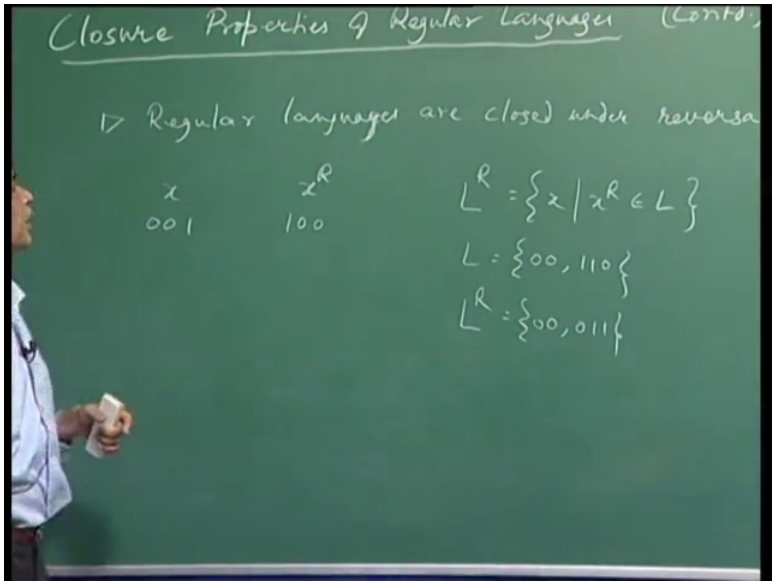
Theory of Computation.
Professor Somenath Biswas.
Department of Computer Science & Engineering.
Indian Institute of Technology, kanpur.
Lecture-14.
Closure Under Reversal, Use of Closure Properties.

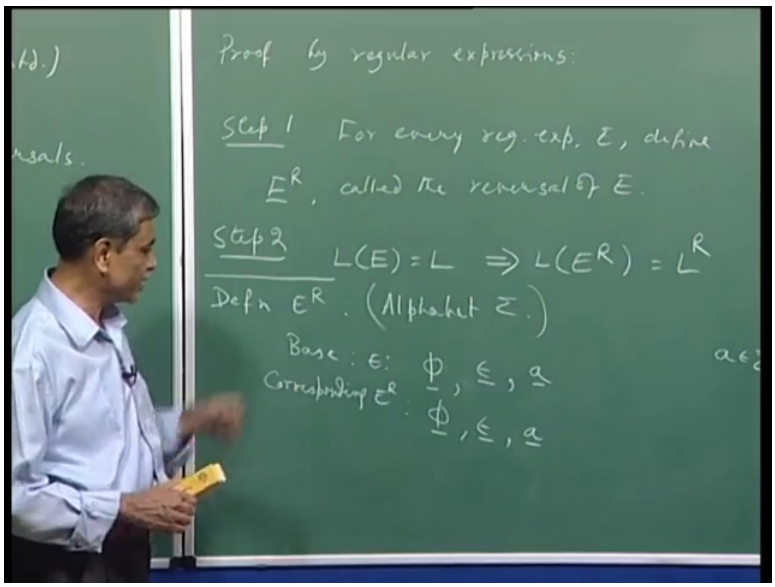
(Refer Slide Time: 0:36)



We will continue our discussion on closure properties of regular languages. The 1st thing I would like to do is to once more review the result that we talked about in the last lecture, which is one of the results that regular languages are closed under, closed under operation of reversals. And recall by reversal we meant, firstly we can talk of any string x and then we can talk of its reversal. For example if the string is 001, then the reverse, reverse string is just a string such other way. And the reversal of the language L is all those xs such that reverse of x is in L. And we give a very simple kind of example, that suppose L had 00, and 110, then L reversal is going to be 00 and 011.

And we also indicated, we can prove this property, that is regular languages are closed under reversals in 2 ways, one was taking a DFA for a regular language L and then reverse all its arrows, the directions of the arrows in the transition diagram, Avenue initial state when Epsilon transitions to the final, old final states of the DFA that accepted that language L and have a new final state as the old final, old initial state of the DFA. So that is something that we discussed. now another way we have proved it or we indicated that this result can be proved by using regular languages.

(Refer Slide Time: 2:54)



So the proof by regular expressions, I would like to dwell on it for a little while because this proof idea is quite natural and simple. The step one in the proof was, for every regular expression say E, define ER called the reversal of E. Then step 2 was, what we intend doing is the step 2 that if the language accepted by E is L, this implies that the language, sorry I should not have said accepted by language, accepted by regular expression, I should have said the language denoted by the regular expression E, Supposing if that language is L, that the language denoted by ER which is defined in this in 1st step is the reversal of the language L.

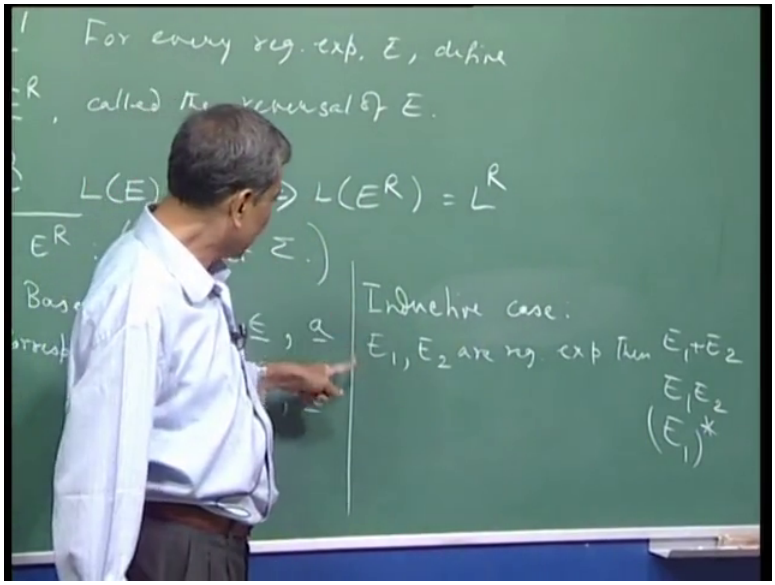
And what steps one was was to define ER, so that was the definition of ER, step 2 we kind of very quickly indicated what we meant but let us now go over this once more. So definition of ER, look what we are trying to do is to give what is ER for every regular expression. Recall that whenever we talk of a regular expression, we keep some alphabet in mind, regular expressions over an alphabet. So let us say the alphabet is some Sigma and so this definition is for this alphabet Sigma and then in this definition there is a base case, this definition itself is an inductive definition, so therefore it has a base case and an induction case.

