Theory of Computation

Prof. Kamala Krithivasan

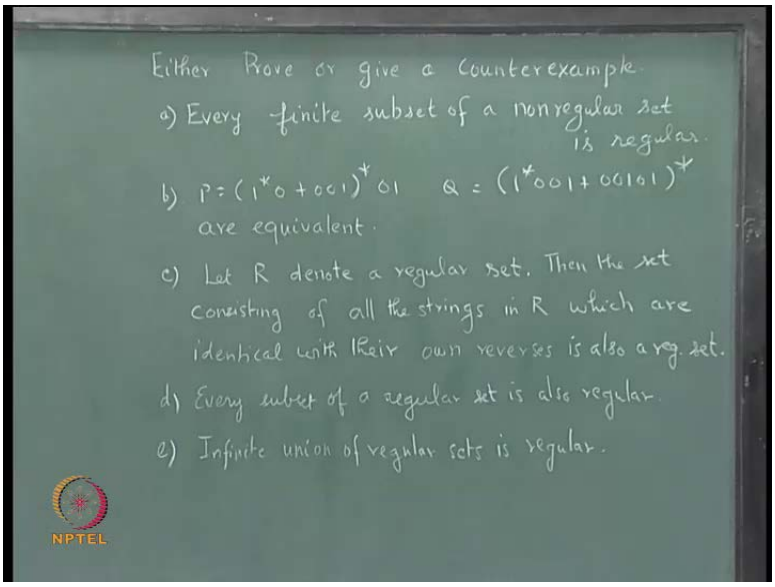
Department of Computer Science and Engineering

Indian Institute of Technology, Madras

Lecture No. # 24

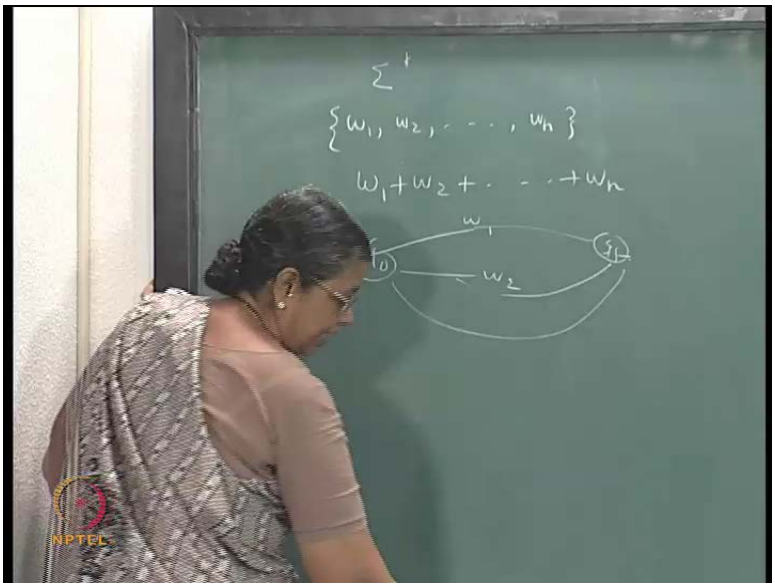
Problem and Solutions

(Refer Slide Time: 00:15)



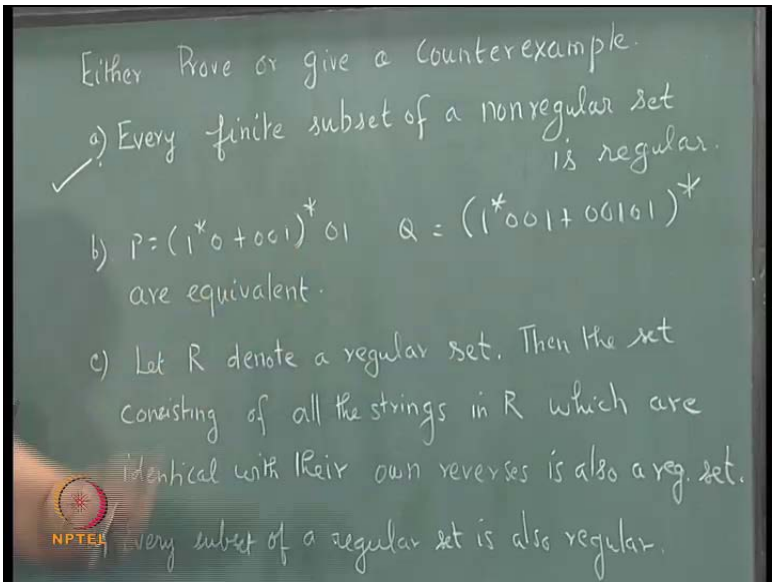
We shall consider some problems today, look at this question for the following statements either prove or give a counter example, either say they are true and if it is true, you have to prove, otherwise you must show it is not true. First statement is every finite subset of a non regular set is regular.

(Refer Slide Time: 00:47)



Look at the statement carefully it says every finite subset of a non regular set is regular. Any finite set let sigma be the alphabet, see when you say alphabets, alphabet is taken as a finite alphabet, alphabet is a finite set of symbols. Now, a finite set over sigma star will have some words like w 1, w 2, w n, it will have a few strings like this. It can be represented by the regular expression w 1 plus w 2 plus w n, this is the regular expression representing it and you can have non-deterministic automata very easily for that. Start from q 0, w 1, q f, w 2, q f, w3, q f, like that we can very easily have a non deterministic automaton for this.

