Theory of Computation

Prof. Kamala Krithivasan

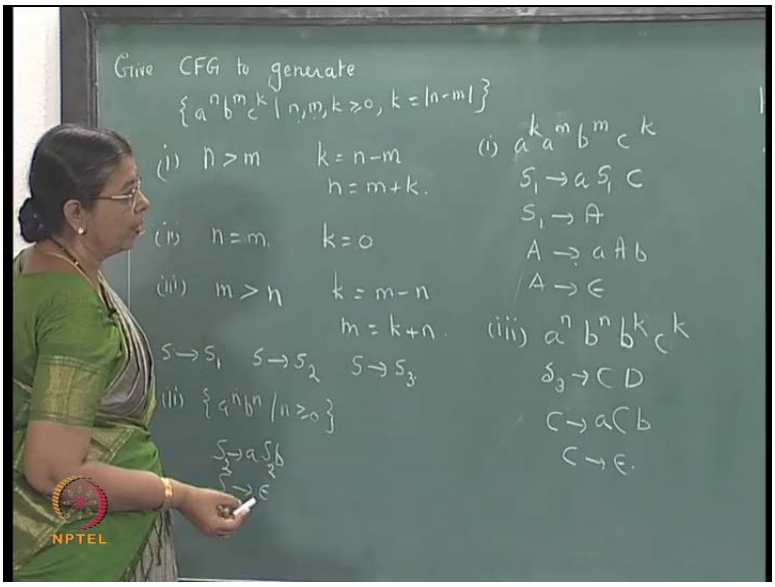
Department of Computer Science and Engineering

Indian Institute of Technology, Madras

Lecture No. # 15

Problems and Solutions

(Refer Slide Time: 00:43)



Today we shall consider some problems on context free grammars, and also on finite state automata. Give a context free grammar to generate a power n b power m c power k. n, m, k greater than or equal to 0 and k is equal to mod of n minus m. So, first of all you must realize that in a context free grammar like s goes to a S b, and S goes to a b there is a way to keep the number of a (s) equal to the number of b (s), when you apply this rule one a will be generated this side, one b will be generated that side. So, ultimately you get a power n b power n.

That is not possible with type 3 or linear grammar when you have rules of this form there is no way to keep count on the number of a (s) and the number of b (s). So, with type 3 such a to generate such a language having some equal number of a (s), and equal number of b (s) or number of a (s) twice the number of b (s) is not possible, you are require a context free grammar for that. And it is not necessity you necessary to go to type 1 rules

where on the left hand side you may have more than one symbol, this is also not necessary. For such things to generate equal number this side and equal number that side. You require only a linear grammar, which is context free you need not have to go to a higher level also.

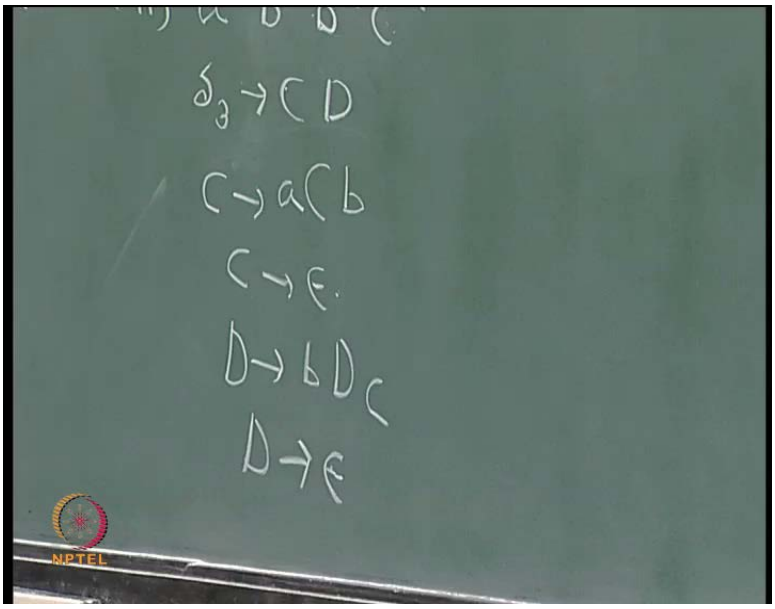
Let us consider this one. Now, the possibility is there are you can split it into two possibilities or three possibilities does not matter. So, n greater than m I will split into 3 if n is greater than m, k is equal to n minus m or n is equal to m plus K this is one possibility. The second possibility is n is equal to m if n is equal to m k is equal to 0. Third instead of considering like that you could have also split it into two like n greater than or equal to m, m greater than or equal to n does not matter that way also it is possible. Then another thing is m greater than n in that case k will be equal to m minus n and m will be equal to k plus n.

Now, for each one of this you can give a context free grammar and the union of them will be the grammar which you require. So, you can have s goes to S 1 where s is the start symbol and S 1 will generate this portion then s goes to S 2. S 2 will generate this portion s goes to S 3 and S 3 will generate this portion. So, second one is very easy when n is equal to m, k is equal to 0. So, what do you get you get two gives rise to the portion a power n, b power n, n greater than or equal to 0 this gives rise to this and this you can very easily generate with s goes to a S b s goes to epsilon the first portion n is equal to m plus k. So, the strings will be of the form a power n, b power m, c power k, but n is m plus k. So, you can write it as actually a power m a power k and so on, or rather a power k, a power m.

So, S 1 should generate that starting from s you can go to S 1 or S 2 or S 3. So, S 2 generates this is S 2 second portion is generated by S 2. So, S 2 goes to a S 2 b S 2 goes to epsilon. Now, this one should be generated by S 1 first you can generate equal number of a (s) and c (s) by having the rule s 1 goes to a S 1 c, after generating equal number of a (s) and c (s) . You may go to a non terminal A of course, there is you can use unit productions only in certain cases you have to avoid them. Now, we are not asked to avoid them. So, you can use the as many as possible S 1 goes to A after generating equal number of a (s) and c (s) it goes to non terminal A then from this you can generate equal number of a (s) and b (s). So that will be a A b, A goes to epsilon.

So, this portion will generate the first of this and now, we have to generate the third one third one s 3 has to generate that m is equal to n plus k. So, you can write it as a power n, b power m, c power k, but this m you can split so, it will be b power n, b power k, c power k m is equal to k plus n so, you can split it like this b. So, you have equal number of a (s) and b (s) than equal number of b (s) and c (s) . So, this has to be generated by S 3 so, S 3 you can make it go to C D some non terminals C and D is a non terminal and D is also a non terminal, then C can generate a power n b power n, c goes to a C b, c goes to epsilon, because n m k are greater than or equal to 0. You can have epsilon in the language and to have the possibility where this is 0 and this is also 0.