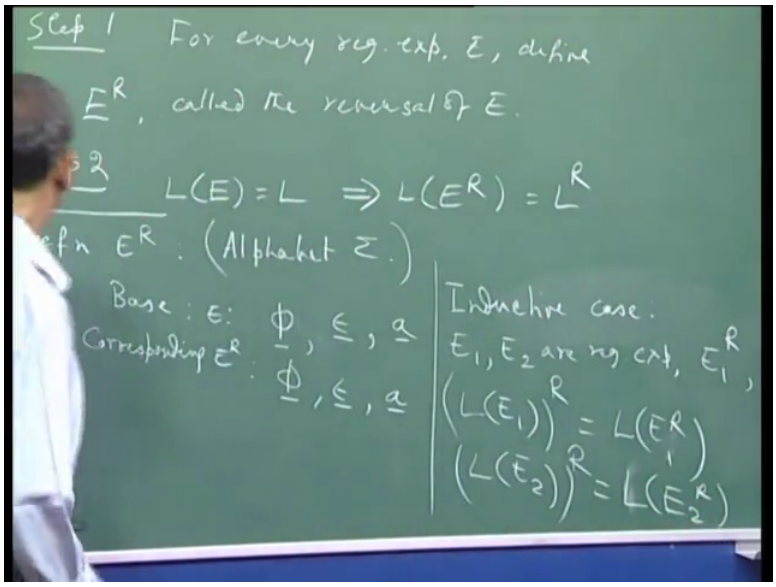
So base case was, you recall what are the basis for regular expressions B over alphabet Sigma, there are 3 base cases, so E can be in the base case either it can be empty or it can be Epsilon or it can be a symbol a where a is an Sigma, right, I put the underline here, denote that these are regular expressions. Corresponding ER is same. now notice in step 2 I am supposed to prove this that if the language denoted by E is L, then language denoted by E of

R is L of R, reversal of L. now what are the reversal, what are the language denoted by these 3 base case regular expressions?

This of course denotes empty set, this denotes just a language with the string Epsilon, this denotes the language with just the string consisting of a single symbol which is a. now such, each of these languages of reversal, here you will get back the same language and therefore this is true, this statement is true for the base case have been defined E and E R for these base cases in this manner, we can claim that this assertion is true for the base case.

(Refer Slide Time: 8:03)



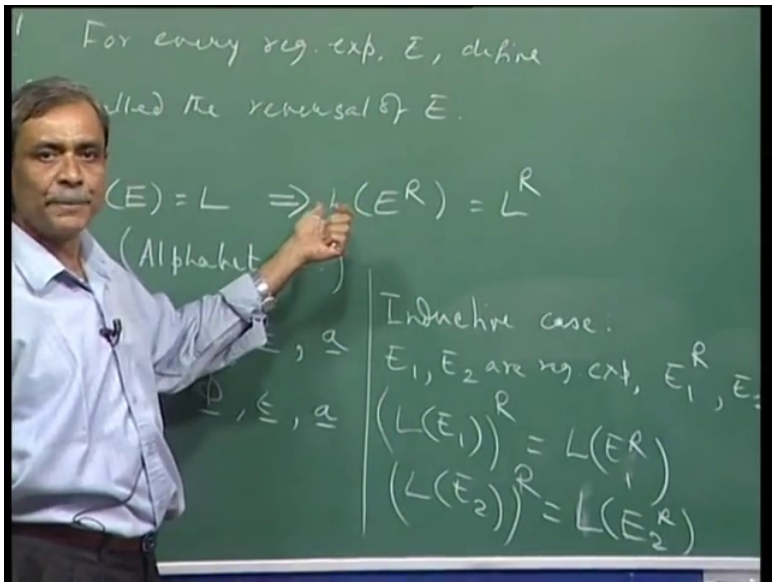


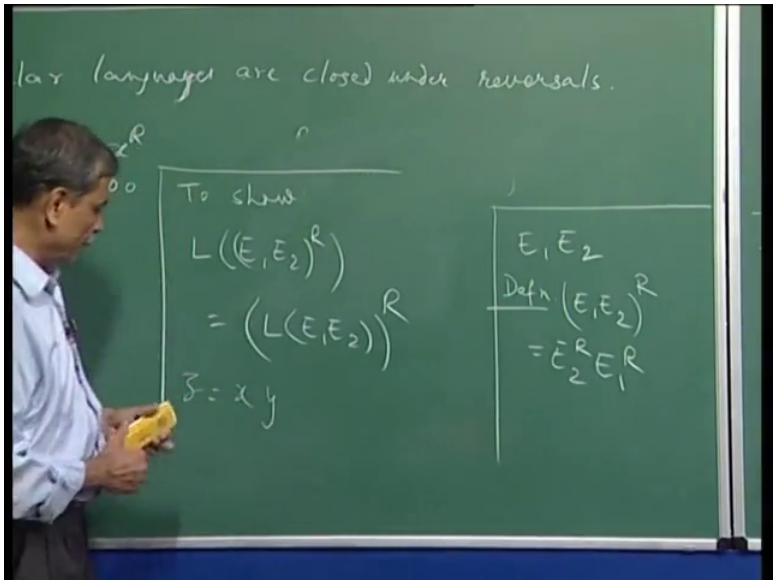
now let us take one of the inductive part case. And now remember once we have defined the base cases, I need to say what about larger and larger regular expressions. Anyway, remember that any other regular expression could have been form either by concatenation or by +

operation by kleene star, star operation, right. So you know inductive cases we had said that E1 and E2 are regular expressions, then E1 + E2, E1 E2 and E1 star, all these are also regular expressions. now what we need to do in our attempt to prove this assertion having proved the base case.