(Refer Slide Time: 01:49)



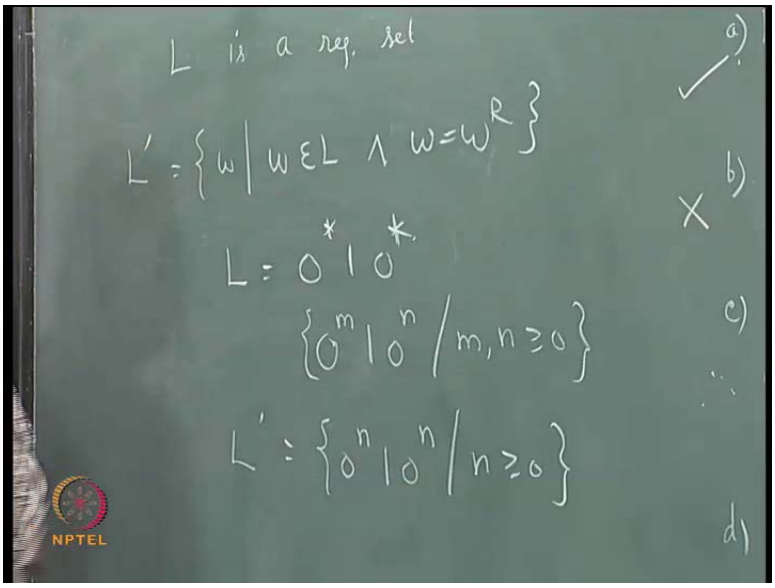
So, any finite set is regular any finite set is regular whether it is a subset of a non regular set or not it does not matter, any finite set is a regular. So, the first statement is true. What about second statement? P is equal to 1 star 0 plus 0 0 1 star 0 1, and Q is equal to 1 star 0 0 1 plus 0 0 1 0 1 star they are equivalent, obviously, they are not equivalent. Immediately you should be able to say that they are not equivalent, because this is something star, that means, epsilon will be there the empty word will be there, here any string should end with a o 1 this means that any string should end with a 0 1 that means, epsilon cannot be there. So, epsilon will belong to Q, but epsilon will not belong to p.

(Refer Slide Time: 03:11)

Any other string also we can take, but they are not equivalent, so these are not equivalent. Take the third one let R denote a regular set, then the set consisting of all the strings in R, which are identical with their own reverses is also a regular set. It is not that the statement you must understand in a clear manner if L is regular L R is a regular.

This is true and that is not the same statement as this, this statement means something different. What is L? L is a regular set, L R consists of all strings w R w belongs to L right and this once you have this state diagram for L, make the initial state the final state, make the final state as a initial state and reverses the arcs you will get the state diagram for L R that is (()).

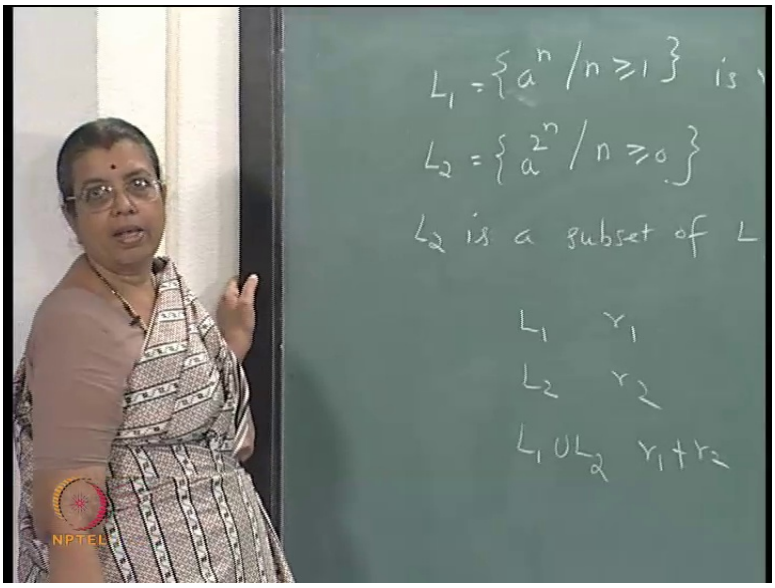
(Refer Slide Time: 04:02)



But this statement is different, what it says is? Let R denote a regular set, then the set consisting of all strings in R which are identical with their own reverses is also a regular set. So, if L is a regular set, by this you mean something like L dash equal to w w belongs to L and w equal to w R. If you define L dash in this manner, L dash consists of strings in L, which are equivalent to their reverses that is w should be equal to w R.

This is different from L R please remember, this is different from that and is this a regular set? This is not regular, need not be a regular set. So, take an example L is equal to take 0 star 1 0 star or as a language it is 0 power m 1 0 power n, m, n greater than or equal to 0, you can take greater than or equal to 1 also does not matter, then what will be L dash? L dash will be 0 power n 1 zero power n, n greater than or equal to 0. If a string has to be equal to its own reverse, then this is not regular right? You can use the Pumping Lemma to show that it is not regular.

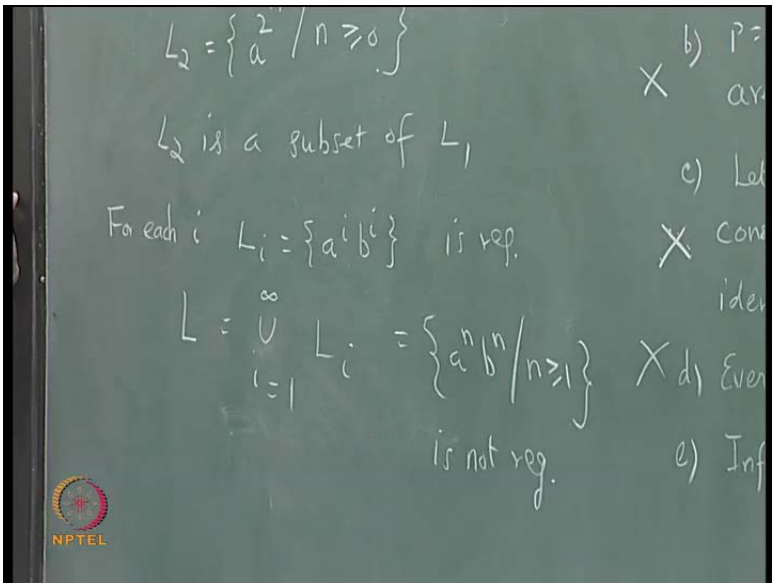
(Refer Slide Time: 06:23)



So, this is also not true. Every subset of a regular set is also regular, that is not correct because suppose I take a power n, n greater than or equal to one. This is regular, if I take a power 2 power n, n greater than or equal to 0 or something like that, this is a sub set of this, I call this as L 1, L 2. L 2 is a subset of L 1, but that it is not regular isn't it. L 2 is not regular. So, this is how do you prove L two is not regular? You can use the Pumping Lemma, you can use Myhill–Nerode theorem or you can show that the Parikh Mapping is not semi linear.