(Refer Slide Time: 07:31)

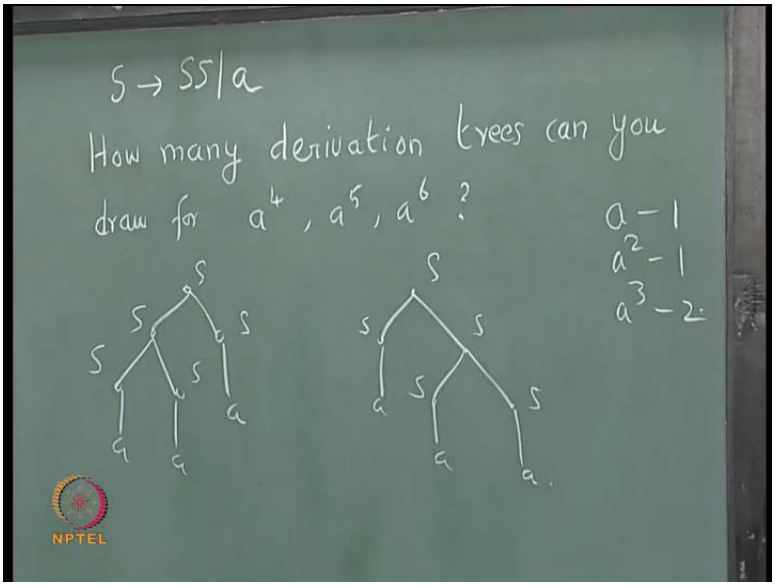


You should have this then from D you can generate equal number of b (s) and d's. So that will be generated by b D c, D goes to epsilon instead of splitting into 3 portions like this. You could have split it into n greater than or equal to m, m greater than or equal to n. In that case some of them will be generated in both like a power n, b power n will be generated in both portions that does not matter. So, this is the solution for this the main thing to understand is when you want to generate some thing in this side and something in this side use a linear row.

So, that keeps count of something occurring on the left hand side and something

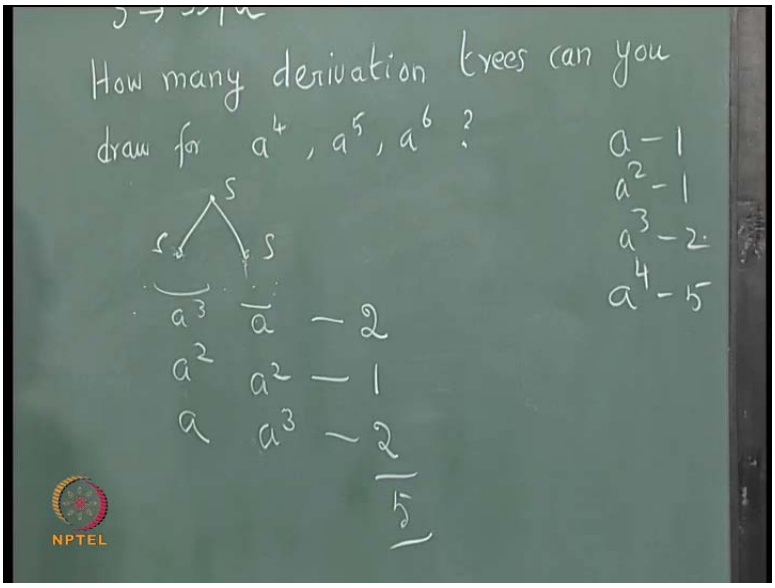
occurring on the right hand side. And that is possible in type two grammars linear in fact linear grammars it is not possible with type 3 and it is not necessary to go to type. When you have 3 of them to make them 3 or 4 of them equal like a power n, b power n, c power n or a power n, b power n, c power n, d power n something like that. When you want to keep equal number of more 3 or more things then you require type one rules.

(Refer Slide Time: 09:07)



Now, how many derivations can you draw for a power 4, a power 5 and a power 6 in this grammar? We have considered this grammar earlier and for a cubed there were 2 derivation trees for a cubed there were 2 derivation trees for a squared for a there is only 1 derivation tree for a squared 1 a cubed 2. Now, how many derivation trees do we have for a power 4? (No audio from 09:41 to 09:47)

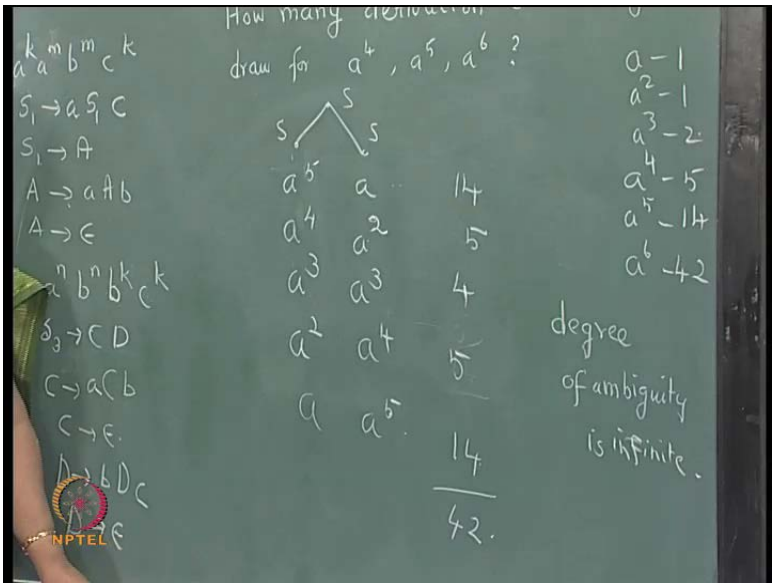
(Refer Slide Time: 09:50)



Now, the first step in the derivation tree will be like this. Now, in this from this s you can derived a cubed and from this s you can derive a, this is one possibility and from this s you can derive a squared and from this s you can derive a squared that is another possibility. From this s you can derive a and from this s you can derive a cubed these are the 3 possibilities. Now if, we use this possibility this a cubed itself can be generated into two ways. So, when you want to generate a cubed from this s and a from this s of course, this can be generated only in one way so, this gives rise to 2 derivation trees.

And when you want to generate 2 a squared from this s and 2 a squared from this. I mean I am sorry when you want to generate a squared from this and a squared from this there is only one possibility so that will give rise to 1 derivation tree. The same thing like this when you want to generate 1 a from here and 3 a (s) from this s this can be done in two ways. So, this will give rise to 2 derivation trees. So, totally for a power 4 you will have 5 derivation trees. Now, what about in a similar manner you have to calculate for a power 5 and a power 6.

(Refer Slide Time: 11:29)



Now, if you take a power 5 the first step of the derivation tree will be like this. Now, from this s you can derive a power 4 and from this s you can derive a that is one possibility. Then from this s you can derive a cubed and from this you can derived a squared from this s you can derive a squared and from this s you can derived a cubed and from this s you can derive a and from this s you can derive a power 4. Now, how many possibilities this will give rise to?