now we have to say that suppose E1 and E 2 are 2 regular expressions, then inductively I can assume that regular expressions, right. And now the assertion will say about E1 R and E2 R the following, right. We will say that languages denoted by E1, its reversal, its language denoted by E1 reversal. So let us let us just spend one moment on this. See from E1 we obtained that the E1 reversal this syntactically. Right, by the rules that I discussed last time. But maybe I will I will just discuss it once more here for one particular case. And this is stating that inductively we can assume these 2 facts.

(Refer Slide Time: 10:53)

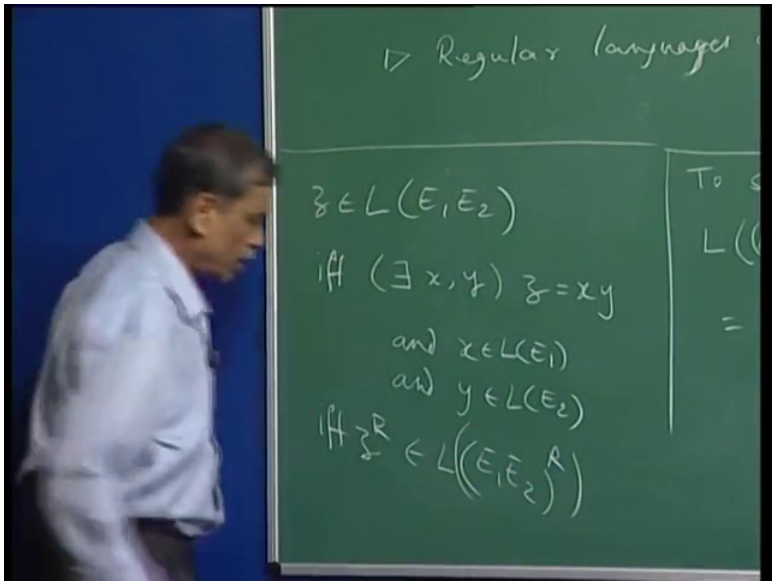


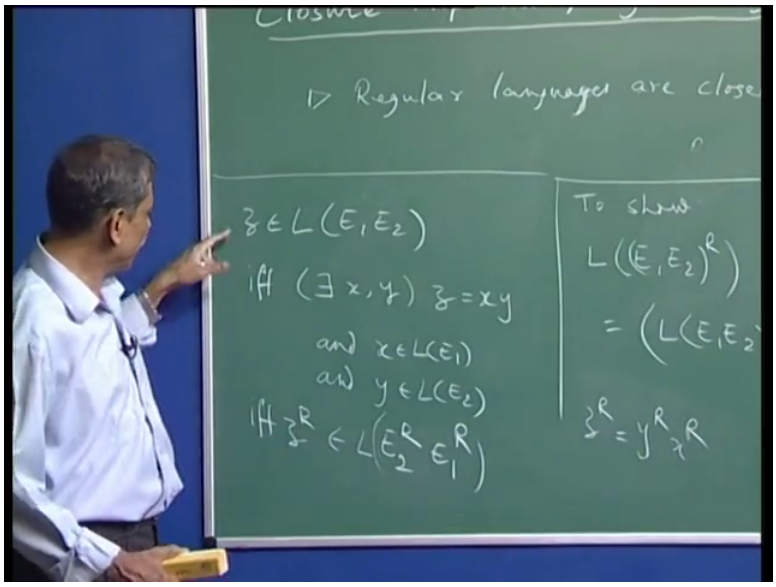


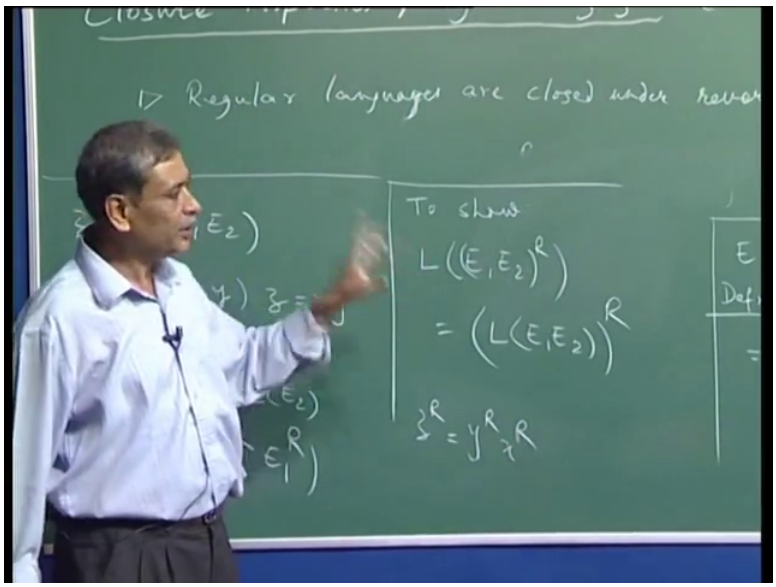
So let us just, looking at that let us just take the house the induction inductive case will go for one interesting part of the 3 cases, namely concatenation. So we know that E1 E2 is the regular expression because E1 is a regular expression and E 2 is a regular expression, right. now we defined, remember that definition of E1 E 2, this regular expression was E 2 reversal followed by E 1 reversal. And now this is how we had defined reversal for a concatenation case. now inductively what we have, what do I need to show now? To show that the language denoted by E1 E2 reversal is the reversal of the language denoted by E1E2.