If it is semi linear then periodically it should jump, but here the jump is not periodic, 2 power n then 2 power n plus 1 jump increases. So, in any one of the 3 ways you can prove. Infinite union of regular sets is regular. A finite union of regular sets is regular, because from regular expression if L1 is represented by r 1, L 2 is represented by r 2, L 1 union L 2 will be represented by r 1 plus r 2. That we know we have seen while considering from regular expression to finite state automata. We have considered this, but the statement here is infinite union of regular sets is regular.

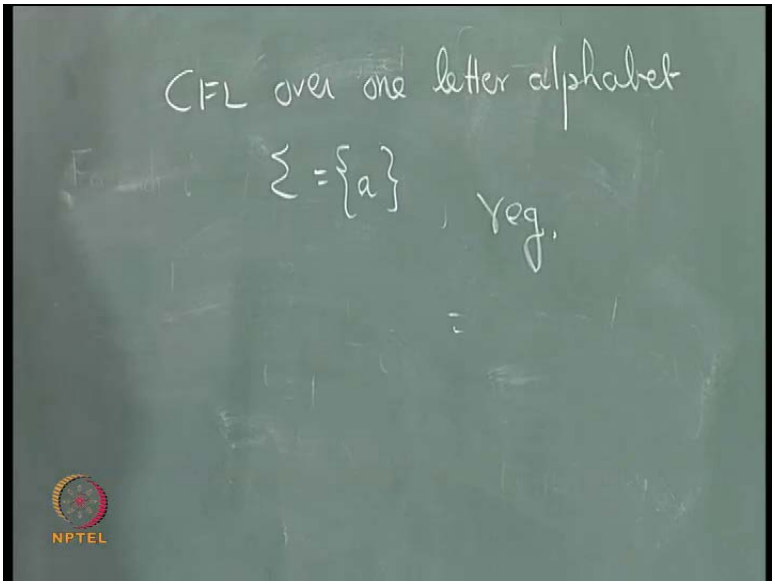
(Refer Slide Time: 08:30)



So, define L i as a single ten a power i, b power i, L i consists of only one string a power i b power i right, if it has only one string it is finite. So, it is regular isn't it. L i has only one string a power i b power i right. So, that is regular L is equal to union i greater than or equal to one L i this is an infinite union, i is equal to one infinity, i is equal one to infinity L i, for each i for each i L i is just a single term and so that is regular.

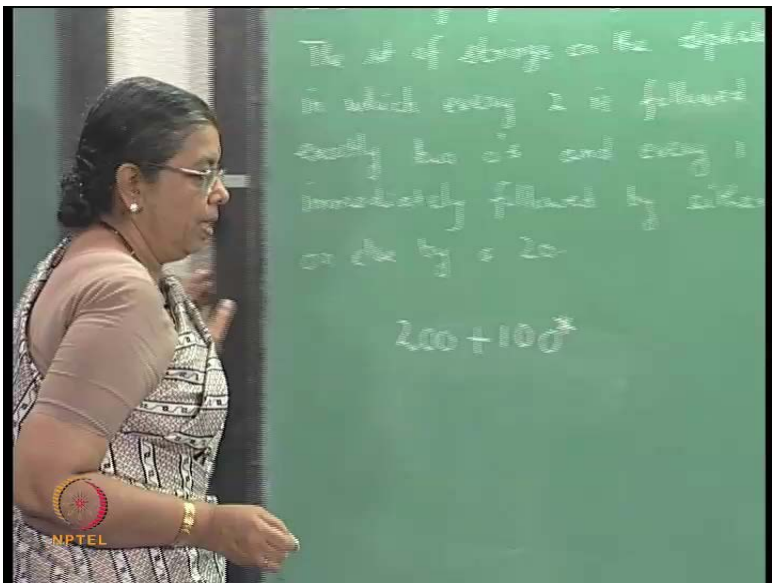
If you will write L is equal to union i is equal to one to infinity it is an infinite union of L i and what is this? That is a power n b power n, n greater than or equal to one that is not regular not regular, again you can use Pumping Lemma or Myhill–Nerode theorem. You cannot use Parikh mapping, Parikh mapping for this is semi linear and it is a context free language, here only a context free language a Parikh mapping is semi linear and here it is not even context-free. So, you can use Parikh mapping for that.

(Refer Slide Time: 10:40)



You can consider the other, this is just one example I am giving one example, you can give any other example one slight deviation. So, if you consider CFL over one letter alphabet, you consider context free language over a one letter alphabet. What can you say about that? That is only one alphabet a, sigma is a that will be regular. Every context free language over a one letter alphabet will be regular because if you take the Parikh mapping ultimately it will be a periodic set of integers and so you can write a regular grammar for that. Work out the details for this, try to work out the complete proof of the statement, every context-free language over one letter alphabet is regular.

