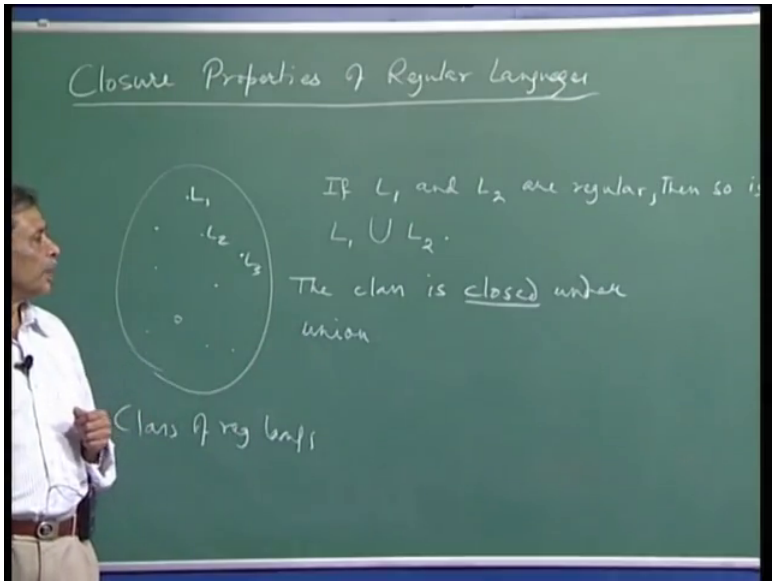
Theory of Computation.
Professor Somenath Biswas.
Department of Computer Science & Engineering.
Indian Institute of Technology, Kanpur.
Lecture-13.
Closure Properties Continued.

(Refer Slide Time: 0:23)



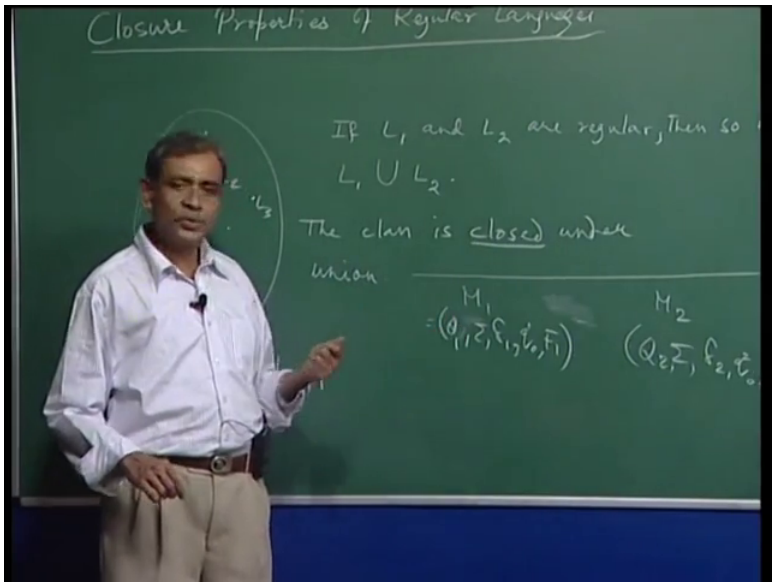
Today we discuss closure properties of regular languages, let us understand what we mean by this phrase closure properties. Remember that the class of regular languages, by that what we mean is all regular languages, no the set of all regular languages that we have been, that is the one we have been different to as the class of regular languages. And now, so this each element of this class is a language and it is regular L1, L2, L3 and so on. Now take some operation, for example union, you take 2 languages here L1 and another language let us say L2 and when you take the union of these 2 languages, you get another language. Right.

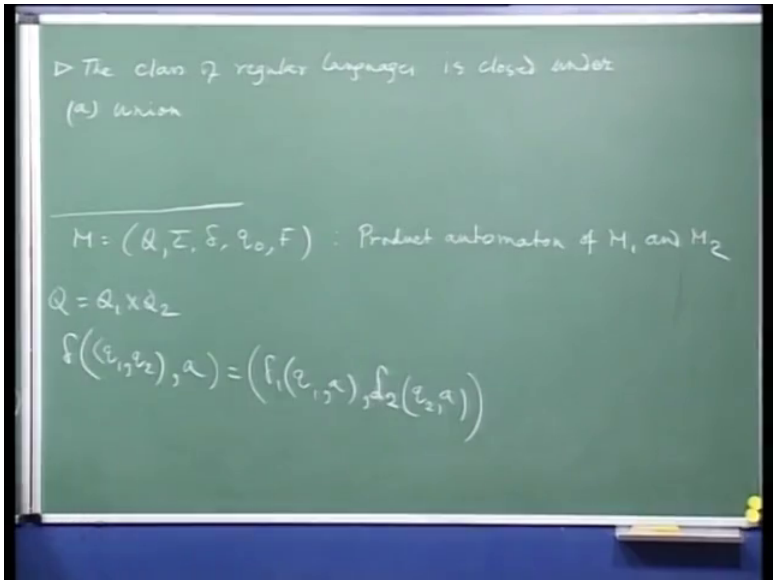
So you can think of this way that union is a binary operation on the set of languages for any class of course. Now as it happens, and we know this fact and we will see that once more that the union of 2 regular languages is regular. So let me write it this way, if L1 and L2 are regular, then so is L1 union L2. So what is happening, we have this binary operation, in this case union, we take 2 languages, any 2 languages from this class and define the new language obtained by taking the union of these 2 languages. And what do I find, that this new language, the result of that binary operation is still in in this class, right.

Because since L1 L2 is regular, L occurs somewhere in this class itself. So in other words, the operation, binary operation union which would give me a language by taking any 2 languages from here. So what we are saying once more is that union is in operation, a binary operation defined on this class which when performed on any 2 languages in the class, the new language that we obtain still belongs to the same old class. So in that sense the operation of union does not give me any new language from this class, but the language formed by taking the union of 2 languages is inside this.

So another way of saying this, what people say that this class is closed, the class is closed under union, because you are not getting any new language when you perform this operation of union. Now there are many possible operations, in particular we will see a few which are binary as well as we will see a few which are union. And we will show that under all those operations, we never manage to get any extra language by performing that operation on this class, so therefore the class is closed under union.

(Refer Slide Time: 4:57)





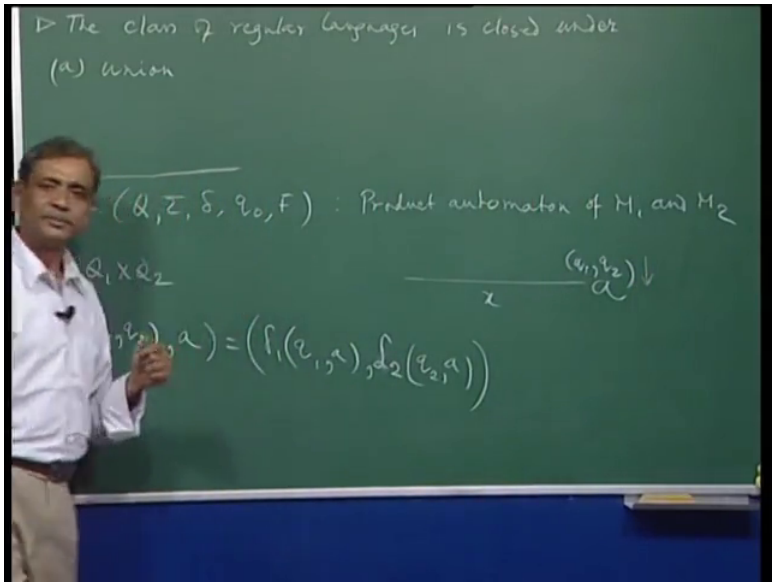
So let us, we started with union, so let us see this, the class of regular languages is closed under. And here I will provide a list of the operations, list of some, not totally exhaustive but a number of operations under which our class is closed. The 1st is union, now what is the proof for that fact that the class, regular, the class of all regular languages is closed under union. Actually we have seen this proof earlier, let me remind you what we did was, suppose I have 2 DFAs M1 and M2 and M1 is Q, say Q1 Sigma Delta1 Q0 and F1 and M2 is Q2, same alphabet Sigma Delta 2 Q0, let me put 1 here, then Q02, subscript n F2, right.