And now we use these 2 assumptions, these 2 assumptions are from fact that we are proving this inductively, so we can assume, given a larger regular expression having done carried out the induction so far that it is assertion is true up to some length and let say E1 and E2, their lengths are within the standard deduction that you do on length of the regular expression. So you see what is the language denoted by E1E2, it is, Supposing I have a string E1E2 in this language, then such string is there in that language because that string can be seen as concatenation of 2 strings.

(Refer Slide Time: 13:32)



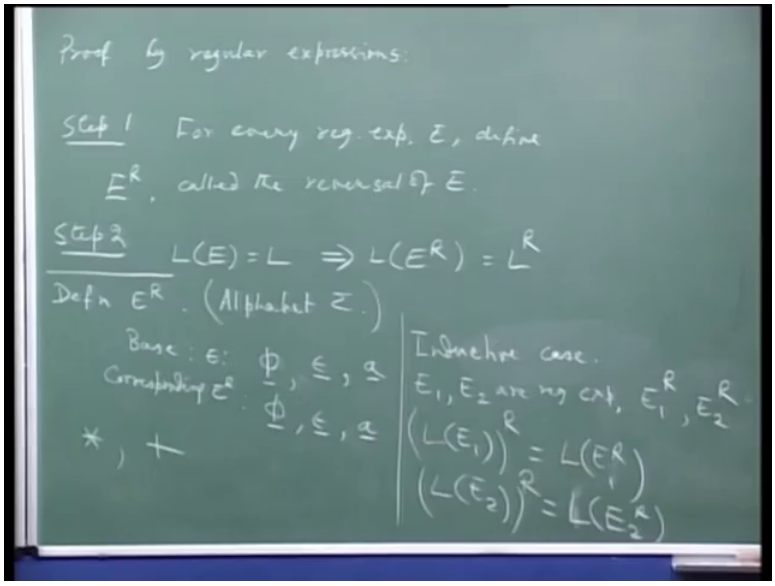


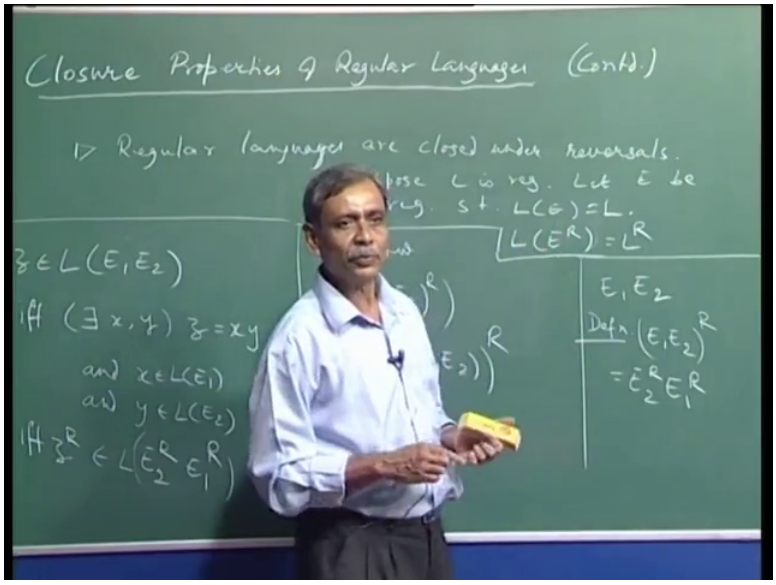


So as we are saying, that suppose z is in this language, language denoted by E1E2, you can see that clearly that z is in the language denoted by the regular expression E1E2 if and only if there are x and y, right, such that x, firstly z is the concatenation of x and y and x is in the language denoted by E1 and y is in the language denoted by E 2. Right, clearly I mean that we know from our understanding of regular expression that when I concatenate 2 regular expressions, then this is what is the meaning of concatenation for regular expressions, the language denoted by the concatenated regular expression can be seen in this manner.

And remember we need to prove this, so what I need to show that z and L, if and only if this, if and only if, I would like to show this, if and only if the reversal of the language zR is in the language denoted by reversal of this. But by definition E1E2, reversal of that at the regular expression, this is our definition, this is same mass, I can I can, just look at that definition, you can see, let me replace this by E2R E1R. now by the way what is z of R, z of R is clearly, z was xy, so you must read it from this side y to x, so yR x of R. now because y is in L E2, language denoted by E2, y of R will be the language denoted by E2 R, right and x of R will be in the language denoted by E1 of R. Right so this is true and therefore I can say this this is proved for the concatenation.

(Refer Slide Time: 17:22)