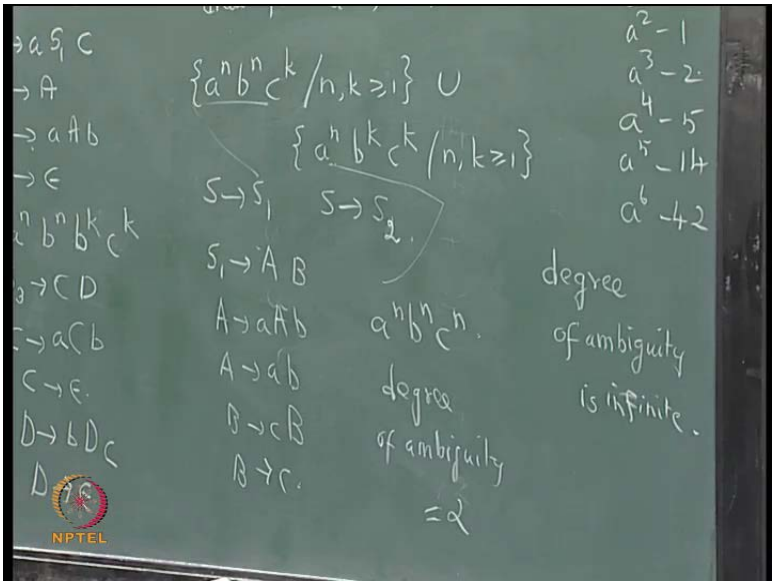
A power 4 itself can be derived in 5 different ways you can have this sub tree itself can be drawn in 5 different ways. So, this will give rise to 5. a cubed can be drawn for a cube we can draw sub tree in two possible ways a squared only one way. So, this will give rise to 2 different trees similarly, this will give rise to 2 different trees and this will give rise to this can be derived in only one way. But the sub tree here deriving a power 4 you can have 5 possibilities because we have already seen that. So, totally 14 different derivation trees you can have for a power 5. (No audio from 13:05 to 13:11)

Now, coming to a power 6 the same argument we can split it as a power 6 a, a power 5 s I am sorry a power 5 a, a power 4 a squared, a cubed a cubed, a squared a power 4, a a power 5. The number of possibilities for this will be 14 for this also it is 14 for this a power 4 can be derived in five ways, a square only one way. So, 5 this cubed can be derived in two different ways this a cubed can also be derived in two different ways. So, 2 into 2, 4 you can have. So, this will add up to 14 plus 14, 28, 38, 42 (No audio from

14:13 to 14:20) Now, for a power 7 obviously the number of trees will be very large a power 8 still more and so on.

As the length of the string increases the number of derivation trees keeps on increasing. So, in this case you say that the degree of ambiguity (No audio from 14:38 to 14:47) is infinite. In this example the degree of ambiguity is infinite keeps on increasing as the length of the string increases.

(Refer Slide Time: 15:07)



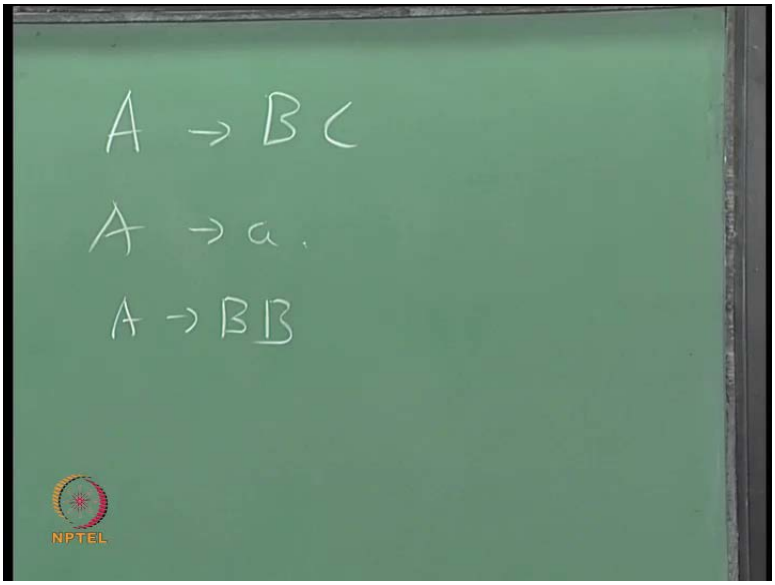
Now, we will consider another language a power n, b power n, c power k. n k greater than or equal to 1 union a power n, b power k, c power k, n k greater than or equal to 1. We have seen that this is an inherently ambiguous language any grammar generating this will be ambiguous. The proof is I did not give you the proof the proof is very lengthy but this has been proved to be an inherently ambiguous that is that is any grammar generating this will be ambiguous. Now, you can have a grammar like this s goes to s 1, s goes to S 1 S 2 sorry, S 1 will generate this portion, S 2 will generate this portion. Now, each one of them this portion S 1 will generate this s 2 will generate this.

Each one of them can be generated unambiguously we can generate each one of them in an unambiguous manner that is s 1 can generate this in unambiguously S 2 can generate this unambiguously it is not very difficult to see for s 1. I will write the grammar s 1 goes to say A B, A goes to a A B, A goes to a b, B goes to c B, B goes to c in a similar manner we can write from S 2 for generating this. So, this is unambiguous but the ambiguity

arises because something like a power n, b power n, c power n a string of the form a power n, b power n, c power n will have a derivation from s 1 and another derivation from s 2.

So, it will have 2 derivations. So, any string will have either 1 derivation or 2 derivations it will not have more than that. So what is the degree of ambiguity here degree of ambiguity for this grammar degree of ambiguity can be finite or infinite. Now, let us consider this problem show that every C F L without epsilon can be generated by a grammar in C N F with additional restrictions. If A goes to B C is a rule then B is not equal to C. If A goes to B C is a rule, X goes to Y B cannot be a rule similarly, Z goes to C W cannot be a rule. That is a Chomsky a context free grammar is in Chomsky normal form if the rules are of this form or of this form.