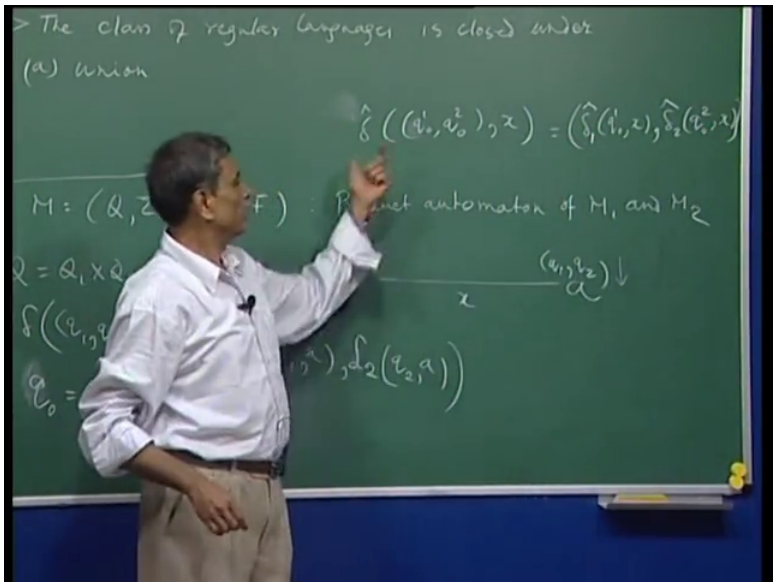
And one of the classes that we had right in the beginning, we had considered taking the product of these 2 sets of states. So let me do it here, sometimes the new machine M is called the product automaton, taking the product of these 2 DFAs. So this has a set of states and that set is the cross product of the 2 DFA set of states Q1 cross Q2. And what is the idea for this product automaton, let me write this, idea is so this is the product of, or a lot of automaton, we did not use this term, let me use it now of M1 and M2 and that we can define in the following manner.

The idea is this machine M will keep track of what happens to both the machines M1 and M2 at the same time. And it does in a very straightforward simple way. In each state it keeps track of the 2 states of the 2 machines and now when a symbol comes, suppose I define this Delta in this way, Delta now see this state, set of states is of pairs, right so each state here is a pair, so let me say Q1, Q2, is not it. A state here is an element of Q and each element here is a pair, so I wrote like that Q1, Q2. And where the 1st of the pair is a state of M1 and the 2nd is the state of M2, right.

And now suppose I define this Delta in this way which we had seen earlier, this is supposed to give me, Delta is supposed to give me a state of this machine M, the product automaton and suppose we define it as Delta 1 of Q1, a and Delta 2 of Q2, a. So Delta1 of Q1, a gives me the state the machine M1 will be in when it sees a symbol a in state Q1. And similarly Delta 2 Q2, a defines the set in which the machine M2 will be when it scans the symbol a on state Q. So you see what is happening, you can see that suppose after scanning a part of the string x, the 2 machines, the machine M1 was in Q1 and the machine M2 was in Q2 and the product automaton or the goal of the product automaton is that it is keeping track of its head, that means in its set of states, what is happening with the 2 individual machines M1 and M2.

(Refer Slide Time: 11:20)

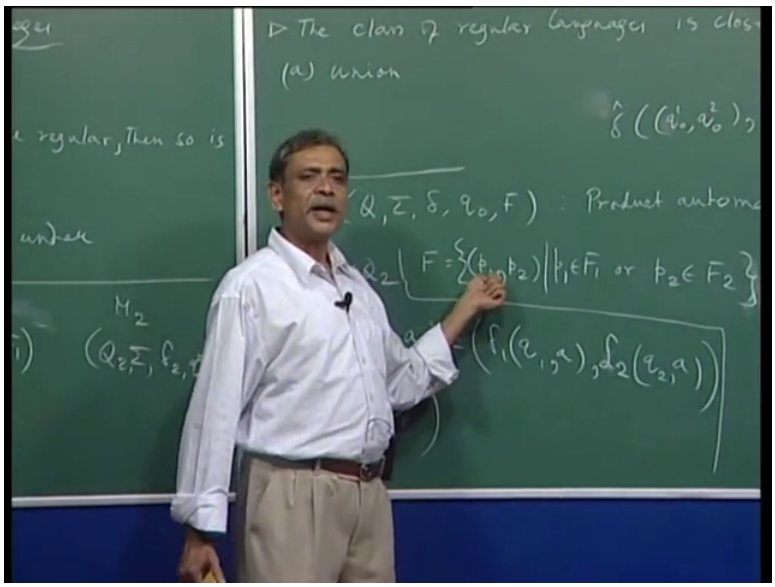


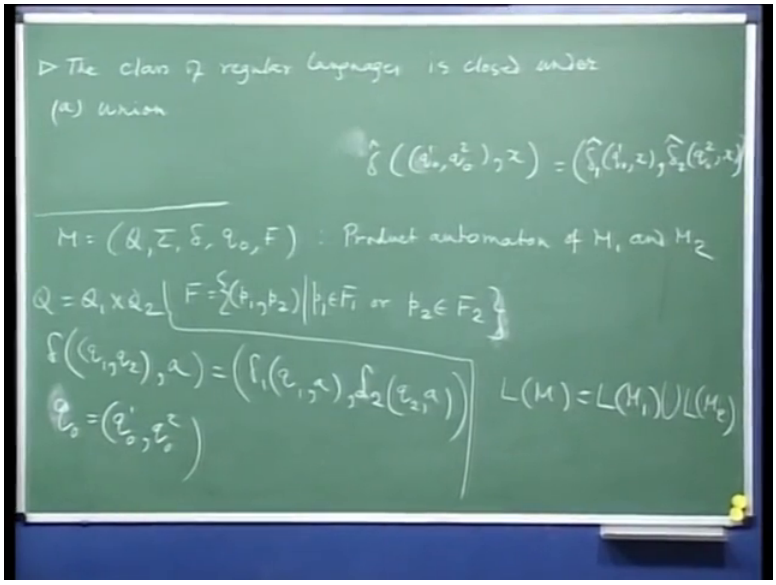


So since machine M1 was in state Q1 at the machine M2 was in state Q2 and now let us say this symbol comes, then at this point clearly this will be the state in which machine M will be there, the product machine. So if you define Q and Sigma this way, Q and delta this way, Sigma is of course, the same Sigma as both M1 and M2, they have. Then Q0 is of course very simple, the initial state is what, initially the 2 machines are M1 and M2, so Delta0, I am sorry the state Q0 is the pair Q0 1 and Q0 2. The index or the suffix 1 mean this is the initial state of machine M1 and the suffix 2 mind that is the initial state of machine M2.

So initially the product machine will be in this state and now it is very easy to see as we had argued earlier that the machine M on scanning a string x we can see Delta hat of for this machine, so let us say Q0, Q01 and 2 on scanning x, clearly it is going to be easy to see Delta hat of Q0 x and Delta hat of, this is one and this is Delta 2 of Q0 2 x. I am saying the same thing in terms of the symbol Delta hat but what we are saying is the machine M on seeing x, it is going to be in a state whose 1st component is a state in which the machine M1 would have been on seeing the string x and whose 2nd component is the state in which the machine M2 would be on scanning the same string x.