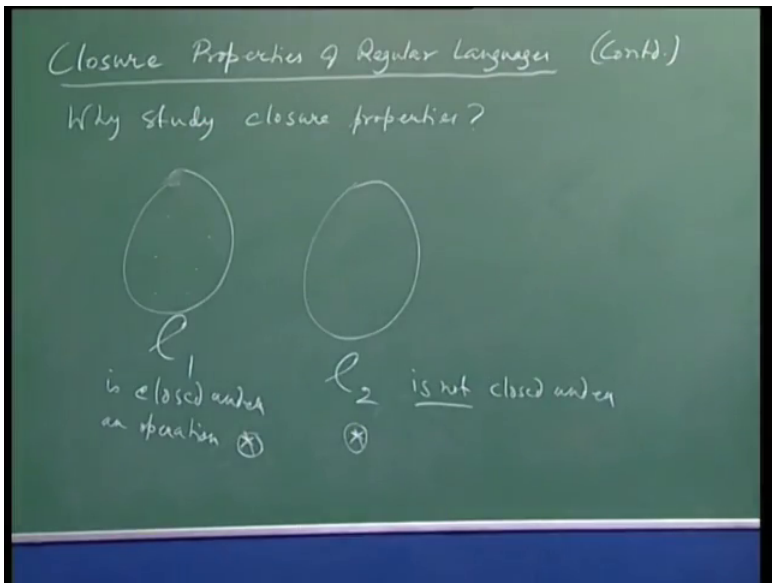




Other 2 cases for induction over regular expressions have star and of course + and these 2 cases are also, in fact + is simpler, star is also simpler than this. So that completes the proof in detail that indeed, so overall idea was that I wanted to prove this, regular languages are closed under reversal. So you take a regular language, so let us say suppose L is regular and then because L is regular and we can say let E be the regular expressions such that language denoted by E is the language L. And then because of what we have shown, we have essentially we have shown that language denoted by E R is defined in this manner is going to be LR, right.

So you take a regular language, its reversal is denoted by a regular expression, there is a regular expression which denotes the reversal language, right. And every regular expression denotes only a regular language. So this language LR is also therefore regular and that is what we needed to prove.

(Refer Slide Time: 18:46)



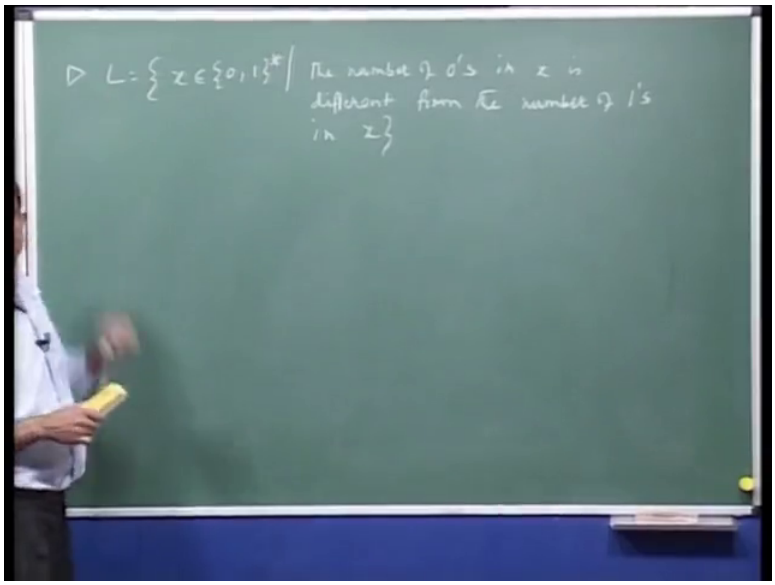
We have seen some closure properties for regular languages but still we need to like to answer this question that why studies such closure properties. There are various reasons, one reason is this, suppose I have 2 classes, let us say C1, this is the class of languages, by that now you know what we mean is C1 is a set containing languages, some languages, so some class of languages C1 and C2 is another class of languages. now one big question is that these classes might have been defined in some manner, we would like to know whether these classes are same, whether you know or not. So one way of proving that they are not same is that suppose I managed to show that C1 is, let us say C1 is closed under an operation, some, let say something.

Let me just denote this operation like this. And C2 is not closed under the same operation. Then immediately I have that the 2 classes could not be identical. So this is called separation of 2 classes C1 and C2, can be proved, finding out of some operation could be unary could be binary, such that one class is closed under that operation, the other class is not closed under that operation. So this is, I can actually illustrate this that we have seen regular expressions that are closed under complementation, right. So C1 if you take regular languages, that class is closed under complementation and now another class which we are going to study soon, that is the class of context free languages, this set consists of all languages which are context free, that class we will see that is not closed under complementation.

And that immediately tells us that these 2 classes are different. Of course there are, we will see in more direct manner that class of regular languages, the class of context free languages are different. This is, but I just wanted to tell you which is one reason people study closure

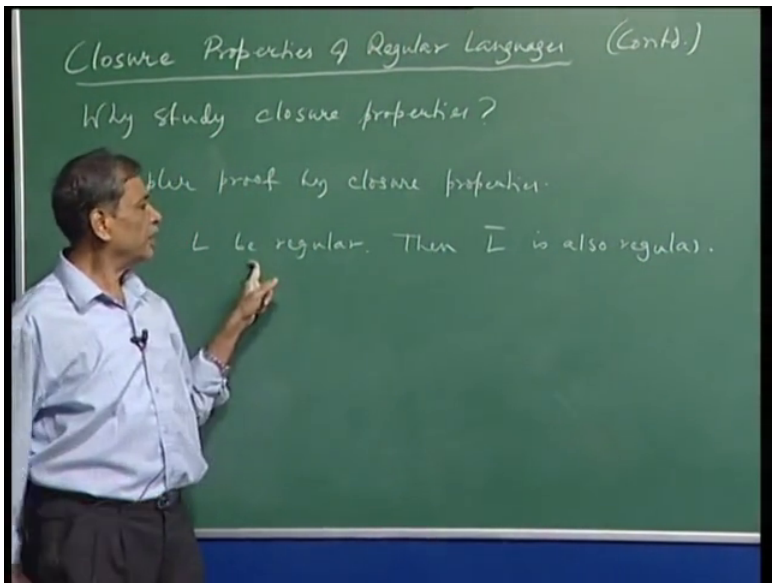
properties and for, there are classes whose separation have been, whose separation has been proved only using closure properties. One more reason which we can see in our context quite easily that some proof might be done simply using closure properties. So let me give you an example.