(Refer Slide Time: 11:56)



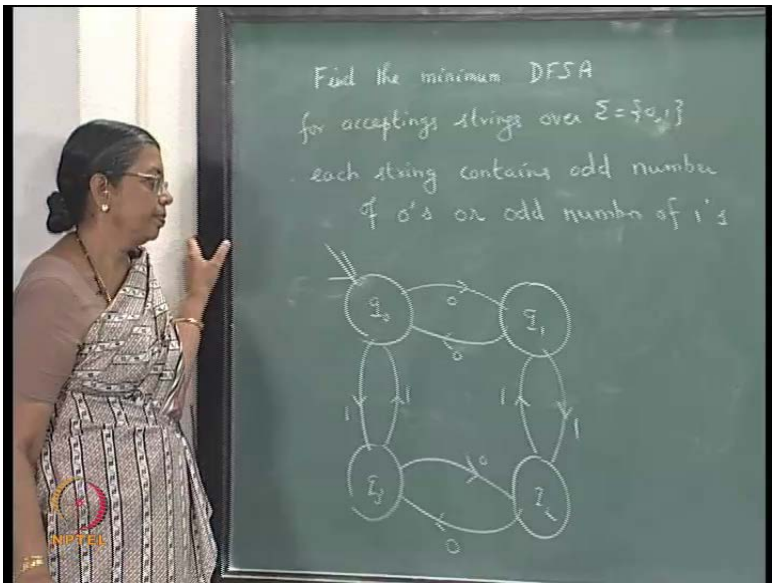
Next we shall consider this question, find a regular expression for the set strings on the alphabet 0, 1, 2 in which every 2 is followed by exactly two 0's and every 1 is immediately followed by either a 0 or else by a two 0's. So, if you have a two, it should be followed by two 0's and you cannot have more 0's I cannot have like this and if a 1 occurs it can be followed by a 0 and more 0's also.

There is no restriction here, every 1 is immediately followed by a 0, but it can be followed by more 0's also no problem or 1 will be followed by a 2 0. If 1 is followed by a 2 0, 2 should be followed by exactly 2 0's. So, you can have only like this right? So, what sort of sub strings you can have? You can have 200, 200, 200, repeatedly occurring that is possible, but if 1 occurs it can be followed by many 0's or a 2 0.

If it is followed by many 0 there should be 1 0 at least it can be 0's. So, 1 0 0 star or 1 should be followed by a 2 0, but 2 should be followed by exactly 2 0's, afterwards either a 1 or a 2 should occur. So, these sub patterns can repeat in the string, any number of them can be occurring like this and in the beginning. So, this will represent strings, where a sub strings are of the form 2 0 0 or 1 followed by many 0's or 1, 2 followed by 2 0's repeated.

But there is no necessity that the string should begin with a 1 or a 2, it can begin with a 0 also and initially you can have many 0's proceedings this right. So, this is the regular expression. Actually in this case writing the regular expression is easier than drawing the finite state automaton and trying to write the equations and solve it. You can draw the finite state automaton for that, but you must draw the deterministic automaton. Then try to write the equation solve that is also possible, but in this particular example, this method is looking at the strings is more easier and you have to write the expression.

(Refer Slide Time: 15:28)

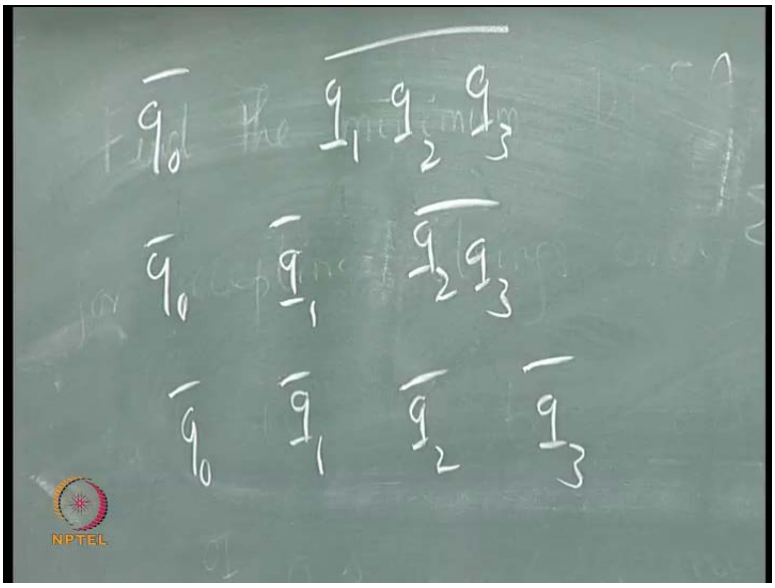


Now, let us consider one more example, find the minimum DFSA for accepting strings over 0, 1 with a condition that each string contains odd number of 0's or odd number of 1. This r is inclusive r, if it is exclusive r you will say it either odd number of 0 or odd number of 1, when you say odd number of 0 or odd number of 1 means it is inclusive r, the string also can contain odd number of 0 and odd number of 1's that is also allowed. What does that mean?

It means that the string cannot contain even number of 0 and even number of 1. The string cannot contain both even number of 0 and even number of 1's and that particular example we have already considered it is a example given in the book. q 0, q 1, q 2, q 3, 0 0 1 1 0 0 1 1. I have not marked the final states, actually the whole class of strings is divided into four equivalence classes by the Myhill–Nerode theorem. You reach this state if you have even number of 1's and even number of 0's.