(Refer Slide Time: 13:55)

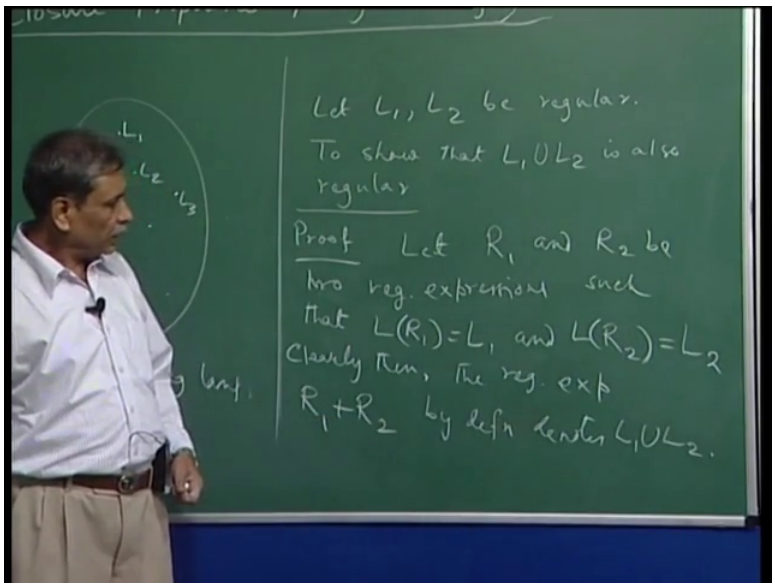




So then it was easy, in fact that is what we did for, if I now define F as F as P1, P2 such that P1 is in F1 or P2 in F2, remember that F1 is the set of final states of the machine M1, F2 is the final states of the machine M2. Now what does it mean, if this product machine is in one of these states F, that means either the machine M1 has released one of its final states of the machine M2 has reached one of its final states, so that means therefore any string which takes the machine M1 to a final state or take the machine M2 to another final states of of course M2, then all such strings will be accepted by this machine M.

So therefore clearly the language accepted by this machine M is language accepted by M1 union language accepted by M2. So if you take any 2 regular languages, of course there will be 2 DFAs corresponding to these 2 regular languages, do this construction to obtain the product machine, define the set of final states of this machine in this manner and you have another DFA which accept the union of the 2 languages. So therefore we can say the class of regular languages is closed under union. But let me give you the same proof more simply now, using the notion of regular expressions.

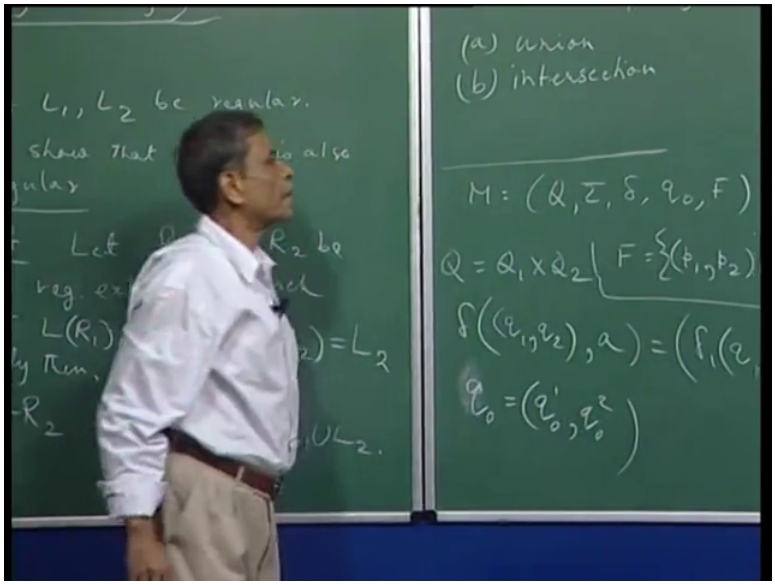
(Refer Slide Time: 16:32)

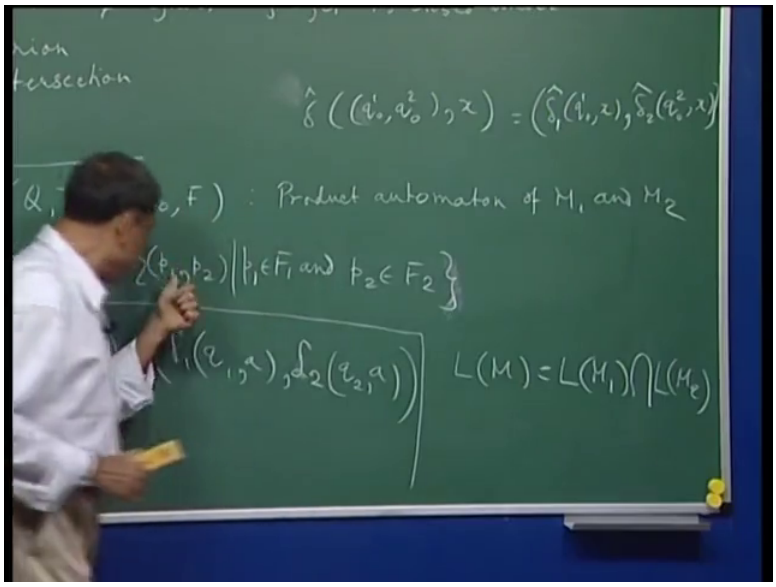


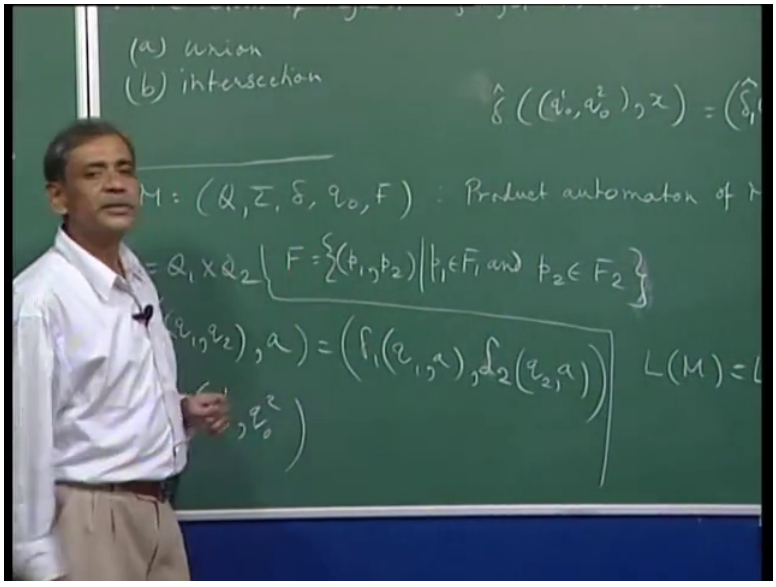
What do we want to prove that let L1, L2 be regular to show that L1 union L2 is also regular. Now that we know the notion of regular expressions and we also know the fact that for every regular language there is a regular expression which denotes that regular language. We can use that fact to prove this much more simply than what we did there. So let me give you the proof using regular expressions. Since L1 and L2 are regular, I can claim the existence of these 2 regular expressions. R1 and R2 be 2 regular expressions such that language denoted by R1 is L1, why can it, why can I see it?

I can say this because L1 is regular and we have proved that for every regular language there is a regular expression which denotes that regular language. And language denoted by the regular expression R2 is L2. Then by the very definition of regular expressions, clearly then the regular expressions R1 + R2 by definition denotes L1 union L2. That is the very definition of the operator + for regular expressions, right. Therefore you see this is a simpler proof for that but that construction is useful when I want to show some other closure properties.