(Refer Slide Time: 18:21)



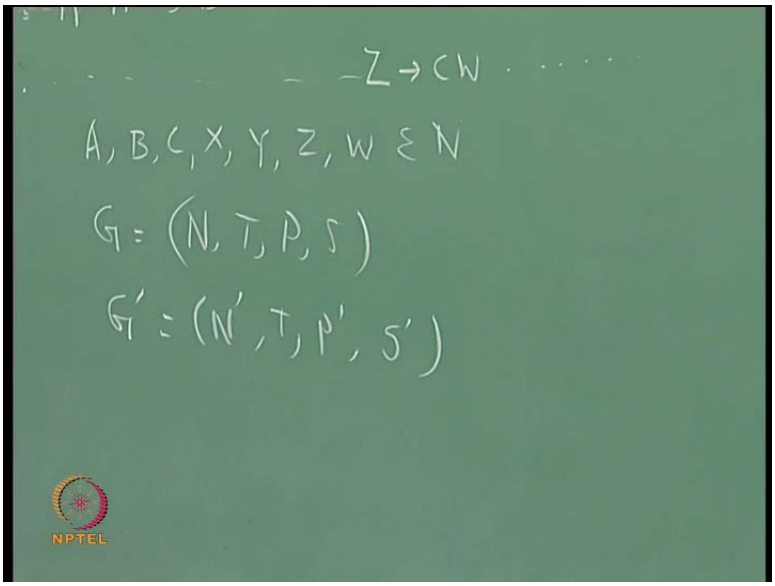
That is on the left hand side for any context free grammar you have a single non terminal on the right hand side you have two non terminals or a single terminal. What this says is if something occurs as the first symbol on the right hand side it cannot occur on the it cannot occur as a second symbol on the right hand side for any rule. And if something occurs as a second symbol on the right hand side it cannot occur as the first symbol on the right hand side. That is why you cannot have this equal to this. You cannot have a rule of the form A goes to B B. Now, of course, in a particular grammar given a particular grammar you look at the rules keep on replacing one of them if go if you have

a rule of the form a goes to B B replace one of the b (s) as B 1 and then proceed and so on. For a particular grammar it is easy you mean there may be a say 7 rules or 8 rules you can do that one by one. You can keep on eliminating ultimately we will be able to do there is no problem about that.

But you have to prove that so, in Greibach normal form we did that when we wanted to convert a context free grammar into Greibach normal form. We used tool m mass we kept on replacing something and then again did some substitution and so on. In a similar manner we can do that but the problem is in Greibach normal form when we did that if you remember what we have done there were 5 steps in step 3 you went from A 1, A 2, A the non terminals you arranged them as A 1, A 2, A n and then in step 3 you went from A 1, A 2 upto A n and in step 4 you went from A n, A n minus 1 upto A 1.

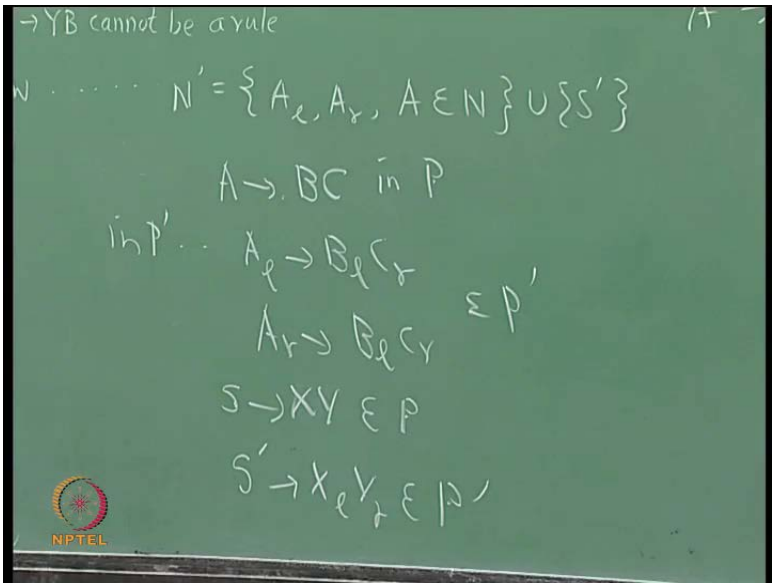
So, after the step 3 all the A n rules will be in Greibach normal form and when you re substitute in step 4 all the a rules will be become rules in Greibach normal form. A similar manner unless you do some arrangement and all that it is difficult to say that the procedure will turn because you may start with a 50 rules or something. When keep on substituting you may end up with more and more rules and ultimately you have to prove that some where it has to stop rather than in this case there is a simple way to prove it.

(Refer Slide Time: 21:08)



What you do is this is a grammar N, T, P, S now, construct a grammar G dash with N dash T, P dash a new symbol S dash new start symbol S dash.

(Refer Slide Time: 21:32)



Now, N dash will be set of non terminals will be of this form

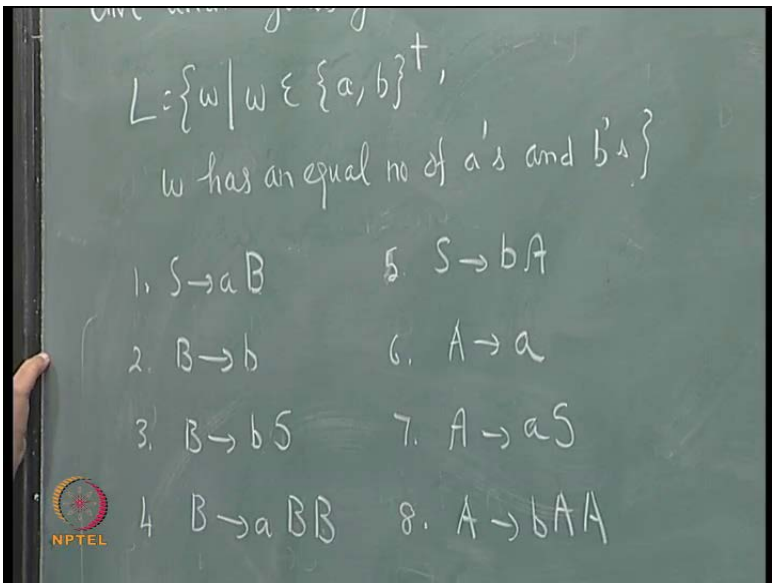
(No audio from 21:36 to 21:53)

For each non terminal A you have two non terminals A l and A r, A left A right and one more start symbol. (No audio from 22:07 to 22:13) So, when you have a rule of the form A goes to B C in P, in P dash you will have A l goes to B l C r, A r goes to B l C r (No audio from 22:40 to 22:46) you have two rules like that. So, this will keep track of the sort of a left and see this way wherever B l occurs there I will be a rule for B l whenever there is A B on the left hand side there will be a rule for the B l and you can apply that. And whenever there is a rule for A C there will be a rule with C r and you can apply it for that.