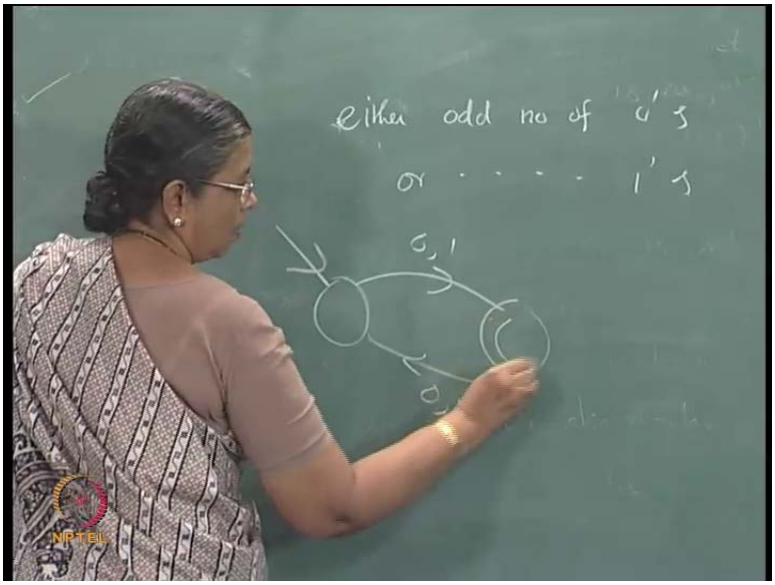
If you make this as the final state that means, the string has even number of 0's and what is the compliment? The compliment will be this, this if you have if you reach this state, that means, you have come across even 1's, even 0's, even 1, even 1 or odd number of 0 you will get here, if 1 odd, odd 0 we will go to this state, if you go to this state you have come across odd number of 1's and even number of 0's.

(Refer Slide Time: 20:21)



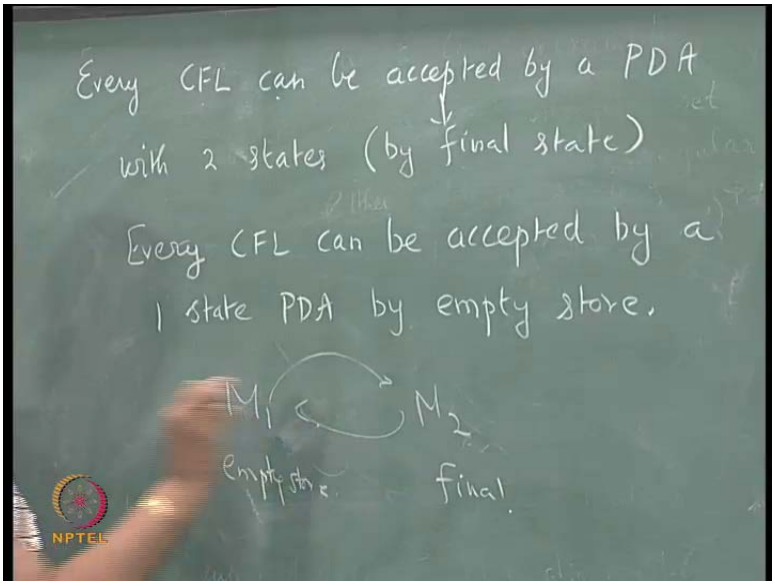
So, this diagram you can draw the question is, is it the minimum state automata or can you minimize? Now, if you follow the minimization procedure first you divide into non final and final states. Now, what are the 0 successes of q 1 q 2 q 3, 0 successor of q 1 is q 0, q 2 is q 3 and q 3 is q 2. So, 0 successor of q 1 here falls into different block q 2 and q 3 falls into a different block. So, q 1 will be separated now q 2 q 3 are in one block. Look at q 2 q 3, what are the one successors of this? One successor of q 3 is q 0, one successor of q 2 is q 1 they are in different blocks.

(Refer Slide Time: 21:23)



So, you have split it so, q 0 q 1 q 2 q 3. So, you cannot minimize it further this is the minimum state automata, but if the statement had been, either odd number of 1 or odd number of 0, either odd number of 0's or odd number of 1's, this will not be a final state, only the other two will be final state. In that case the minimum state automaton will have only two states. The reason is instead of saying this if you have either odd number of 1 or odd number of 0 that means, the other one should be even, if you have odd number of 1 number of 0 should be even, if you have odd number of 0's number of 1 should be even that way. So, the length of the string will be odd, one is odd another one is even. So, the length of the string will be odd.

(Refer Slide Time: 22:53)



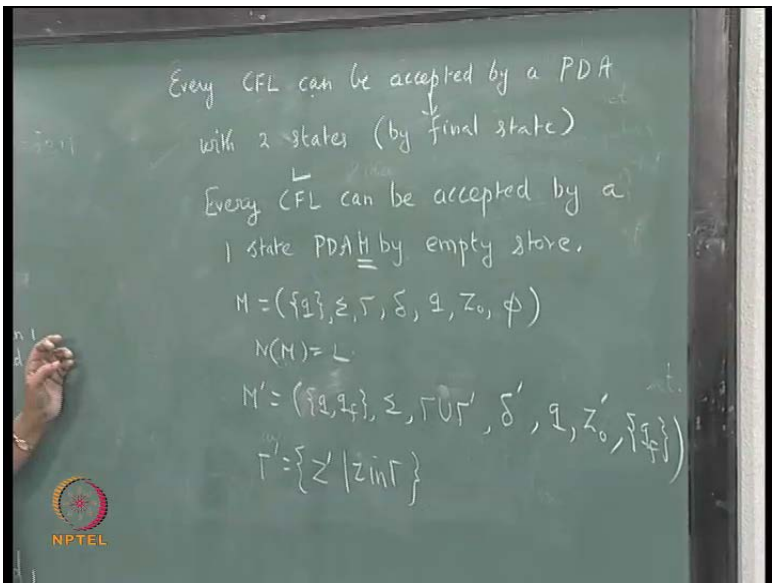
So, actually in the sense you are constructing an automaton which will accept strings whose length is odd and that we can do with two states, is it not? We can have something like this and you can merge these two, but not in the way we have considered. Let us consider one more problem, every CFL can be accepted by a PDA with 2 states by final state accepted by final state by a 2 state PDA.

It is enough if you have 2 states you can accept every CFL by final state, obviously one should be the initial state another should be the final state 2. How do you prove this? So, every CFL can be accepted by final state by a PDA with 2 states. Now you know that we have considered the construction of converting a context free grammar into a PDA which accepts by empty store.