(Refer Slide Time: 19:46)





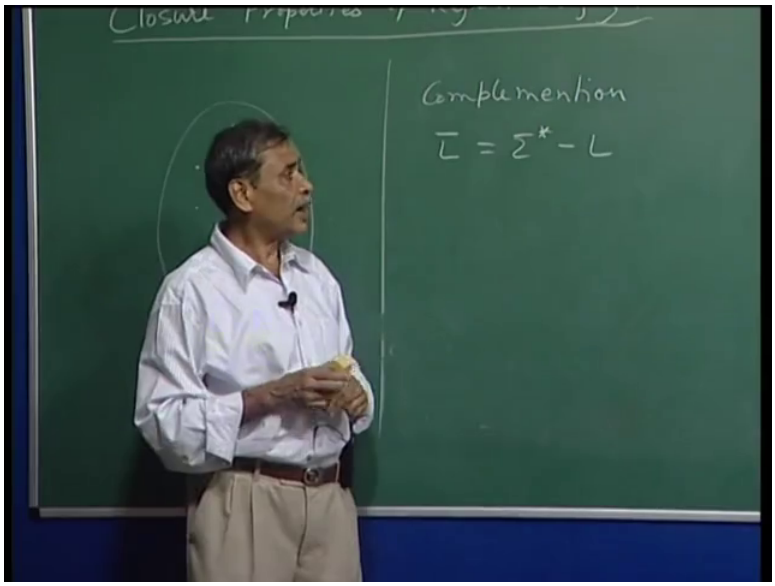


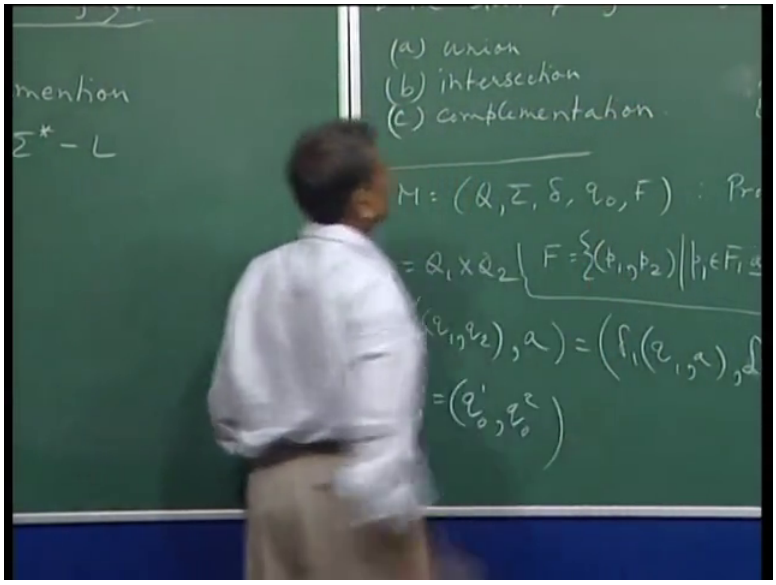
In fact the very next closure property that we would like to consider will be intersection. So here what we would like to prove, that if we are given 2 regular languages L1 and L2, their intersection, L1 Intersection L2 is also regular. Suppose we managed to prove this, then clearly I can say that the class of regular language is closed also under intersection. Now as it happens, this kind of proof through regular expressions will not work as simple as that because we do not have any regular expression operation corresponding to intersection.

However this product automaton idea will work. What is class of, what is the set of strings which are there in this set L1 Intersection L 2, these are precisely those things which are both in L1 as well as in L2 by definition of intersection. So what would such strings to do the machines M1 and M2? Each such string which is both in the language accepted by the machine M1 as well as in the language accepted by M2, all such strings will take the product automaton to a pair of states where both the states are such that P1 is in F1 and P2 is in F2.

That is all I need to do in the definition, that is all the change I need from the union, definition of the product automaton to the you know obtaining a product automaton for the intersection of the languages, this Delta definition remains same, Q0 definition remains same and now because of this change, this is going to be the language accepted by this product automaton where the set of final states is defined in this manner. Just change the previous Or to And, that machine M is going to accept intersection of these 2 languages. So you see the product automaton became useful in this case.

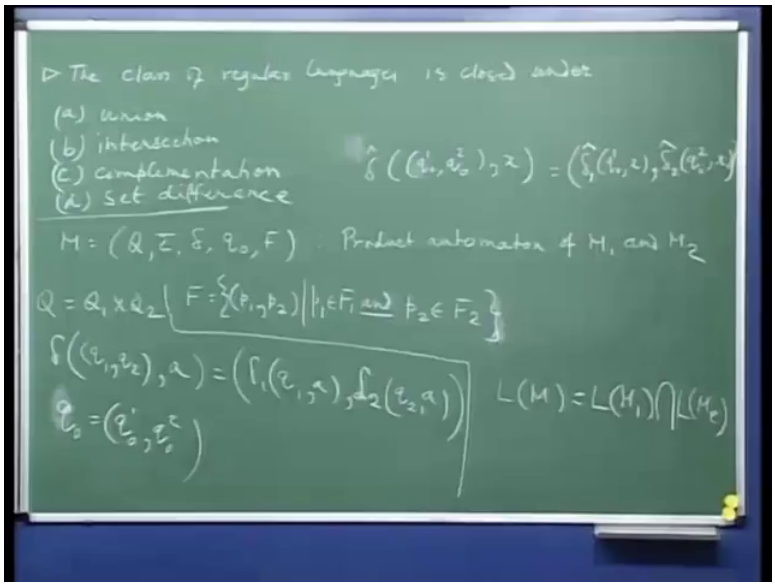
(Refer Slide Time: 23:06)





And the what about, you know there are other Boolean Boolean operations. So let me take another one which is complementation. Now what does it mean, that I take a language L, the complement recall was defined to be Sigma Star - L, this is a unary operation as opposed to union and intersection with binary operation on the class of regular languages. Here I take any language, I perform this operation of complementation, I get a new language and we say that this last is closed under complementation if whenever, sorry, this is Sigma Star - L, whenever L is regular, its complement is also regular.

(Refer Slide Time: 26:03)

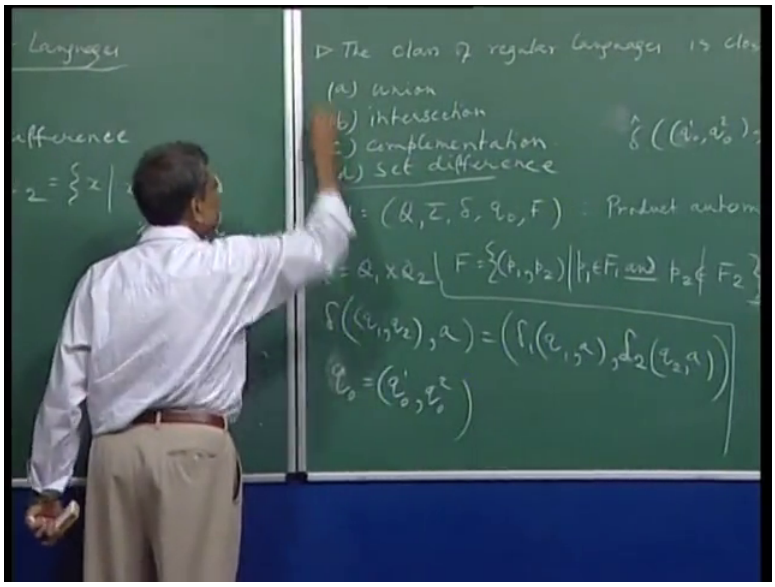


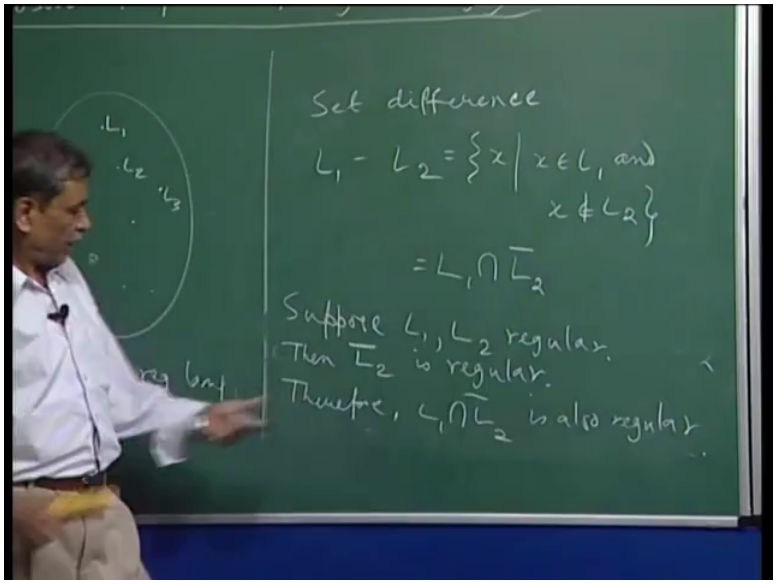
In fact that is the case and how do we know that, and again this is a fact, something we have seen before. If I take a DFA for health, I swap the final states, the accepting states with the non-accepting states, the resulting DFA will accept precisely the complement of L, we have

seen that. So let me therefore say the class of regular languages is also closed under complementation. Now there is a binary operation which is something obtained by notion of difference of 2 sets and that is set difference, right.

