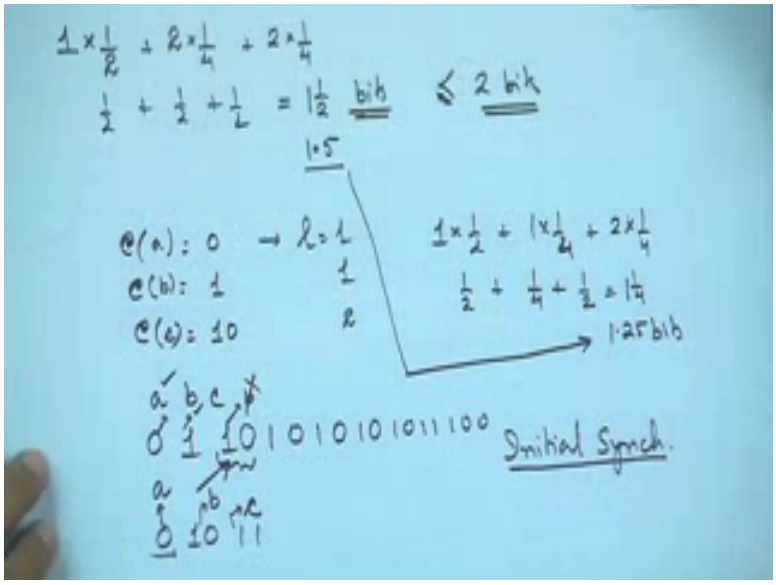
Modern Digital Communication Techniques
Prof. Suvra Sekhar Das
G. S. Sanyal School of Telecommunication
Indian Institute of Technology, Kharagpur

Lecture - 08
Source Coding (Contd.)

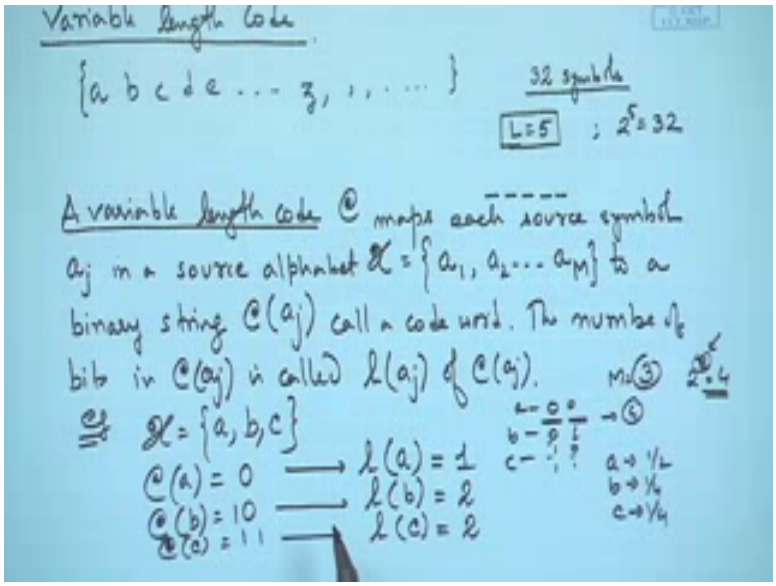
So, in the previous lecture what we saw is that instead of using fixed length code if we use variable length code, and you know the probability model of the source; that means, you know with what probability a particular symbol comes out, in that case you can use a better mechanism, you can exploit this knowledge and you can encode the source in such a way that you reduce the number of bits required to effectively encode the source and with the particular example.

(Refer Slide Time: 00:50)



That we discussed we showed that instead of 2 bits that are required for fixed length code, you can effectively use on an average 1.5 bits to encode this source.