So, from CFG to m such that N of M is equal to L of G this we have done and in this construction how many states did we have for M just one state. So, in this construction which you consider M has 1 state. So, every CFL can be accepted by a push down automata, by empty store with just one state. So I will write it like this, every CFL can be accepted by a one state PDA by empty store. If we know that if something can be accepted by empty store it can accepted by final state, if M 1 accepts by language by empty store M 2 can be constructed such that this accepts same language by final state this is empty store. We know given M 1 how to construct M 2 given M 2 how to construct M 1 that also we have considered.

Now, given M 1 when you constructed M 2 you added 2 states whatever be the set of states here you added 2 more states q 0 dash and q f right. So, if you follow this procedure then with 1 state you must add 2 more states, 3 states you will get, but what you are excepted do is to do the same thing with just 2 states.

(Refer Slide Time: 26:16)

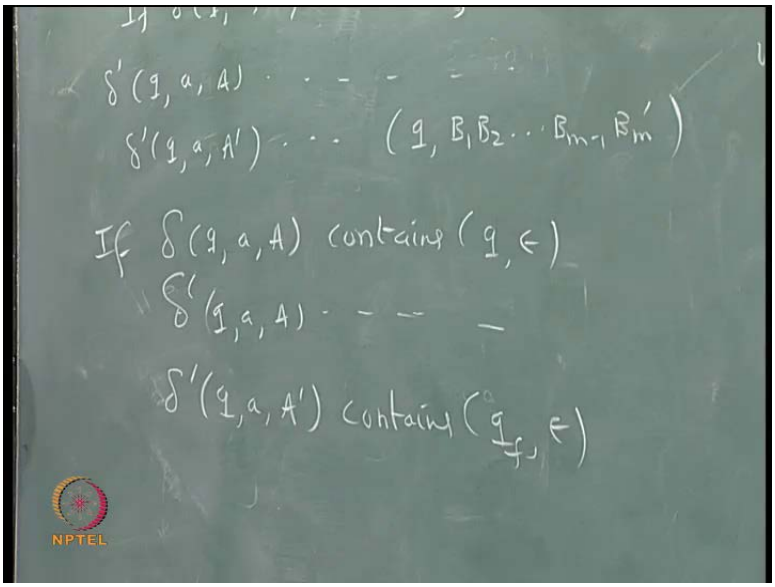


So, how do you do that? So, M I will call this as M, M is just one state q, q, sigma, gamma, delta, q, z 0 it has got only one states and input alphabet accepted. This N of M is equal to L, the given L if it is every CFL. Now, you construct M dash in such a way that M dash is q, q f two states sigma, gamma dash. I will write it as union gamma dash, what you have is every symbol z in gamma, you have marked version, a primed version of it.

So, if you have k pushed down symbols now you are having 2 k pushed down symbols, for symbol every pushed down symbol you are having a marked version or a primed version of that. Delta dash we have to define, this q is the initial state. Now, z 0 dash the primed version of z 0 will be the initial pushed down symbol and q f will be the only final state. How do you define?

The whole idea is like this, whatever it is you start with the original case you start with z 0, when you read this string something will be done on the stack and finally, everything will be erased. Now, you start with z 0 dash same simulation is done here except that whenever in the original thing you have to say, original automaton which is accepting by empty stores at any stage suppose you have B 1, B 2, B m on the stack. In this automaton you will have up to B m minus one the same thing, but the last symbol you always mark it, you use the primed version for that symbol. So, ultimately in the last step when you remove this you go to a final state, when you remove that primed version you go to a final state.

(Refer Slide Time: 29:41)

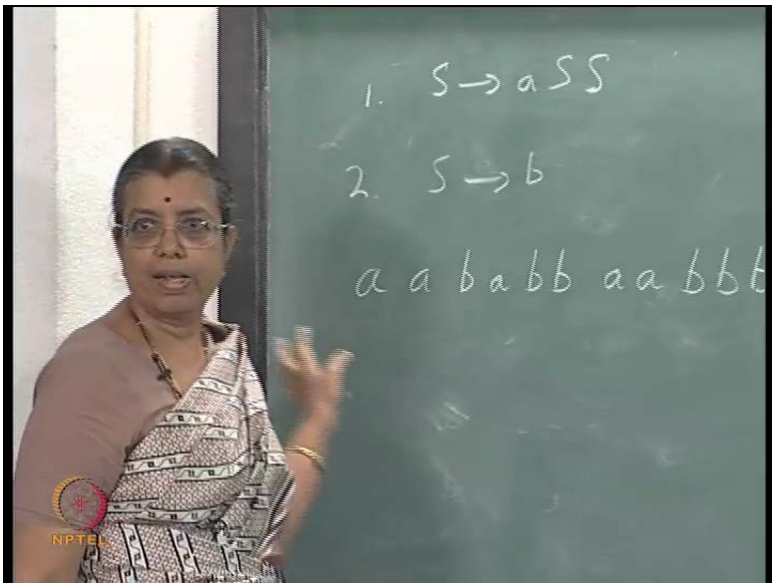


So, the mappings are defined like this if delta of q, a, A contains q there is only a one state please remember that, q is the same, m has this mapping, then delta dash q, a, A contains this, the same mapping will be here and delta dash, q, a, A dash. Suppose, the stack contains only this symbol A dash, then this should be replaced by B 1, B 2, B m right, but the last symbol you will mark it, if this is marked then it contains q, B 1, B 2, B

m minus 1, B m dash. Whatever mapping is here it is also there right. So, in the top most symbol is not the last symbol, the stack contains more than one symbol, m dash behaves just like m.

But when the stack contains only one symbol it is a marked symbol, in that case when you do something you always make it a point to mark the last symbol. Then if you have something like this if delta of q, a, A, if contains q Epsilon it is a popping move, you pop the symbol a, then the same thing you will have delta dash, q, a, A contains this map will be there in delta dash also if you are popping something from the push down store which is not the last symbol, you have the same mapping right.