If you take L1 and L2, 2 languages which are of course 2 sets of strings, L1 - L 2 by definition is the set of all strings such that x is in L1 and x is not in L2. Right. So again it is the case, that the class of regular languages is closed under set difference, which is a binary operation and that is something which is again immediately obvious from this product machine construction, so again we just need to change the set of final states and now you will say what, P1 is in F1 and P2 is in F2. So all those strings x which takes the 1st machine, that is the machine for the language L1 to one of its final states but does not take the 2nd machine to one of its final states.

(Refer Slide Time: 27:04)





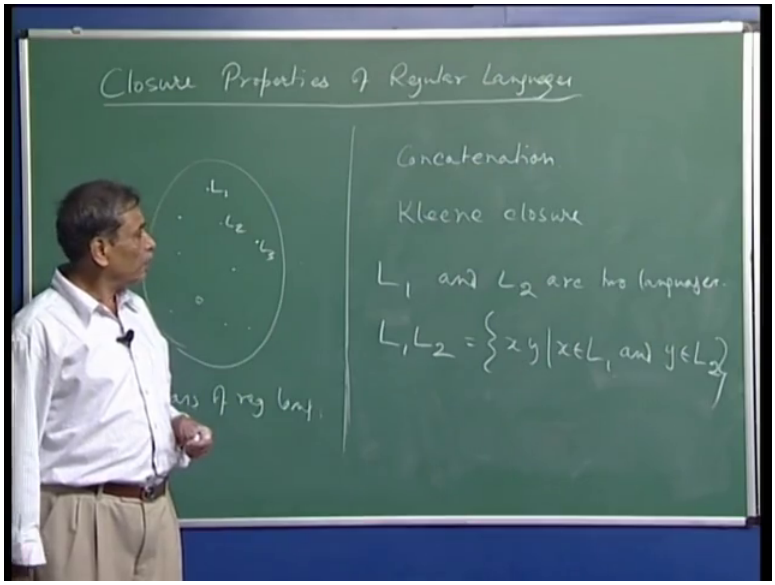
That means the 2nd machine does not accept the spring but the 1st machine accepts. That means if the 1st machine was for L1 and the 2nd machine was for L2, therefore this machine will accept all strings which are in L1 but not in L2. Actually now that we have done these, we could have said it more simply this result without going through the notion of product automaton. Because, you see there is another way of saying this would be what? L1 Intersection L2 complement, is not this. Which are the strings which are here, all those things which are in L1 as well as in L2 complement? That means all those things which are in L1 but not in L2.

However since we have seen that regular languages are closed under complementation, so if I take a regular language L2, I do the complementation, I get a new language, but that is also regular. Why we are saying that suppose L1 and L2 are regular, let me give that argument once more, then we have already proved that L2 complement is also regular. And that follows from the fact that the class of regular languages is closed under complementation. So since L2 is regular, L2 complement is regular, therefore L1 Intersection L2 complement is also regular. Why, because again the same thing that I have 2 regular languages L1 and L2 complement and I am taking their intersection.

And regular languages are closed under intersection, so therefore the new language I get is also regular. Therefore the conclusion is that class of regular languages is closed under set difference. Once more, because I would take any 2 regular languages, I consider the difference L1 - L2 and that will be regular because of this argument. See all these operations union, intersection, complementation, set difference, these operations are defined concepts are, however there are certain operations which are defined because these are sets of strings,

in fact we know a language is a set of strings and each such string in the set will be finite strings.

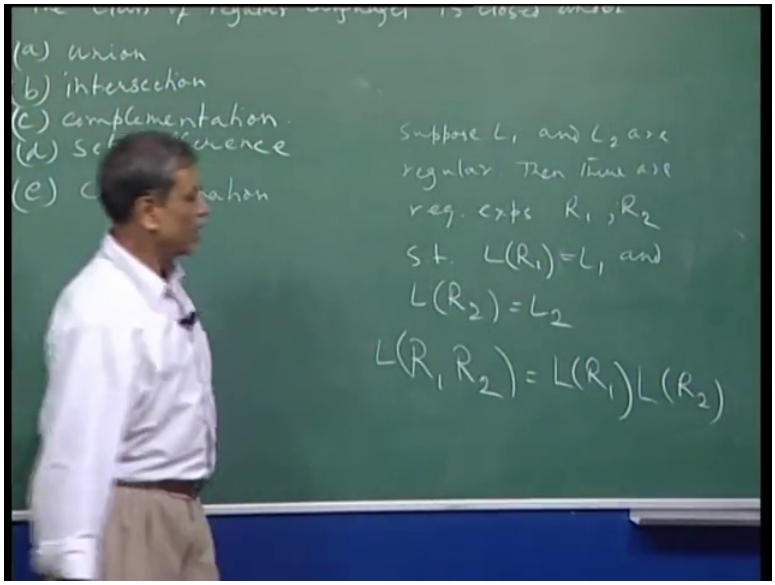
(Refer Slide Time: 30:30)



And you have seen some of these 2 most important concatenation and the Kleene star or Kleene closure, we will remind ourselves of this definition once more. That is, we see that we have 2 languages, L1 and L2 are 2 languages, remember what was the definition of the language obtained by concatenating these 2 languages, by definition it was set of all strings x, y such that x is in L1 and y is in L2. In other words the concatenation is a binary operation on class of languages obtained by, the new language is obtained by taking a string from L1, taking another string from, another string or one string from L2 during the concatenation and these are the kinds of strings which will be there in the language which is the concatenation of L1 L2.

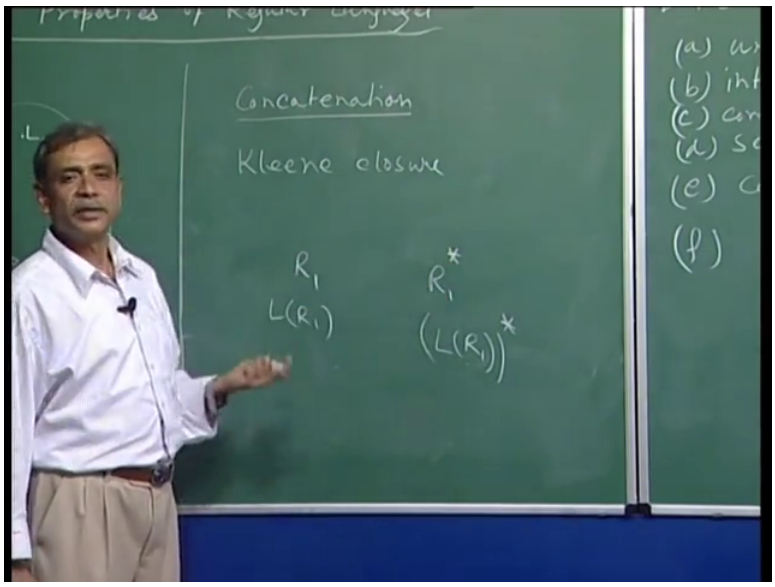
Now can we say immediately that the class of regular languages is closed under concatenation, the binary operation concatenation? Yes, indeed because again just think of regular expressions. Since to prove this that the class of regular languages is closed under concatenation, again we appeal 2 regular expressions that will give us the simplest proof.