But while splitting like this B r will occur only as the second symbol, B l will occur only as the first symbol. So, there is no question of the first symbol occurring as a second symbol or the second symbol occurring as a first symbol. And initially if s goes to some A B or x goes to I will use some other symbol X Y belongs to P start symbol going into something instead. You also for this you will have s l goes to X l Y r, S r goes to X l Y r two rules will be there for that apart from that you also have s dash goes to X l Y r belongs to P dash. (No audio from 24:01 to 24:10) This will make sure that the rules whatever symbol is occurring or as a first symbol on the left hand side will not occur as the second symbol and whatever is appearing as the second symbol will not occur as the

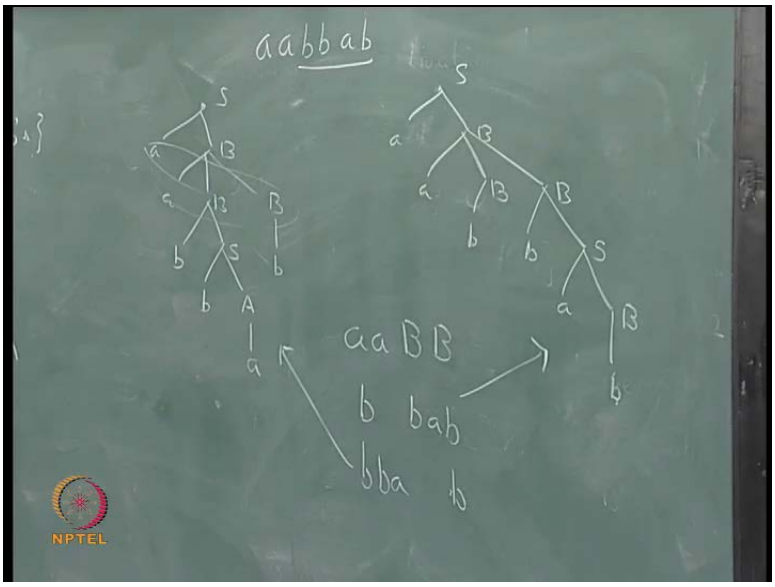
first symbol that is all.

(Refer Slide Time: 24:47)



So, let us come to some more problems give unambiguous grammar for w, w is a string of a (s) and b (s) and w has an equal number of a (s) and b (s). This example we have considered several times the grammar which we considered was this S goes to a B, B goes to b, B goes to b S, B goes to a B B and S goes to b A not 2, 5, 6 A goes to a, A goes to a S, 8 A goes to b A A. We also proved using induction that this generated equal number of a (s) and b (s) and nothing else.

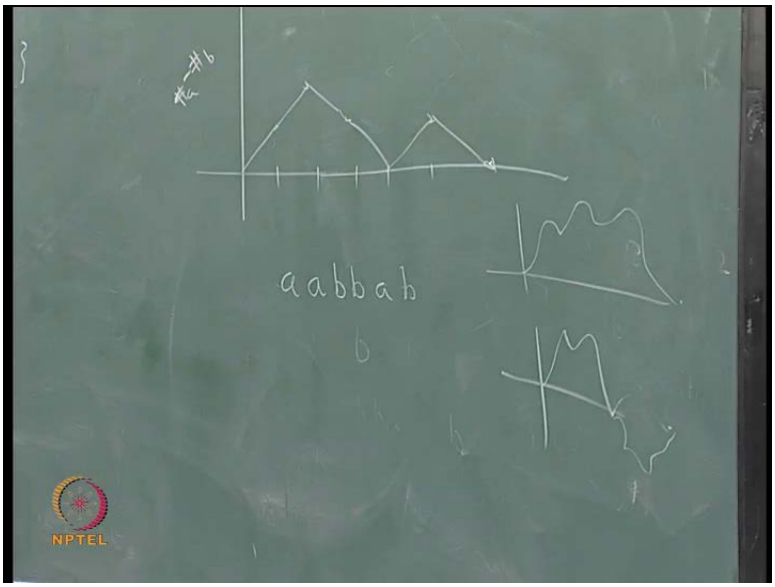
(Refer Slide Time: 26:11)



The induction hypothesis we proved that from s if a string is derived if and only if it has got equal number of a (s) and b (s). From a if you derive a string it will have one more a than it has b (s) from b if you derive a string it will have one more B than it has a (s), And consider this is the ambiguous because if you consider a a b b a b the 2 derivation trees possible. S goes to a B then a B B then this one you can have as b goes to B S, S goes to a B correct no S goes to b A, A goes to a, B goes to b this is 1 derivation tree.

Another 1 will be S goes to a B, a B B this will go to b this is goes to b s a B b. So, a a b b a b will be generated in two different ways. So, this is an ambiguous grammar we want to give an unambiguous grammar for the language. The reason for this is at this stage you have the sentential form B B and from these 2 B B's you have to derive this from a B you can derive one string which has got one more B it has a (s) is it not. So, this you can either split it as b b a b which will give raise to this derivation tree or you can split it as b b a b, which will give raise to this derivation tree that is why the ambiguity comes and in the proof while proving this? (No audio from 28:01 to 28:08)

(Refer Slide Time: 28:09)

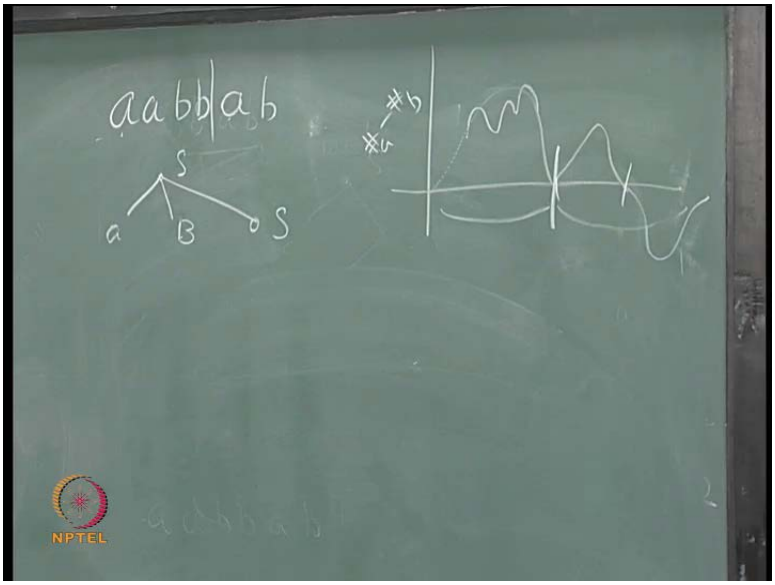


We proved here by induction but also (No audio from 28:10 to 28:22) for another example if you denote number of a (s) minus number of b (s) and the length of the string here. And take any string suppose it is a a the same string a b b a b the number a (s) minus b (s). As you go along this string will be a a b b a b because the number of b (s) is equal to the number of a (s) it will start at the origin and it will touch the x axis in the

end. In between there are several possibilities the diagram may just be like this or it can be like this or you may have something and then something or this portion even may go below this to have unambiguity you have to change.

So, there is no hard and fast rule how you find the unambiguous grammar? You have to just think and see how it can be done that is all there is you because it is undesirable to find out whether a given grammar is ambiguous or not. And depending upon the structure of the sentences, depending upon the language you are given you having to look into that and see how it can be generated unambiguously is it possible or not?