(Refer Slide Time: 33:03)



But delta dash, q, a, A dash, if you are popping the last symbol which is a marked, then you go to the final state. So, with this sort of a construction with two states you will able to accept any context free language by final state. S goes to a S S, S goes to b, for this grammar let us consider the string a, a b, a b b, a a, b b b, right is this the string? For this string find the left parse, right parse and the Parikh mapping this was the question?

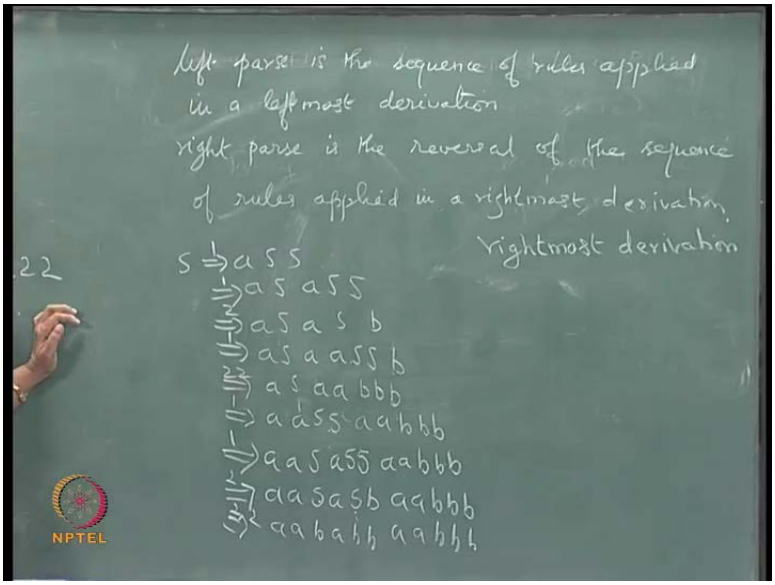
Now, why are the numbers given for the rules to find the left parse. What is a left parse? Left parse is the sequence of rules applied in a left most derivation, right parse is the reversal of the sequence of rules applied in a right most derivation. What is the definition of left parse and what is the definition right parse? It is not that drawing I explained how it is used in top down parsing and bottom of parsing right, that does not mean that we

have to draw the tree, there will be only one tree is this grammar ambiguous or not first of all is this grammar ambiguous or not?

Ambiguous.

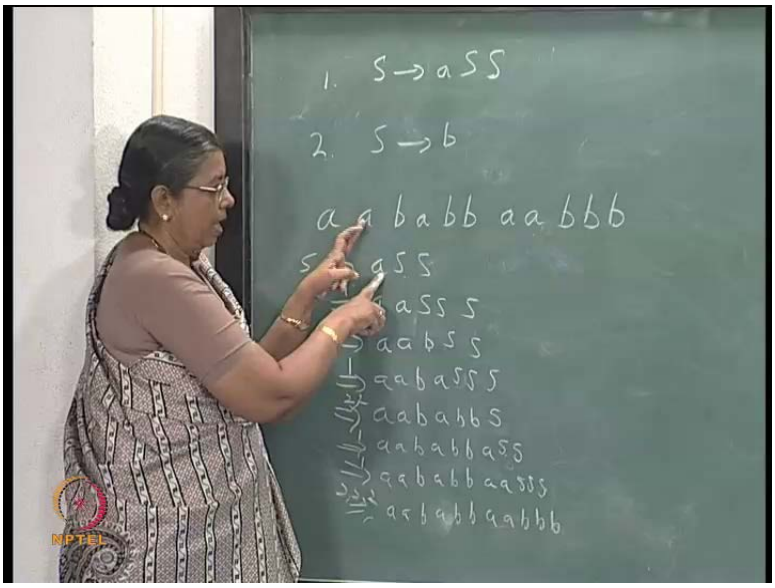
It is not it is unambiguous. So, for any string you will have only one derivation tree. So, in the parsing how you draw the tree and in the top down parsing and bottom of parsing how you draw the derivation tree is different, but ultimately you will get the same derivation right? Left parse and right parse means this, left parse is the sequence of rules applied in a left most derivation.

(Refer Slide Time: 35:17)



Right parse is the reversal of the sequence of rules applied in a right most derivation. So, if you consider the left most derivation a S S rule 1, then a a S S is written as a S S that is again 1 a a then this S goes to b to this S goes to a S S rule 1 this is a leftmost derivation, a a b, a then this S goes to b rule 2, another rule 2 have this then rule 1 a a b, a b b, a S S.

(Refer Slide Time: 36:32)



Now, this again rule 1 a a b, a b b, a a S S. Now, these three S S will go to b one by one, 2 2 2 you will get a a b, a b b, a a b b, when you want to see first one a has been generated, this again you have to generate a that means, you have to use the first rule only if you have to generate b you have to use the second rule. So, it is a sort of deterministic parsing. So, that is why it is unambiguous grammar is unambiguous. So, what is the sequence of rules applied it is 1 1 2 1 22 1 1 2 2 2 this is the left parse. What is the right parse? Look at the right most derivation a S S rule 1.

Then a S this is this is rewritten a S S rule one again, a S a this goes to b rule 2, a S a this goes to a a S S b this is the rule 1, then two rules or two is applied to convert them into b a S a a b b b then 1 a a S S a a b b b a a S this goes to a S S a a b b b. So, this S rewritten a S S that is rule 1 then a a S a S b a a b b b, that is rule 2 again for this again, you can apply rule 2 and again for this s you can apply rule 2 a a b a b b a a b b b.