(Refer Slide Time: 22:15)



Consider this language L which is over the alphabet, binary alphabet such that the number of zeros in x is different from the number of 1s, okay. So all those binary strings where, when you count the number of zeros and the number of ones, these 2 numbers come out to be different. So we are collecting all those strings and putting them in this language L and we would like to show L is not regular. And one method we know of making, providing such proofs is by pumping lemma. We can prove this through pumping lemma, let me just indicate the proof.

(Refer Slide Time: 24:30)



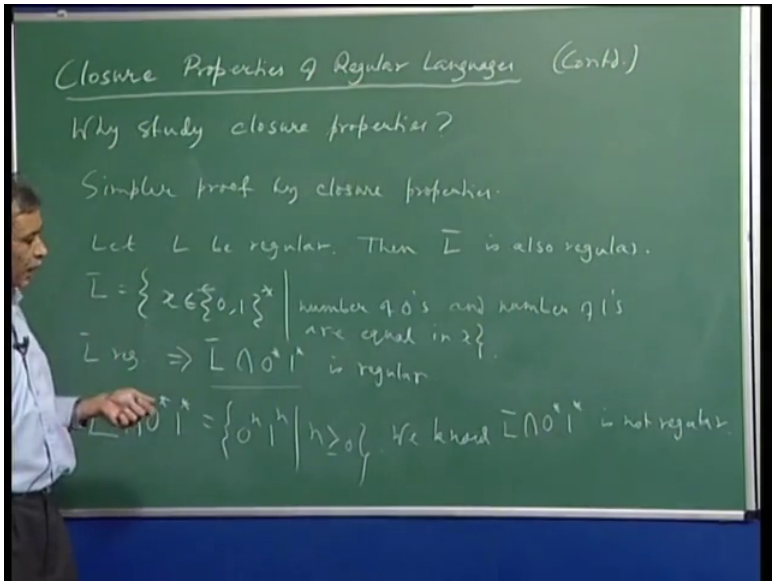
Proof by pumping lemma, remember that pumping lemma proofs start with, it is a proof by contradiction, we say let L be regular. Then we say let k be the pumping lemma constant for L. And now the way such proofs will go that I will take some z which is in L, right whose length is greater than or equal to k and get z1 on pumping z such that z1 is not in L. And it is not immediate at least, all the other pumping lemma proofs we have seen, things were pretty clear what the z you should take and what is the pumping you should do so that you get string not in the language. now here you can take z to be, let say 0k, 1k factorial + k.

And you see it is not, I will not give the details of the proof but you should be able to see this, you see what is happening is that now u v part is here and it is the V part that we pump. So V is some number which is between 1 and k, that is your V. All such numbers are divisible by k factorial, right. So what you can do, so in fact let me indicate this. So this 0k is UV, UV, let us say length of V is t and now I can write this as 0k - T, then 0T, this is your UV. Well, there is something else here also, UV 0k is UV and some part of W, right.

So let me let me write it as 0L UV and some part of W and that part of W which is in 0, so this is of, this I will write it as 0k. And now because V is there which is the V part which you can pump, you should be able to prove, you see for example if you pump twice, then what is the length of the string that I will get, the only 0s part will be k + T, right. Because you have pumped this part twice, so this part will become 0T, so k + T, right. So in this manner if you pump from R + 1 times, what you are going to get is some, you can see it that it will become k + some Rt and since t divides k factorial and this k k part will cancel and therefore I will be

able to get the z 1 which is not in the language because then the number of zeros will become exactly equal to this.

(Refer Slide Time: 28:42)



I leave the details but the point I am kind to make is this is not a very immediate thing. On the other hand, so let me reassure you a simpler proof using closure properties. Let me now give a simpler proof of this fact that this language L is not regular using closure properties, simpler proof by closure properties. This proof also starts by assuming that L is regular, let L be regular, right. now we say then L complement is also regular, right. now how did I get this fact, from this fact because regular languages are closed under complementation, so therefore L complement is also regular, right. However what is L complement, you can see L complement will be the set of all strings in which number of zeros and number of ones are equal.

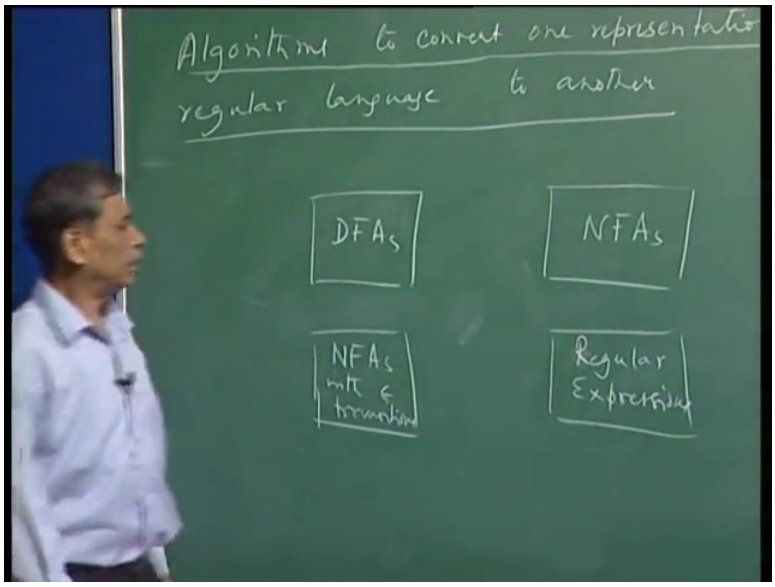
So L complement is all strings over the binary alphabets such that number of 0s and number of 1s are equal in x. Okay. So this is L complement. now since L complement is regular, this implies L complement intersection 0 star, 1 star is also regular. Remember 0 star 1 star is the, all strings which are of the form some number of zeros followed by some number of 1s, L complement. Why can I, why, what is the reason I can claim this that L complement intersection 0 star 1 star is regular, L complement is regular, 0 star 1 star, this language is also regular, regular languages are closed under intersection, so therefore this language is regular.