(Refer Slide Time: 30:21)

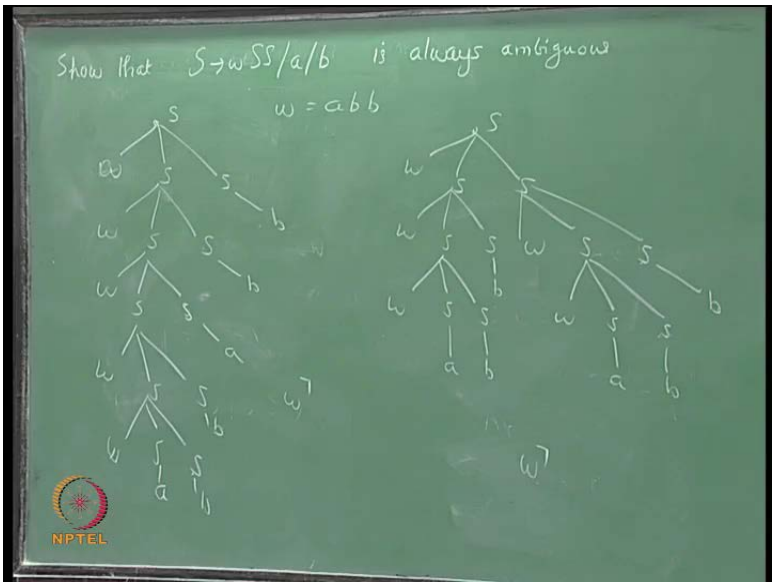


Here the first rule if you change s goes to a B s and s goes to b A S. Let us take the same string a a b b a b start from s. You apply the rule a B s actually the idea is like this. Suppose, this number of a minus number of b is this then if the graph is like this. It may have ups and downs like this. The first time it touches the x axis this portion will have equal number of a (s) and b (s) and because of that this portion also will have equal number of a (s) and b (s). So, the idea is use the first 2 rules if it touches the x axis in in the middle the first portion you try to generate this and recursively it may again the graph may go like this then.

So, this portion will be generated by the this s again you can repeatedly apply. (No audio from 31:33 to 31:45) If, you do that it has to be done since a a b b the first time this portion has equal number of a (s) and b (s) and the other portion a b also has equal

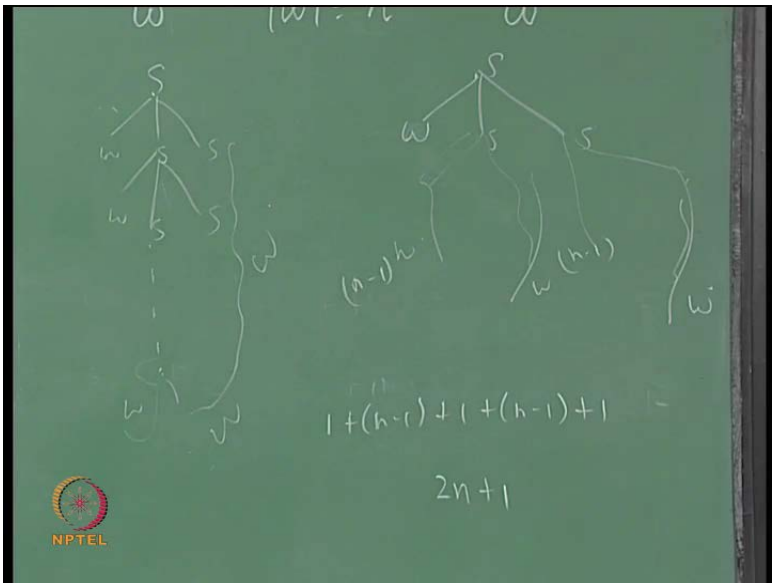
number of s. So, from this s you should be able to derive this a b from this portion you should be able to derive this. (No audio from 32:00 to 32:13) If, you do this any string can be derived only in one possible way. Now, let us it is a slightly different from the example of the example you can show that this is always ambiguous.

(Refer Slide Time: 32:35)



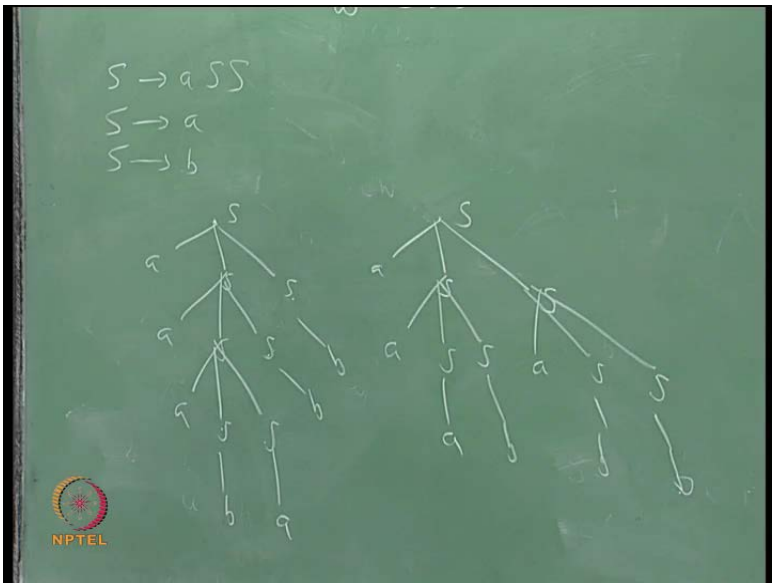
How do you show that s a suppose, I will take an example rather than suppose w is a b b some string. The rules are s goes to w s s, s goes to a, s goes to b. So, you can always have something which can be derived in two different ways in this case how is that possible? a I will put w w s s and then s can go to a or b again you can have another w s s. Now, 3 symbols can be derived from that. (No audio from 33:23 to 33:29) Another way of doing it will be s goes to w s s, w s s, w s s, w s s, w s s. I have taken w to be a b b so, here you can make it go to a b b, a b b. Similar manner if you do w s s make it go to a b b, w s s, w s s, a b b this will go to a, a b b. How many times you get a b b 1, 2, 3, 4, 5, 6, 7 times you get this is w power 7. How many times you get w here 1, 2, 3, 4, 5, 6, 7 so, this also generates w power 7.

(Refer Slide Time: 35:07)



In general if you take from w then consider length of w to be n if you consider the length of w to be n take w power 2 n plus 1. In one way you can derive like s goes to w s s w s s and so on. And finally, these s's should contribute to 2 w's here you will generate n minus 1, 2 n minus 1. w's and this will generate 2 n symbols that is 2 w's you can generate them in another manner from s goes to w s s. you can generate again from this n symbols and n minus 1 w's. So, 1 w will be n minus 1 w's here and here again n minus 1 w's and with the s's 1 more w. So, totally the number of w's will be generated is 1 plus n minus 1 plus 1 plus n minus 1 plus 1 this is 2 n plus 1 the string generated will be of the form w power 2 n plus 1. If you take such a string it is always possible to derive in 2 different ways so such a grammar will always be ambiguous whatever may be w, but if the length is 2 or more you take w power 2 n plus 1 if it is single letter for example.