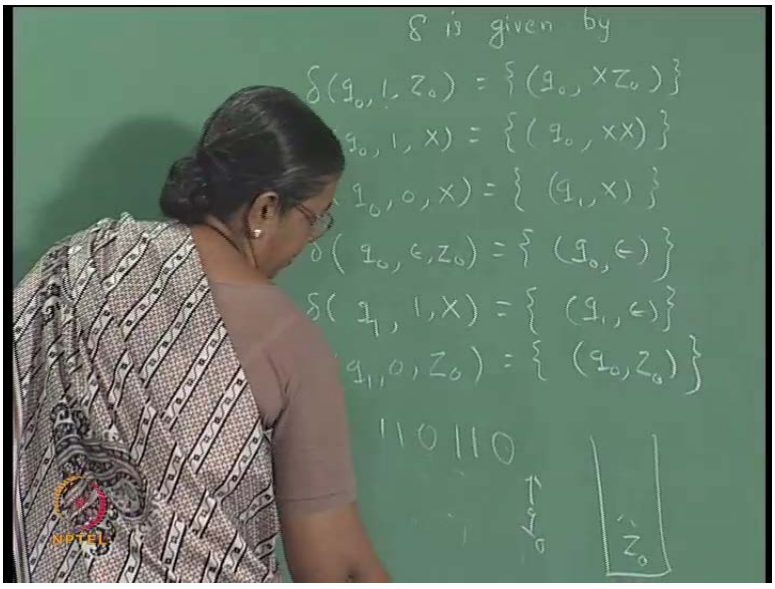
So, right most derivation the sequence of rules is applied a sequence is 1 1 2 1 22 1 1 2 2 2. The reversal of this is 2 2 2 1 1 2 2 1 2 1 1 this is the right parse, this is the left most derivation at each step the left most non terminal is replaced, the sequence of rules applied is this and that is called the left parse. Here this is the right most derivation, at each step the right most non terminal is replaced. The sequence of rules applied here is this, the reversal of that sequence is the this is the right parse and right parse is used for

bottom of parsing left parse is used for top down parsing. This particular thing can be parsed in a deterministic manner in a top down one this called it has a L L 1 grammar.

This is the L L1 grammar, we say this grammar is L L1 because when you have S and a you know that you have to apply the first rule, when you have S and b you know that you have apply the second rule and you can deterministically say that, that is why it is called the L 1, another feature of this is something similar to the dyke set for this and the dyke set there is some similarity is there. Are you able to realise that, as the length of the string increases.

See here at 1 stage you will have three S S, three S S in a sentential form at one stage and similarly here also somewhere you will three S S, where is it here as the length of the string increases somewhere you will have to get four S S, five S S and so on. You cannot bind the number of non terminals, somewhere it will increase like the dyke set. So, this is not a language, the grammar generates the language which is not of finite index. L of G is not of finite index like the dyke set.

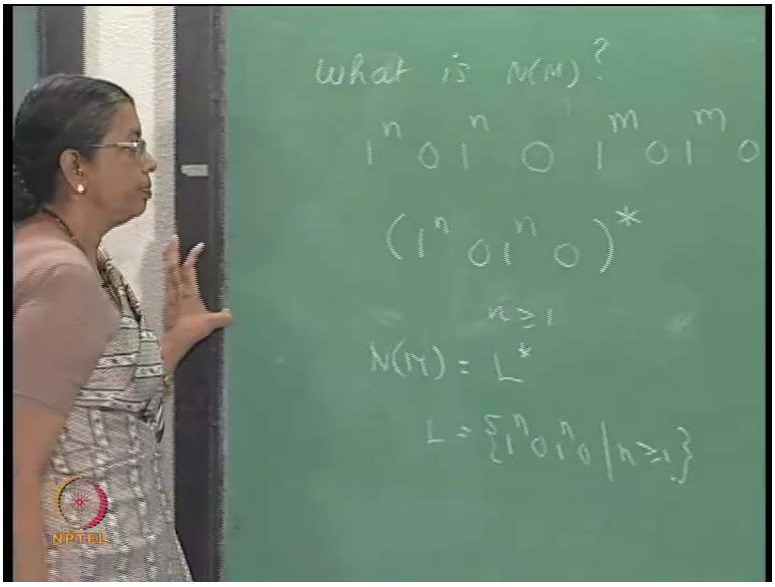
(Refer Slide Time: 43:47)



There is another feature of this particular grammar. This question what will be the how does it work on the string 1 1 0 1 1 0 you start in q 0 the stack contains z 0, how will this machine work on that particular string.

So, I will write this string here 1 1 0 1 1 0, it starts in say q 0 reading a 1. So, on the push down store you have z 0 then when you see a 1 you add a x, again when you see this one you add x and in q 0 when you see a 0 you go to q 1. In q 1 you start removing the x's, in q 1 when you see a 0 you go to q 0 and in q 0 you can remove the symbol and this string will be accepted.

(Refer Slide Time: 46:21)



So, this will be accepted by this machine. What is the language accepted by the this machine? What is N of M? When you are reading one in q 0 you are adding x, then when you see a 0, you change a state from q 0 to q 1 and in q 1, when you read ones, you keep on removing the x. So, the number of 1's occurring before the one and after the 0 are the same. So, you will have 1 power n 0, 1 power n 0. Finally, when you read this 0 you go to initial configuration again then what happens? First of all is this deterministic or non deterministic. First of all is this PDA deterministic or non deterministic

Non deterministic.

It is non deterministic, even though these are all single terms, this is non deterministic because when you are having a q 0 z 0 combination you can have a true in put move or Epsilon move both are allowed, in a deterministic machine that should not be allowed. So, after reading this you can erase the z 0 and end the whole process or you can start again. So, when you start again you will get something like 1 power m 0, 1 or m 0 again you can start and get 1 power k 0, 1 power k 0 and so on.

So, ultimately the language accepted is 1 power n 0 1 power n 0 star something n greater than or equal 1. You can just erase z 0 also see from this you can just erase that. So, epsilon also will be accepted. So, the language accepted N of M is something like this or you can put it N of M is L star where L is 1 power n 0 1 power n 0 n greater than or equal to 1. Now, construct the grammar from this. Now, construct the context-free grammar from these delta mappings.