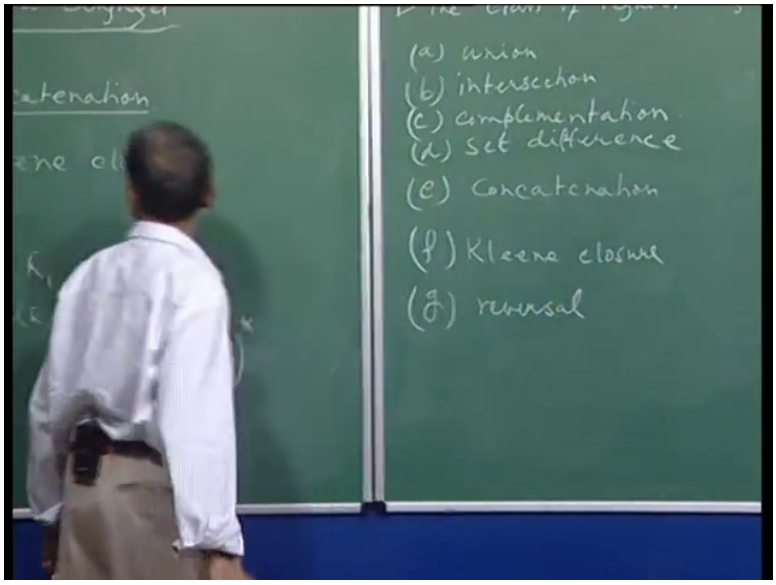
(Refer Slide Time: 32:27)



So what we will say that suppose L1 and L2 are regular, then there are regular expressions R1, R2 such that the language denoted by R1 is L1 and the language denoted by R2 is L2, we know this is, again, the idea is for every regular language there is a regular expression, which denotes that regular language. And what is the regular expression R1 R2, we know this that this regular expression will denote the language obtained by concatenating the 2 languages L R1, language denoted by R1 and language denoted by R2. Again this is the definition of concatenation for regular expressions.

(Refer Slide Time: 34:28)

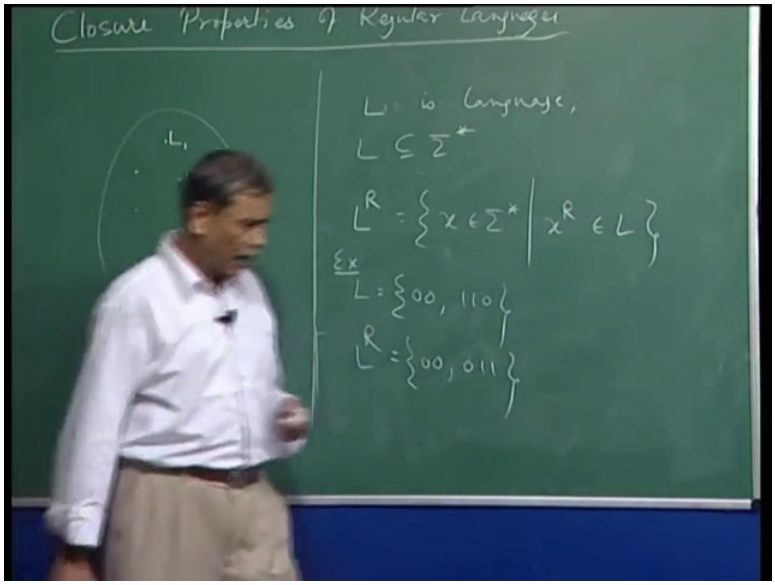




So since I have a regular expression for the language L R1, L R2 concatenated, therefore I claim that the regular languages, the class of regular languages is closed also under concatenation. Same with Kleene closure, that proof will be very similar, only thing then we will appeal to the fact that if R1 is a regular expression, then by definition so is R1 star such that the regular the regular language denoted by this is the closure, Kleene closure of the language L R1 where L R1 is the language denoted by this regular expression. If I take any regular expression R1 and I take star of that, of course that gives me the language which is the closure of the language denoted by R1.

So therefore again the class of regular languages is closed under Kleene closure. There is one more example of closure that we have seen earlier, which is the operation of reversal. If we see this, that this operation is a binary operation on the class of regular languages, this is also a binary operation on the class of regular languages, this is a unary operation on the class of regular languages, take one language, complement it you will get another language. This is a binary operation, this is a binary operation, you concatenate 2 languages and this is again a unary operation, you take a language L and then you top of it star, so that is a unary operation.

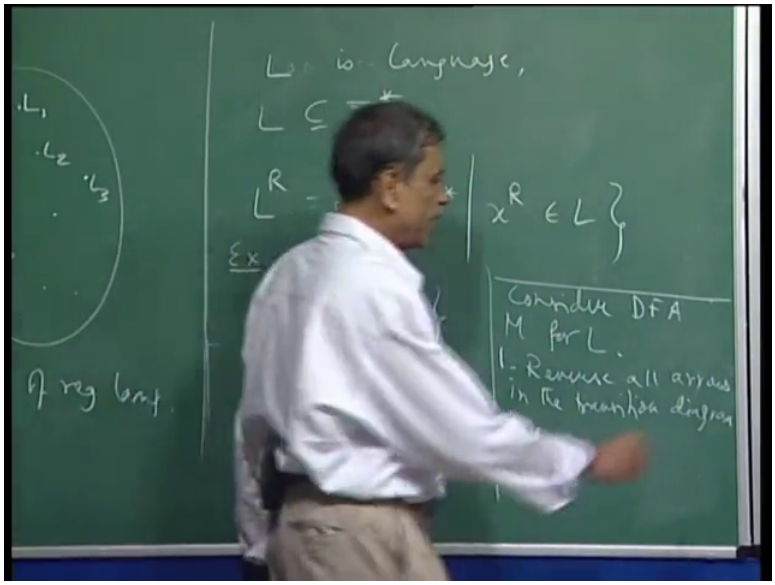
(Refer Slide Time: 36:27)

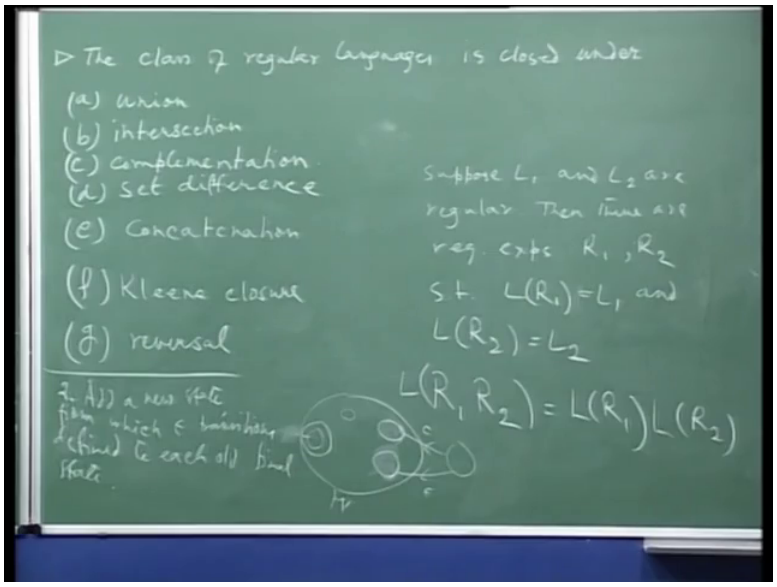


This is again going to be a unary operation. And let me define that once more. So L is a language and let L in the over the alphabet Sigma. Then what is LR, reversal of L, we will write this way, is the set of all strings x in Sigma star such that when you reversed this string XR, on reversing x to get XR, that is in language L. We have seen this but let us remind ourselves once more by taking a very simple example. Supposing L is the 2 strings 00 and 110, let me make it, just very simple language with 2 strings.