(Refer Slide Time: 01:08)



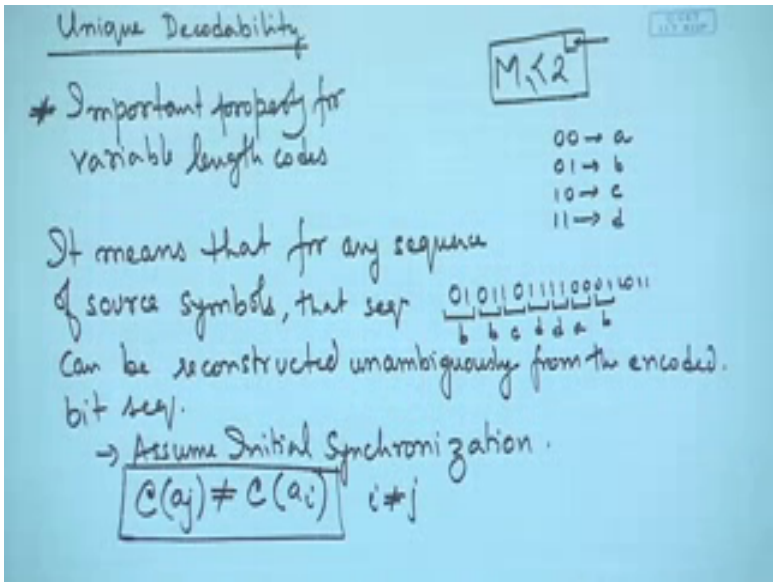
Now, if we move ahead with this particular example and what we had chosen is a code which is in this form; like instead of this if suppose we would have chosen a code which is c of a is equal to let us say 0, and c of b is equal to 1 and c code this is the code map of c is equal to 1 0 what is the problem or what is the advantage. What we do by making designing of code in this manner is that previously you can see that we had 1 code with 1 bit and 2 codes with 2 bits. Now we have a better situation this has a length of 1, this is the length of 1 this is a length of 2. So, if you are going to calculate the average number of bits that is required what do you get going by the example that we calculated 1 multiplied by half plus 1 multiplied by half plus 2 multiplied by 1 by 4.

So, that would lead to a half sorry 1 multiplied by 1 by 4 this is the probability of occurrence of b. So, that is half plus 1 by 4 plus 1 by half so; that means, 1 and 1 forth or 1.25 bits. So, this particular method of encoding that we have done over here it reduces this to 1.5 bits, but if I look at the stream of bits coming in at the decoder if I have send a b and c in this case I would get a 0, 1 and 1 0. The receiver or let us say I get a few further sequences and so on would not know whether this means a b or whether the receiver should read this together to mark it as a c right. So that means, these codes have a problem of ambiguity whereas, if we take this particular code the way we have marked it and we want to send a b c d the sequence that would come is 0 followed by b which is 1 0 and we have c which is 11.

Now, the receiver can use a method saying that I will read the code and immediately go and look up of course, for all this point initial synchronization is assumed. So, initial synch is always assumed it will go into the lookup table and it will find that 0 is a code word. If 0 is a code word it will immediately decode this to a c to a, next it gets a 1 it goes to the code book it finds there is no code word with one.

So, it waits it reach the next codebook next incoming bit it goes to the code book it finds there is an entry of 1 0 through this entry of 1 0 it maps it to b and it decodes it to b right next it finds a 1 it goes to the code book it sees there is no entry called 1 it comes back it reach the next entry which is 1 it goes back it finds 1 1, it decodes it as c right. Now you could say do the same thing for the previous code and it would let us see a 0 it goes to the code book, it reaches a no problem then it finds a 1 it goes there it is it is b no problem, then it finds a 1 it goes there reaches that b again now this is wrong because the transmitter has send a 1 0 for c so; that means, this particular code word cannot be decoded immediately.

(Refer Slide Time: 05:43)



So, therefore, we can say that there is a decodability issue with this particular code and this brings us to the motivation or this gives us the need for defining something known as unique decodability. So, at this point if we try to recall the fixed length code words, so for fixed length code words you would remember that we said we require 2 to the power of l which is equal to or m should be less than or equal to 2 to the power of l this is the

particular criteria that we had said for fixed length code whereby we said that l is the minimum integer for which this condition is satisfied. So, such that each of the symbols is assigned a unique code word, otherwise we did not have use that particular expression right.

So, we already have defined that in case of fixed length codes that we had choosing the number of bits required to represent the symbol so that each symbol gets a unique code word. So, for a fixed length code it is almost design into the system that each symbol will be unique. So, whenever I get a sequence of bits the receiver knows that it is a three bit sequence or it is a five bit sequence or it is a 7 bit sequence; that means, depending on the value of the l. So, the receiver would know the value of l. So, it will wait for so many bits go to the lookup table and read it off it is very simple.

