you know that yes prefix free code can be constructed, then you form the tree and once you form the tree with appropriate lengths as you need then you can easily choose the code words as the length or the leaf nodes and you can construct the tree.

So, now the proof of this is there it is a there is a logical proof for this particular statement and there are many ways to go around this particular proof. So, one particular way of doing it is you can consider a binary string this is a this is a decimal point. So, as this now this can be expressed as the real number this binary string can be expressed as m equals to 1 to let us say l y of m 2 to the power of minus m so; that means, if I write a point one that I get one multiplied by 2 to the power of minus 1. So, half and this has an interval 2 to the power of minus l. So, if there are l bits in it the interval is 2 to the power of minus l. So, for example, if you have 0 point or if you have point 0 1 1, this would map to 0 plus 1 by 4 plus 1 by 8 because this 0 one multiplied by 2 to the power of minus 2 plus one multiplied by 2 to the power of minus 3 and this has an interval of 1 upon 8.

Now, so; that means, this particular sequence it covers an interval which is marked by 2 to the power of y of m, 2 to the power of minus m, m equals to 1 to l to m equals to 1 to l y of m 2 to the power of minus m plus 2 to the power of minus l because this is the interval length which we have said so; that means, this particular one covers this region from this to this plus 1 upon 8. So, this is the interval which it covers now at this point you can note that all the codes that form part of this are unique. So that means, this codes do not overlap; that means, there is no prefix when we say it is a prefix free code. So, if they are all unique that means, they do not overlap because they form different intervals these different intervals and if you see the length that it would cover.

(Refer Slide Time: 29:00)



So, it would cover a length from 0 up to 1 right that is the full length that it can cover and every length every length being unique; that means, this interval formed by a particular code is different from another interval formed by a code is it is a full tree is going to cover this full length between 0 and 1, and if it not a full tree it is not going to cover this full length from 0 to 1 it will be less than 1.

So, that is intuitive way of describing that you have a parts which add up to form a maximum value of one or it can form value which is less than 1. In other very simple way of looking at it is could be when the code tree starts it starts from here, and if this is the minimum code tree that you can have. So, these are the code word lengths that gets added and what we have seen that when you have such a source these symbols are probabilistic of course, they come with equal probability still there is certain amount probability, but they are probabilistic in nature and what we will find later on is that somewhat connectivity of this length with this probability. So, what we see is that in this a full tree we can assign it a value of one over here, and you can say there are every node it splits into a value of half right this is 0 this is 1 and whenever there is a split these values add up to the value of the node.

So, again this will be half and this will be half. So, even if it does not exists it does not extend this arm is half this arm is half, this is split into half of half and half of half so; that means, at this point the value is 1 by 4 at this point the value is 1 by 4 like if this one

splits further this is half. So, this half of this value half of 1 by 4 half of 1 by 4 so; that means, this value is equal to 1 by 8, this value is equal to 1 by 8 right. So, again that is also available from here; that means, we are basically doing the same thing in another way. So, at every node you have the same value and even if it splits the some value up to that node remains the same and the starting point is this is 0 and this is 1. So, there is half and half length assigned to these.

So, again if you add this up it comes to one. So, what we have over here is it is kind of more of logical proof at this point and what it summarizes to tell us is that if this particular inequality that we have over here or we used over here is satisfied by this set of lengths, all it tells you is that you can construct a prefix free code. Prefix free code is very important which we have established earlier, now once you know that the prefix free code can be constructed you can construct binary tree with appropriate lengths. Once you construct a binary tree with appropriate lengths you just assign symbols to the leaf nodes and you gets codes corresponding to those code words.

So, we stop this particular discussion at this point and we continue on source coding in the next lecture.

Thank you.