Now what we are saying, then LR would be, we are saying that you take a string from here, reverse it, write the string the other way round, from left to right you write it from right to left. For the 1st string of course you will get the same string back on reversal, for the second string you will get to 011. And this constitutes, these 2 strings will constitute the language which is the reversal of language L. I like to remind you what we did to show that if L is regular, then so will be reversal language LR.

(Refer Slide Time: 38:28)





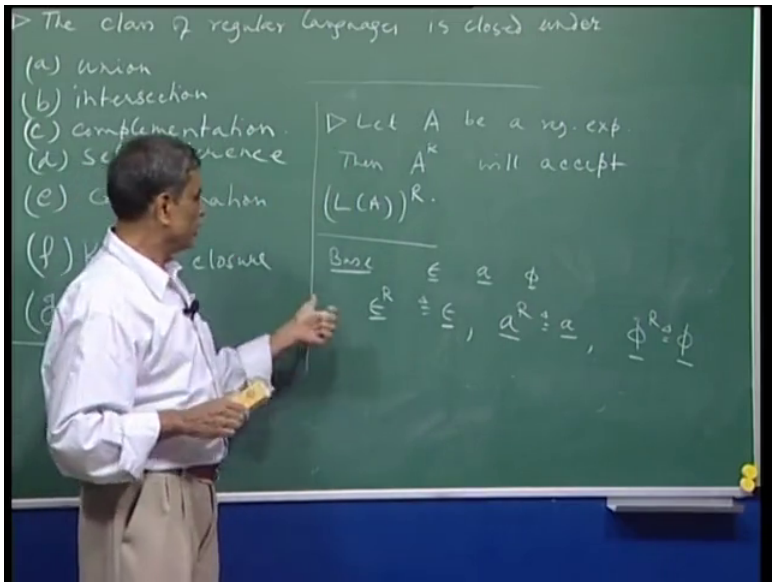
Briefly what we did was, consider the DFA M for L, we are calling what we did to show that this L is regular, then so will be LR. We say that look, consider a DFA for L, we call it M and there what you do, very briefly I am trying to recall what we do is reverse all arrows in the transition diagram of M, that is one thing that you do. 2nd thing that we will do is, a picture would be better. So this was Q0 in the of M and let say we have number of final states, the 2nd thing that we do is add a new state, right.

And there are Epsilon transitions to the old final states from which Epsilon transitions defined to each old final state, they are now of course no longer final states, so they are just ordinary states, only final states will be the old initial states. And this new machine M dash we claim would have accepted the language which is the reversal of the language accepted by the old

machine M. Now that was approved with the we had given then and it was more as an illustration of non (())(40:53). Remember non (())(40:54) came about in 2 ways because you had Epsilon transitions as well as when you reverse the directions of the arrows in transition diagram from the same state, from the same symbol, now there could be more than one state that one could go to.

So we would get a NFA in general and that NFA it is not difficult to see would have accepted the reversal language. But like some of the proofs before, now that we know regular expressions, this proof also can be made much simpler using the notion of regular expressions. So let me provide the proof now. Again we will say that since a language L is regular, you see this is what I am trying to show, that if the language L is regular, then reversal LR will also be regular. Now this proof will be a little different from the previous one because we did not have a direct operation for reversal as we had for concatenation as well as for Kleene star with our regular expression notation.

(Refer Slide Time: 42:33)

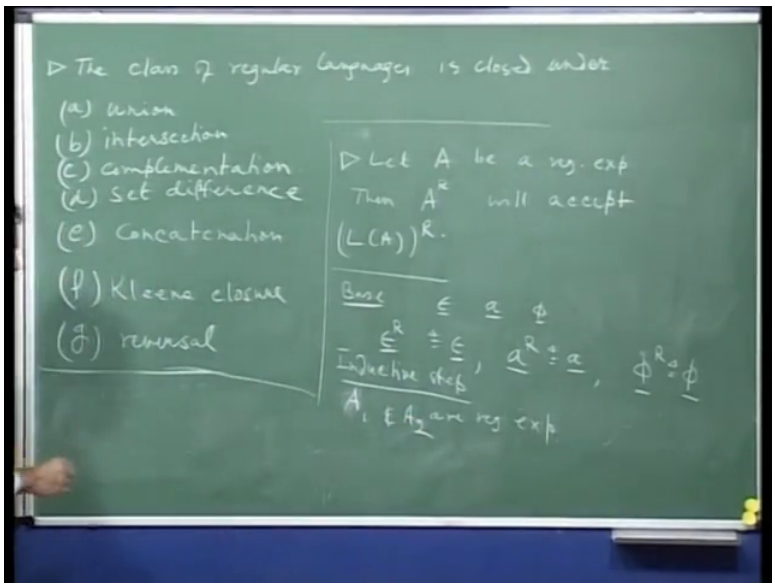


What we can do is that we say that for every regular expression, let me claim this, write it properly, let R be a regular expression, then, now this notation is a little funny because R on top of I am writing R. So let me make this instead of R, let me make it a, let A be a regular expression, then AR, the reversal of the regular expression. Now this is something I am going to define, will accept L of A reversal. But I have not yet defined what I mean by given regular expression what is its reversal. This I will do inductively. See I want to now provide the definition of regular expressions reversal.

And these kinds of proofs we have seen earlier, that I want to claim something, here I am trying to define something overall regular expressions and we will do that inductively, the way regular expressions themselves were defined inductively. So remember the base cases, based cases were that I had the regular expression Epsilon a and phi. Now intuitively what we are trying to do is that if we have a regular expression for the language, I am trying to define the regular expression for the reversal of that language. So I have here Epsilon, this regular expression stands for the language, so let me write it, language denoted by this regular expression is of course this.

Now what is the reversal language plane from here? Take the empty string in reversal, you will get back Epsilon, right. So clearly let me define the reversal of this is nothing but E R. So this is the definition, remember, let me write it, put it this way so that we know that this is the definition. We are defining the reversal of Epsilon to be, sorry, to be itself. And the same kind of reasoning will be true for these 2 also. Reversal of the regular expression a is the regular expression a itself by definition and reversal of regular expression phi is again by definition is going to be phi, all right.

(Refer Slide Time: 46:19)



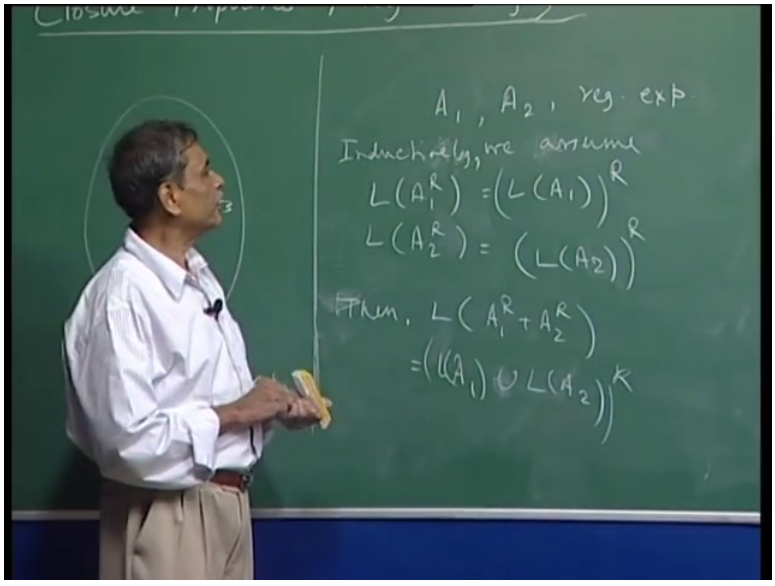
Now what I have done, that I have defined for the base cases of regular expression definition what the reversal means. Now the inductive step, recall that we built regular expressions also inductively, there after the base cases I had 3 ways of getting new regular expressions out of all regular expressions. So we say that if the R, let me now call it again A1 and A2, A1 and A2 are regular expressions, right. Then I know A2 here, then I know that there are 3 separate ways I can build bigger regular expressions out of A1 and A2. And they were A1 and A1 +

A2 concatenation A1 A2 and let say A1 star. So this is of course unary and this is these 2 are binary.