But what is this language, L complement intersection 0 star 1 star, this language is nothing but 0n, 1n, n, right. And do you see the contradiction, because this is a very simple language

to prove that this is not regular. So I can say you know this language L complement intersection 0 star 1 star is not regular. And therefore we have come to contradiction, why because I started with a regular language L and we did some closure properties operation, operations which preserve regularity and we got a language which is not regular. Which means our assumption must be wrong and therefore I claim, therefore L is not regular. So here I got a contradiction, therefore L is not regular.

you will definitely see that this proof is much simpler than directly applying the pumping lemma. How does one figure out the string, you have to rack your brains to find such a spring and then we need to, you know argue, once of course we found that name, that argument was also little involved. But here all the arguments are much simpler because we use closure properties. you recall we have 4 different presentations for regular languages, what are they?

(Refer Slide Time: 33:52)

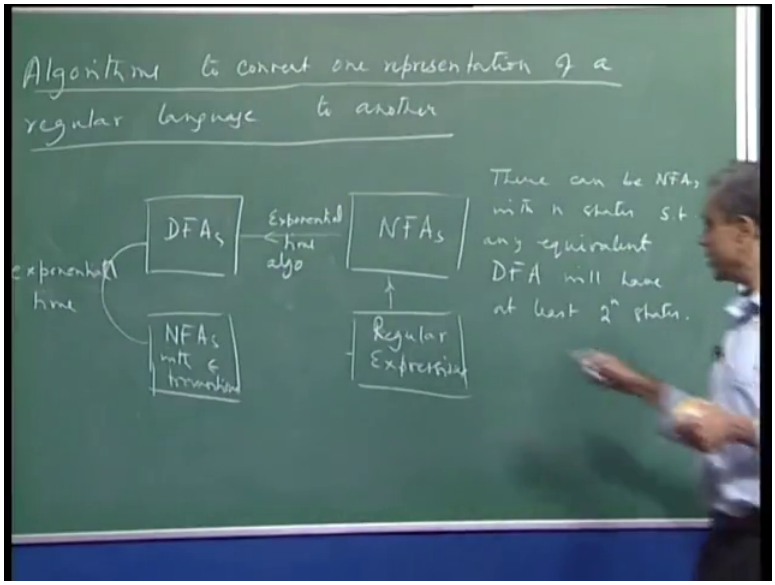


We can represent DFAs, this is one way of representing regular languages, NFA, correct, then you have NFAs with Epsilon transitions and regular expressions. The point I am saying that you can take a regular language L, I can represent that regular language by providing a DFA that would accept that regular language and NFA which will accept the same regular language, possibly an NFA with Epsilon transitions for the same regular language and of course I can give a regular expressions which will denote the same language. now what we are interested is in knowing, can I go from one representation to another and what is the time complexity, how efficiently can I do these things?

now by the way DFA to NFA, DFA to NFA with Epsilon transitions, these are immediate, right because all, after all every DFA is an NFA, why we have said this? Because NFA says simply that from every state there can be 0 or more number of transitions on a symbol, DFA just uses one transition per symbol from a string, exactly one transition. That is fine, the DFA is also an NFA and NFA is also an NFA with Epsilon transitions. So these conversations DFA to NFA, DFA to NFA with Epsilon transitions or NFAs to NFAs with Epsilon transitions, they are not of interest because one is a more generalised version of the other.

NFA is a more generalised version of DFA, NFA with Epsilon transition is a more generalised version of NFAs. So that is this way it is simple.

(Refer Slide Time: 37:05)



What about going from NFA to DFA, we know the construction and we also know that if and, there can be NFAs such that let me write it down there can be NFAs with n states such that any DFA accepting, or let me just use equivalent, you will understand what I mean, any equivalent DFA, that means the new DFA as accept the same language as the NFA. So there can be, what I am saying is there can be NFAs with n states such that any equivalent DFA will have at least 2 to the power n states. And we have seen an example like, you know let say kth bit from the right-hand is 1 for binary strings, recall that example.

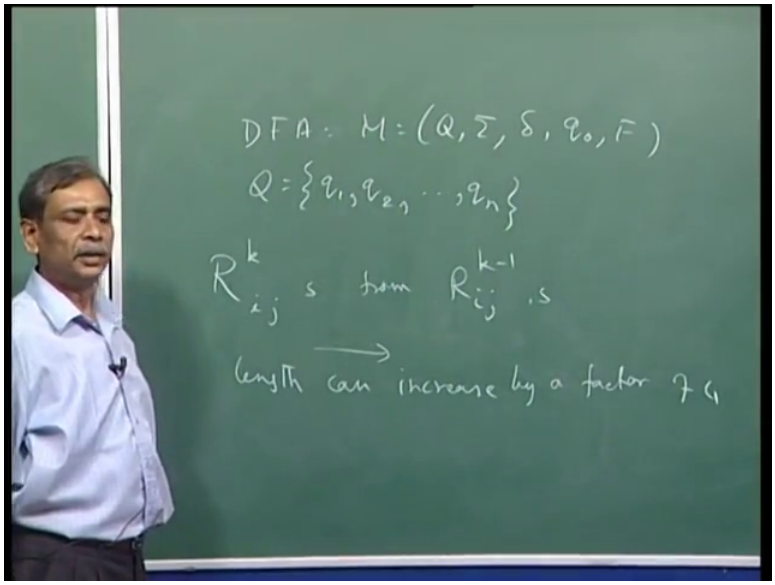
NFA has you know not do many states but the corresponding DFA and 2 to the power n states and this is happening again because of the subset construction going from DFA to NFA. Because that our way of doing that was the DFA will have the number of, the set of states of DFA will be set of all subset of states of NFA. So that way, there is an exponential growth as