(Refer Slide Time: 37:13)

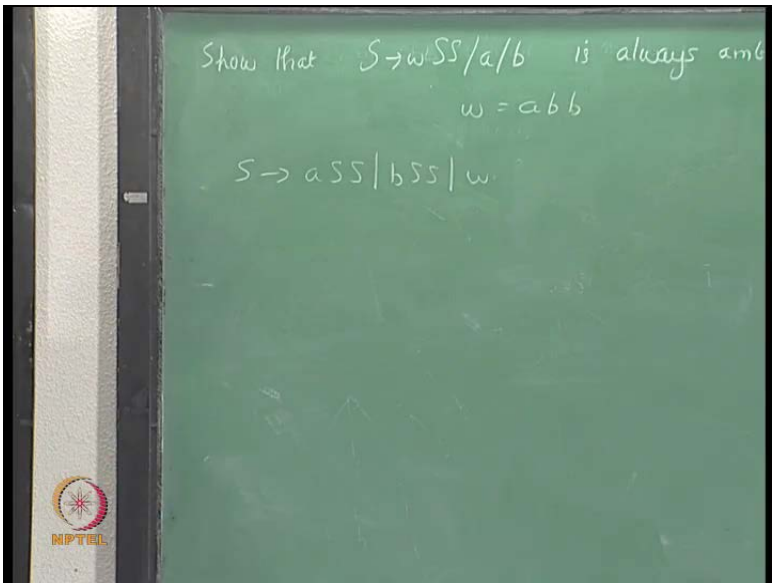


If you take a single letter (No audio from 37:04 to 37:10) s goes to a S S, s goes to a, s goes to b here again you will have ambiguity but the string you have to be careful a s s, a s s then whatever you generate another string s a s s. Now, to generate this you will have another a s s in two different ways you have to show a s s then these symbols can go to different symbols and so on so. Minimum length 7 you have to take. Now, again a s s you have to use this rule.

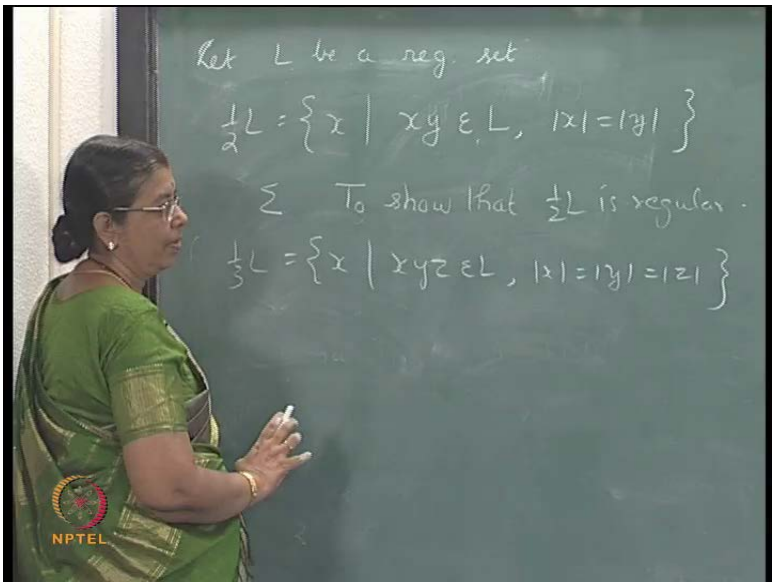
So, this has to go to a and this has to go to a these 3 you can make it go to a or b suppose I make it go to b b b b. So, I had the minimum length I have to consider is length 7. (No audio from 38:32 to 38:42) So that it is not a power 2 n plus 1 you are not considering a power 5. (No audio from 38:52 to 39:06) In a similar manner same sort of argument you can use for proving this s goes to a s s, b s s this will also be always ambiguous.

(No audio from 38:50 to 39:07)

(Refer Slide Time: 39:18)



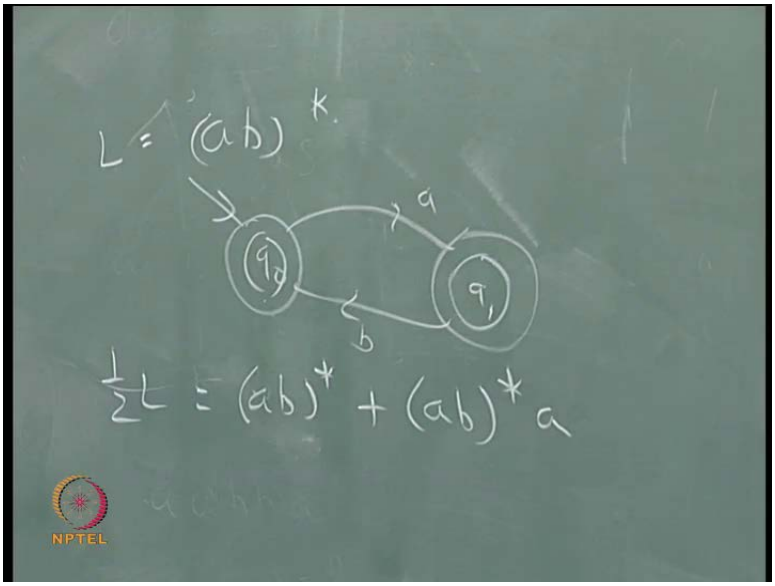
(Refer Slide Time: 39:33)



Next we will consider this problem. Let L be a regular set then half of L is defined as x, x y belongs to L length of x is equal to length of y. Of course; you can consider an alphabet sigma over an alphabet sigma. What it really means is from L you take all strings of even length take all strings of even length and in each string you consider the middle one I am the first half only then such strings will form a regular set half of L is you have to show that half L is regular. How do you prove this in fact like that you can prove 1 third l is regular and so on?

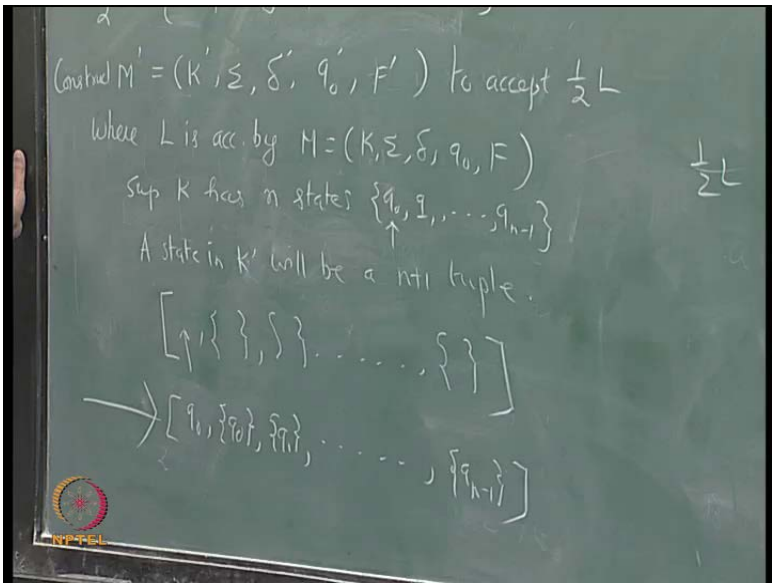
When you can define like that what will be 1 third L in that case you can also define 1 third L, x, x y z belongs to L length of x is equal to length of y is equal to length of z. That is from L (No audio from 41:04 to 41:13) you take strings of length 3. And you just pick up strings of which are multiples of length 3 I mean the length of the strings is a multiple of 3 then from each of such strings you take the first one third portion alone.