Now let us say inductively we have defined A1 R and A2 R, right. These are already defined, then I define the reversal of this to be, you take the regular expression A1, take its reversal and now put A2, take its reversal and similarly in this case, and this is again a definition, so the definition for reversal of concatenation language is going to be what? Then I will come to the reason why we are defining it this way. A2 reversal concatenated with A1 reversal and A1, the reversal of Kleene closure is going to be, we are defining it as, take A1 and its reversal and then do the closure.

Now the claim that we have is, what we have done so far. We have A and now we are defining something called reversal of the regular expression and what we will finally claim that this regular expression will denote the language which is the reversal of the language denoted by the regular expression A. And we will very clearly see this once we understand the motivation behind defining the reversal this way for regular expressions. For the base case I have already given you the idea, right. This is what I am trying to ensure. Now let us take this case.

(Refer Slide Time: 49:53)



Suppose I have A1 and A2, 2 regular expressions, then inductively we have, or let us say we assume language denoted by reversal of A1, the regular expression A1 is, take the language denoted by A1, its reversal, now this is no longer a definition, this is a claim because inductively we are making this claim, inductively, we assume this and similarly L A2 R is the

language obtained by reversing the language denoted by A2. So let us see A is the language A1 + A2, this is not the language, the regular expression. I have A1 as the regular expression, A2 is the regular equation, and the denoting by this regular expression.

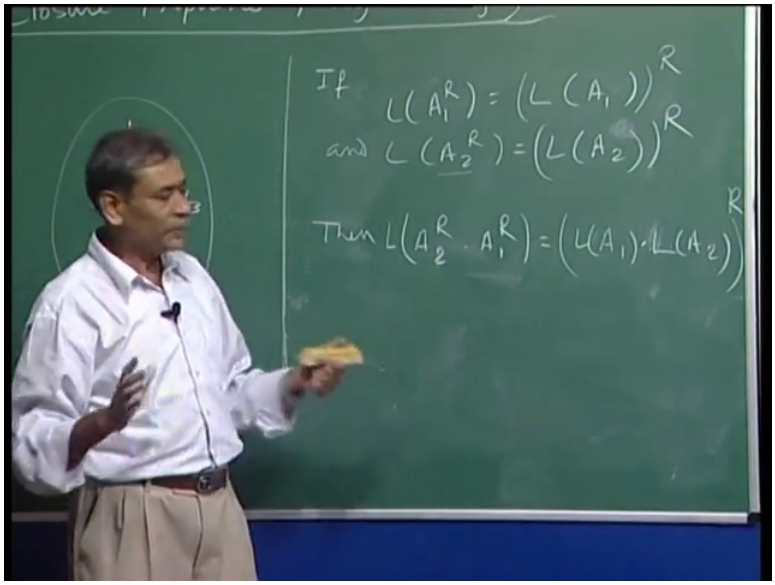
So what is the language, this denotes the language which is denoted by this regular expression and what is its reversal. By definition it is set of all strings, right, which, whose reversal will be either in, let me write it this way, that of all strings x such that XR is in LA1 or XR is in L A2, okay. Remember this simple A, capital A is the regular expression A1 + A2. So do you see what we are doing here? Inductively the language denoted by that A1 R is the reversal of the language L A1 and the language denoted by A2 R is the reversal of the language L A2.

So if I just take the + operation for these 2 regular expressions, then clearly I will have this, because by definition the language denoted by these regular expressions will be the set of all strings which are either in this regular expression, in the language denoted by this regular expression or denoted by this regular expression. Right. So therefore the reversal, we defined in this manner, and now you know inductively I can claim that is A1 and A2 are regular expressions, then the, this regular expression A1 reversal and A2 reversal will indeed denote the language, let me let me write it like this.

L, right, denotes the reversal of, here this is union, so what we are saying is if you take the 2 regular expressions A1 R and A2 R, simply put a + between them, the new regular expression that you get, that will generate all strings which are either the reversal of a string in A1, the language denoted by A1 or of a string denoted by A2. So therefore what I am claiming now is that if I have 2 regular expressions A1 and A2, then their reversals, their individual reversals, if I now just put this, then how did I define this? For this, you know it was defined this way and this is going to be the language which is the reversal of the union of the 2 languages denoted by A1 and A2. Right.

So you see this is a simpler of this, we take this case, concatenation. What we did was that by definition I had a regular expression A1 A2, the next reversal I defined it as, take the reversal of the expression which is the regular expression of the 2nd one A2 reversal followed by the regular expression reversal of A1. Now again it is not difficult to see that if you inductively assume that A1 R denotes the set of all strings which are in the language, which constitute the language, reversal of LA1.

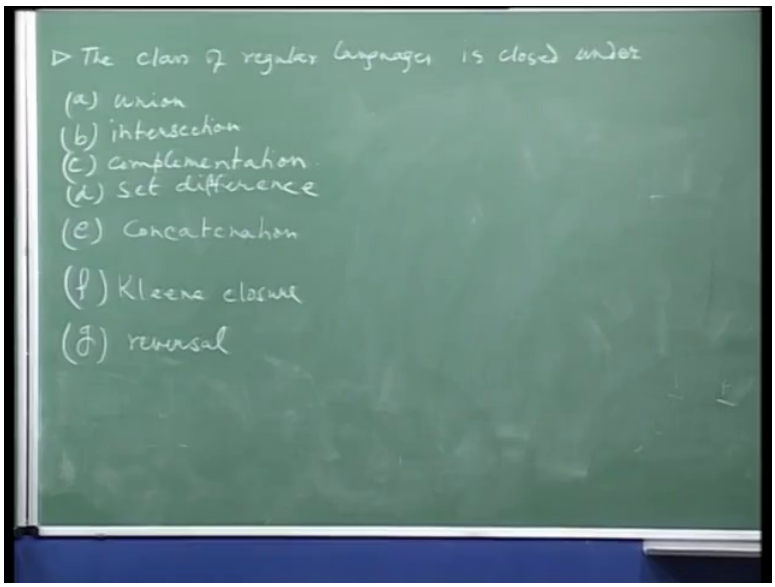
(Refer Slide Time: 56:23)



So assume that is the language and L A2 R, language denoted by this regular expression is language, sorry, if this is the language, we take reversal here, then the reversal operation is on the language, right. When I do this, I am getting a syntactic, I am referring to the syntactic expression, which we defined in this manner. Now if this holds, then A1, so let me claim this A2 R A1 R, okay the language denoted by this is indeed the language obtained by A1 A2, L of A1, L of A2 and I concatenate them and now take the reverse. This is also fairly easy to see. See what is happening is when you take the concatenation of 2 languages and then you do reverse that language, that means initially we will get a string which is in the reversal of A2, right.

And next, that is what you were getting here, A2 R, A2 R inductively we will generate all strings which are in the language, this followed by, right, you have reversed it, right. So on the reversal finally the last part will be the string in the language denoted by A1. And therefore this will be the situation. So similarly for Kleene closure and therefore what I am trying to say is that using regular expression we can also prove the fact that class of regular languages is closed under reversal. So what we have done in today's lecture is that 1st of all we defined an important notion called closure of a class.

(Refer Slide Time: 59:35)



And then we considered the class of regular languages and the prove, in fact most of the results were already known by us as examples of DFA, NFA, etc. that the class of regular languages is closed under union, intersection, complementation, set difference, concatenation, Kleene closure as well as close on the reversal. Then we have found another interesting thing in this process, that the fact that the class of regular languages can be captured by regular expressions, that helped us improving some of these results in a manner which is different or more simple.

So that will be the case always that since for the same class I have different ways of looking at, either as regular expressions or as language or as you know or through automaton. So sometimes automaton will give us easier proofs, sometimes regular expressions will give easier proofs for this. We will see some reason why other than of course the fact that these are nice things to know, why closure properties are useful.