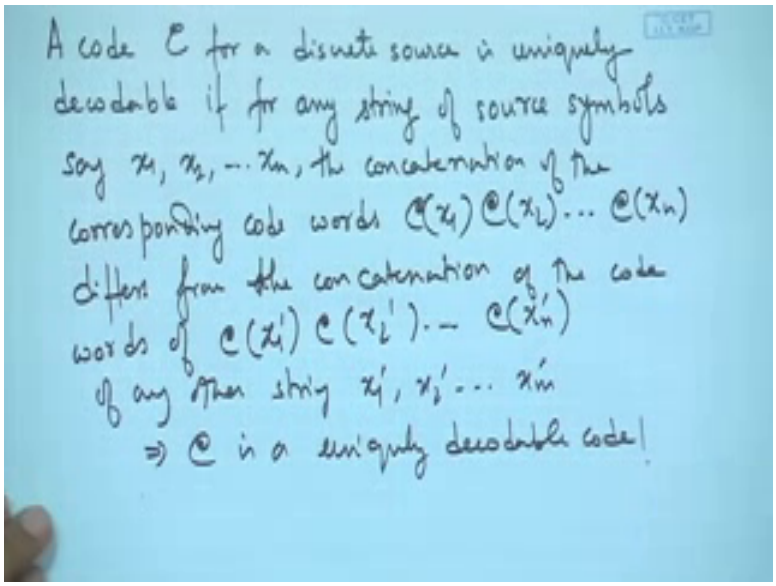
So, suppose we have 0 0 to indicate a, 0 1 to indicate b, 1 0 to indicate c and 1 1 to indicate d. So, if there is a string of this coming it will know that it has to read 2 at a time. So, it is going to read this go there it is a b, bit again it will read this go there it is a b, it will read this 1 0 it is a c 1 1 it is a d again 1 1 it is a d, 0 0 it is an a, 0 1 it is a b. So, there is no confusion about decoding. So, fixed length has no problem, but whenever you designing a variable length code it is very important that we bring in this notion of unique decodability into the picture right. It is an important property for variable length codes, this is very important what it means we can write down that it means that for any sequence of source symbols that sequence can be reconstructed unambiguously from the encoded bit sequence.

Of course we would assume initial synchronization so, that that is definitely assumed. So, what we are trying to say is that it requires code for a j not equal to code for a i, when i is not equal to j right. So, this is a clear requirement that these the 2 codes do not match there is a primary requirement and that is obviously, met in case of a fixed length code, in case of variable length code in 1 case we have said yes it is true and there is also another very important property which we are coming to that is known as prefix free condition.

So, that would definitely mean that any code word cannot be a prefix of another code word so; that means, in this particular example that we have taken here what we have seen you see here we are going to define all these things that here. 0 is not a prefix of any

other code word, 1 0 is definitely not 1 1 is definitely not, but if you look at this particular set of code words what we find that this 1 is a prefix of this code word so that means, code word for b is the prefix for code word of c at that head caused confusion. So, with this we move forward and we would define these things further.

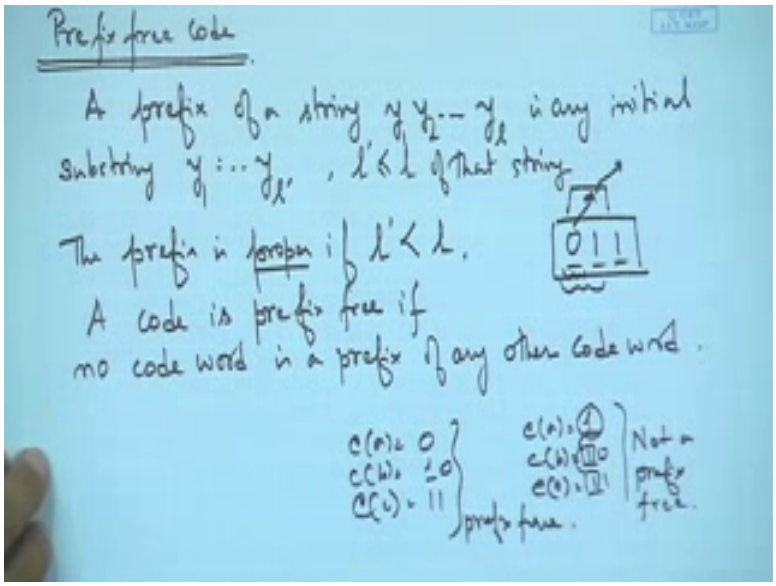
(Refer Slide Time: 11:10)