(Refer Slide Time: 41:39)



Such strings will form a regular set it is a simple example look at a b star half of L accepted by this machine. I am not marking that dead state a b, a b, a b. So, if you take strings of even length and take the first half what will you get if you take a b. You will get only a, if you get a b, a b you will get a b. If you get a b, a b, a b you will get a b a. If you get a b, a b, a b, a b 4 times you will get a b, a b twice. So, half this is L half of L will be a b star plus I mean can be represented by this regular expression epsilon will also be there. You can have a b, you can have epsilon, you can have a, you can have a b, you can have a b a, you can have a b a b, you can have a b a b a. This is the regular set to show us an example and this will hold for every regular set and how can you prove that half of l is regular.

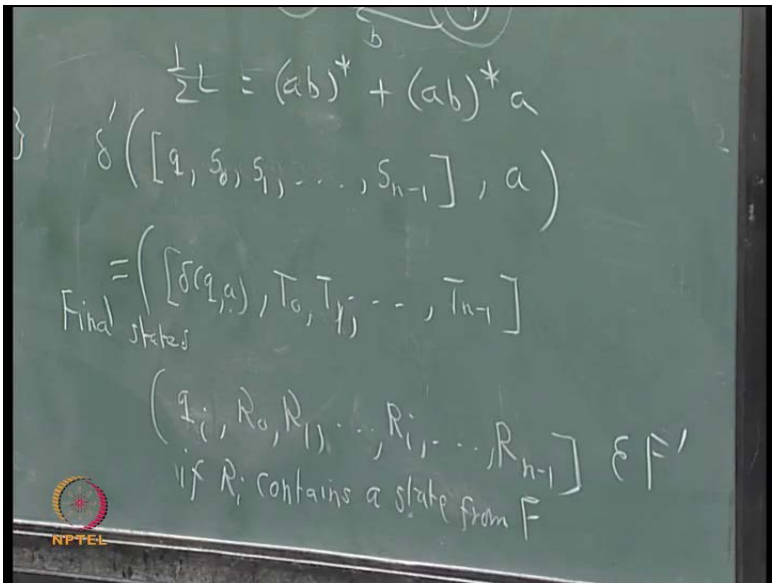
(Refer Slide Time: 43:23)



So, we have to show that half L is regular how do you prove that? this is what you understood you construct m dash K dash sigma delta dash q naught dash F dash construct a machine m dash is equal to this to accept half of L. (No audio from 43:47 to 43:56) Where L is accepted by M is equal to K sigma delta q naught F. Now, I will not write down step by step I will give the construction. Generally a construction and then illustrative examples the K dash the set if states K dash they will be of the forms they will be suppose K has n states it has got n states q naught q 1 q n minus 1 you have to some order where this is the initial state any of them can be final state.

Suppose k has n states then a state in K dash will be a n plus 1 tuple you take the state in K dash as a n plus 1 tuple. The first one will be a state this will belong to K the second, third up to the other one they will all be sets subsets of one. The first one will be a single state the remaining n are subsets of k and the initial state will be (No audio from 46:01 to 46:10) q naught q naught q 1 they are all subsets of single terms this will be the initial state the initial state will be like this.

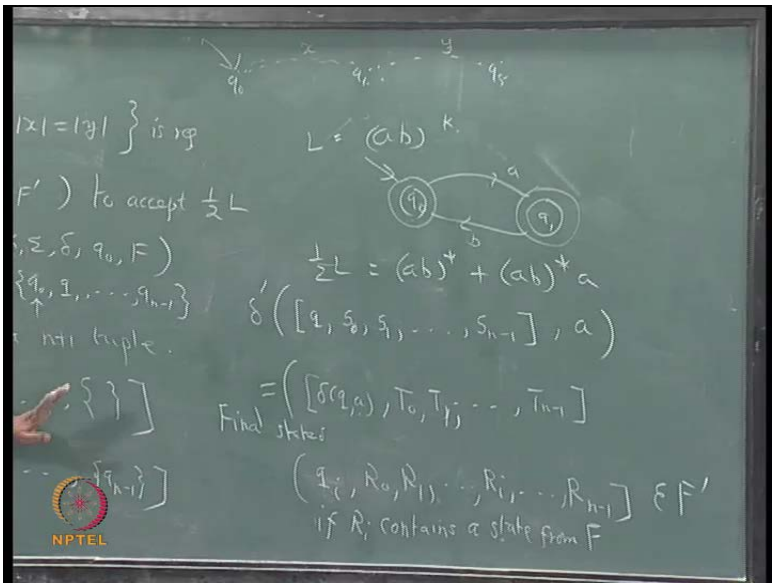
(Refer Slide Time: 46:38)



How do you define the mappings? I have a state is of the form q, s 1, s. I will put is as s naught rather s naught s 1 s n minus 1 the state will be of this form. Where q is a single state s naught s 1 s 2 s n minus 1 are subsets of states from K. And how do you define delta dash this a where a is a symbol this you define as the first one will be delta of q a from q if you read a you go to say state p say which is represented by delta of q a you go to that state. And here you will go to say t naught t 1 t n minus they will be subsets how these subsets are defined. From any one of the states in s naught, if you get a symbol any symbol not necessarily a by a single move single transition what are the states to which you can go from any one of them by reading one symbol?

It can be a b c anything what are the sets of states to which you can go that is given by t naught and s 1 represents a set from s 1 by dash by one move that is by reading one symbol what are the set of states to which you can go that is t 1. So, you can define the mappings like this and what are the final states? Final states are of the form sum q i then, say R naught R 1 there are all subsets R i R n minus 1. If R i contains a state from F this will be a final state this belongs to F dash or if this a final state if R i contains a state from F. What does that mean starting from this initial state this is the initial state so, starting from this?

(Refer Slide Time: 49:49)



If, you reach a final state what does that mean after reading some portion you have reached q i and from q i after reading the same number of symbols you have reached the final state in the original automata. If, you reach a final state like that in this automaton starting from this initial state that means starting from q naught after reading the original machine starting from q naught after reading from x you have reached q q i. And because R i contains a final state starting from q i after reading and a string of equal length you have reached a final state.

So, from q i after reading some string you have reached a final state here. But the number of moves is the same what number of moves you are making from here to here each time you are also counting just a one transition. So, the number of symbols you have read from going to q f is the same as the number of symbols you have read q naught to q i. So, this automaton will accept half of l this is a general construction the same sort of an idea you can use for 1 third L, 1 forth L, 1 nth L and so on.