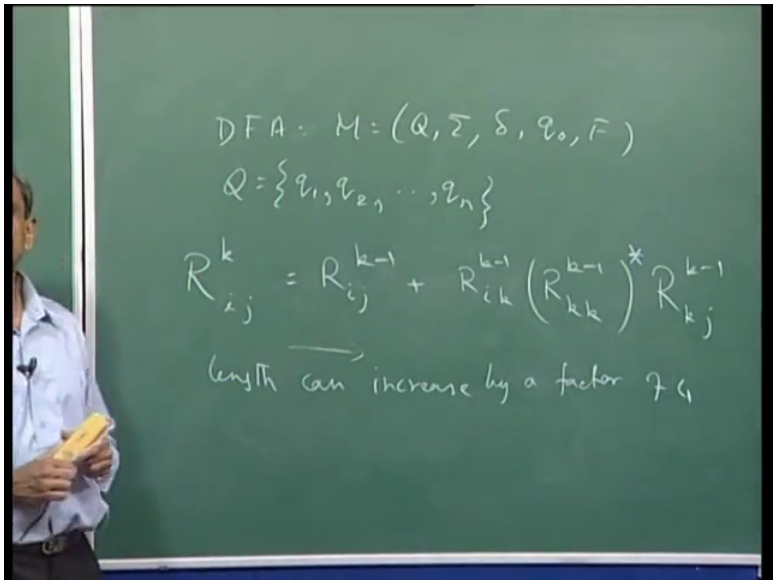
we go from NFA to DFA, there can be such exponential growth in general. So this is an exponential algorithm. That is, we cannot do better than that because of this fact. What about going from NFAs with Epsilon transitions to DFAs?

The only different thing between NFA and NFA with Epsilon transitions says that we need to take closure, right or states or set of states. The closure is not an expensive operation, however because anyway we are going from NFA, so in this case also that subset constructions will be there on top of that there will be that closure operation but closure operation is efficient, right because it is kind of reachable. Closure of a state is the set of all states reachable from that State using only Epsilon transitions. So reachability can be done in sufficient time. And therefore this also however because we are going from NFA to DFA, the number of states can blow up, so this is again exponential time.

What about regular expression to DFA, this? Actually we what we did was, what we had proved was this that you can go from a regular expression we can get an NFA and that can be done quite efficiently. Because that inductive proof that we gave, that we gave NFAs for the base cases of regular expressions and then we showed for the inductive cases how the NFAs will, we built up from the component NFAs of the smaller regular expression. But this can be done efficiently, so in fact linear time at let me just write this conversion can be done in polynomial time.

(Refer Slide Time: 41:34)

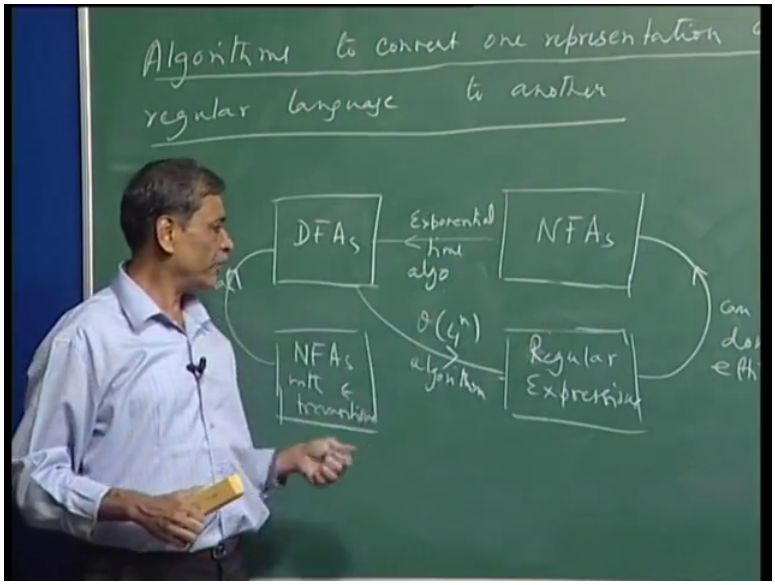




This can be done efficiently. Now remember we also discussed going from DFA to regular expressions and same idea can be can be used for NFAs regular expressions or NFAs to Epsilon, NFAs with Epsilon transitions to regular expressions. What about this algorithm? Remember what we did was we took a DFA and then we wrote a number of regular expressions which are of this type that if the DFA was q, Sigma, Delta, q0, F and I will just indicate this so that you remember what we did. We took the set of states as q1, q2, qn and we defined regular expressions of this form R ij k to be denoting all those strings over Sigma which can take the machine M from qI to qJ without passing through any states whose you know this this this number, whose number is larger than k. Right.

We built this thing and then finally when we got R ij n, using some of those R ij ns we defined, we got the regular expressions which is which denotes the same language as the language accepted by M. And how, what did we do, we obtained R ij k from R ij k - 1s. And here if you go back to that proof, the length of R ij k, the length can increase, right. Recall, let me just write R ij k was defined to be firstly R ij k -1, the regular expression for this + R ik k -1, Rkk k -1, this whole star. Then Rkj k -1. You see if the regular expression with superscript k -1s we had, then we could build the regular expressions with ² k.

(Refer Slide Time: 45:53)



But in the process 1, 2, 3, 4 such things were using, so in this, in this definition the length can increase by factor of 4. And so therefore you see what happens is as you go from here to here, how many times this factor of 4 increase happens, as many times the number of states. So you have a 4, 4 to the power n order algorithm. So this is we cannot do better going from DFA to regular expression or from NFA to regular expression, there can be a blow up of, exponential blow up in the length of the expression. And therefore the algorithm is exponential.