So, for unique decodability you would say that A code C let us say we define it as c for a discreet source is uniquely decodable if for any string of source symbols say x 1 x 2 up to x n the concatenation of the corresponding course code words c of x 1 c of x 2 up to c of x n differs from the concatenation of the code words of c of x 1 prime, c of x 2 prime, c of x n prime by prime you mean a different a choice of symbols of any other string x 1 prime x 2 prime up to x n prime of source symbols.

So; that means, is in that case in that case we would say that c is a uniquely decodable code. Now this is by definition of a uniquely decodable code. So, with these 2 definitions this is an explanation that we have given and is the definition that we have given for uniquely decodable code we are going to use this in further designing variable length codes.

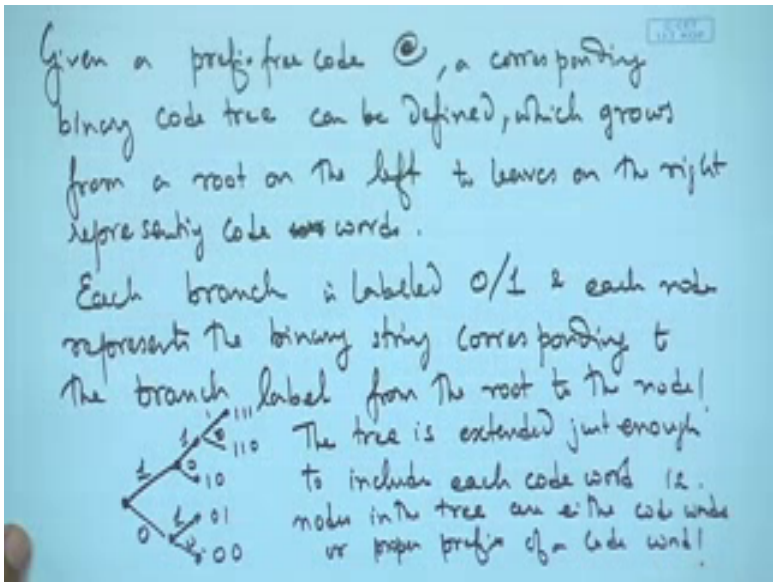
(Refer Slide Time: 13:31)



So, not only this as we just mentioned about the prefix free conditions which we have explained over here that here this is a prefix of another code words. So, we need to now go into the definition of prefix free code. So, by a prefix free code we have to first define a prefix. So, will say a prefix of a string y 1 y 2 up to y of l is any initial substring y 1 up to y l, u to y l prime where l prime is less than or equal to l of that string. So, what we mean is it suppose I have 0 1 1 is a code. So, there are 3.

So, 0 is a prefix 0 1 is a prefix. So, prefix free code should not if it has this as a code word should not have 0 as a code word, and it should not have 0 1 as a code word because these are prefix and we go further go on the prefix is proper if l prime is less than l, then we say a code is prefix free if no code word is a prefix that is what we just said of any other code word right. So, if we take the earlier example that we had given; that means, we are already taken a prefix condition now if we take this example that we had used here what we find 0 is not a prefix of these 2 and there is no code word which is 1 now suppose I had chosen suppose in this example. So, instead if I choose c of a is equal to 1, c of b is equal to 1 0, code of c is equal to 1 1 clearly it violates the prefix precondition because 1 is a code which is a prefix over here right this is not a prefix free code whereas, this particular example c of a is equal to 0, c of b is equal to 1, and code of c equals to 1 1 this is a prefix free code all right. So, this is how we can distinguish between a prefix free and a non prefix free code right.

(Refer Slide Time: 16:39)



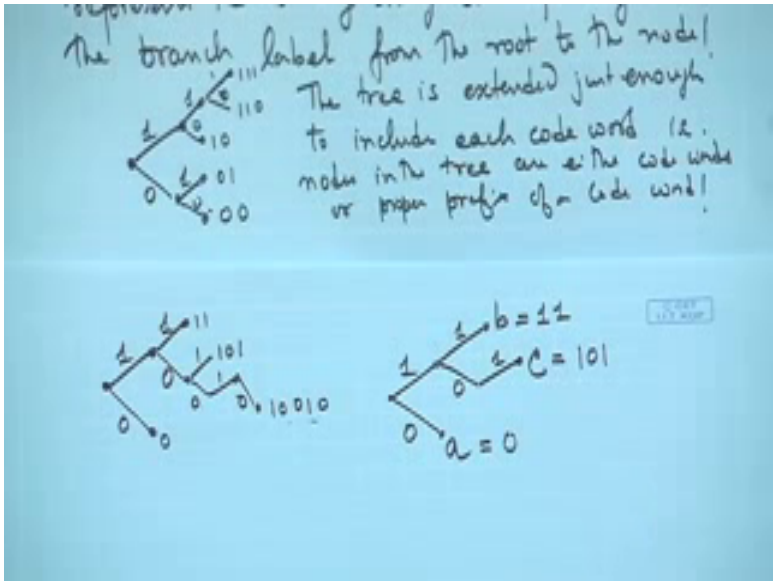
So, we can move on further and we would like to clarify that given a prefix free code c a corresponding this is the important let us define it first and then we will explain a corresponding binary code tree can be defined this is very interesting which close. So, we have defining it first which goes from a route on the left to leaves on the right representing code words. Further we say that each branch is labelled 0 or 1 and each node represents the binary string corresponding to the branch label from the route to the node.

See if you read this given a prefix free code see a corresponding binary code tree can be defined which grows root on the left to leaves on the right, and these leaves they form the code words. So, there will be code words and these are nodes. So, each node or each branch first we say each branch is labelled a 1 or a 0 each branch is labelled a 1 or a 0 all right and each node represents the code string from the route. So, if this is a node. So, you can grow it further if this is a node this node would represent the code which is 1 1, and the code at this point would be 1 1 1, the code at this point would be 1 1 0, the code at this point is 1 0 code at this point is 1 0, code at this point is 0 0 and so on and so forth.

So, you can form a binary tree and you could use this in designing prefix free codes as we going to see shortly; and further say that the tree is extended just enough to include each code word that is nodes in the tree are either code words or proper prefix of a code

word. Now what does it mean is that that tree is not arbitrarily designed; that means, you can say that I start of a binary tree from here.

(Refer Slide Time: 20:18)



From the root it grows towards the right and then I keep on doing like this. So that means, is this is 0 this is 1 this is 1 this is 0 1 0 1 0. So, this would mean that the code word here it is 1 1, code word here 1 0 1, here it is 1 0 0 1, it is not ending it is here a 1 0 0 1 1 0 0 1 and 0, and here it is 0 now this is well this is fine, but it says that the tree is extended just enough to include to include each code word in the node in the tree are that is to include each code word that is the nodes in the tree can either the code words or proper prefix free proper prefix of a code word nothing else. So, in this case what we find is that this tree is extended beyond the need, it is gone beyond the need and there is unnecessary extension over here which will be clear when we discuss about the other things. So, one particular another particular example that we can take this we have 1 and 0 we have 1 and 0 and this goes there.

So, we could label from our example this as the symbol a this with the symbol b and this with the symbol c. So, b has the code word of in this case 1 1, c has the code word 1 0 1 1 0 1 this has the code word a is equal to 0. So, in this fashion if we design a code book then we can find that none of the codes are prefix of any other code even if you take this the code word 0 0 is not a prefix 0 is is not there in any other code 1 1 is not present as a prefix there is no 1 1 sequence over here even 1 0 1 is not a prefix over here. So, if you

are going to design codes in this fashion then you can make a variable length codes which are not prefix of any other code. But however, there are certain more conditions on this binary code tree which we are going to see, so that we do not have an unnecessarily large code word and that would be able to help us in minimizing the number of bits that we provide on an average each symbol in a probabilistic manner. So, that we reduce the average number of bits required to represent a source symbol through variable length code.

So, we are going to stop this particular lecture at this point and will move further in the next lecture trying to see how one can calculate the minimum number of bits required to represent a code word for a given particular source.

Thank you.